

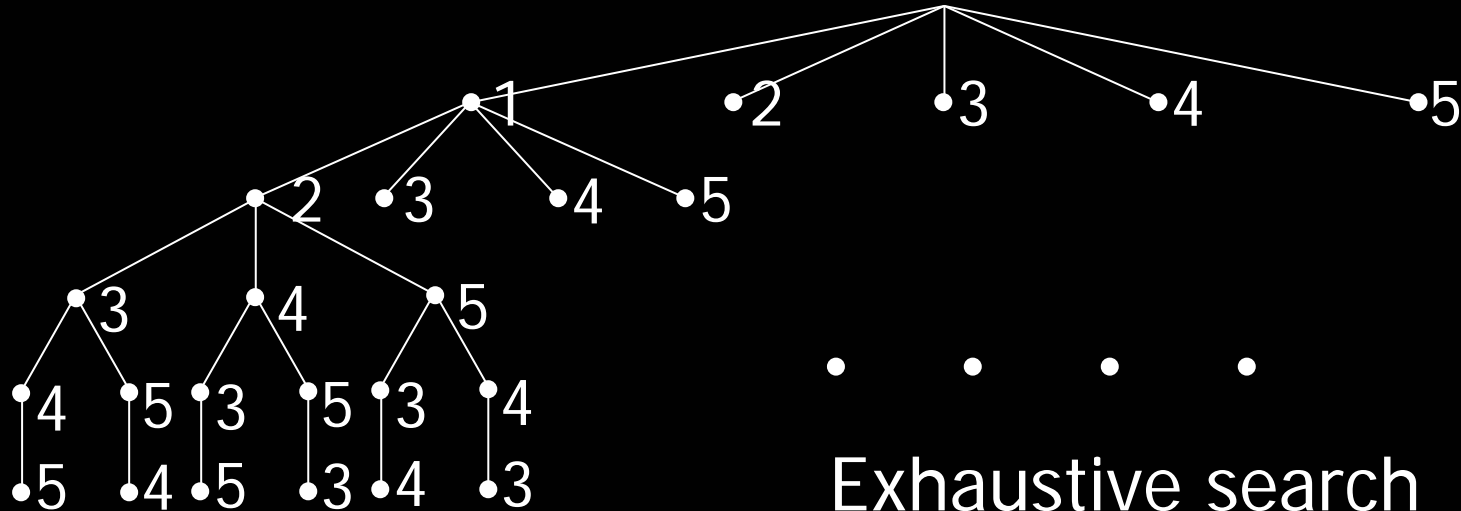
Permutation, Combination, and Related Problems - backtracking algorithms

Pei-yih Ting

Introduction

- You might need to generate permutations or combinations to enumerate all possible configurations in order to find the optimal solutions (bottomline solutions) of some problems
- Procedural programming is about data processing.
- To design the program:
 - figure out how the data/configuration changes
 - represent the data suitably in a program
 - figure out the “process” that transforms the data step by step

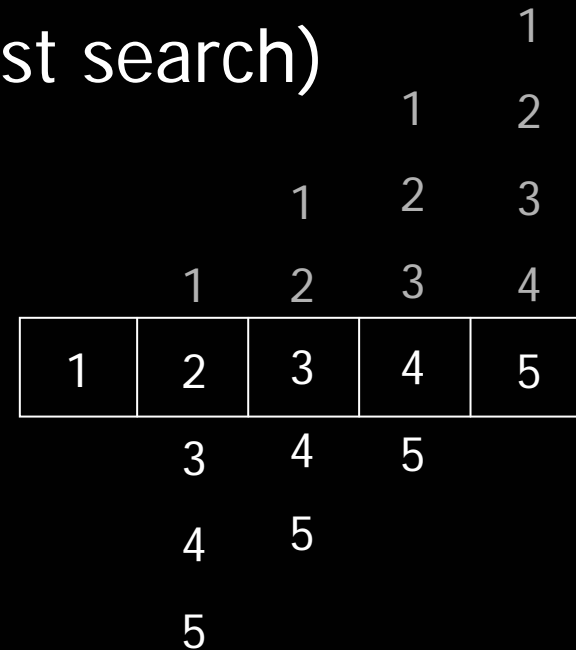
Generating Permutations



Exhaustive search
(depth first search)

$5! = 120$
permutations

- 1 2 3 4 5
- 1 2 3 5 4
- 1 2 4 3 5
- 1 2 4 5 3
- 1 2 5 3 4
- 1 2 5 4 3
- ...

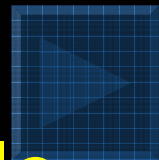


n-layer **for** loop Implementation

```
int i1, i2, i3, i4;
for (i1=1; i1<=4; i1++) {
    for (i2=1; i2<=4; i2++) {
        if (i2 == i1) continue;
        for (i3=1; i3<=4; i3++) {
            if ((i3==i2)||(i3==i1)) continue;
            for (i4=1; i4<=4; i4++) {
                if ((i4==i3)||(i4==i2)||(i4==i1)) continue;
                printf("%d %d %d %d\n", i1, i2, i3, i4);
            }
        }
    }
}
```

i1	i2	i3	i4
----	----	----	----

This is a quick implementation but **NOT scalable**.



2-layer **for** loop Implementation

- Consider this simplified loop without collision constraints

```
int i1, i2, i3, i4;
for (i1=1; i1<=4; i1++) {
    for (i2=1; i2<=4; i2++) {
        for (i3=1; i3<=4; i3++) {
            for (i4=1; i4<=4; i4++) {
                printf("%d %d %d %d\n",
                    i1, i2, i3, i4);
            }
        }
    }
}
```

i1	i2	i3	i4
1	1	1	1
1	1	1	2
1	1	1	3
1	1	1	4
1	1	2	1
1	1	2	2
1	1	2	3
1	1	2	4
1	1	3	1

...

- Is there other *scalable* program structure to generate the same (i1, i2, i3, i4) counting-up sequence? **yes**

Equivalent 2-layer **for** loop

```
void nextIndex(int index[], int n) {  
    int i;  
    for (i=n-1; i>0; i--)  
        if (index[i]<n)  
            { index[i]++; return; }  
    else  
        index[i] = 1;  
    index[0]++;  
}
```

index

1	1	1	1
1	1	1	2
1	1	1	3
1	1	1	4
1	1	2	1
1	1	2	2
1	1	2	3
1	1	2	4
1	1	3	1

```
int index[4], n=4;
```

```
for (i=0; i<n; i++)
```

```
    index[i] = 1;
```

```
    for (; index[0]<=n; nextIndex(index, n))
```

```
        printArray(index, n);
```

Scalable

2-layer **for** loop for Permutation

```
int index[4], n=4;
for (i=0; i<n; i++)
    index[i] = i+1;
for (; index[0]<=n; nextIndex(index, n))
    if (isValid(index, n))
        printPerm(index, n);
```

```
int isValid(int index[], int n) {
    int i, j;
    for (i=0; i<n-1; i++)
        for (j=i+1; j<n; j++)
            if (index[i]==index[j])
                return 0;
    return 1;
}
```

```
void nextIndex(int index[], int n) {
    int i;
    for (i=n-1; i>0; i--)
        if (index[i]<n)
            ...
            index[0]++;
}
```

index	1	2	3	4
	1	2	4	3
	1	3	2	4
	1	3	4	2
	1	4	2	3
	1	4	3	2
	2	1	3	4
	2	1	4	3
			...	

Another Implementation

	num[0]	num[-1]	num[n-1]		
	0	0	1	2	3
		0	1	3	2
		0	2	1	3
		0	2	3	1
		0	3	1	2
		0	3	2	1
		1	0	2	3
			...		

Another Implementation

```
void print(int m, int num[]);
void next(int n, int num[], int it);
int main() {
    int i, n=4, num0[] = {0, 0, 1, 2, 3}, *num=num0+1;
    while (num[-1]==0) {
        print(n, num);
        i=n-1;
        do
            next(n, num, --i);
            while (i>=0&&num[i]==n);
            while (++i<n)
                num[i]=-1, next(n, num, i);
        }
    }
    return 0;
}
```

	num[0]	num[-1]		num[n-1]	
	0	0	1	2	3
		0	1	3	2
		0	2	1	3
		0	2	3	1
		0	3	1	2
		0	3	2	1
		1	0	2	3
			...		

Another Implementation

```
void print(int m, int num[]);
void next(int n, int num[], int it);
int main() {
    int i, n=4, num0[] = {0, 0, 1, 2, 3}, *num=num0+1;
    while (num[-1]==0) {
        print(n, num);
        i=n-1;
        do
            next(n, num, --i);
            while (i>=0&&num[i]==n);
            while (++i<n)
                num[i]=-1, next(n, num, i);
        }
    }
    return 0;
}
```

```
void print(int n, int num[]) {
    for (int i=0; i<n; i++)
        printf("%d ", num[i]);
    printf("\n");
}
```

	num[0]	num[1]	num[2]	num[3]	
num[-1]	0	0	1	2	3
	0	0	1	3	2
	0	2	1	3	
	0	2	3	1	
	0	3	1	2	
	0	3	2	1	
	1	0	2	3	
			...		

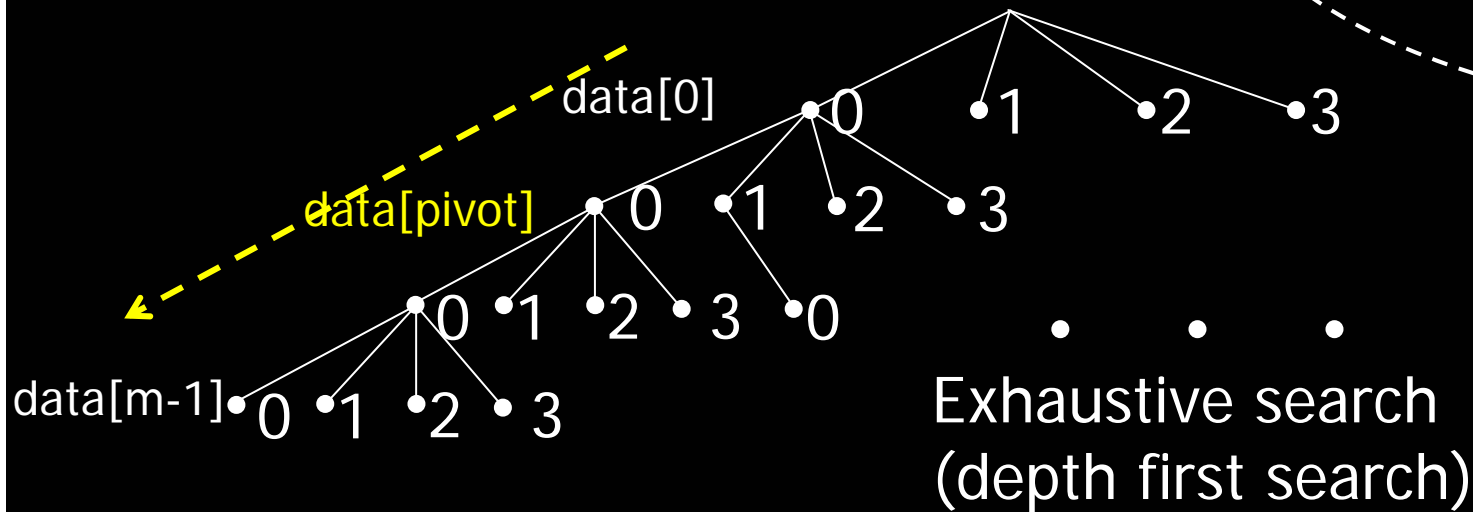
```
void next(int n, int num[], int it) {
    int i;
    for (num[it]++; num[it]<n; num[it]++) {
        for (i=0; i<it; i++)
            if (num[i]==num[it]) break;
        if (i==it) return;
    }
}
```

找到比 **num[it]** 大的下一個可用數字，找不到就設為 **n**

Recursive Counting Up



- 0 0 0 0
- 0 0 0 1
- 0 0 0 2
- 0 0 0 3
- 0 0 1 0
- ...
- 0 0 3 3
- 0 1 0 0
- 0 1 0 1
- 0 1 0 2
- 0 1 0 3
- 0 1 1 0
- ...
- 0 1 3 2
- 0 1 3 3
- 0 2 0 0
- ...
- 1 0 0 0
- ...
- 3 3 3 2
- 3 3 3 3



Recursive Counting Up

```
01 int main() {
02     int data[10];
03     countUp(4, data, 4, 0);
04     return 0;
05 }
06
07 void countUp(int n, int data[], int m, int pivot) {
08     if (pivot >= m)
09         print(data, m);
10     else
11         for (data[pivot]=0; data[pivot]<n; data[pivot]++)
12             countUp(n, data, m, pivot+1);
13 }
```



```
0 0 0 0
0 0 0 1
0 0 0 2
0 0 0 3
0 0 1 0
...
0 0 3 3
0 1 0 0
0 1 0 1
0 1 0 2
0 1 0 3
0 1 1 0
...
0 1 3 2
0 1 3 3
0 2 0 0
...
1 0 0 0
...
3 3 3 2
3 3 3 3
```

Permutation

- If you view the `index[]` as an n -digit n -ary number (the digits set is $\{1, 2, \dots, n\}$), the previous program generates all possible increasing numbers (from $111\dots 1$ to $nnn\dots n$) and eliminates all invalid numbers with repeating digits on the way of counting up.

Permutation

- If you view the `index[]` as an n -digit n -ary number (the digits set is $\{1, 2, \dots, n\}$), the previous program generates all possible increasing numbers (from $111\dots 1$ to $nnn\dots n$) and eliminates all invalid numbers with repeating digits on the way of counting up.
- Problem: **slow** as n gets to 10 or larger. n^n vs. $n!$

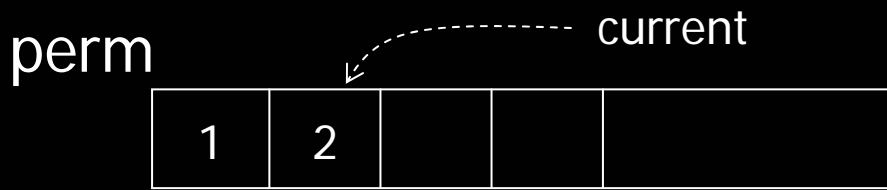
Permutation

- If you view the `index[]` as an n -digit n -ary number (the digits set is $\{1, 2, \dots, n\}$), the previous program generates all possible increasing numbers (from $111\dots 1$ to $nnn\dots n$) and eliminates all invalid numbers with repeating digits on the way of counting up.
- Problem: **slow** as n gets to 10 or larger. n^n vs. $n!$
- Next, you will see another implementation. It tries to generate only valid increasing numbers with the `next()` function. Later, we will have some more algorithms to generate permutations.

```

01 #include <stdio.h>
02
03 void main()
04 {
05     int size, perm[12] = {0}, current=0, solCount=0, i;
06
07     printf("Please input number of elements: ");
08     scanf("%d", &size);
09
10     while (current >= 0)
11     {
12         current += next(size, current, perm);
13         if (current == size)
14         {
15             solCount++;
16             printf("%4d: ", solCount);
17             for (i=0; i<size; i++)
18                 printf("%d ", perm[i]);
19             printf("\n");
20             current = size-1;
21         }
22     }
23     printf("Total %d permutations\n", solCount);
24 }

```



```

26 int next(int size, int pivot, int perm[])
27 {
28     int i, collision;
29
30     while (perm[pivot]++ < size)
31     {
32         collision = 0;
33         for (i=0; i<pivot; i++)
34             if (perm[pivot] == perm[i])
35             {
36                 collision = 1;
37                 break;
38             }
39         if (!collision) return 1;
40     }
41     perm[pivot] = 0;
42     return -1;
43 }

```




start of program

0				
---	--	--	--	--

1	0			
---	---	--	--	--

start of program

0				
---	--	--	--	--

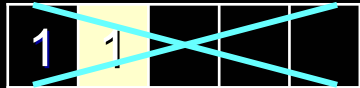
1	0			
---	---	--	--	--

start of program

1	1			
---	---	--	--	--

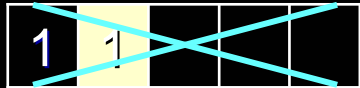


start of program





start of program



0				
---	--	--	--	--

1	0			
---	---	--	--	--

start of program

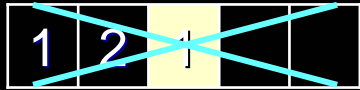
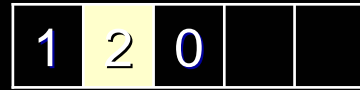
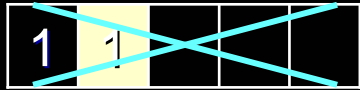
1	1			
--------------	--------------	--	--	--

1	2	0		
---	---	---	--	--

1	2	1		
---	---	---	--	--

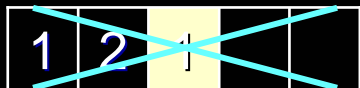
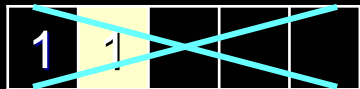


start of program



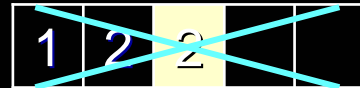
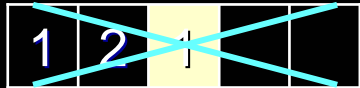
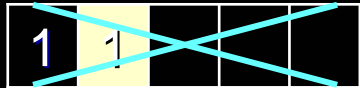


start of program



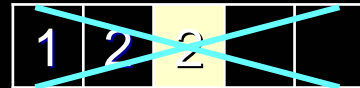
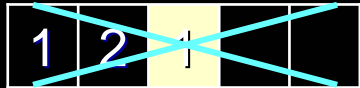
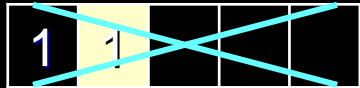


start of program





start of program



0				
---	--	--	--	--

1	0			
---	---	--	--	--

start of program

1	1			
--------------	--------------	--	--	--

1	2	0		
---	---	---	--	--

1	2	1		
--------------	--------------	--------------	--	--

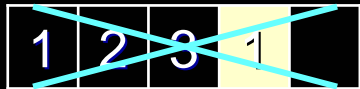
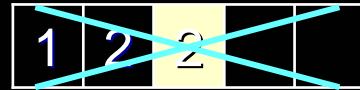
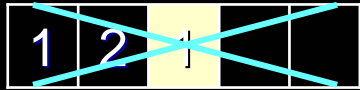
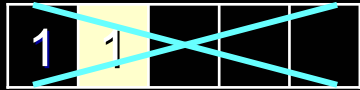
1	2	2		
--------------	--------------	--------------	--	--

1	2	3	0	
---	---	---	---	--

1	2	3	1	
---	---	---	---	--

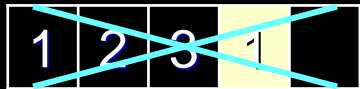
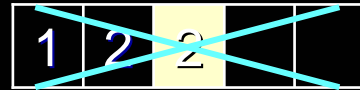
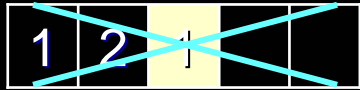
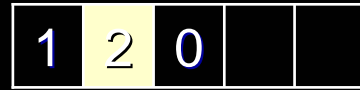
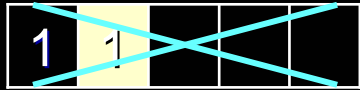


start of program





start of program



0				
---	--	--	--	--

1	0			
---	---	--	--	--

start of program

1	1			
--------------	--------------	--	--	--

1	2	0		
---	---	---	--	--

1	2	1		
--------------	--------------	--------------	--	--

1	2	2		
--------------	--------------	--------------	--	--

1	2	3	0	
---	---	---	---	--

1	2	3	1	
--------------	--------------	--------------	--------------	--

1	2	3	2	
--------------	--------------	--------------	--------------	--

0				
---	--	--	--	--

1	0			
---	---	--	--	--

start of program

1	1			
--------------	--------------	--	--	--

1	2	0		
---	---	---	--	--

1	2	1		
--------------	--------------	--------------	--	--

1	2	2		
--------------	--------------	--------------	--	--

1	2	3	0	
---	---	---	---	--

1	2	3	1	
--------------	--------------	--------------	--------------	--

1	2	3	2	
--------------	--------------	--------------	--------------	--

...

1	2	3	4	0
---	---	---	---	---

0				
---	--	--	--	--

1	0			
---	---	--	--	--

start of program

1	1			
--------------	--------------	--	--	--

1	2	0		
---	---	---	--	--

1	2	1		
--------------	--------------	--------------	--	--

1	2	2		
--------------	--------------	--------------	--	--

1	2	3	0	
---	---	---	---	--

1	2	3	1	
--------------	--------------	--------------	--------------	--

1	2	3	2	
--------------	--------------	--------------	--------------	--

...

1	2	3	4	0
---	---	---	---	---

1	2	3	4	1
---	---	---	---	---

0				
---	--	--	--	--

1	0			
---	---	--	--	--

start of program

1	1			
--------------	--------------	--	--	--

1	2	0		
---	---	---	--	--

1	2	1		
--------------	--------------	--------------	--	--

1	2	2		
--------------	--------------	--------------	--	--

1	2	3	0	
---	---	---	---	--

1	2	3	1	
--------------	--------------	--------------	--------------	--

1	2	3	2	
--------------	--------------	--------------	--------------	--

...

1	2	3	4	0
---	---	---	---	---

1	2	3	4	1
--------------	--------------	--------------	--------------	--------------

0				
---	--	--	--	--

1	0			
---	---	--	--	--

start of program

1	1			
--------------	--------------	--	--	--

1	2	0		
---	---	---	--	--

1	2	1		
--------------	--------------	--------------	--	--

1	2	2		
--------------	--------------	--------------	--	--

1	2	3	0	
---	---	---	---	--

1	2	3	1	
--------------	--------------	--------------	--------------	--

1	2	3	2	
--------------	--------------	--------------	--------------	--

...

