Chapter 1

Induction, Recurrence Equations, and Recursive Calls

COMP2119C

Triangular numbers

n	1	2	3	4	5	6	7	8	9	10	…
$T(n)$	1	3	6	10	15	21	28	36	45	55	

How to describe this sequence of numbers to a friend?

Triangular numbers

n	1	2	3	4	5	6	7	8	9	10	…
$T(n)$	1	3	6	10	15	21	28	36	45	55	

The 1st number is ‘1’,
The 2nd number is ‘3’,
The 3rd number is ‘6’, …

The description is long yet incomplete.

Triangular numbers

n	1	2	3	4	5	6	7	8	9	10	...
$T(n)$	1	3	6	10	15	21	28	36	45	55	

The 1st number is '1'.

The n-th number ($T(n)$) is obtained by adding n to the previous number ($T(n-1)$).

sufficient to reconstruct the whole sequence.

Recurrence equation

A mathematical function is *recursive* if it is defined in terms of itself,

e.g.,

$$f(n) = \begin{cases} 0, & n = 0 \\ f(n-1) + n^2, & n > 0 \end{cases}$$

is *well defined* for non-negative integers. We call the above equation a *recurrence equation*.

Unwrapping

The value of a recursive function given certain parameter value(s) can be obtained by "unwrapping" until a base case is reached, e.g., $f(3)$ can be computed as follows:

$$
\begin{aligned}
f(3) &= f(2) + 3^2 \\
&= f(1) + 2^2 + 9 \\
&= f(0) + 1^2 + 13 \\
&= 14
\end{aligned}
$$

Two rules

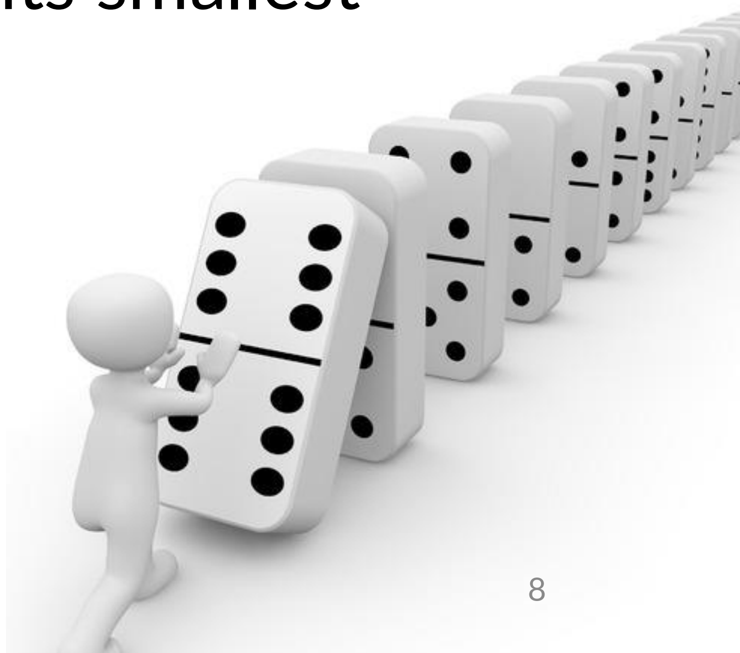
2 important rules in defining a recursive function:

- A *base case*, i.e., the case for which the value of the function is directly known without resorting to recursion.
- *Progress*. For the cases that are to be solved recursively, the recursive call must always be to a case that makes progress towards a base case.

Mathematical Induction

Mathematical Induction is a general way to prove that some statement $S(n)$ about the integer n is true for all $n \geq n_o$ (a certain constant). It involves two steps:

- Step 1: we prove the statement $S(n)$ when n has its smallest value, n_o; this is called the *basis*.
- Step 2: we prove that if $S(n_o), \ldots, S(k)$ is true for some $k \geq n_o$, then $S(k+1)$ is also true. This is called *induction*.



Example

One can easily prove (by M.I.) that

$$f(n) = \frac{n(n+1)(2n+1)}{6}, \qquad \forall n \geq 0$$

Recursive function

Similar to M.I. and recurrence equations, in a computer program, a function/procedure can be recursive, e.g., $f(n)$ can be computed by:

```
int f(n) {          /* assume n >= 0 */
  if (n == 0) return (0);
  else return(f(n-1) + n*n);
}
```

Note that in defining a recursive function/procedure, one should observe the rules of recursion (i.e., "*base case*" and "*progress*").

Recursive function

In a sense, in recursive programming, we use a solution of a *smaller* problem to solve a *larger* problem.

For example, if one knows the solution of $f(n–1)$ (a smaller problem), one can compute $f(n)$ (a larger problem) fairly easily.

$$f(n) = \begin{cases} 0, & n = 0 \\ f(n-1) + n^2, & n > 0 \end{cases}$$

Example (recursive programming)

Problem: given an array $A[1, n]$, reverse the elements in A.