1	2	3	4	0
---	---	---	---	---

1	2	3	4	1
--------------	--------------	--------------	--------------	--------------

1	2	3	4	2
---	---	---	---	---

0				
---	--	--	--	--

1	0			
---	---	--	--	--

start of program

1	1			
--------------	--------------	--	--	--

1	2	0		
---	---	---	--	--

1	2	1		
--------------	--------------	--------------	--	--

1	2	2		
--------------	--------------	--------------	--	--

1	2	3	0	
---	---	---	---	--

1	2	3	1	
--------------	--------------	--------------	--------------	--

1	2	3	2	
--------------	--------------	--------------	--------------	--

...

1	2	3	4	0
---	---	---	---	---

1	2	3	4	1
--------------	--------------	--------------	--------------	--------------

1	2	3	4	2
--------------	--------------	--------------	--------------	--------------

0				
---	--	--	--	--

1	0			
---	---	--	--	--

start of program

1	1			
--------------	--------------	--	--	--

1	2	0		
---	---	---	--	--

1	2	1		
--------------	--------------	--------------	--	--

1	2	2		
--------------	--------------	--------------	--	--

1	2	3	0	
---	---	---	---	--

1	2	3	1	
--------------	--------------	--------------	--------------	--

1	2	3	2	
--------------	--------------	--------------	--------------	--

...

1	2	3	4	0
---	---	---	---	---

1	2	3	4	1
--------------	--------------	--------------	--------------	--------------

1	2	3	4	2
--------------	--------------	--------------	--------------	--------------

...

0				
---	--	--	--	--

1	0			
---	---	--	--	--

start of program

1	1			
--------------	--------------	--	--	--

1	2	0		
---	---	---	--	--

1	2	1		
--------------	--------------	--------------	--	--

1	2	2		
--------------	--------------	--------------	--	--

1	2	3	0	
---	---	---	---	--

1	2	3	1	
--------------	--------------	--------------	--------------	--

1	2	3	2	
--------------	--------------	--------------	--------------	--

...

1	2	3	4	0
---	---	---	---	---

1	2	3	4	1
--------------	--------------	--------------	--------------	--------------

1	2	3	4	2
--------------	--------------	--------------	--------------	--------------

...

1	2	3	4	5
---	---	---	---	---

0				
---	--	--	--	--

1	0			
---	---	--	--	--

start of program

1	1			
--------------	--------------	--	--	--

1	2	0		
---	---	---	--	--

1	2	1		
--------------	--------------	--------------	--	--

1	2	2		
--------------	--------------	--------------	--	--

1	2	3	0	
---	---	---	---	--

1	2	3	1	
--------------	--------------	--------------	--------------	--

1	2	3	2	
--------------	--------------	--------------	--------------	--

...

1	2	3	4	0
---	---	---	---	---

1	2	3	4	1
--------------	--------------	--------------	--------------	--------------

1	2	3	4	2
--------------	--------------	--------------	--------------	--------------

...

1	2	3	4	5
---	---	---	---	---

1	2	3	4	6
---	---	---	---	---

0				
---	--	--	--	--

1	0			
---	---	--	--	--

start of program

1	1			
--------------	--------------	--	--	--

1	2	0		
---	---	---	--	--

1	2	1		
--------------	--------------	--------------	--	--

1	2	2		
--------------	--------------	--------------	--	--

1	2	3	0	
---	---	---	---	--

1	2	3	1	
--------------	--------------	--------------	--------------	--

1	2	3	2	
--------------	--------------	--------------	--------------	--

...

1	2	3	4	0
---	---	---	---	---

1	2	3	4	1
--------------	--------------	--------------	--------------	--------------

1	2	3	4	2
--------------	--------------	--------------	--------------	--------------

...

1	2	3	4	5
---	---	---	---	---

1	2	3	4	6
--------------	--------------	--------------	--------------	--------------

0

1 0

start of program

~~1 1~~

1 2 0

~~1 2 1~~

~~1 2 2~~

1 2 3 0

~~1 2 3 1~~

~~1 2 3 2~~

... 1 2 3 4 0

~~1 2 3 4 1~~

~~1 2 3 4 2~~

... 1 2 3 4 5 ~~1 2 3 4 6~~

1 2 3 5 0

0

1 0

start of program

~~1 1~~

1 2 0

~~1 2 1~~

~~1 2 2~~

1 2 3 0

~~1 2 3 1~~

~~1 2 3 2~~

... 1 2 3 4 0

~~1 2 3 4 1~~

~~1 2 3 4 2~~

... 1 2 3 4 5 ~~1 2 3 4 6~~

1 2 3 5 0

1 2 3 5 1

0

1 0

start of program

~~1 1~~

1 2 0

~~1 2 1~~

~~1 2 2~~

1 2 3 0

~~1 2 3 1~~

~~1 2 3 2~~

... 1 2 3 4 0

~~1 2 3 4 1~~

~~1 2 3 4 2~~

... 1 2 3 4 5

~~1 2 3 4 6~~

1 2 3 5 0

~~1 2 3 5 1~~

start of program

0

1 0

~~1 1~~

1 2 0

~~1 2 1~~

~~1 2 2~~

1 2 3 0

~~1 2 3 1~~

~~1 2 3 2~~

... 1 2 3 4 0

~~1 2 3 4 1~~

~~1 2 3 4 2~~

... 1 2 3 4 5

~~1 2 3 4 6~~

1 2 3 5 0

~~1 2 3 5 1~~

... 1 2 3 5 4

start of program

0				
---	--	--	--	--

1	0			
---	---	--	--	--

1	1			
--------------	--------------	--	--	--

1	2	0		
---	---	---	--	--

1	2	1		
--------------	--------------	--------------	--	--

1	2	2		
--------------	--------------	--------------	--	--

1	2	3	0	
---	---	---	---	--

1	2	3	1	
--------------	--------------	--------------	--------------	--

1	2	3	2	
--------------	--------------	--------------	--------------	--

...

1	2	3	4	0
---	---	---	---	---

1	2	3	4	1
--------------	--------------	--------------	--------------	--------------

1	2	3	4	2
--------------	--------------	--------------	--------------	--------------

...

1	2	3	4	5
---	---	---	---	---

1	2	3	4	6
--------------	--------------	--------------	--------------	--------------

1	2	3	5	0
---	---	---	---	---

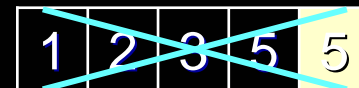
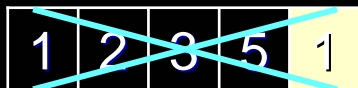
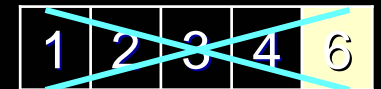
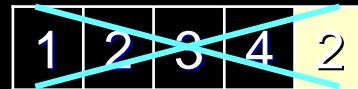
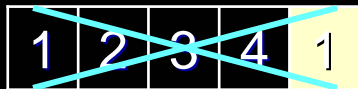
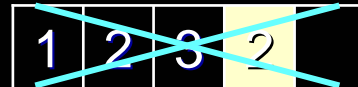
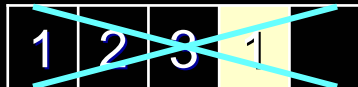
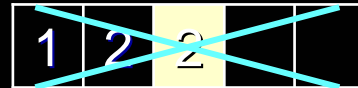
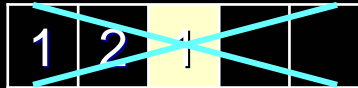
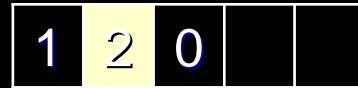
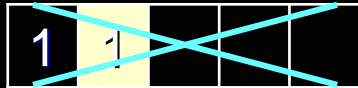
1	2	3	5	1
--------------	--------------	--------------	--------------	--------------

...

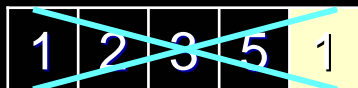
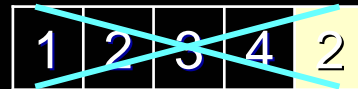
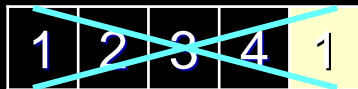
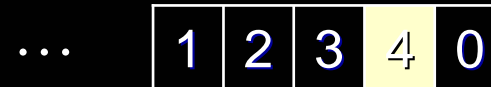
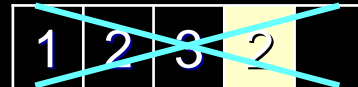
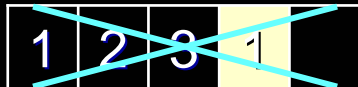
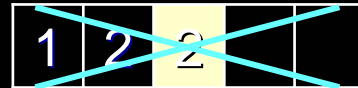
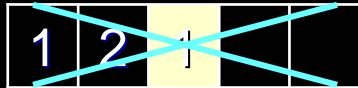
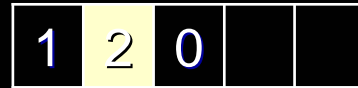
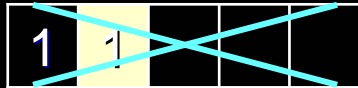
1	2	3	5	4
---	---	---	---	---

1	2	3	5	5
---	---	---	---	---

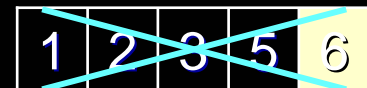
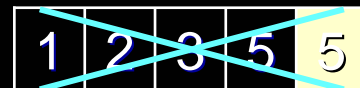
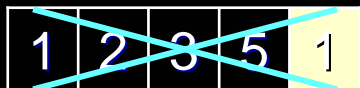
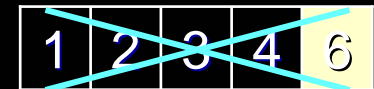
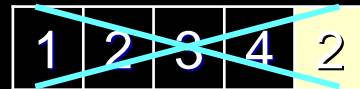
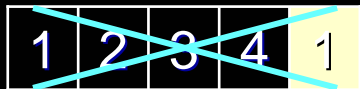
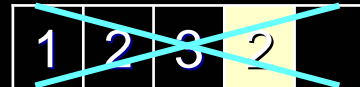
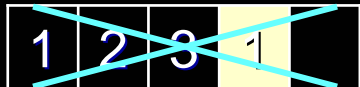
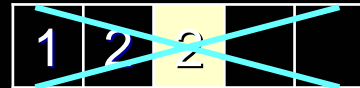
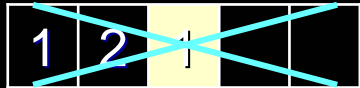
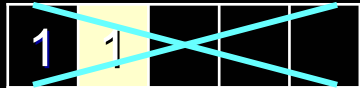
start of program



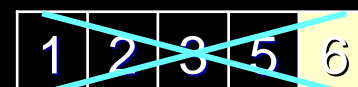
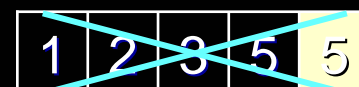
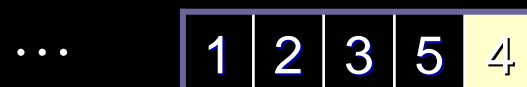
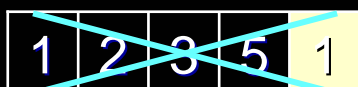
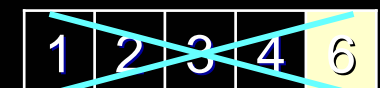
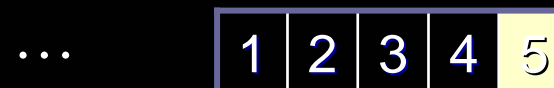
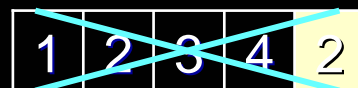
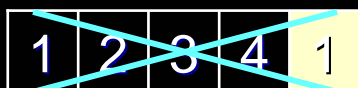
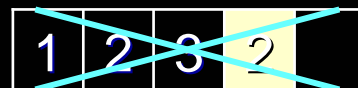
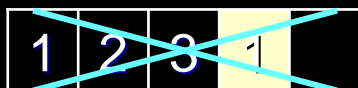
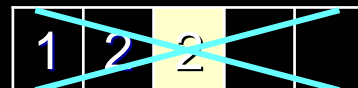
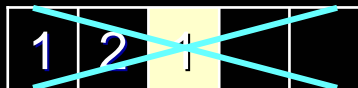
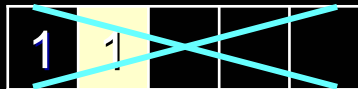
start of program



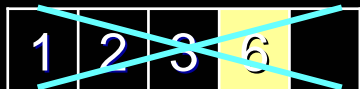
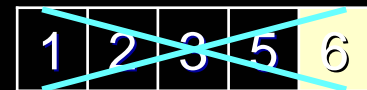
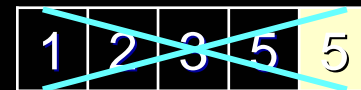
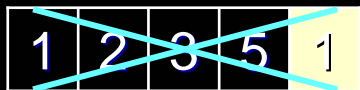
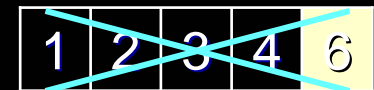
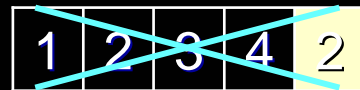
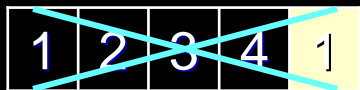
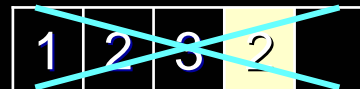
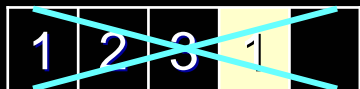
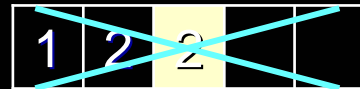
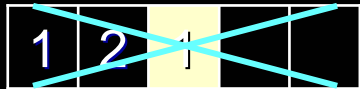
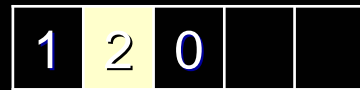
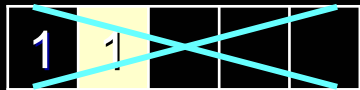
start of program



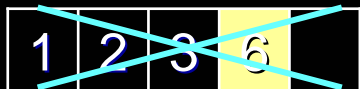
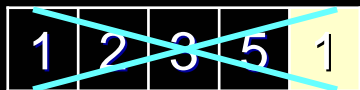
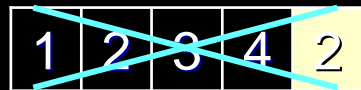
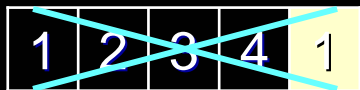
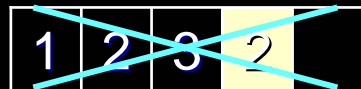
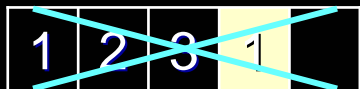
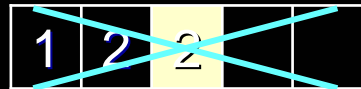
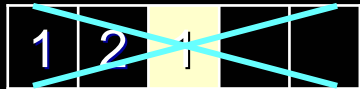
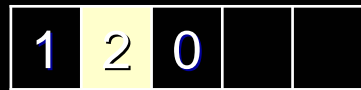
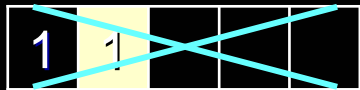
start of program



start of program



start of program



start of program

0				
---	--	--	--	--

1	0			
---	---	--	--	--

1	1			
--------------	--------------	--	--	--

1	2	0		
---	---	---	--	--

1	2	1		
--------------	--------------	--------------	--	--

1	2	2		
--------------	--------------	--------------	--	--

1	2	3	0	
---	---	---	---	--

1	2	3	1	
--------------	--------------	--------------	--------------	--

1	2	3	2	
--------------	--------------	--------------	--------------	--

...

1	2	3	4	0
---	---	---	---	---

1	2	3	4	1
--------------	--------------	--------------	--------------	--------------

1	2	3	4	2
--------------	--------------	--------------	--------------	--------------

...

1	2	3	4	5
---	---	---	---	---

1	2	3	4	6
--------------	--------------	--------------	--------------	--------------

1	2	3	5	0
---	---	---	---	---

1	2	3	5	1
--------------	--------------	--------------	--------------	--------------

...

1	2	3	5	4
---	---	---	---	---

1	2	3	5	5
--------------	--------------	--------------	--------------	--------------

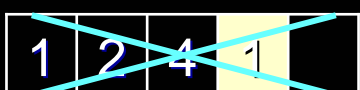
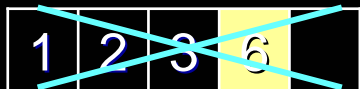
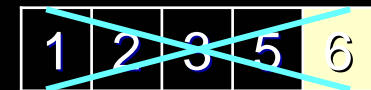
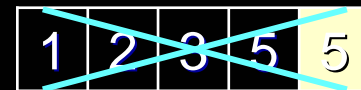
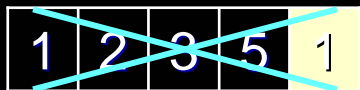
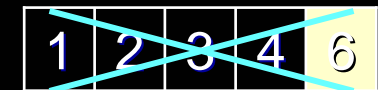
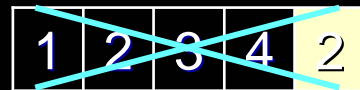
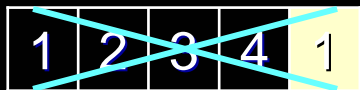
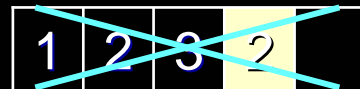
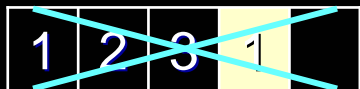
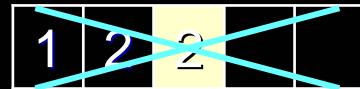
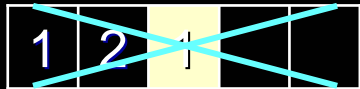
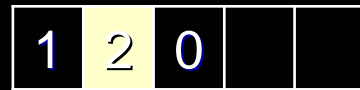
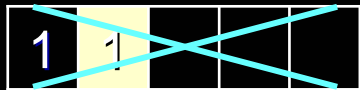
1	2	3	5	6
--------------	--------------	--------------	--------------	--------------

1	2	3	6	
--------------	--------------	--------------	--------------	--

1	2	4	0	
---	---	---	---	--

1	2	4	1	
---	---	---	---	--

start of program



start of program

0

1 0

~~1 1~~

1 2 0

~~1 2 1~~

~~1 2 2~~

1 2 3 0

~~1 2 3 1~~

~~1 2 3 2~~

... 1 2 3 4 0

~~1 2 3 4 1~~

~~1 2 3 4 2~~

... 1 2 3 4 5

~~1 2 3 4 6~~

1 2 3 5 0

~~1 2 3 5 1~~

... 1 2 3 5 4

~~1 2 3 5 5~~

~~1 2 3 5 6~~

~~1 2 3 6~~

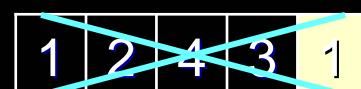
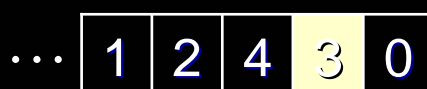
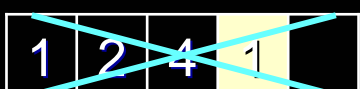
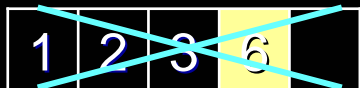
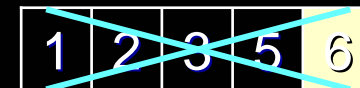
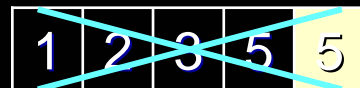
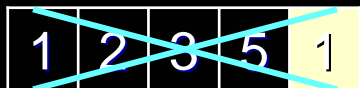
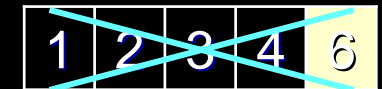
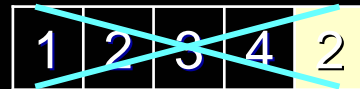
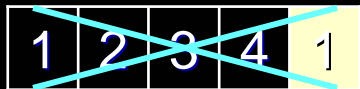
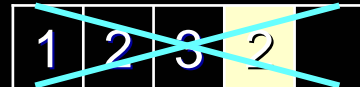
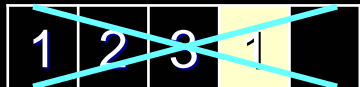
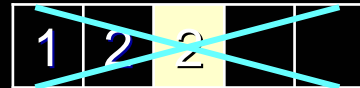
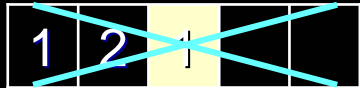
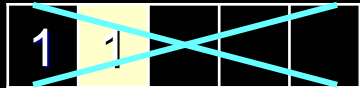
1 2 4 0

~~1 2 4 1~~

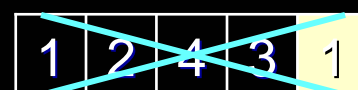
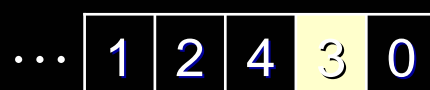
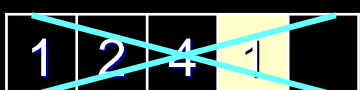
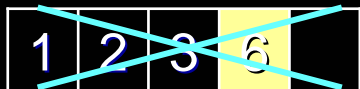
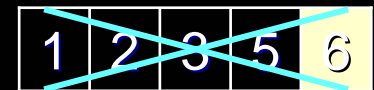
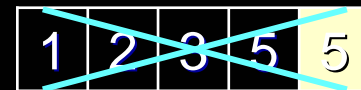
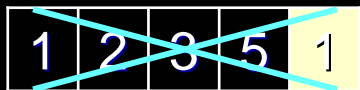
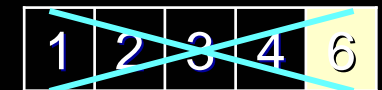
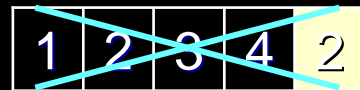
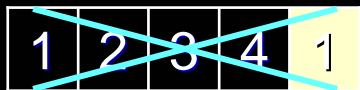
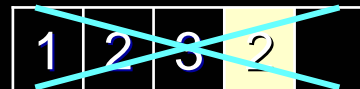
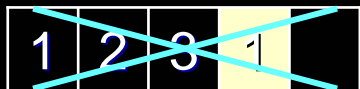
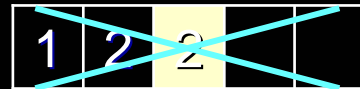
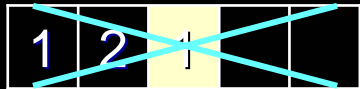
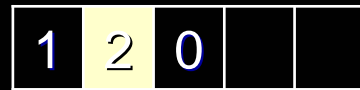
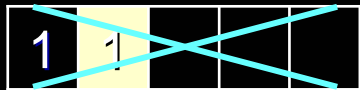
... 1 2 4 3 0

1 2 4 3 1

start of program



start of program



...

5	4	3	2	1
---	---	---	---	---

...

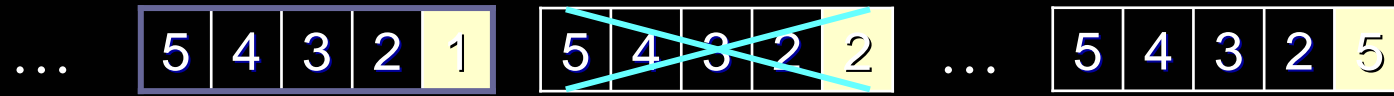
5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
---	---	---	---	---

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---



...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

...

5	4	3	2	5
--------------	--------------	--------------	--------------	---





...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

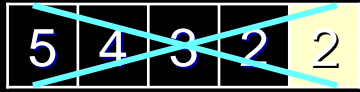
...

5	4	3	2	5
--------------	--------------	--------------	--------------	---

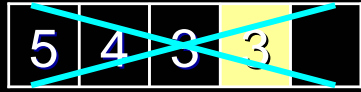
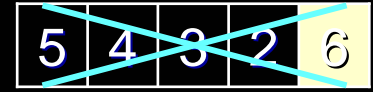
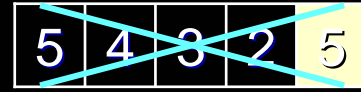
5	4	3	2	6
--------------	--------------	--------------	--------------	---

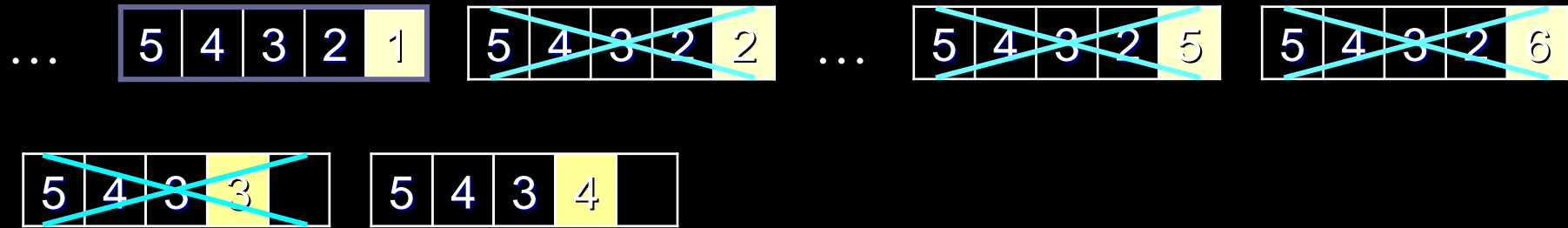
5	4	3	3	
---	---	---	---	--

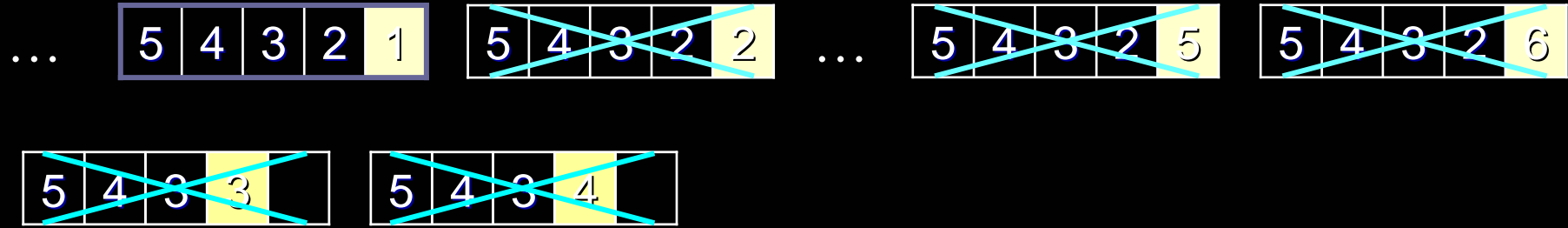
...

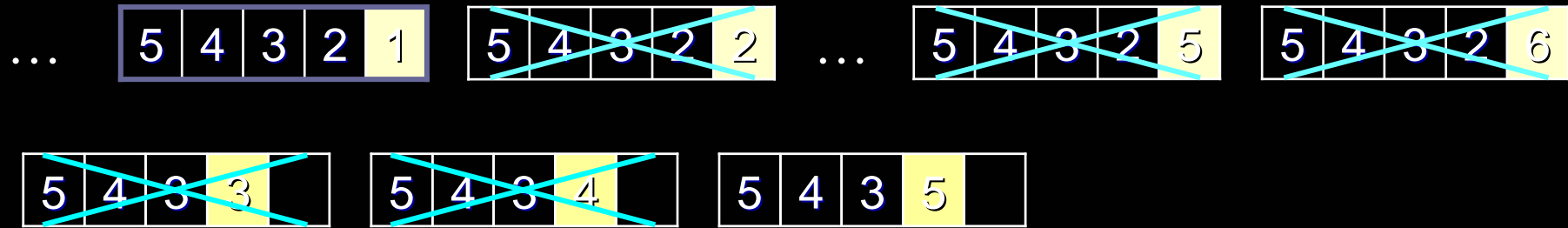


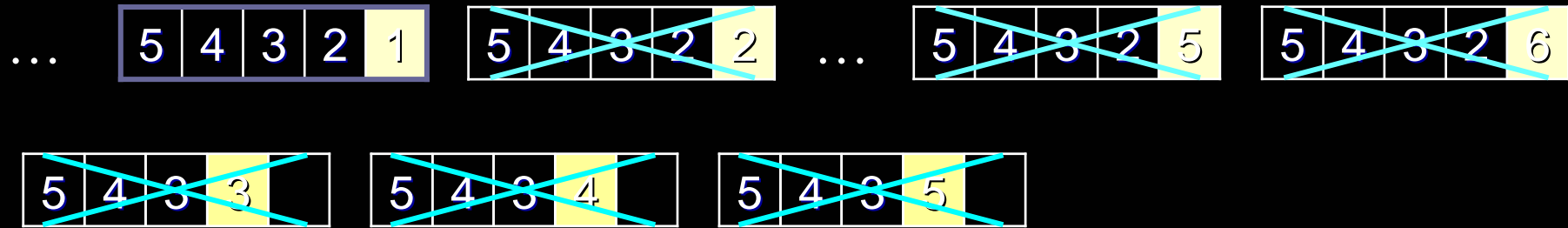
...











...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

 ...

5	4	3	2	5
--------------	--------------	--------------	--------------	---

5	4	3	2	6
--------------	--------------	--------------	--------------	---

5	4	3	3	
--------------	--------------	--------------	---	--

5	4	3	4	
--------------	--------------	--------------	---	--

5	4	3	5	
--------------	--------------	--------------	---	--

5	4	3	6	
---	---	---	---	--

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

 ...

5	4	3	2	5
--------------	--------------	--------------	--------------	---

5	4	3	2	6
--------------	--------------	--------------	--------------	---

5	4	3	3	
--------------	--------------	--------------	---	--

5	4	3	4	
--------------	--------------	--------------	---	--

5	4	3	5	
--------------	--------------	--------------	---	--

5	4	3	6	
--------------	--------------	--------------	---	--

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

 ...

5	4	3	2	5
--------------	--------------	--------------	--------------	---

5	4	3	2	6
--------------	--------------	--------------	--------------	---

5	4	3	3	
--------------	--------------	--------------	---	--

5	4	3	4	
--------------	--------------	--------------	---	--

5	4	3	5	
--------------	--------------	--------------	---	--

5	4	3	6	
--------------	--------------	--------------	---	--

5	4	4		
---	---	---	--	--

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

 ...

5	4	3	2	5
--------------	--------------	--------------	--------------	---

5	4	3	2	6
--------------	--------------	--------------	--------------	---

5	4	3	3	
--------------	--------------	--------------	---	--

5	4	3	4	
--------------	--------------	--------------	---	--

5	4	3	5	
--------------	--------------	--------------	---	--

5	4	3	6	
--------------	--------------	--------------	---	--

5	4	4		
--------------	--------------	---	--	--

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

 ...

5	4	3	2	5
--------------	--------------	--------------	--------------	---

5	4	3	2	6
--------------	--------------	--------------	--------------	---

5	4	3	3	
--------------	--------------	--------------	---	--

5	4	3	4	
--------------	--------------	--------------	---	--

5	4	3	5	
--------------	--------------	--------------	---	--

5	4	3	6	
--------------	--------------	--------------	---	--

5	4	4		
--------------	--------------	---	--	--

5	4	5		
---	---	---	--	--

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

 ...

5	4	3	2	5
--------------	--------------	--------------	--------------	---

5	4	3	2	6
--------------	--------------	--------------	--------------	---

5	4	3	3	
--------------	--------------	--------------	---	--

5	4	3	4	
--------------	--------------	--------------	---	--

5	4	3	5	
--------------	--------------	--------------	---	--

5	4	3	6	
--------------	--------------	--------------	---	--

5	4	4		
--------------	--------------	---	--	--

5	4	5		
--------------	--------------	---	--	--

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

 ...

5	4	3	2	5
--------------	--------------	--------------	--------------	---

5	4	3	2	6
--------------	--------------	--------------	--------------	---

5	4	3	3	
--------------	--------------	--------------	---	--

5	4	3	4	
--------------	--------------	--------------	---	--

5	4	3	5	
--------------	--------------	--------------	---	--

5	4	3	6	
--------------	--------------	--------------	---	--

5	4	4		
--------------	--------------	---	--	--

5	4	5		
--------------	--------------	---	--	--

5	4	6		
---	---	---	--	--

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

 ...

5	4	3	2	5
--------------	--------------	--------------	--------------	---

5	4	3	2	6
--------------	--------------	--------------	--------------	---

5	4	3	3	
--------------	--------------	--------------	---	--

5	4	3	4	
--------------	--------------	--------------	---	--

5	4	3	5	
--------------	--------------	--------------	---	--

5	4	3	6	
--------------	--------------	--------------	---	--

5	4	4		
--------------	--------------	---	--	--

5	4	5		
--------------	--------------	---	--	--

5	4	6		
--------------	--------------	---	--	--

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

 ...

5	4	3	2	5
--------------	--------------	--------------	--------------	---

5	4	3	2	6
--------------	--------------	--------------	--------------	---

5	4	3	3	
--------------	--------------	--------------	---	--

5	4	3	4	
--------------	--------------	--------------	---	--

5	4	3	5	
--------------	--------------	--------------	---	--

5	4	3	6	
--------------	--------------	--------------	---	--

5	4	4		
--------------	--------------	---	--	--

5	4	5		
--------------	--------------	---	--	--

5	4	6		
--------------	--------------	---	--	--

5	5			
---	---	--	--	--

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

 ...

5	4	3	2	5
--------------	--------------	--------------	--------------	---

5	4	3	2	6
--------------	--------------	--------------	--------------	---

5	4	3	3	
--------------	--------------	--------------	---	--

5	4	3	4	
--------------	--------------	--------------	---	--

5	4	3	5	
--------------	--------------	--------------	---	--

5	4	3	6	
--------------	--------------	--------------	---	--

5	4	4		
--------------	--------------	---	--	--

5	4	5		
--------------	--------------	---	--	--

5	4	6		
--------------	--------------	---	--	--

5	5			
--------------	---	--	--	--

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

 ...

5	4	3	2	5
--------------	--------------	--------------	--------------	---

5	4	3	2	6
--------------	--------------	--------------	--------------	---

5	4	3	3	
--------------	--------------	--------------	---	--

5	4	3	4	
--------------	--------------	--------------	---	--

5	4	3	5	
--------------	--------------	--------------	---	--

5	4	3	6	
--------------	--------------	--------------	---	--

5	4	4		
--------------	--------------	---	--	--

5	4	5		
--------------	--------------	---	--	--

5	4	6		
--------------	--------------	---	--	--

5	5			
--------------	---	--	--	--

5	6			
---	---	--	--	--

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

 ...

5	4	3	2	5
--------------	--------------	--------------	--------------	---

5	4	3	2	6
--------------	--------------	--------------	--------------	---

5	4	3	3	
--------------	--------------	--------------	---	--

5	4	3	4	
--------------	--------------	--------------	---	--

5	4	3	5	
--------------	--------------	--------------	---	--

5	4	3	6	
--------------	--------------	--------------	---	--

5	4	4		
--------------	--------------	---	--	--

5	4	5		
--------------	--------------	---	--	--

5	4	6		
--------------	--------------	---	--	--

5	5			
--------------	---	--	--	--

5	6			
--------------	---	--	--	--

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

 ...

5	4	3	2	5
--------------	--------------	--------------	--------------	---

5	4	3	2	6
--------------	--------------	--------------	--------------	---

5	4	3	3	
--------------	--------------	--------------	---	--

5	4	3	4	
--------------	--------------	--------------	---	--

5	4	3	5	
--------------	--------------	--------------	---	--

5	4	3	6	
--------------	--------------	--------------	---	--

5	4	4		
--------------	--------------	---	--	--

5	4	5		
--------------	--------------	---	--	--

5	4	6		
--------------	--------------	---	--	--

5	5			
--------------	---	--	--	--

5	6			
--------------	---	--	--	--

6				
---	--	--	--	--

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
---	---	---	---	---

 ...

5	4	3	2	5
---	---	---	---	---

5	4	3	2	6
---	---	---	---	---

5	4	3	3
---	---	---	---

5	4	3	4
---	---	---	---

5	4	3	5
---	---	---	---

5	4	3	6
---	---	---	---

5	4	4
---	---	---

5	4	5
---	---	---

5	4	6
---	---	---

5	5
---	---

5	6
---	---

6

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
---	---	---	---	---

 ...

5	4	3	2	5
---	---	---	---	---

5	4	3	2	6
---	---	---	---	---

5	4	3	3
---	---	---	---

5	4	3	4
---	---	---	---

5	4	3	5
---	---	---	---

5	4	3	6
---	---	---	---

5	4	4
---	---	---

5	4	5
---	---	---

5	4	6
---	---	---

5	5
---	---

5	6
---	---

6

--	--	--	--	--	--

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
--------------	--------------	--------------	--------------	---

 ...

5	4	3	2	5
--------------	--------------	--------------	--------------	---

5	4	3	2	6
--------------	--------------	--------------	--------------	---

5	4	3	3	
--------------	--------------	--------------	---	--

5	4	3	4	
--------------	--------------	--------------	---	--

5	4	3	5	
--------------	--------------	--------------	---	--

5	4	3	6	
--------------	--------------	--------------	---	--

5	4	4		
--------------	--------------	---	--	--

5	4	5		
--------------	--------------	---	--	--

5	4	6		
--------------	--------------	---	--	--

5	5			
--------------	---	--	--	--

5	6			
--------------	---	--	--	--

6				
---	--	--	--	--

--	--	--	--	--

...

5	4	3	2	1
---	---	---	---	---

5	4	3	2	2
---	---	---	---	---

 ...

5	4	3	2	5
---	---	---	---	---

5	4	3	2	6
---	---	---	---	---

5	4	3	3
---	---	---	---

5	4	3	4
---	---	---	---

5	4	3	5
---	---	---	---

5	4	3	6
---	---	---	---

5	4	4
---	---	---

5	4	5
---	---	---

5	4	6
---	---	---

5	5
---	---

5	6
---	---

6

--

end of program

Counting w/o invalid() check

Counting w/o invalid() check

perm

1	2	3
---	---	---

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;
```

Counting w/o invalid() check

perm

1	2	3
---	---	---

```
int perm[15], used[16]={0}, n=3;
for (i=0; i<n; i++)
    perm[i] = i+1, used[i] = 1;
do
    printArray(perm, n);
while (next(perm, n));
```



```
void next (int perm[], int used[], int n) {
```

Counting w/o invalid() check

perm

1	2	3
---	---	---

```
}
```

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

```
void next (int perm[], int used[], int n) {  
    int i, j, k;
```

Counting w/o invalid() check

perm

1	2	3
---	---	---

```
}
```

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

```
void next (int perm[], int used[], int n) {
```

```
    int i, j, k;
```

```
    for (i=n-1; i>=0; i--) {
```

Counting

w/o invalid() check

perm

1	2	3
---	---	---

```
}
```

```
}
```

```
int perm[15], used[16]={0}, n=3;
```

```
for (i=0; i<n; i++)
```

```
    perm[i] = i+1, used[i] = 1;
```

```
do
```

```
    printArray(perm, n);
```

```
while (next(perm, n));
```

```
void next (int perm[], int used[], int n) {
```

```
    int i, j, k;
```

```
    for (i=n-1; i>=0; i--) {
```

```
        used[perm[i]] = 0;
```

```
    }
```

```
}
```

Counting

w/o invalid() check

perm

1	2	3
---	---	---

```
int perm[15], used[16]={0}, n=3;
```

```
for (i=0; i<n; i++)
```

```
    perm[i] = i+1, used[i] = 1;
```

```
do
```

```
    printArray(perm, n);
```

```
while (next(perm, n));
```

Counting

w/o invalid() check

```
void next (int perm[], int used[], int n) {
```

```
    int i, j, k;
```

```
    for (i=n-1; i>=0; i--) {
```

```
        used[perm[i]] = 0;
```

```
        do
```

```
            perm[i] ++;
```

```
        while ((perm[i]<=n)&&used[perm[i]]);
```

perm

1	2	3
---	---	---

```
    }
```

```
}
```

```
int perm[15], used[16]={0}, n=3;
```

```
for (i=0; i<n; i++)
```

```
    perm[i] = i+1, used[i] = 1;
```

```
do
```

```
    printArray(perm, n);
```

```
while (next(perm, n));
```

Counting

w/o invalid() check

```
void next (int perm[], int used[], int n) {
```

```
    int i, j, k;
```

```
    for (i=n-1; i>=0; i--) {
```

```
        used[perm[i]] = 0;
```

```
        do
```

```
            perm[i] ++;
```

```
        while ((perm[i]<=n)&&used[perm[i]]);
```

```
        if (perm[i]<n) {
```

```
            }
```

```
        }
```

```
    }
```

perm

1	2	3
---	---	---

```
int perm[15], used[16]={0}, n=3;
```

```
for (i=0; i<n; i++)
```

```
    perm[i] = i+1, used[i] = 1;
```

```
do
```

```
    printArray(perm, n);
```

```
while (next(perm, n));
```

Counting

w/o invalid() check

```
void next (int perm[], int used[], int n) {
```

```
    int i, j, k;
```

```
    for (i=n-1; i>=0; i--) {
```

```
        used[perm[i]] = 0;
```

```
        do
```

```
            perm[i] ++;
```

```
        while ((perm[i]<=n)&&used[perm[i]]);
```

```
        if (perm[i]<n) {
```

```
            used[perm[i]] = 1;
```

```
        }
```

```
    }
```

```
}
```

perm

1	2	3
---	---	---

```
int perm[15], used[16]={0}, n=3;
```

```
for (i=0; i<n; i++)
```

```
    perm[i] = i+1, used[i] = 1;
```

```
do
```

```
    printArray(perm, n);
```

```
while (next(perm, n));
```

Counting

w/o invalid() check

perm

1	2	3
---	---	---

```
void next (int perm[], int used[], int n) {
```

```
    int i, j, k;
```

```
    for (i=n-1; i>=0; i--) {
```

```
        used[perm[i]] = 0;
```

```
        do
```

```
            perm[i] ++;
```

```
        while ((perm[i]<=n)&&used[perm[i]]);
```

```
        if (perm[i]<n) {
```

```
            used[perm[i]] = 1;
```

```
            for (k=0, j=i+1; j<n; j++) {
```

```
                }
```

```
            }
```

```
        }
```

```
    }
```

```
int perm[15], used[16]={0}, n=3;
```

```
for (i=0; i<n; i++)
```

```
    perm[i] = i+1, used[i] = 1;
```

```
do
```

```
    printArray(perm, n);
```

```
while (next(perm, n));
```


Counting

w/o invalid() check

perm

1	2	3
---	---	---

```
void next (int perm[], int used[], int n) {
```

```
    int i, j, k;
```

```
    for (i=n-1; i>=0; i--) {
```

```
        used[perm[i]] = 0;
```

```
        do
```

```
            perm[i] ++;
```

```
        while ((perm[i]<=n)&&used[perm[i]]);
```

```
        if (perm[i]<n) {
```

```
            used[perm[i]] = 1;
```

```
            for (k=0, j=i+1; j<n; j++) {
```

```
                while (used[++k]);
```

```
                perm[j] = k, used[k] = 1;
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

```
int perm[15], used[16]={0}, n=3;
```

```
for (i=0; i<n; i++)
```

```
    perm[i] = i+1, used[i] = 1;
```

```
do
```

```
    printArray(perm, n);
```

```
while (next(perm, n));
```