Example:

Example

If we want to come up with a solution using recursion, what are the two issues we need to address?

Example

If we want to come up with a solution using recursion, what are the two issues we need to address?

We need:

- Some *base cases*, and
- A strategy for making *progress*.

Example

What is a base case?

Example

What is a base case?

A base case is one whose solution is readily known/available.
Note that there could be more than 1 base case.

In our example problem, we know that:

- if the array is empty, we don't have to do anything to reverse it;
- similarly, if there is only one element in the array, we don't need to do anything at all.

Example

Recall that the idea of recursion is to use the solution of a *smaller* problem to solve a *larger* problem.

So,

- what do we mean by “small” and “large”?
  - We need a measure of *problem size*.
- what would be a good measure of the problem size in the example?

Example

Recall that the idea of recursion is to use the solution of a *smaller* problem to solve a *larger* problem.

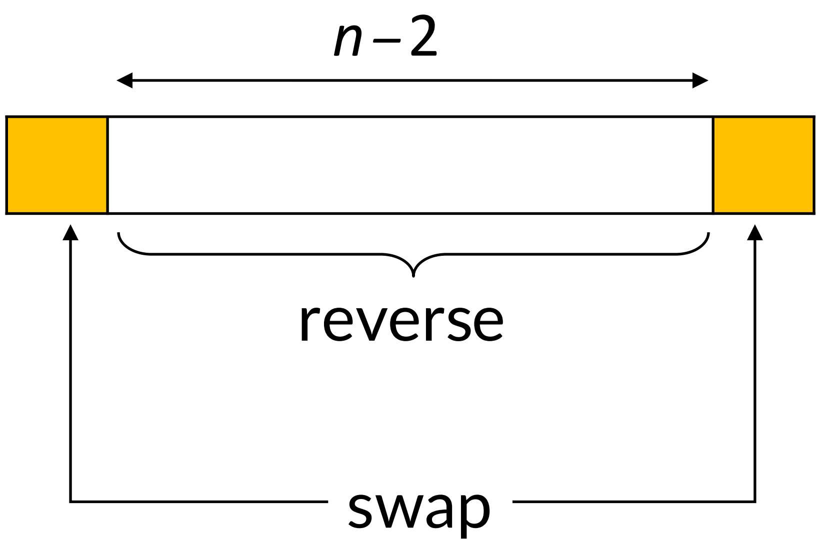
So,

- what do we mean by “small” and “large”?
    - We need a measure of *problem size*.
- what would be a good measure of the problem size in the example?

Perhaps ***n***, the number of elements in the array.

Example

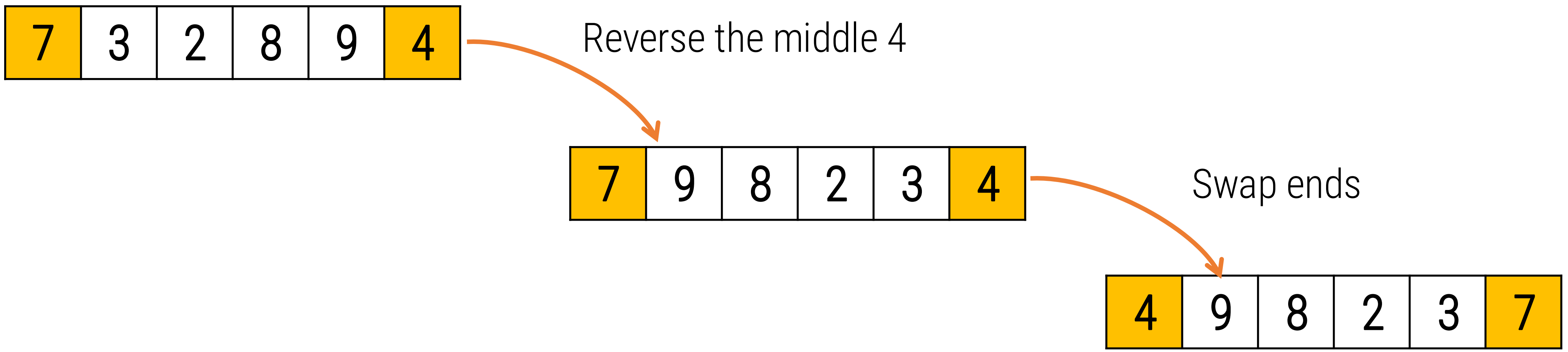
Insights: if we know how to reverse an array with $n-2$ elements, we know how to reverse an array with n elements.



If we know how to reverse a (smaller) segment of the array, we know how to reverse the whole array!

Example

Insights: if we know how to reverse an array with $n-2$ elements, we know how to reverse an array with n elements.