Counting

w/o invalid() check

perm

1	2	3
---	---	---

```
void next (int perm[], int used[], int n) {
```

```
    int i, j, k;
```

```
    for (i=n-1; i>=0; i--) {
```

```
        used[perm[i]] = 0;
```

```
        do
```

```
            perm[i] ++;
```

```
        while ((perm[i]<=n)&&used[perm[i]]);
```

```
        if (perm[i]<n) {
```

```
            used[perm[i]] = 1;
```

```
            for (k=0, j=i+1; j<n; j++) {
```

```
                while (used[++k]);
```

```
                perm[j] = k, used[k] = 1;
```

```
            }
```

```
            return 1;
```

```
        }
```

```
    }
```

```
}
```

```
int perm[15], used[16]={0}, n=3;
```

```
for (i=0; i<n; i++)
```

```
    perm[i] = i+1, used[i] = 1;
```

```
do
```

```
    printArray(perm, n);
```

```
while (next(perm, n));
```

Counting

w/o invalid() check

perm

1	2	3
---	---	---

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i] ++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

Counting

w/o invalid() check

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i] ++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

perm

1	2	3
---	---	---

1

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

Counting

w/o invalid() check

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i] ++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

perm

1	2	3
1	3	

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

Counting

w/o invalid() check

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i] ++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

perm

1	2	3
1	3	2

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

Counting

w/o invalid() check

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i] ++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

perm	1	2	3
	1	3	2
	2		

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

Counting w/o invalid() check

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i] ++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

perm	1	2	3
	1	3	2
	2	1	

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```


Counting w/o invalid() check

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i] ++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

perm	1	2	3
	1	3	2
	2	1	3

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

Counting w/o invalid() check

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i] ++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

perm	1	2	3
	1	3	2
	2	1	3
	2		

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

Counting w/o invalid() check

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i] ++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

perm	1	2	3
	1	3	2
	2	1	3
	2	3	

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

Counting w/o invalid() check

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i] ++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

perm	1	2	3
	1	3	2
	2	1	3
	2	3	1

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

Counting w/o invalid() check

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i] ++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

perm	1	2	3
	1	3	2
	2	1	3
	2	3	1
	3		

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

Counting w/o invalid() check

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i]++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

perm	1	2	3
	1	3	2
	2	1	3
	2	3	1
	3	1	

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

Counting w/o invalid() check

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i] ++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

perm	1	2	3
	1	3	2
	2	1	3
	2	3	1
	3	1	2

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

Counting w/o invalid() check

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i] ++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

perm	1	2	3
	1	3	2
	2	1	3
	2	3	1
	3	1	2
	3		

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```


Counting w/o invalid() check

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i] ++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

perm	1	2	3
	1	3	2
	2	1	3
	2	3	1
	3	1	2
	3	2	

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

Counting w/o invalid() check

```
void next (int perm[], int used[], int n) {  
    int i, j, k;  
    for (i=n-1; i>=0; i--) {  
        used[perm[i]] = 0;  
        do  
            perm[i] ++;  
        while ((perm[i]<=n)&&used[perm[i]]);  
        if (perm[i]<n) {  
            used[perm[i]] = 1;  
            for (k=0, j=i+1; j<n; j++) {  
                while (used[++k]);  
                perm[j] = k, used[k] = 1;  
            }  
            return 1;  
        }  
    }  
    return 0;  
}
```

perm	1	2	3
	1	3	2
	2	1	3
	2	3	1
	3	1	2
	3	2	1

```
int perm[15], used[16]={0}, n=3;  
for (i=0; i<n; i++)  
    perm[i] = i+1, used[i] = 1;  
do  
    printArray(perm, n);  
while (next(perm, n));
```

Permutation from Swapping

- **Recursive**

1 + permutations of {2, 3, 4}

2 + permutations of {1, 3, 4}

3 + permutations of {2, 1, 4}

4 + permutations of {2, 3, 1}

```
for (i=0; i<n; i++) a[i] = i+1;  
permutation(a, 0, n-1);
```

```
void permutation(int perm[], int start, int end) {  
    if (start == end) printPerm(perm, end+1);  
    for (int i=start; i<=end; i++) {  
        swap(&perm[start], &perm[i]);  
        permutation(perm, start+1, end);  
        swap(&perm[start], &perm[i]);  
    }  
}
```

1	2	3	4
1	2	4	3
1	3	2	4
1	3	4	2
1	4	3	2
1	4	2	3
2	1	3	4
2	1	4	3
...
3	2	1	4
3	2	4	1
...
4	2	3	1
4	2	1	3
...
4	1	2	3

Permutation from Rotation

0 1 2 3	2 0 1 3	1 2 0 3	1 0 2 3	2 1 0 3	0 2 1 3
3 0 1 2	3 2 0 1	3 1 2 0	3 1 0 2	3 2 1 0	3 0 2 1
2 3 0 1	1 3 2 0	0 3 1 2	2 3 1 0	0 3 2 1	1 3 0 2
1 2 3 0	0 1 3 2	2 0 3 1	0 2 3 1	1 0 3 2	2 1 3 0
0 1 2 3	2 0 1 3	1 2 0 3	1 0 2 3	2 1 0 3	0 2 1 3
		0 1 2 3			1 0 2 3
					0 1 2 3

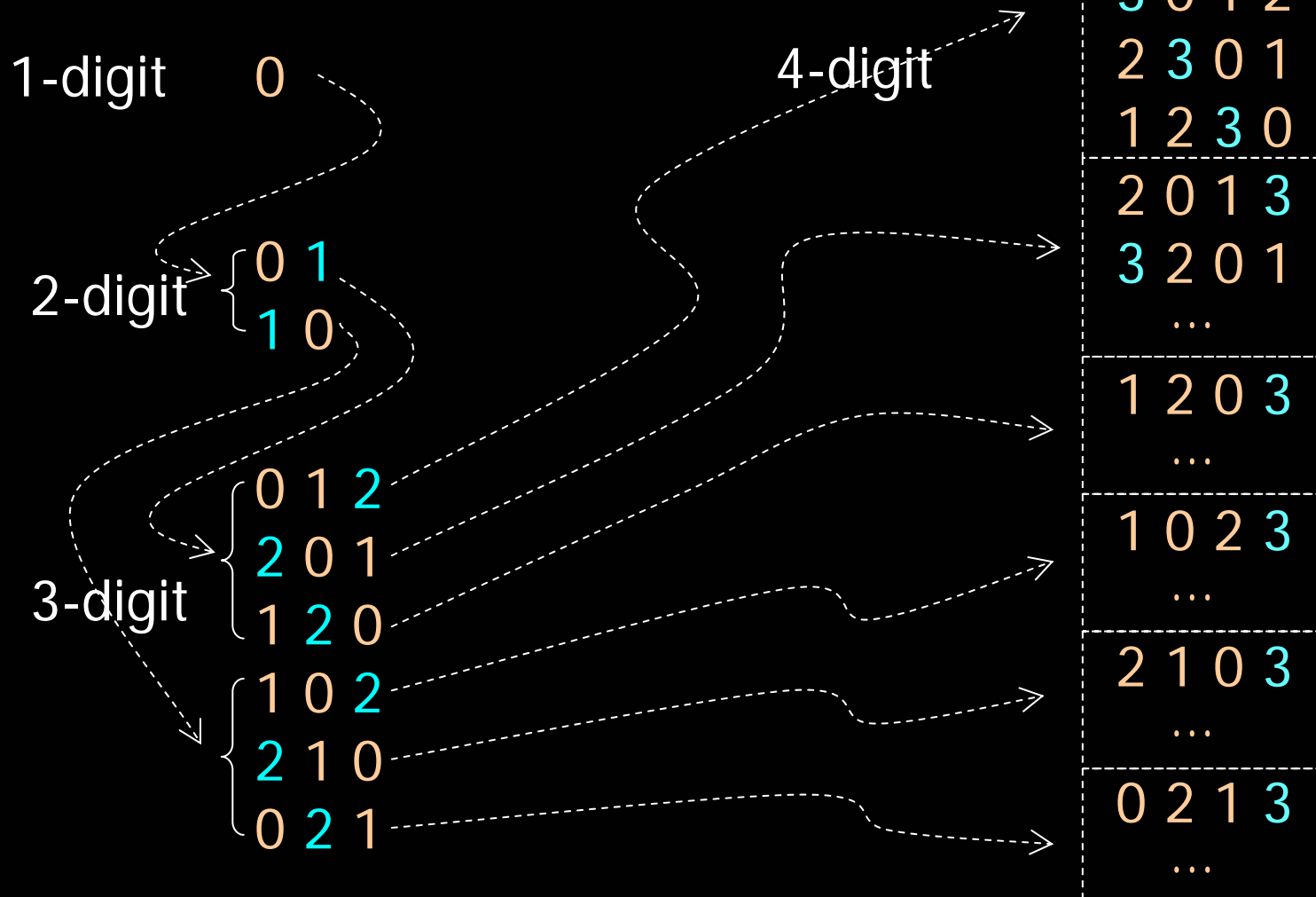
```

int rotate_n_check(int size, char data[]) {
    int i;
    for (i=size; i>=2; i--) {
        rotate(i, data);
        if (data[i-1] != i-1) return 1;
    }
    return 0;
}

for (i=0; i<num; i++) digit[i] = i;
do
    printPermutation(num, digit);
while (rotate_n_check(num, digit));
    
```

Rotation (cont')

- Why does it work?



Permutation by Exchanging Neighbors

perm	pos	perm	pos
3 2 1 0	0 0 0	3 2 0 1	0 0 1
2 3 1 0	1 0 0	2 3 0 1	1 0 1
2 1 3 0	2 0 0	2 0 3 1	2 0 1
2 1 0 3	3 0 0	2 0 1 3	3 0 1
3 1 2 0	0 1 0	3 0 2 1	0 1 1
1 3 2 0	1 1 0	0 3 2 1	1 1 1
1 2 3 0	2 1 0	0 2 3 1	2 1 1
1 2 0 3	3 1 0	0 2 1 3	3 1 1
3 1 0 2	0 2 0	3 0 1 2	0 2 1
1 3 0 2	1 2 0	0 3 1 2	1 2 1
1 0 3 2	2 2 0	0 1 3 2	2 2 1
1 0 2 3	3 2 0	0 1 2 3	3 2 1



```

for (i=0;i<num;i++)
    perm[i]=num-1-i, position[i]=0;
do
    printPermutation(num, perm);
while (shift_n_check(num, perm, position))
    
```

```

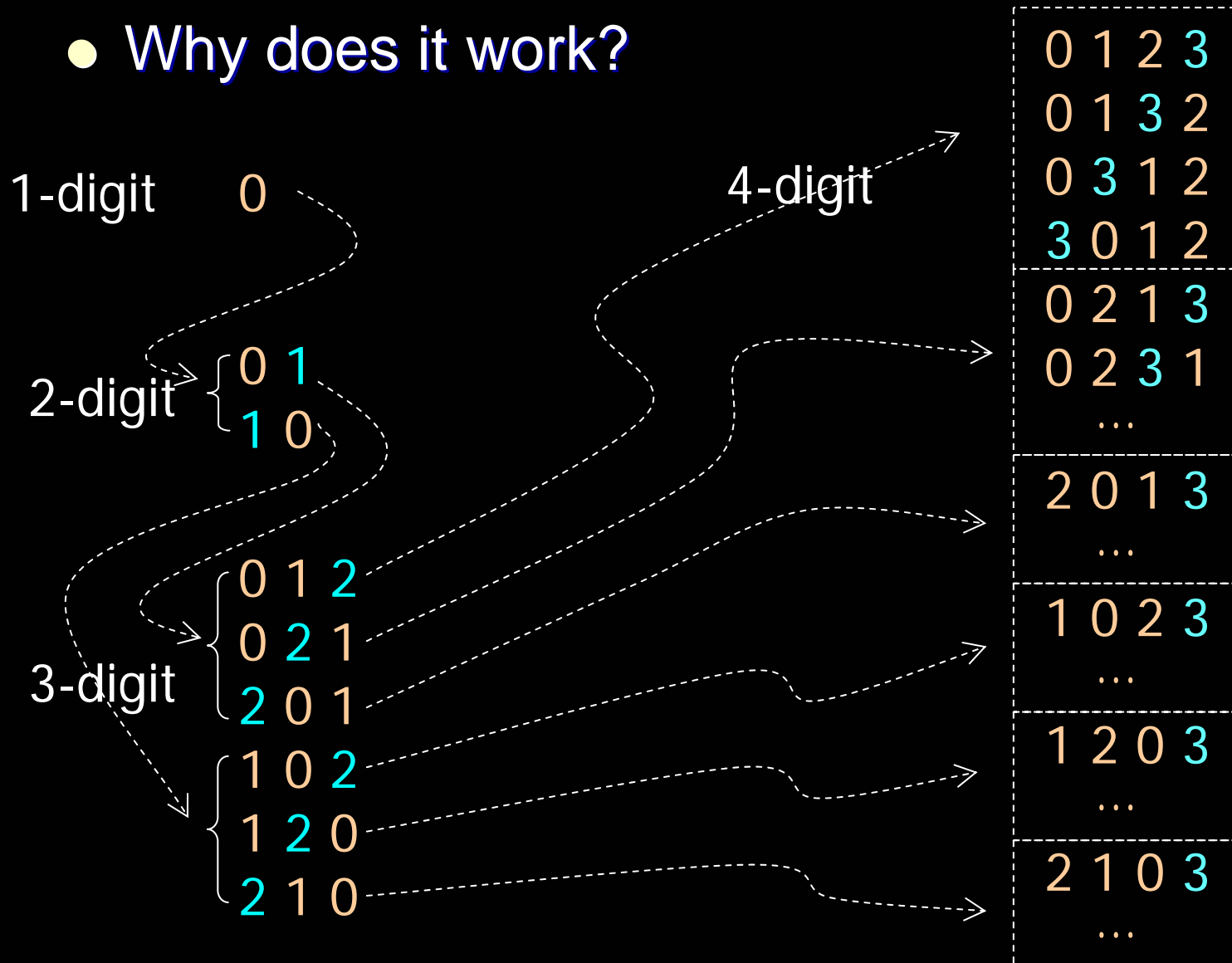
void shiftRight(int size, char perm[],
                char *pos){
    char tmp = perm[*pos];
    if (*pos < size-1) {
        perm[*pos] = perm[*pos+1];
        perm[++(*pos)] = tmp;
    }
    else {
        for (int i=size-2; i>=0; i--)
            perm[i+1] = perm[i];
        perm[*pos=0] = tmp;
    }
}
    
```

```

int shift_n_check(int size, char perm[],
                  char pos[]) {
    for (int i=size; i>=2; i--) {
        shiftRight(i, &perm[size-i], &pos[size-i]);
        if (pos[size-i] > 0) return 1;
    }
    return 0;
}
    
```

Exchanging Neighbors (cont'd)

- Why does it work?



Generate Disjoint Partitions

- A partition of a set X is a set of non-empty subsets of X such that every element x in X is in exactly one of these subsets.
 - e.g. The set $X=\{1, 2, 3\}$ has five partitions:
 - $\{\{1\}, \{2\}, \{3\}\}$, sometimes written $1 | 2 | 3$.
 - $\{\{1, 2\}, \{3\}\}$, or $12 | 3$.
 - $\{\{1, 3\}, \{2\}\}$, or $13 | 2$.
 - $\{\{1\}, \{2, 3\}\}$, or $1 | 23$.
 - $\{\{1, 2, 3\}\}$, or 123
 - The total number of partitions is the Bell numbers B_n

$$B_{n+1} = \sum_{k=0}^n C_k^n B_k$$

e.g. $B_0 = 1, B_1 = 1, B_2 = 2, B_3 = 5, B_4 = 15, B_5 = 52, B_6 = 203, \dots$

$B_{20} = 51724158235372 \approx 5 \times 10^{13}, B_{30} = 846749014511809332450147 \approx 8 \times 10^{23},$

$B_{40} = 157450588391204931289324344702531067 \approx 2 \times 10^{35},$

$B_{50} = 185724268771078270438257767181908917499221852770 \approx 2 \times 10^{47}.$

Generate Partitions (cont'd)

- A simple algorithm through counting up with dynamically adjusted ceiling, partition / max e.g., X is a 4-element set $\{a_0, a_1, a_2, a_3\}$

1: **0,0,0,0** / 0,1,1,1 $\implies \{a_0, a_1, a_2, a_3\}$

2: **0,0,0,1** / 0,1,1,1 $\implies \{a_0, a_1, a_2\}, \{a_3\}$

3: **0,0,1,0** / 0,1,1,2 $\implies \{a_0, a_1, a_3\}, \{a_2\}$

4: **0,0,1,1** / 0,1,1,2 $\implies \{a_0, a_1\}, \{a_2, a_3\}$

5: **0,0,1,2** / 0,1,1,2 $\implies \{a_0, a_1\}, \{a_2\}, \{a_3\}$

6: **0,1,0,0** / 0,1,2,2 $\implies \{a_0, a_2, a_3\}, \{a_1\}$

7: **0,1,0,1** / 0,1,2,2 $\implies \{a_0, a_2\}, \{a_1, a_3\}$

8: **0,1,0,2** / 0,1,2,2 $\implies \{a_0, a_2\}, \{a_1\}, \{a_3\}$

9: **0,1,1,0** / 0,1,2,2 $\implies \{a_0, a_3\}, \{a_1, a_2\}$

10: **0,1,1,1** / 0,1,2,2 $\implies \{a_0\}, \{a_1, a_2, a_3\}$

11: **0,1,1,2** / 0,1,2,2 $\implies \{a_0\}, \{a_1, a_2\}, \{a_3\}$

12: **0,1,2,0** / 0,1,2,3 $\implies \{a_0, a_3\}, \{a_1\}, \{a_2\}$

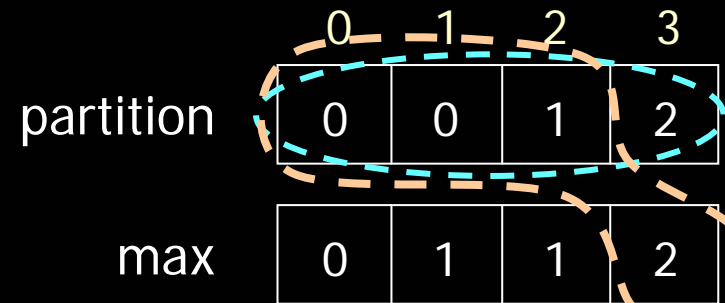
13: **0,1,2,1** / 0,1,2,3 $\implies \{a_0\}, \{a_1, a_3\}, \{a_2\}$

14: **0,1,2,2** / 0,1,2,3 $\implies \{a_0\}, \{a_1\}, \{a_2, a_3\}$

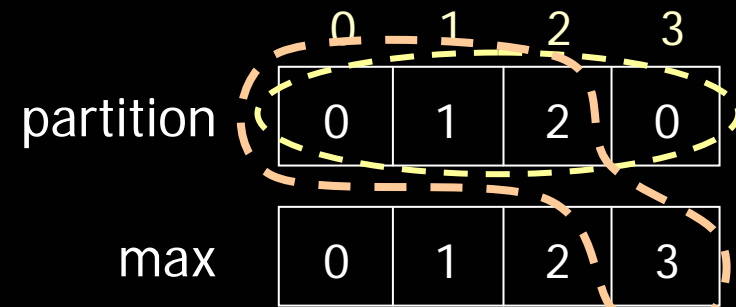
15: **0,1,2,3** / 0,1,2,3 $\implies \{a_0\}, \{a_1\}, \{a_2\}, \{a_3\}$

Implementation with 2 arrays

0,0,1,2 label the partitions for each set elements a_i



each $\max[i]$ is greater than all prior partitions



Implementation

```
01 int next(int size, int pivot, int partition[], int max[]) {
02     int i, j, maxVal;
03     while (pivot>0 && (partition[pivot] >= max[pivot]))
04         pivot--;
05     if (pivot > 0) {
06         partition[pivot]++;
07         for (i=pivot+1; i<size; i++) {
08             partition[i] = 0;
09             maxVal = 0;
10             for (j=1; j<i; j++)
11                 if (partition[j]>maxVal)
12                     maxVal = partition[j];
13             max[i] = maxVal+1;
14         }
15         return size;
16     }
17     return pivot;
18 }
```

pivot

0 1 2 3

partition

0	1	1	2
---	---	---	---

max

0	1	2	2
---	---	---	---

0 1 2 3

partition

0	1	2	0
---	---	---	---

max

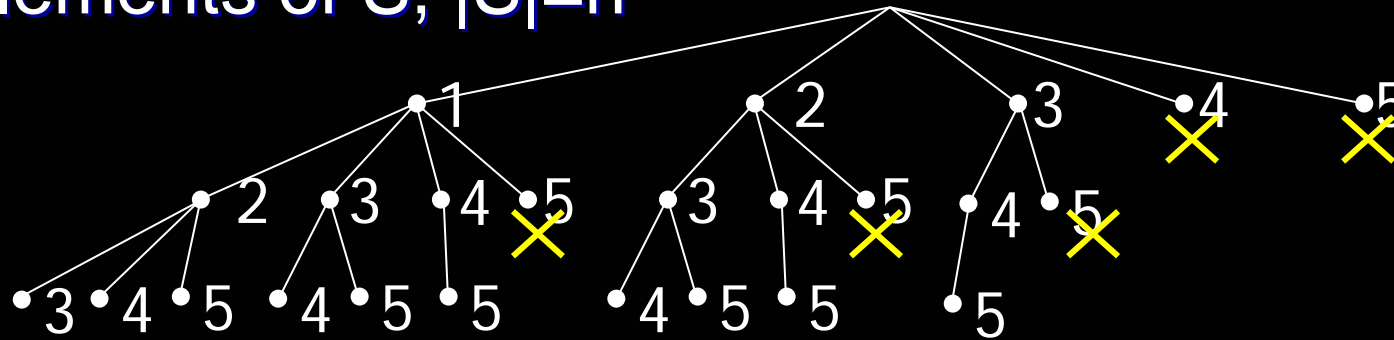
0	1	2	3
---	---	---	---

Generate (n,k) -Combinations

- (n,k) -combination of a set S is a subset of k distinct elements of S , $|S|=n$

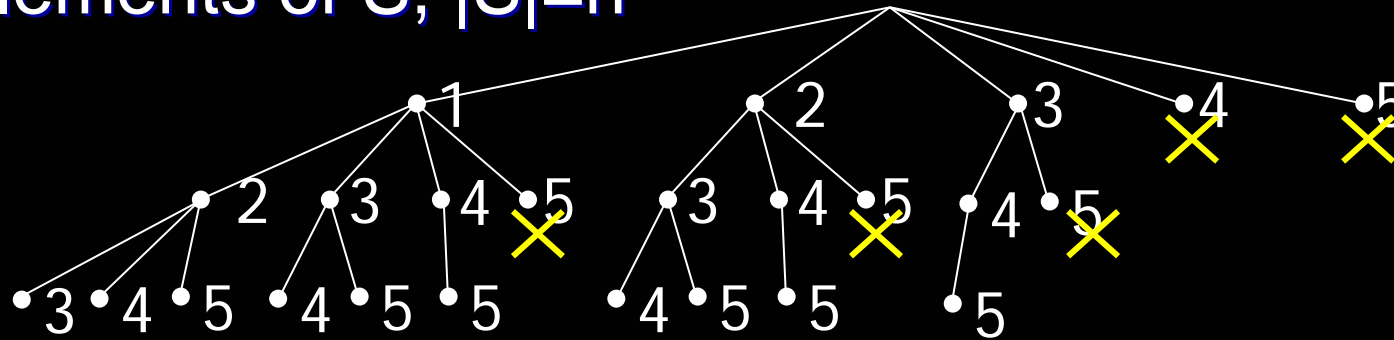
Generate (n,k) -Combinations

- (n,k) -combination of a set S is a subset of k distinct elements of S , $|S|=n$



Generate (n,k)-Combinations

- (n,k)-combination of a set S is a subset of k distinct elements of S, |S|=n

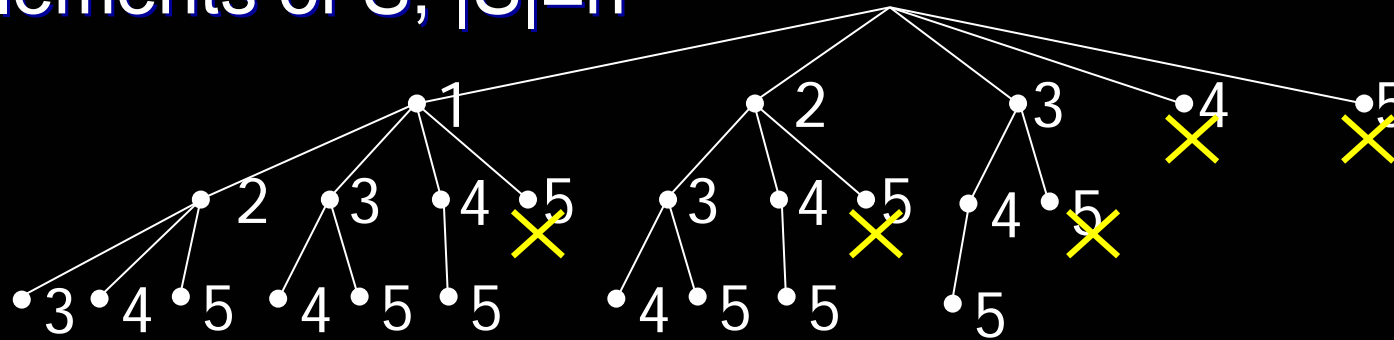


$$C_3^5 = 10$$

(5,3)-combinations

Generate (n,k)-Combinations

- (n,k)-combination of a set S is a subset of k distinct elements of S, |S|=n

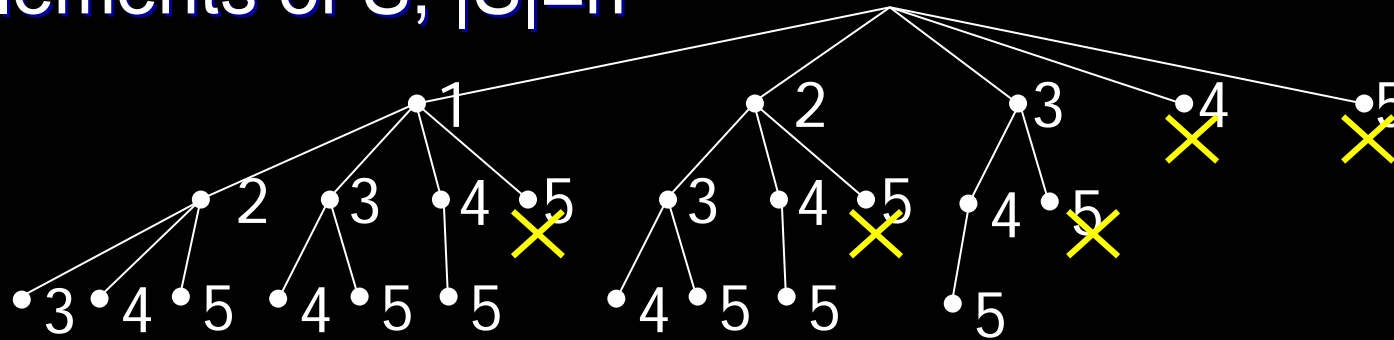


$C_3^5 = 10$
(5,3)-combinations

1	2	3
1	2	4
1	2	5
1	3	4
1	3	5

Generate (n,k)-Combinations

- (n,k)-combination of a set S is a subset of k distinct elements of S, $|S|=n$



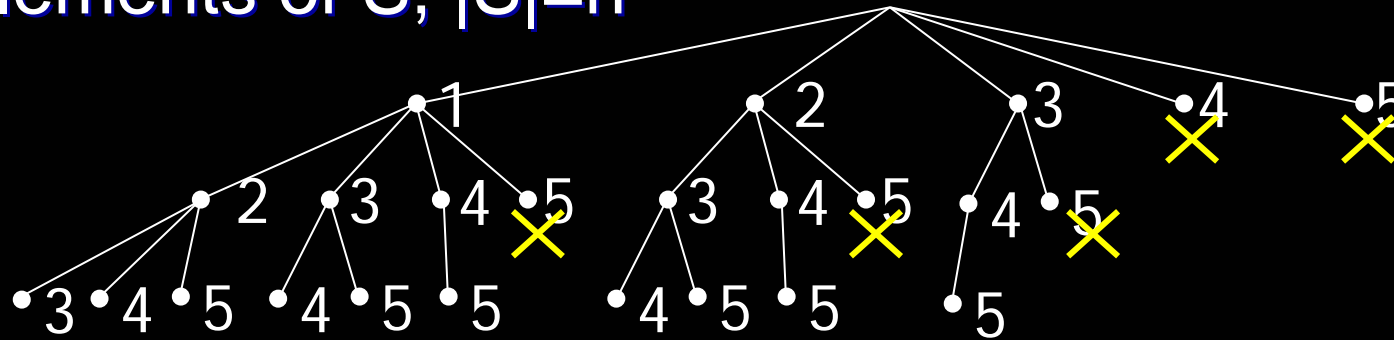
$$C_3^5 = 10$$

(5,3)-combinations

1	2	3	1	4	5
1	2	4	2	3	4
1	2	5	2	3	5
1	3	4	2	4	5
1	3	5	3	4	5

Generate (n,k)-Combinations

- (n,k)-combination of a set S is a subset of k distinct elements of S, |S|=n



$$C_3^5 = 10$$

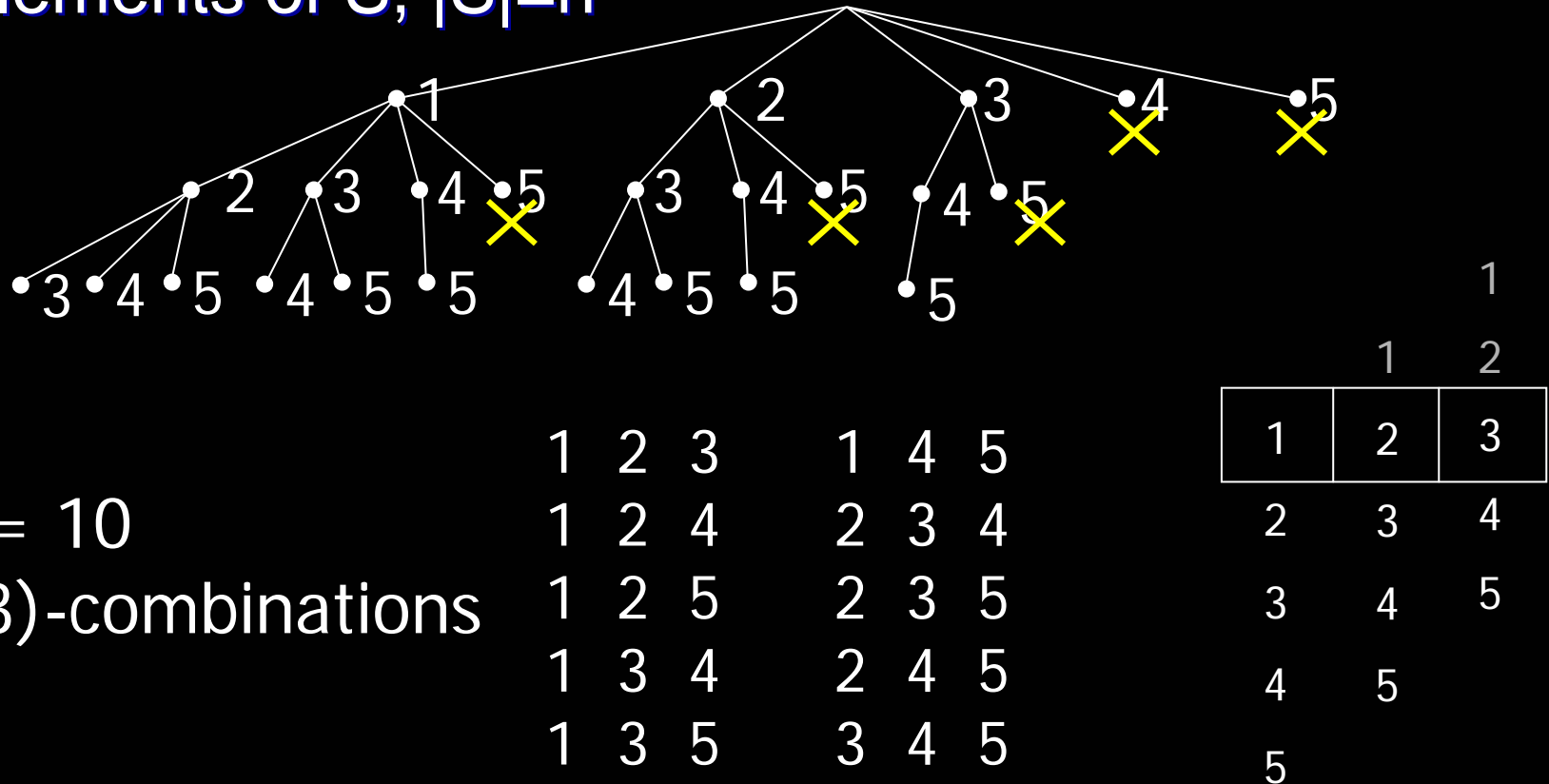
(5,3)-combinations

1	2	3	1	4	5
1	2	4	2	3	4
1	2	5	2	3	5
1	3	4	2	4	5
1	3	5	3	4	5

	1	2
1	2	3
2	3	4
3	4	5
4	5	
5		

Generate (n,k)-Combinations

- (n,k)-combination of a set S is a subset of k distinct elements of S, |S|=n



- Extend the counting-up program and avoid those non-increasing sequences

Efficient Counting w/o Invalid()

Efficient Counting w/o Invalid()

comb

1	2	3
---	---	---

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;
```

Efficient Counting w/o Invalid()

comb

1	2	3
---	---	---

```
int comb[18], n=5, k=3;
for (i=0; i<n; i++)
    comb[i] = i+1;
do
    printArray(comb, k);
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {
```

comb

1	2	3
---	---	---

```
}
```

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {
```

comb

1	2	3
---	---	---

```
    }  
}
```

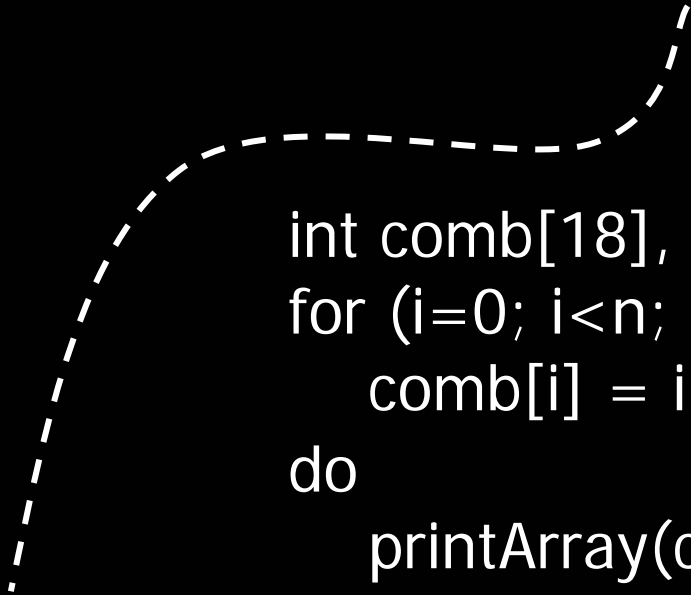
```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
        }  
}
```

comb

1	2	3
---	---	---



```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb

1	2	3
---	---	---

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```


Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb

1	2	3
1	2	

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb

1	2	3
1	2	4

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	5

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	5
	1		

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	5
	1	3	

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	5
	1	3	4

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	5
	1	3	4
	1	3	

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```


Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	5
	1	3	4
	1	3	5

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	5
	1	3	4
	1	3	5
	1		

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	5
	1	3	4
	1	3	5
	1	4	

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	5
	1	3	4
	1	3	5
	1	4	5

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	5
	1	3	4
	1	3	5
	1	4	5
	2		

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	5
	1	3	4
	1	3	5
	1	4	5
	2	3	

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	5
	1	3	4
	1	3	5
	1	4	5
	2	3	4

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	5
	1	3	4
	1	3	5
	1	4	5
	2	3	4
	2	3	

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```


Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	5
	1	3	4
	1	3	5
	1	4	5
	2	3	4
	2	3	5

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Efficient Counting w/o Invalid()

```
void next (int comb[], int n, int k) {  
    int i, j;  
    for (i=k-1; i>=0; i--)  
        if (comb[i]<n-(k-1-i)) {  
            comb[i]++;  
            for (j=i+1; j<k; j++)  
                comb[j] = comb[j-1]+1;  
            return 1;  
        }  
    return 0;  
}
```

comb	1	2	3
	1	2	4
	1	2	5
	1	3	4
	1	3	5
	1	4	5
	2	3	4
	2	3	5
	⋮		

```
int comb[18], n=5, k=3;  
for (i=0; i<n; i++)  
    comb[i] = i+1;  
do  
    printArray(comb, k);  
while (next(comb, n, k));
```

Recursive (n,k)-Combinations

$c(5,3)$

1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

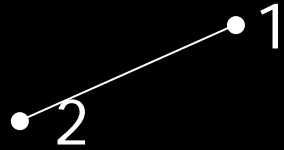
$c(5,3)$

• 1

1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

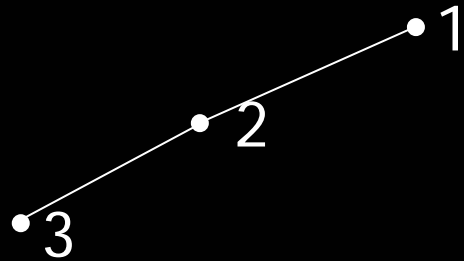
$c(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

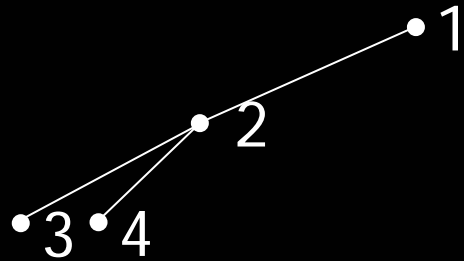
$c(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

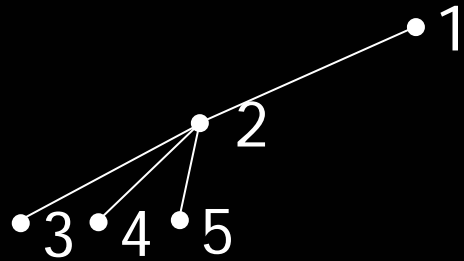
$c(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

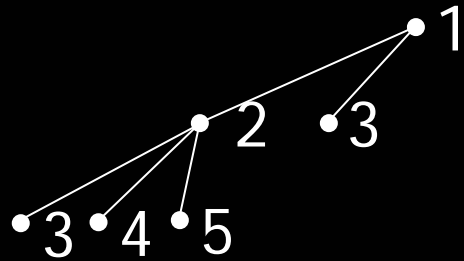
$c(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

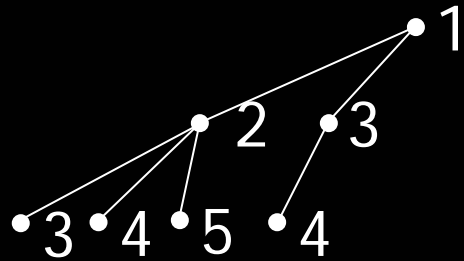
$c(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

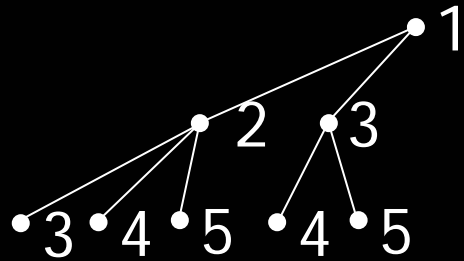
$c(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

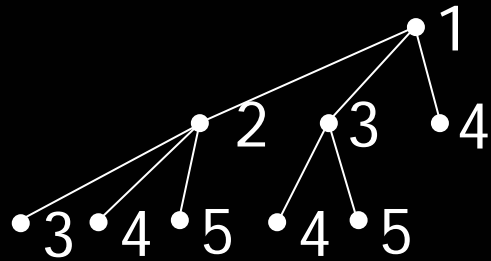
$c(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

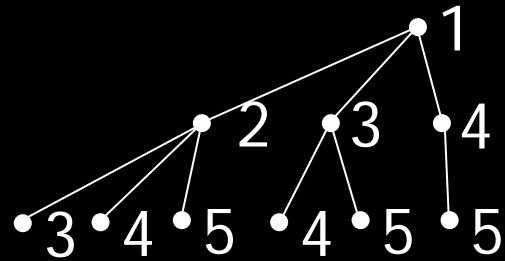
$c(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

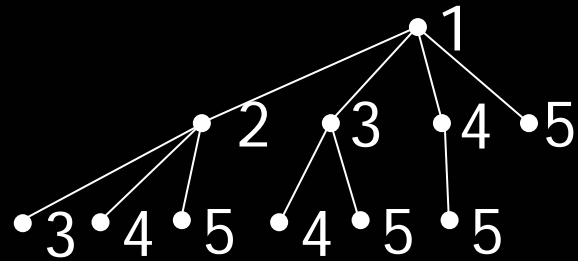
$c(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

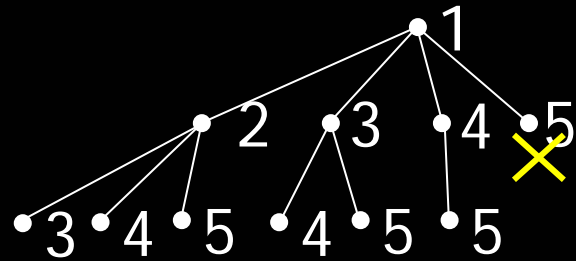
$c(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

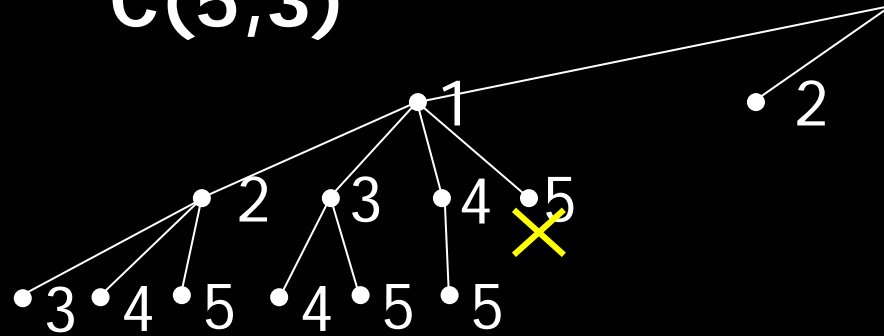
$c(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

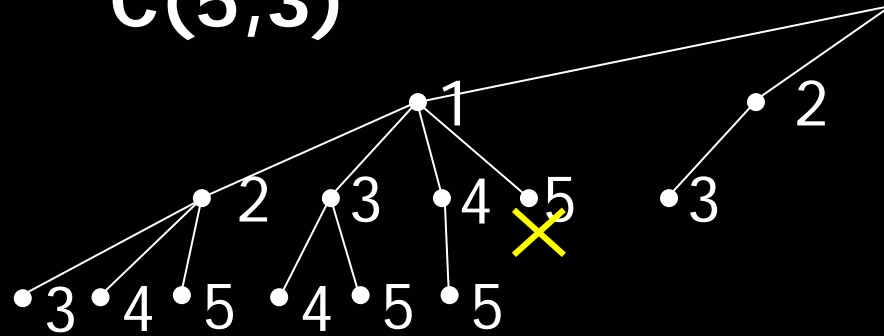
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

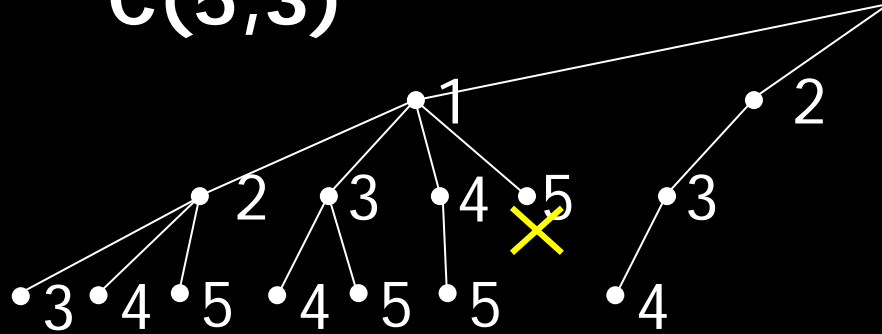
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

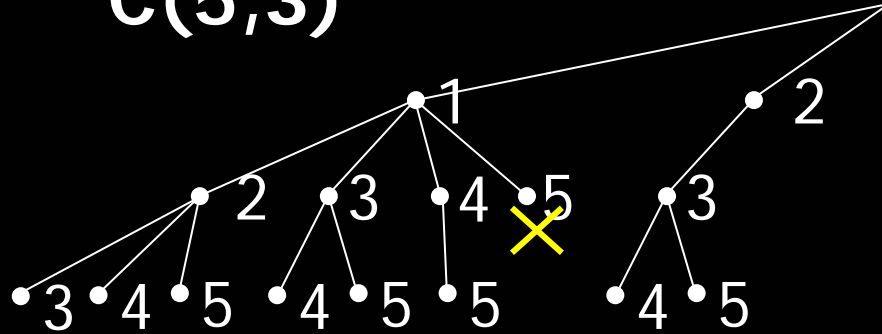
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

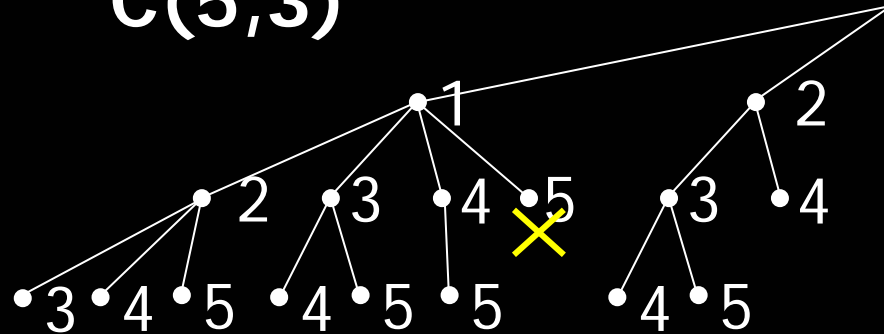
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

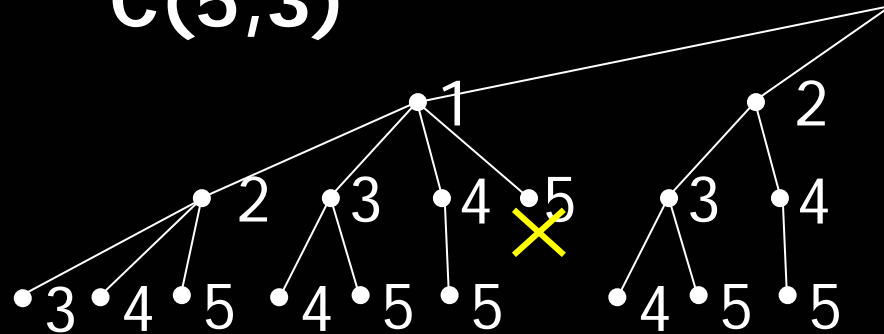
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

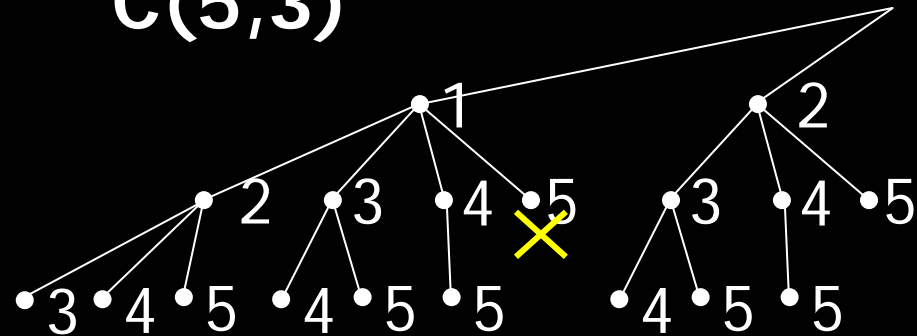
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

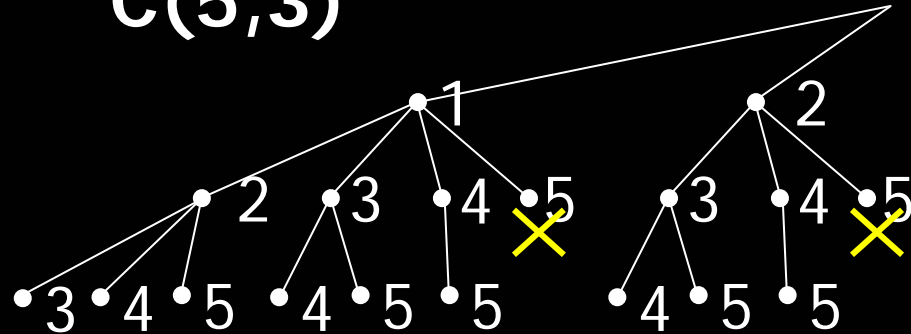
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

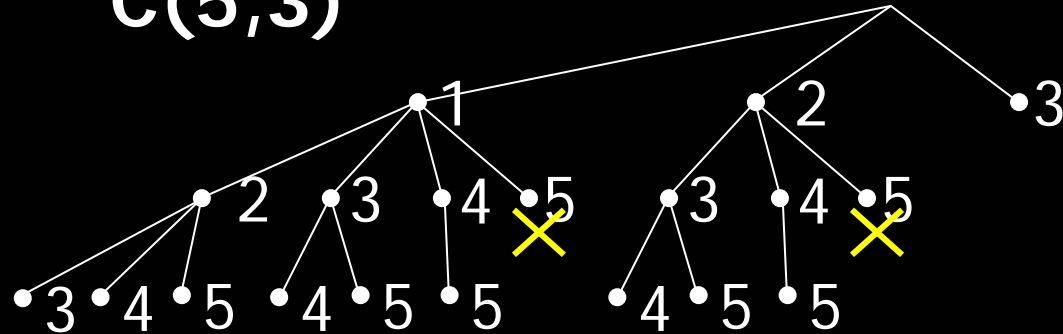
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

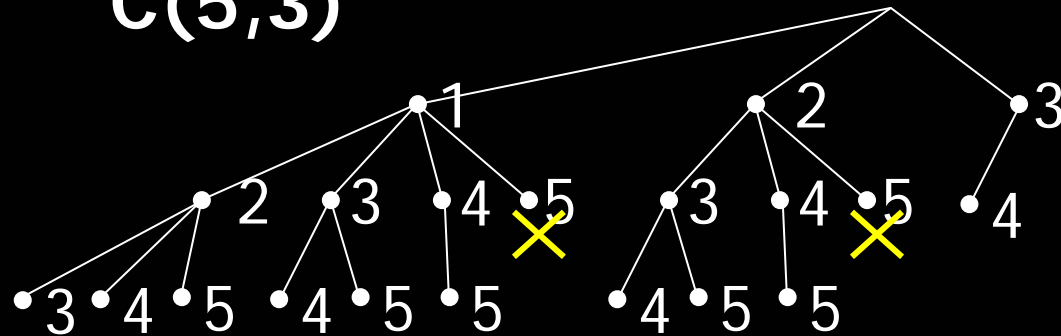
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

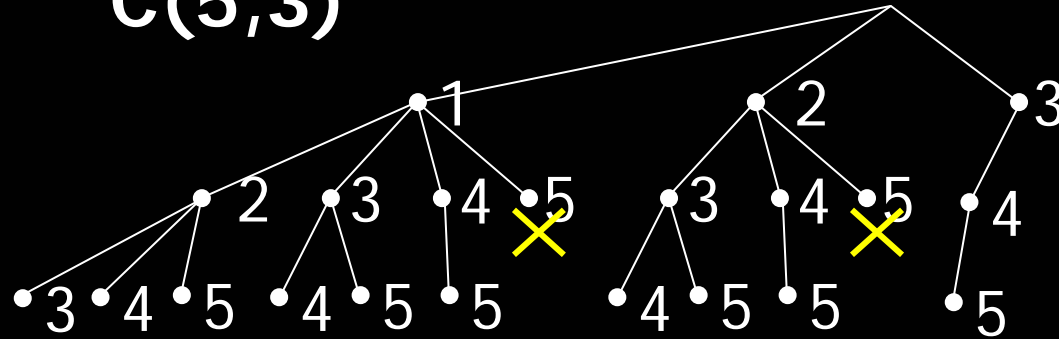
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

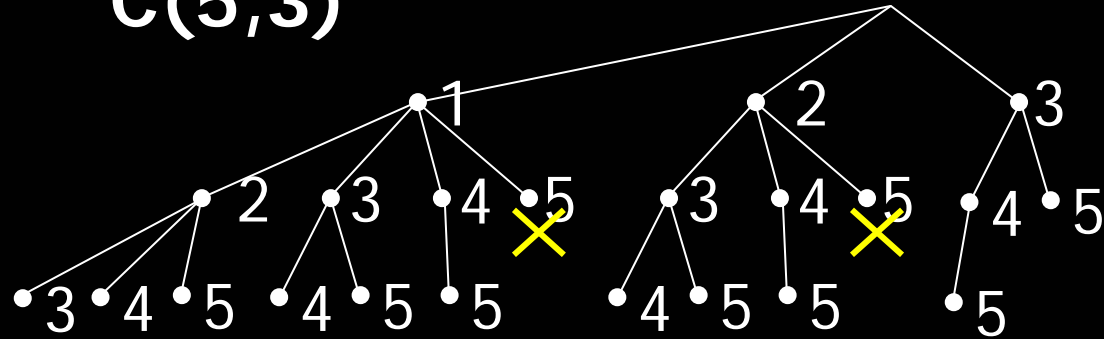
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

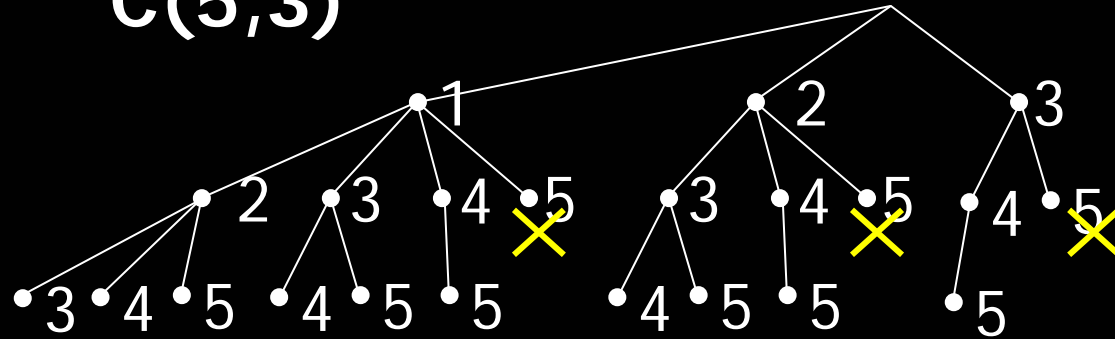
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

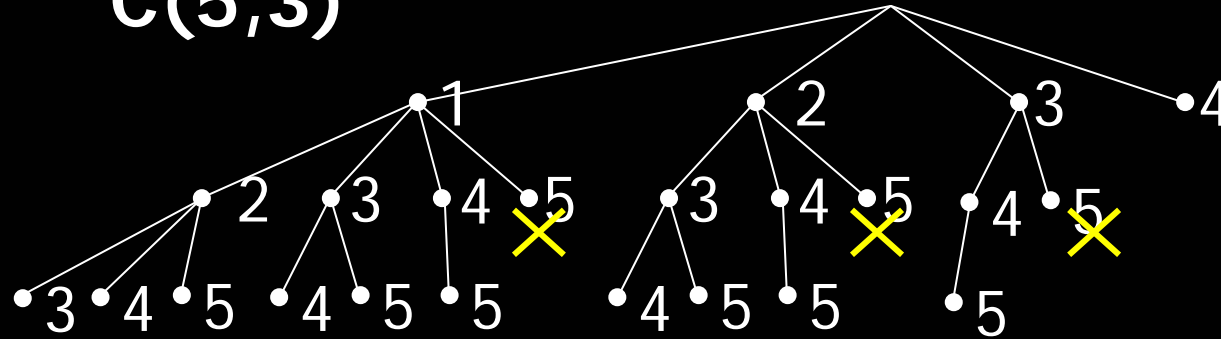
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

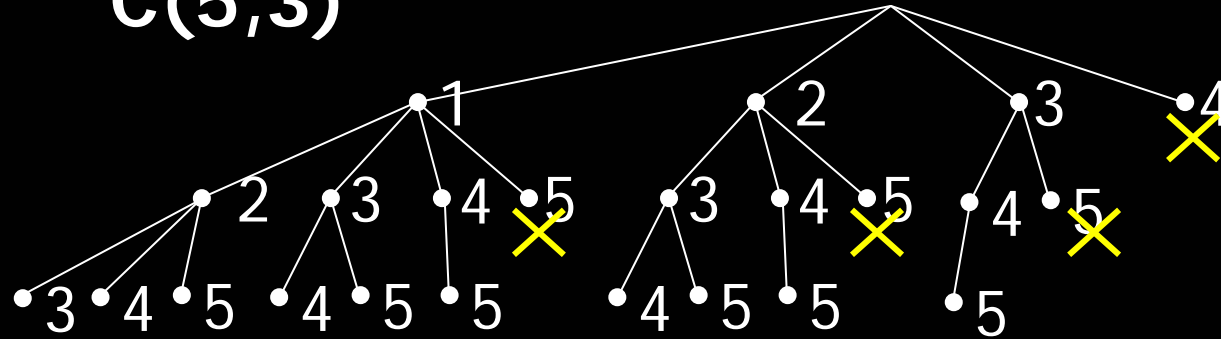
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

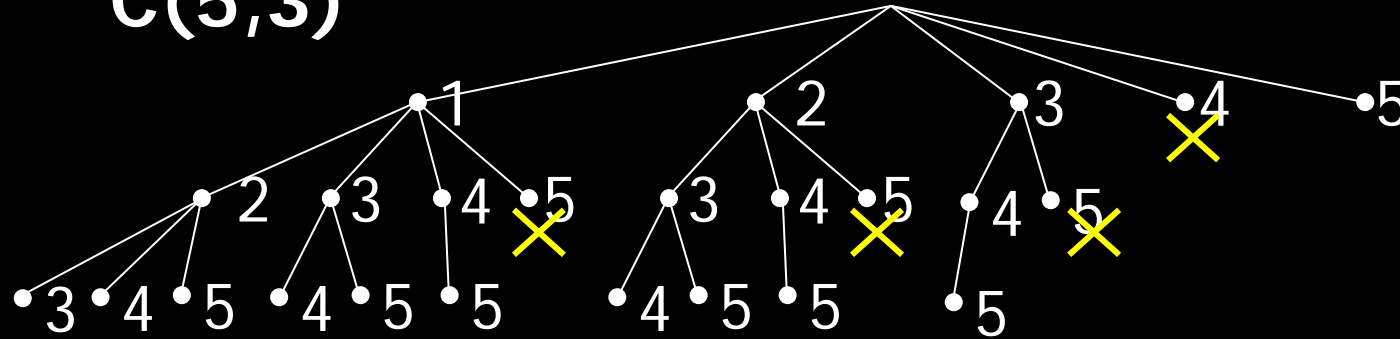
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

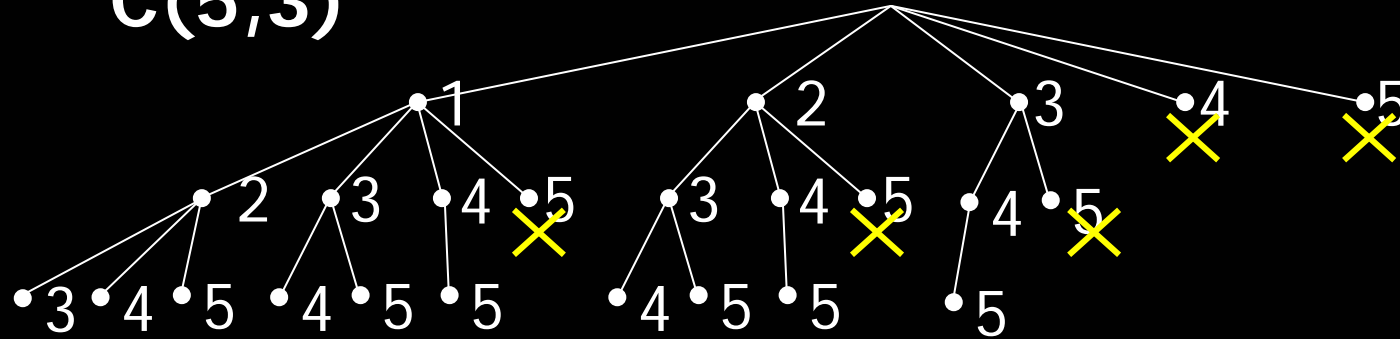
$C(5,3)$



1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

$C(5,3)$

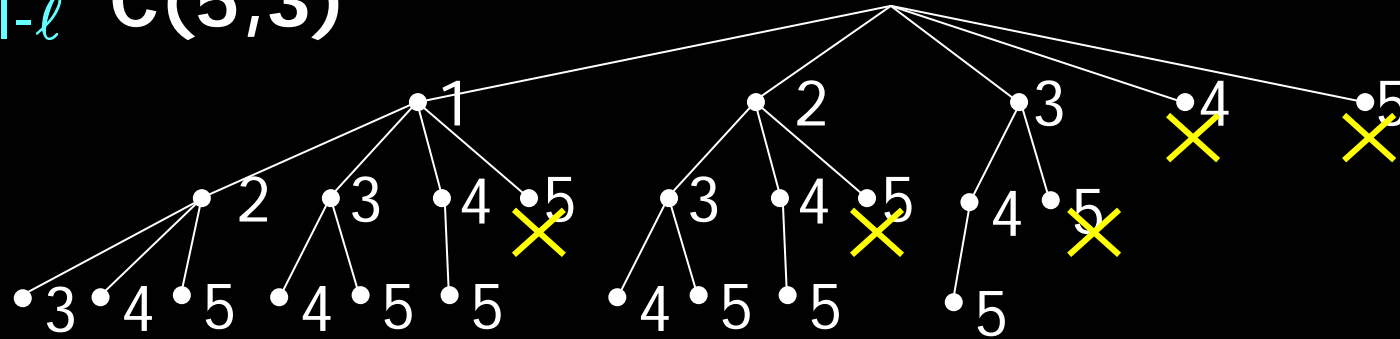


1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

level- ℓ $C(5,3)$

0



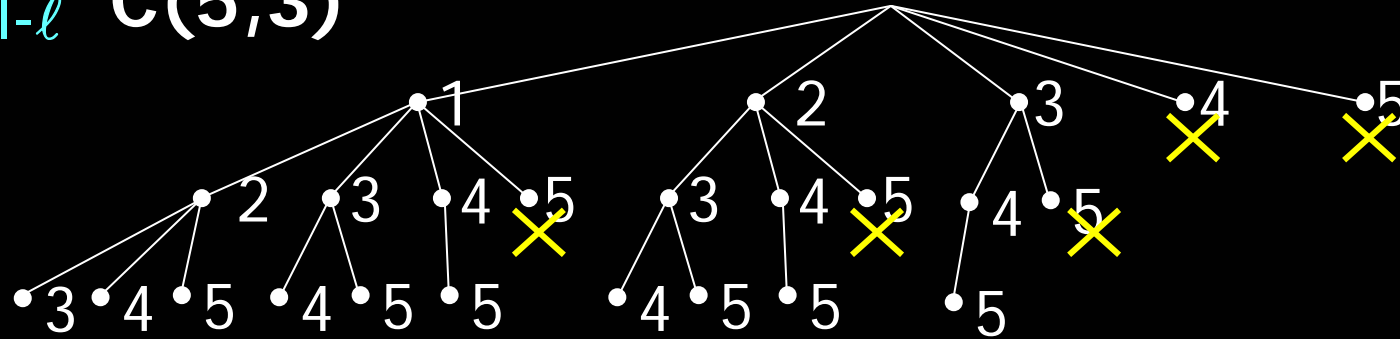
1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

level- ℓ $C(5,3)$

0

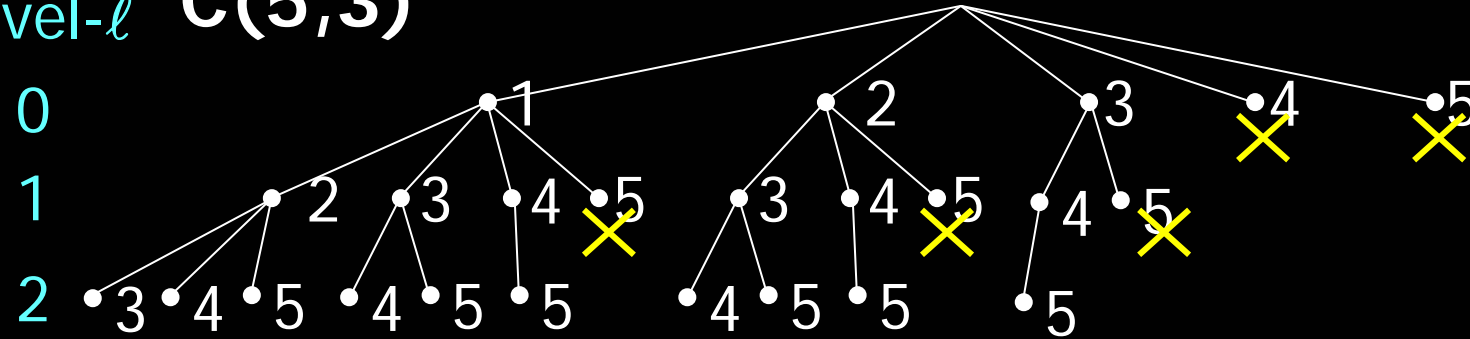
1



1	2	3	4	5
---	---	---	---	---

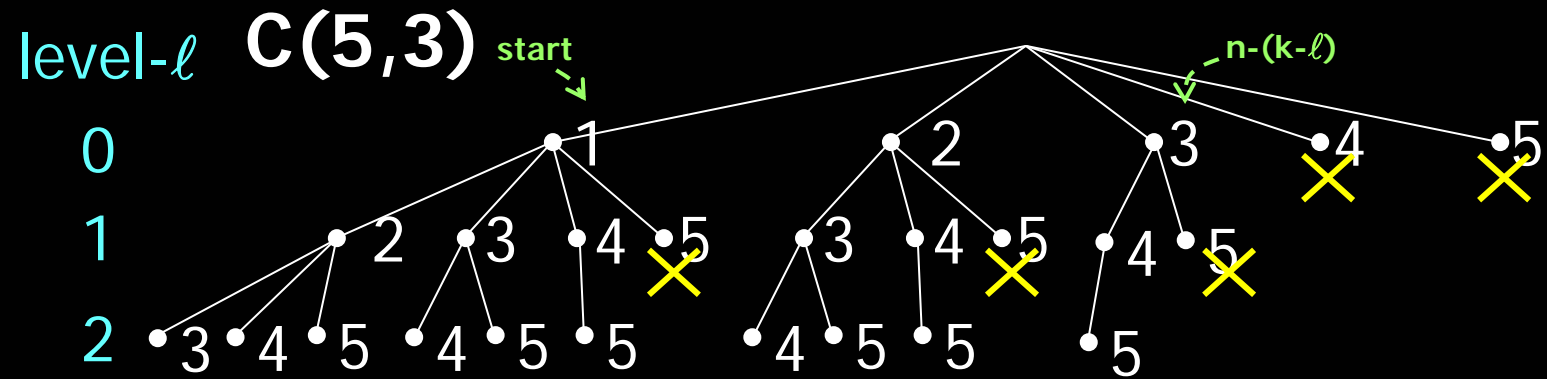
Recursive (n,k)-Combinations

level- ℓ $C(5,3)$



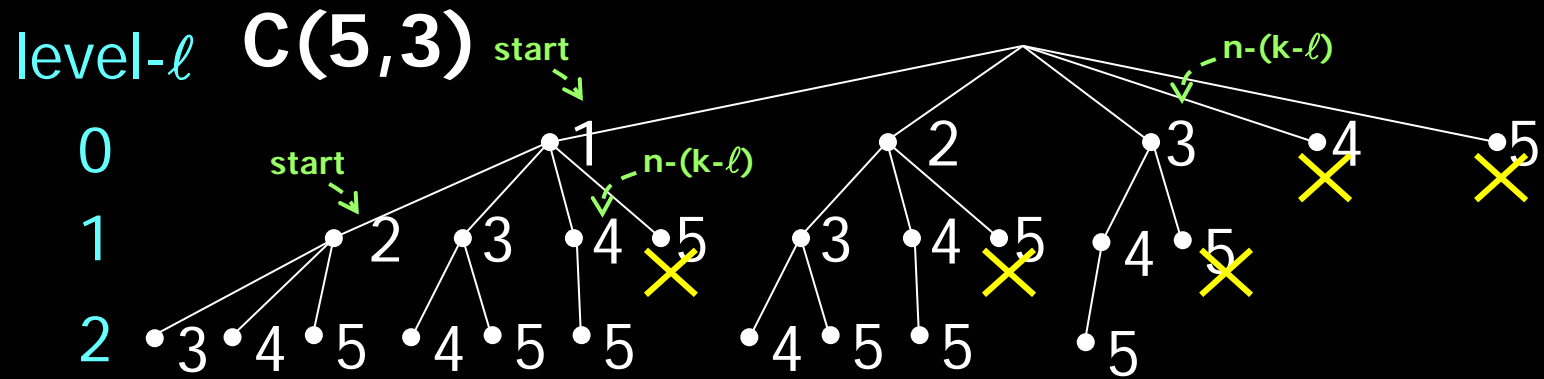
1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations



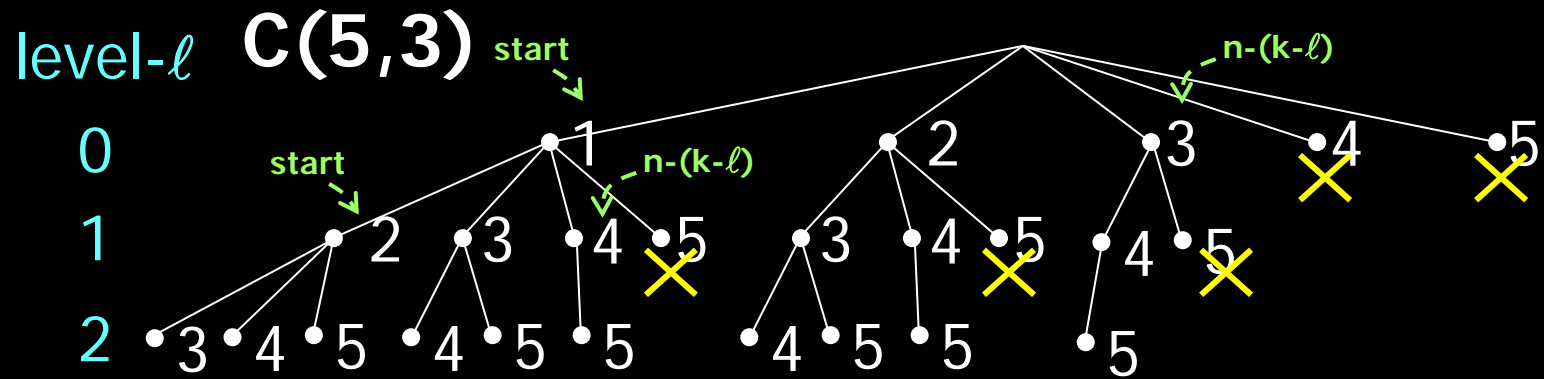
1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations

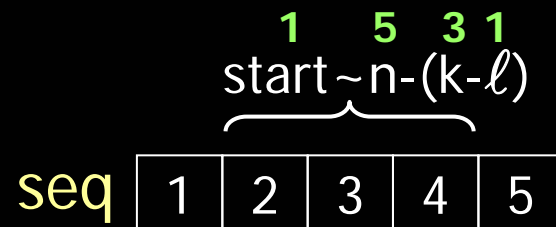


1	2	3	4	5
---	---	---	---	---

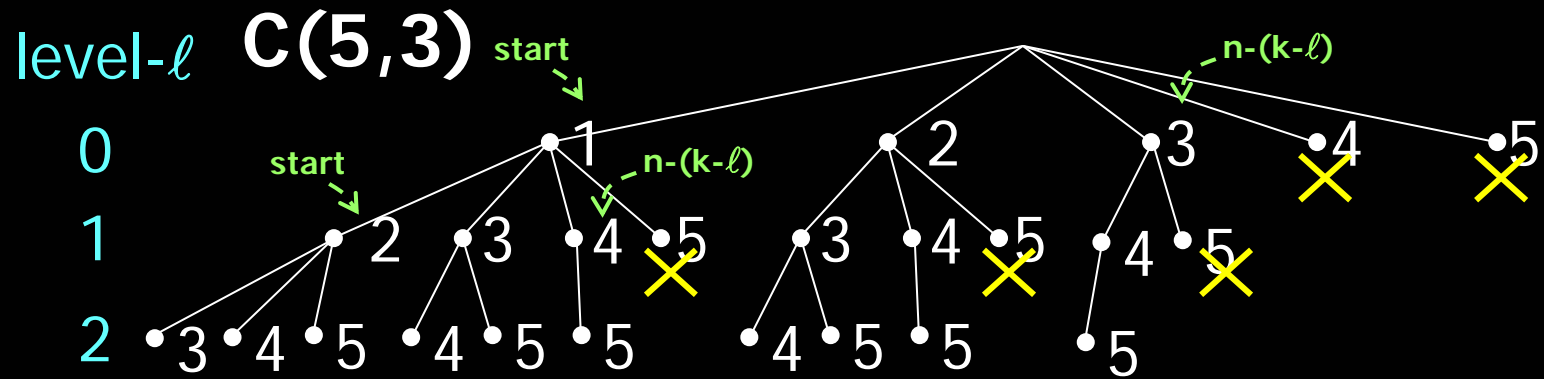
Recursive (n,k)-Combinations



- select level- ℓ elements from $\text{seq}[\text{start}] \sim \text{seq}[\text{n}-(\text{k}-\ell)]$



Recursive (n,k)-Combinations



- select level- ℓ elements from $\text{seq}[\text{start}] \sim \text{seq}[\text{n}-(\text{k}-\ell)]$
- **comb(n,k,seq,0,0,result);**
 void **comb**(int n, int k, const int seq[], int start, int ℓ , int result[]) {

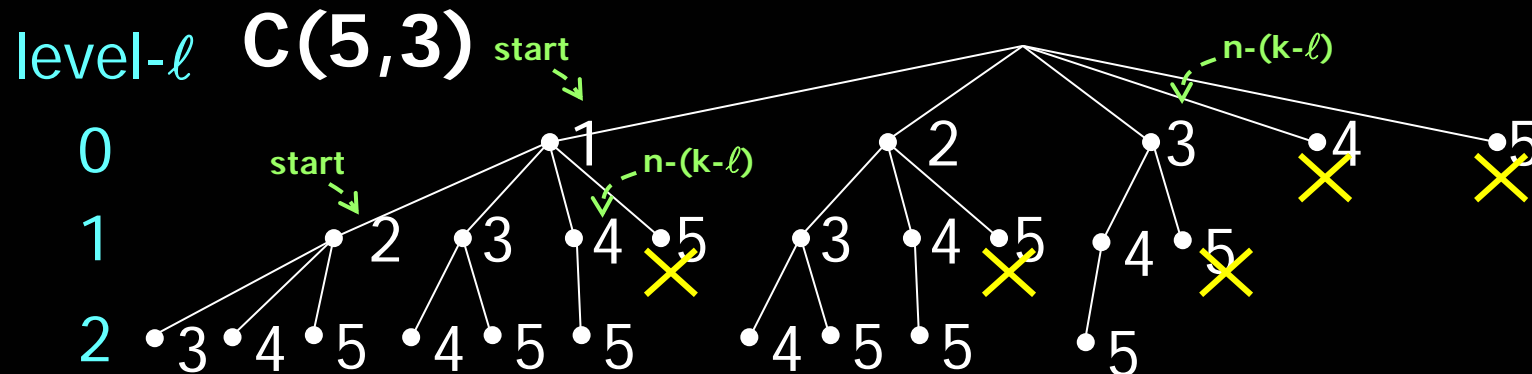
}

1 5 3 1
 $\text{start} \sim \text{n}-(\text{k}-\ell)$

seq

1	2	3	4	5
---	---	---	---	---

Recursive (n,k)-Combinations



- select level- ℓ elements from $\text{seq}[\text{start}] \sim \text{seq}[\text{start} + n - (k - \ell)]$
- put the result to $\text{result}[\ell] \sim \text{result}[\ell + k - 1]$
- **comb(n,k,seq,0,0,result);**

```
void comb(int n, int k, const int seq[], int start, int  $\ell$ , int result[]) {
```

```
}
```

1 5 3 1
start \sim start + n - (k - ℓ)

seq

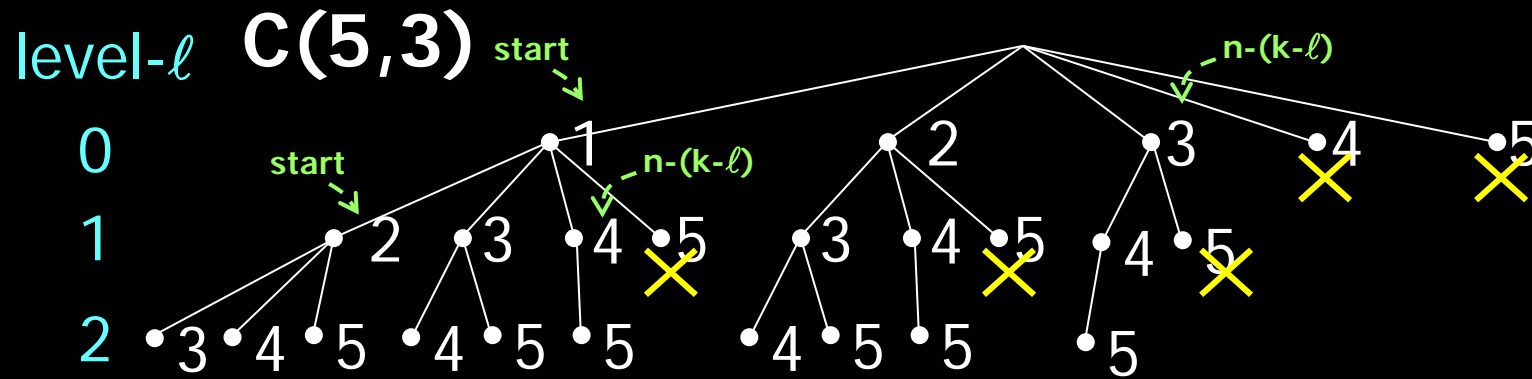
1	2	3	4	5
---	---	---	---	---

1 3
 ℓ \sim $\ell + k - 1$

result

1		
---	--	--

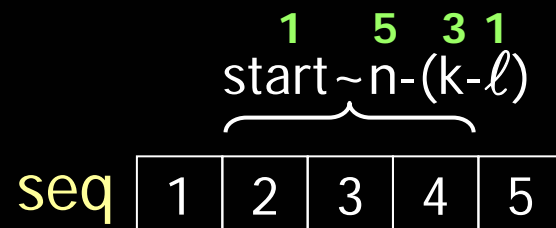
Recursive (n,k)-Combinations



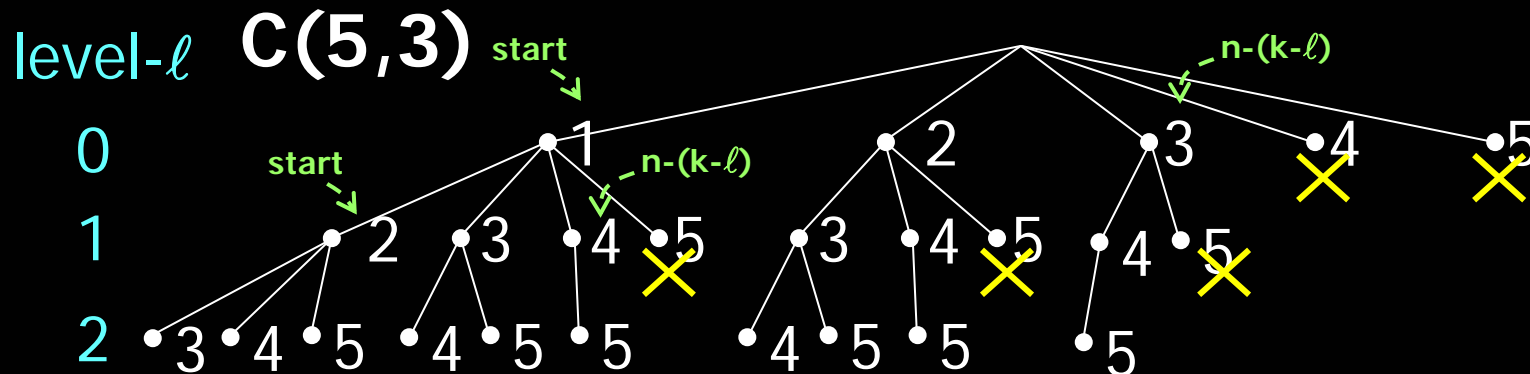
- select level- ℓ elements from $seq[start] \sim seq[n-(k-\ell)]$
- put the result to $result[\ell] \sim result[k-1]$
- **comb(n,k,seq,0,0,result);**

```
void comb(int n, int k, const int seq[], int start, int  $\ell$ , int result[]) {
    if ( $\ell == k$ ) print(result, k); // result[0] ~ result[k-1]
    else
```

}



Recursive (n,k)-Combinations

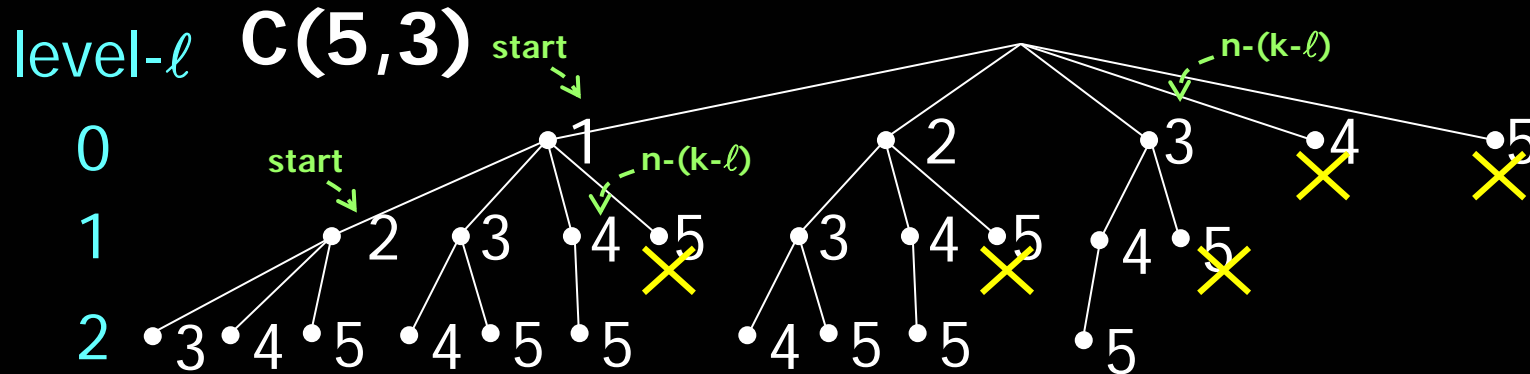


- select level- ℓ elements from $\text{seq}[\text{start}] \sim \text{seq}[\text{n}-(\text{k}-\ell)]$
- put the result to $\text{result}[\ell] \sim \text{result}[\text{k}-1]$
- **comb(n,k,seq,0,0,result);**

```
void comb(int n, int k, const int seq[], int start, int  $\ell$ , int result[]) {
    if ( $\ell == k$ ) print(result, k); // result[0] ~ result[k-1]
    else for (int i=start; i<=n-(k- $\ell$ ); i++)
        result[ $\ell$ ] = seq[i],
        comb(n, k, seq, i+1,  $\ell+1$ , result);
}
```



Recursive (n,k)-Combinations

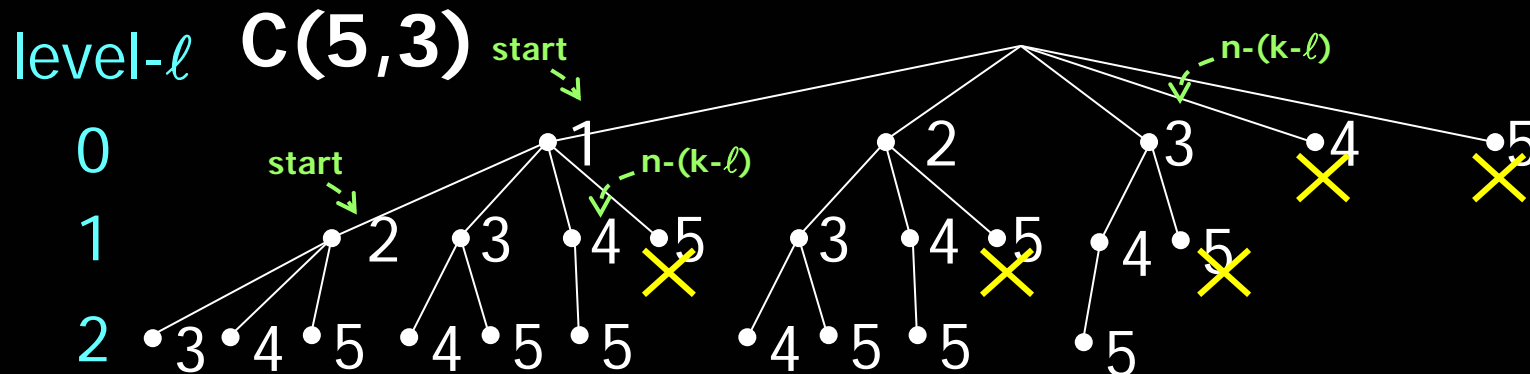


- select level- ℓ elements from $\text{seq}[\text{start}] \sim \text{seq}[n-(k-\ell)]$
- put the result to $\text{result}[\ell] \sim \text{result}[k-1]$
- **comb(n,k,seq,0,0,result);**

```
void comb(int n, int k, const int seq[], int start, int  $\ell$ , int result[]) {
    if ( $\ell == k$ ) print(result, k); // result[0] ~ result[k-1]
    else for (int i=start; i<=n-(k- $\ell$ ); i++)
        result[ $\ell$ ] = seq[i],
        comb(n, k, seq, i+1,  $\ell+1$ , result);
}
```



Recursive (n,k)-Combinations



- select level- ℓ elements from $\text{seq}[\text{start}] \sim \text{seq}[\text{n}-(\text{k}-\ell)]$
- put the result to $\text{result}[\ell] \sim \text{result}[\text{k}-1]$
- **comb(n,k,seq,0,0,result);**

```
void comb(int n, int k, const int seq[], int start, int  $\ell$ , int result[]) {
    if ( $\ell == k$ ) print(result, k); // result[0] ~ result[k-1]
    else for (int i=start; i<=n-(k- $\ell$ ); i++)
        result[ $\ell$ ] = seq[i],
        comb(n, k, seq, i+1,  $\ell+1$ , result);
}
```



Generate Power Set

- The power set of a set is the collection of all subsets of S , including the empty set and S itself.
 - e.g. $S = \{1, 2, 3\}$,
 $2^S = \{\{\}, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$
- We can extend the program for generating (n,k)-combinations to generate the whole power set.
- ```
int n=3, seq[]={1,2,3}, result[3], k;
for (k=0; k<=n; k++)
 comb(n, k, seq, 0, 0, result);
```

# 正整數 $N$ 的加法拆解(整數劃分)

- 每個正整數  $N$  都可以寫成比它小的數字和, 請把所有不重複整數加總等於  $N$  的方法寫出來

# 正整數 $N$ 的加法拆解(整數劃分)

- 每個正整數  $N$  都可以寫成比它小的數字和, 請把所有不重複整數加總等於  $N$  的方法寫出來

Sample Input

6

Sample Output

1 2 3

1 5

2 4

6

# 正整數 $N$ 的加法拆解(整數劃分)

- 每個正整數  $N$  都可以寫成比它小的數字和, 請把所有不重複整數加總等於  $N$  的方法寫出來

Sample Input

6

Sample Output

1 2 3

1 5

2 4

6

Sample Input

10

Sample Output

1 2 3 4

1 2 7

1 3 6

1 4 5

2 3 5

3 7

4 6

10



# 正整數 $N$ 的加法拆解(整數劃分)

- 每個正整數  $N$  都可以寫成比它小的數字和, 請把所有不重複整數加總等於  $N$  的方法寫出來

Sample Input

6

Sample Output

1 2 3  
1 5  
2 4  
6

Sample Input

10

Sample Output

1 2 3 4  
1 2 7  
1 3 6  
1 4 5  
2 3 5  
3 7  
4 6  
10

Sample Input

15

Sample Output

1 2 3 4 5  
1 2 3 9  
1 2 4 8  
1 2 5 7  
1 2 12  
1 3 4 7  
1 3 5 6  
1 3 11  
1 4 10  
1 5 9  
1 6 8  
1 14

2 3 4 6  
2 3 10  
2 4 9  
2 5 8  
2 6 7  
2 13  
3 4 8  
3 5 7  
3 12  
4 5 6  
4 11  
5 10  
6 9  
7 8  
15

# 正整數 $N$ 的加法拆解(整數劃分)

- 每個正整數  $N$  都可以寫成比它小的數字和, 請把所有不重複整數加總等於  $N$  的方法寫出來

Sample Input

6

Sample Output

1 2 3  
1 5  
2 4  
6

Sample Input

10

Sample Output

1 2 3 4  
1 2 7  
1 3 6  
1 4 5  
2 3 5  
3 7  
4 6  
10

Sample Input

15

Sample Output

1 2 3 4 5  
1 2 3 9  
1 2 4 8  
1 2 5 7  
1 2 12  
1 3 4 7  
1 3 5 6  
1 3 11  
1 4 10  
1 5 9  
1 6 8  
1 14

2 3 4 6  
2 3 10  
2 4 9  
2 5 8  
2 6 7  
2 13  
3 4 8  
3 5 7  
3 12  
4 5 6  
4 11  
5 10  
6 9  
7 8  
15

- 輸出請依照範例以一般化字典序 (lexicographic order) 排列

# Sudoku

- **Sudoku:** In these three examples, 81 cells are divided into 9 blocks each with 9 cells (3-by-3). A player is required to fill in the blank cells such that integers in each row, each column, and each block are permutations of  $\{1,2,3,\dots,9\}$ , i.e. no duplication of numbers in each row, column, or block.

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 7 | 8 | 9 |   | 2 |   |   |   | 5 |
| 6 |   |   |   | 5 |   | 8 |   |   |
|   |   |   |   |   |   | 1 |   |   |
| 2 |   |   | 8 |   |   | 5 | 1 |   |
|   |   | 5 | 7 | 1 | 6 |   |   |   |
| 9 | 1 |   |   | 6 |   |   |   | 3 |
|   | 7 |   |   |   |   |   |   |   |
|   | 5 |   | 9 |   |   |   |   | 6 |
| 8 |   |   |   | 1 | 5 | 3 | 7 |   |

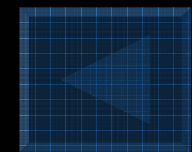
number of lines  
row, column, value  
row, column, value

**Initial configuration**

30  
0, 0, 7  
0, 1, 8  
0, 2, 9  
0, 4, 2  
...  
0, 8, 5  
1, 0, 6  
1, 5, 5  
1, 7, 8  
...

|   |   |   |   |   |   |   |   |  |
|---|---|---|---|---|---|---|---|--|
|   |   | 6 | 1 | 3 | 4 |   |   |  |
|   |   | 3 |   | 8 |   |   |   |  |
| 5 | 4 |   | 7 |   | 1 | 2 |   |  |
|   |   | 2 |   |   |   |   | 4 |  |
|   | 5 |   | 3 | 9 |   | 7 |   |  |
| 7 |   |   |   | 4 |   |   |   |  |
|   | 3 | 7 |   | 5 |   | 4 | 2 |  |
|   |   |   | 8 |   | 7 |   |   |  |
|   |   | 1 | 4 | 7 | 8 |   |   |  |

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 3 |   | 8 |   |   |   | 6 |   |   |
|   | 6 |   |   |   |   | 4 |   |   |
|   | 9 |   | 8 | 5 |   |   | 1 |   |
| 2 |   |   | 9 |   |   |   |   |   |
| 4 |   | 5 |   |   |   | 9 |   | 2 |
|   |   |   |   |   | 7 |   |   | 6 |
|   | 2 |   |   | 1 | 3 |   | 6 |   |
|   |   | 9 |   |   |   |   | 3 |   |
|   |   | 1 |   |   |   | 5 |   | 4 |



# Sudoku (cont'd)

- Extension of permutation generation

|   |   |   |   |  |   |   |  |  |
|---|---|---|---|--|---|---|--|--|
|   |   | 6 | 1 |  | 3 | 4 |  |  |
|   |   | 3 |   |  | 8 |   |  |  |
| 5 | 4 |   |   |  |   |   |  |  |
|   |   |   |   |  |   |   |  |  |
|   |   |   |   |  |   |   |  |  |
|   | 5 |   |   |  |   |   |  |  |
| 7 |   |   |   |  |   |   |  |  |
|   | 3 |   |   |  |   |   |  |  |
|   |   |   |   |  |   |   |  |  |
|   |   |   |   |  |   |   |  |  |

|   |   |   |   |   |   |   |   |  |
|---|---|---|---|---|---|---|---|--|
|   |   | 6 | 1 | 3 | 4 |   |   |  |
|   |   | 3 |   | 8 |   |   |   |  |
| 5 | 4 |   | 7 |   | 1 | 2 |   |  |
|   |   | 2 |   |   |   |   | 4 |  |
|   | 5 | 3 | 9 |   | 7 |   |   |  |
| 7 |   |   | 4 |   |   |   |   |  |
|   | 3 | 7 |   | 5 |   | 4 | 2 |  |
|   |   | 8 |   |   | 7 |   |   |  |
|   |   | 1 | 4 | 7 | 8 |   |   |  |

# Sudoku (cont'd)

- Extension of permutation generation

|   |   |   |   |  |   |   |  |  |
|---|---|---|---|--|---|---|--|--|
|   |   | 6 | 1 |  | 3 | 4 |  |  |
|   |   | 3 |   |  | 8 |   |  |  |
| 5 | 4 |   |   |  |   |   |  |  |
|   |   |   |   |  |   |   |  |  |
|   | 5 |   |   |  |   |   |  |  |
| 7 |   |   |   |  |   |   |  |  |
|   | 3 |   |   |  |   |   |  |  |
|   |   |   |   |  |   |   |  |  |
|   |   |   |   |  |   |   |  |  |

...

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
|   |   | 6 | 1 | 3 | 4 |   |   |
|   |   | 3 |   | 8 |   |   |   |
| 5 | 4 |   | 7 |   | 1 | 2 |   |
|   |   | 2 |   |   |   |   | 4 |
|   | 5 |   | 3 | 9 |   | 7 |   |
| 7 |   |   |   | 4 |   |   |   |
|   | 3 | 7 |   | 5 |   | 4 | 2 |
|   |   |   | 8 |   | 7 |   |   |
|   |   | 1 | 4 | 7 | 8 |   |   |

more constraints  
on the set of  
values to be filled  
in each cell

# Sudoku (cont'd)

- Extension of permutation generation

|   |   |   |   |  |   |   |  |  |
|---|---|---|---|--|---|---|--|--|
| 2 |   | 6 | 1 |  | 3 | 4 |  |  |
|   |   | 3 |   |  | 8 |   |  |  |
| 5 | 4 |   |   |  |   |   |  |  |
|   |   |   |   |  |   |   |  |  |
|   | 5 |   |   |  |   |   |  |  |
| 7 |   |   |   |  |   |   |  |  |
|   | 3 |   |   |  |   |   |  |  |
|   |   |   |   |  |   |   |  |  |
|   |   |   |   |  |   |   |  |  |

...

✕  
 2  
 ✕  
 ✕  
 ✕  
 ✕  
 ✕  
 ✕  
 8  
 9

|   |   |   |   |   |   |   |     |
|---|---|---|---|---|---|---|-----|
|   |   | 6 | 1 | 3 | 4 |   |     |
|   |   | 3 |   | 8 |   |   |     |
| 5 | 4 |   | 7 |   | 1 | 2 |     |
|   |   | 2 |   |   |   |   | 4   |
|   | 5 |   | 3 | 9 |   | 7 |     |
| 7 |   |   |   | 4 |   |   |     |
|   | 3 | 7 |   |   | 5 |   | 4 2 |
|   |   |   | 8 |   |   | 7 |     |
|   |   | 1 | 4 |   | 7 | 8 |     |

more constraints on the set of values to be filled in each cell

# Sudoku (cont'd)

- Extension of permutation generation

|   |   |   |   |  |   |   |  |  |
|---|---|---|---|--|---|---|--|--|
| 2 | 7 | 6 | 1 |  | 3 | 4 |  |  |
|   |   | 3 |   |  | 8 |   |  |  |

|   |   |  |
|---|---|--|
| 5 | 4 |  |
|   |   |  |
|   | 5 |  |
| 7 |   |  |
|   | 3 |  |
|   |   |  |
|   |   |  |

...

|   |              |
|---|--------------|
| × | ×            |
| 2 | <del>2</del> |
| × | ×            |
| × | ×            |
| × | ×            |
| × | ×            |
| × | ×            |
| 8 | 7            |
| 9 | 8            |
|   | 9            |

|   |   |   |   |   |   |   |     |
|---|---|---|---|---|---|---|-----|
|   |   | 6 | 1 | 3 | 4 |   |     |
|   |   | 3 |   | 8 |   |   |     |
| 5 | 4 |   | 7 |   | 1 | 2 |     |
|   |   | 2 |   |   |   |   | 4   |
|   | 5 |   | 3 | 9 |   | 7 |     |
| 7 |   |   |   | 4 |   |   |     |
|   | 3 | 7 |   |   | 5 |   | 4 2 |
|   |   |   | 8 |   |   | 7 |     |
|   |   | 1 | 4 |   | 7 | 8 |     |

more constraints on the set of values to be filled in each cell

# Sudoku (cont'd)

- Extension of permutation generation

|   |   |   |   |     |   |   |  |  |
|---|---|---|---|-----|---|---|--|--|
| 2 | 7 | 6 | 1 |     | 3 | 4 |  |  |
| 1 |   | 3 |   |     | 8 |   |  |  |
| 5 | 4 |   |   |     |   |   |  |  |
|   |   |   |   | ... |   |   |  |  |
|   | 5 |   |   |     |   |   |  |  |
| 7 |   |   |   |     |   |   |  |  |
|   | 3 |   |   |     |   |   |  |  |
|   |   |   |   |     |   |   |  |  |
|   |   |   |   |     |   |   |  |  |

|              |              |              |
|--------------|--------------|--------------|
| ×            | ×            | 1            |
| 2            | <del>2</del> | <del>2</del> |
| ×            | ×            | ×            |
| <del>3</del> | ×            | ×            |
| ×            | ×            | ×            |
| ×            | ×            | ×            |
| ×            | 7            | <del>7</del> |
| 8            | 8            | ×            |
| 9            | 9            | 9            |

|   |   |   |   |   |   |   |     |
|---|---|---|---|---|---|---|-----|
|   |   | 6 | 1 | 3 | 4 |   |     |
|   |   | 3 |   | 8 |   |   |     |
| 5 | 4 |   | 7 |   | 1 | 2 |     |
|   |   | 2 |   |   |   |   | 4   |
|   | 5 |   | 3 | 9 |   | 7 |     |
| 7 |   |   |   | 4 |   |   |     |
|   | 3 | 7 |   |   | 5 |   | 4 2 |
|   |   |   | 8 |   |   | 7 |     |
|   |   | 1 | 4 |   | 7 | 8 |     |

more constraints on the set of values to be filled in each cell



# Sudoku (cont'd)

- Extension of permutation generation

|   |   |   |   |     |   |   |  |  |
|---|---|---|---|-----|---|---|--|--|
| 2 | 7 | 6 | 1 |     | 3 | 4 |  |  |
| 1 | 9 | 3 |   |     | 8 |   |  |  |
| 5 | 4 |   |   |     |   |   |  |  |
|   |   |   |   | ... |   |   |  |  |
|   | 5 |   |   |     |   |   |  |  |
| 7 |   |   |   |     |   |   |  |  |
|   | 3 |   |   |     |   |   |  |  |
|   |   |   |   |     |   |   |  |  |
|   |   |   |   |     |   |   |  |  |

|   |   |   |   |
|---|---|---|---|
| × | × | 1 | × |
| 2 | × | × | × |
| × | × | × | × |
| × | × | × | × |
| × | × | × | × |
| × | × | × | × |
| × | 7 | × | × |
| 8 | 8 | × | × |
| 9 | 9 | 9 | 9 |

|   |   |   |   |   |   |   |     |
|---|---|---|---|---|---|---|-----|
|   |   | 6 | 1 | 3 | 4 |   |     |
|   |   | 3 |   | 8 |   |   |     |
| 5 | 4 |   | 7 |   | 1 | 2 |     |
|   |   | 2 |   |   |   |   | 4   |
|   | 5 |   | 3 | 9 |   | 7 |     |
| 7 |   |   |   | 4 |   |   |     |
|   | 3 | 7 |   |   | 5 |   | 4 2 |
|   |   |   | 8 |   |   | 7 |     |
|   |   | 1 | 4 |   | 7 | 8 |     |

more constraints on the set of values to be filled in each cell

# Determinant of an n-by-n Matrix

- Leibniz formula

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

# Determinant of an n-by-n Matrix

- Leibniz formula

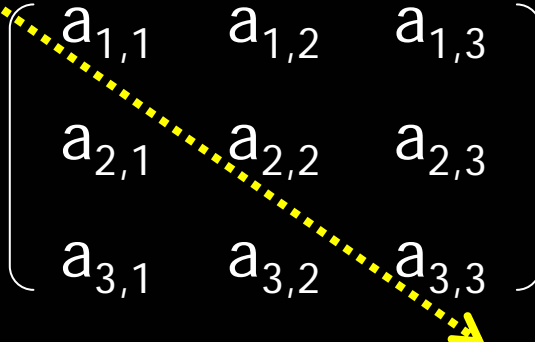
$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

- Determinant

$$\det(\mathbf{A}) =$$

# Determinant of an n-by-n Matrix

- Leibniz formula

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$


- Determinant

$$\det(\mathbf{A}) = a_{1,1}a_{2,2}a_{3,3}$$

# Determinant of an n-by-n Matrix

- Leibniz formula

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

- Determinant

$$\det(\mathbf{A}) = a_{1,1}a_{2,2}a_{3,3} + a_{1,2}a_{2,3}a_{3,1}$$

# Determinant of an n-by-n Matrix

- Leibniz formula

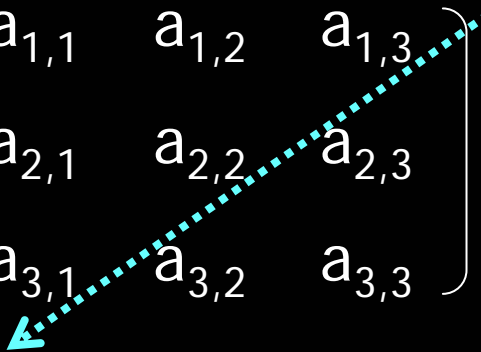
$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

- Determinant

$$\det(\mathbf{A}) = a_{1,1}a_{2,2}a_{3,3} + a_{1,2}a_{2,3}a_{3,1} + a_{1,3}a_{2,1}a_{3,2}$$

# Determinant of an n-by-n Matrix

- Leibniz formula

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$


- Determinant

$$\det(\mathbf{A}) = a_{1,1}a_{2,2}a_{3,3} + a_{1,2}a_{2,3}a_{3,1} + a_{1,3}a_{2,1}a_{3,2} \\ - a_{1,3}a_{2,2}a_{3,1}$$

# Determinant of an n-by-n Matrix

- Leibniz formula

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

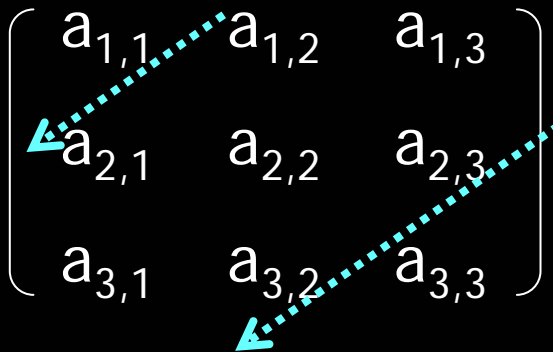
- Determinant

$$\det(\mathbf{A}) = a_{1,1}a_{2,2}a_{3,3} + a_{1,2}a_{2,3}a_{3,1} + a_{1,3}a_{2,1}a_{3,2} \\ - a_{1,3}a_{2,2}a_{3,1} - a_{1,2}a_{2,1}a_{3,3}$$



# Determinant of an n-by-n Matrix

- Leibniz formula

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$


- Determinant

$$\det(\mathbf{A}) = a_{1,1}a_{2,2}a_{3,3} + a_{1,2}a_{2,3}a_{3,1} + a_{1,3}a_{2,1}a_{3,2} \\ - a_{1,3}a_{2,2}a_{3,1} - a_{1,2}a_{2,1}a_{3,3} - a_{1,1}a_{2,3}a_{3,2}$$

# Determinant of an n-by-n Matrix

- Leibniz formula

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

- Determinant

$$\det(\mathbf{A}) = a_{1,1}a_{2,2}a_{3,3} + a_{1,2}a_{2,3}a_{3,1} + a_{1,3}a_{2,1}a_{3,2} \\ - a_{1,3}a_{2,2}a_{3,1} - a_{1,2}a_{2,1}a_{3,3} - a_{1,1}a_{2,3}a_{3,2}$$

- Permutation?

$$\det(\mathbf{A}) = a_{1,1}a_{2,2}a_{3,3} + a_{1,2}a_{2,3}a_{3,1} + a_{1,3}a_{2,1}a_{3,2} \\ - a_{1,3}a_{2,2}a_{3,1} - a_{1,2}a_{2,1}a_{3,3} - a_{1,1}a_{2,3}a_{3,2}$$

# Determinant of an n-by-n Matrix

- Leibniz formula

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

- Determinant

$$\det(\mathbf{A}) = a_{1,1}a_{2,2}a_{3,3} + a_{1,2}a_{2,3}a_{3,1} + a_{1,3}a_{2,1}a_{3,2} \\ - a_{1,3}a_{2,2}a_{3,1} - a_{1,2}a_{2,1}a_{3,3} - a_{1,1}a_{2,3}a_{3,2}$$

- Permutation?

$$\det(\mathbf{A}) = a_{1,1}a_{2,2}a_{3,3} + a_{1,2}a_{2,3}a_{3,1} + a_{1,3}a_{2,1}a_{3,2} \\ - a_{1,3}a_{2,2}a_{3,1} - a_{1,2}a_{2,1}a_{3,3} - a_{1,1}a_{2,3}a_{3,2}$$

- $\text{sign}(\sigma) = \begin{cases} 1 & \text{if } \sigma \text{ has even number of misplaced pairs} \\ -1 & \text{if ... odd ...} \end{cases}$

e.g.  $\text{sign}(123)=1$ ,  $\text{sign}(231)=1$ ,  $\text{sign}(312)=1$   
 $\text{sign}(321)=-1$ ,  $\text{sign}(213)=-1$ ,  $\text{sign}(132)=-1$

# Applications

1. 給定  $n$  個各種長度的木棒, 是否可排出正方形? 是否可排出指定長寬比例的矩形?
2. 有  $m$  條星狀連接的路徑, 在各路徑途中指定位置有不同金額的獎勵, 如果油料有限制, 最多只能夠走  $n$  公里, 請找到由中心點出發能夠取回最大獎勵的方式
3.  $n$  個小偷竊得  $m$  個不同價值的物品, 怎樣才能比較公平地分贓而不至於內鬩呢?
4.  $n$  組數字, 由每一組數字中各取一個, 總和最大的取法?
5. 尋找  $n \times n$  魔方陣, 洛書
6. Tetromino & pentomino
7. 輸入整數  $N$ ,  $2 \leq N \leq 9$ , 印出一  $N$  位數字  $X$ , 其  $N$  個數字都不重複且  $\text{mod } 10^i$  皆能被  $N$  整除, 例如  $N = 8$ ,  $X = 21579648$ 、 $X \text{ mod } 10^7 = 1579648$ 、 $579648$ 、 $79648$ 、 $9648$ 、 $648$ 、 $48$ 、 $8$  皆可以被  $8$  整除且各位數字不重複出現。