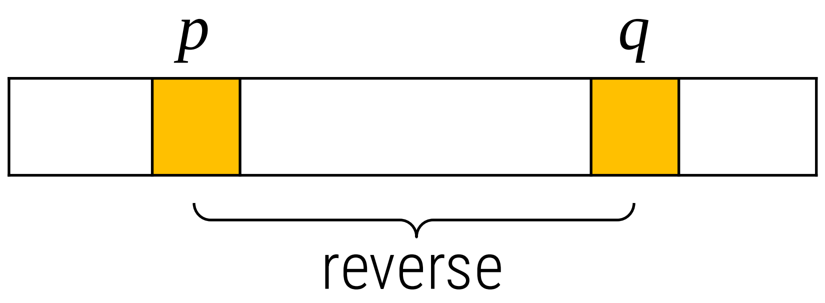
Example

Let's define a recursive function reverse(A, p, q) where A is an array and p, q are two indexes. The semantics is to reverse the elements in the segment $A[p, q]$.

```
reverse(A,p,q) {
  if (p >= q) return;          /* base cases */
  swap(A[p],A[q]);             /* swap the elements A[p] and A[q] */
  reverse(A,p+1,q-1);
}
```

To reverse the whole array, call

reverse(A,1,n);



Another example

Problem: given n characters stored in the array $A[1..n]$, generate all the permutations of the n characters.

E.g., if A is $[a,b,c]$ then the permutations are:

$[a,b,c]$, $[a,c,b]$, $[b,a,c]$, $[b,c,a]$, $[c,a,b]$ $[c,b,a]$.

Permutation

Again, if we want to use a recursive solution, we need:

- one (or more) base case,
- a recursion strategy (i.e., how to make progress towards the base case).

Permutation

Can you come up with some base cases?

Example

Recursion strategy (progress):

- The permutations can be divided into n groups.
  All permutations in the i-th group are ended with the i-th character.
- Each group can be formed by permuting the other $n-1$ characters and then adding the special one at the end.

Example

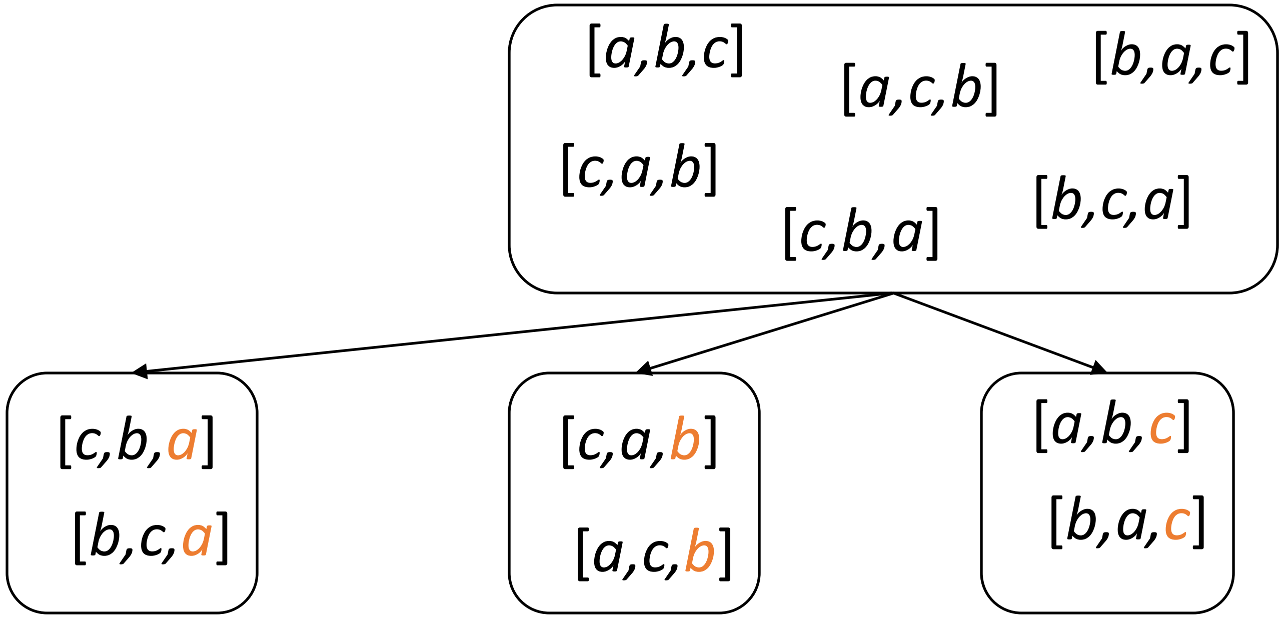
Consider the 6 permutations generated from 3 characters:

$[a,b,c]$ $[a,c,b]$ $[b,a,c]$

$[c,a,b]$ $[c,b,a]$ $[b,c,a]$

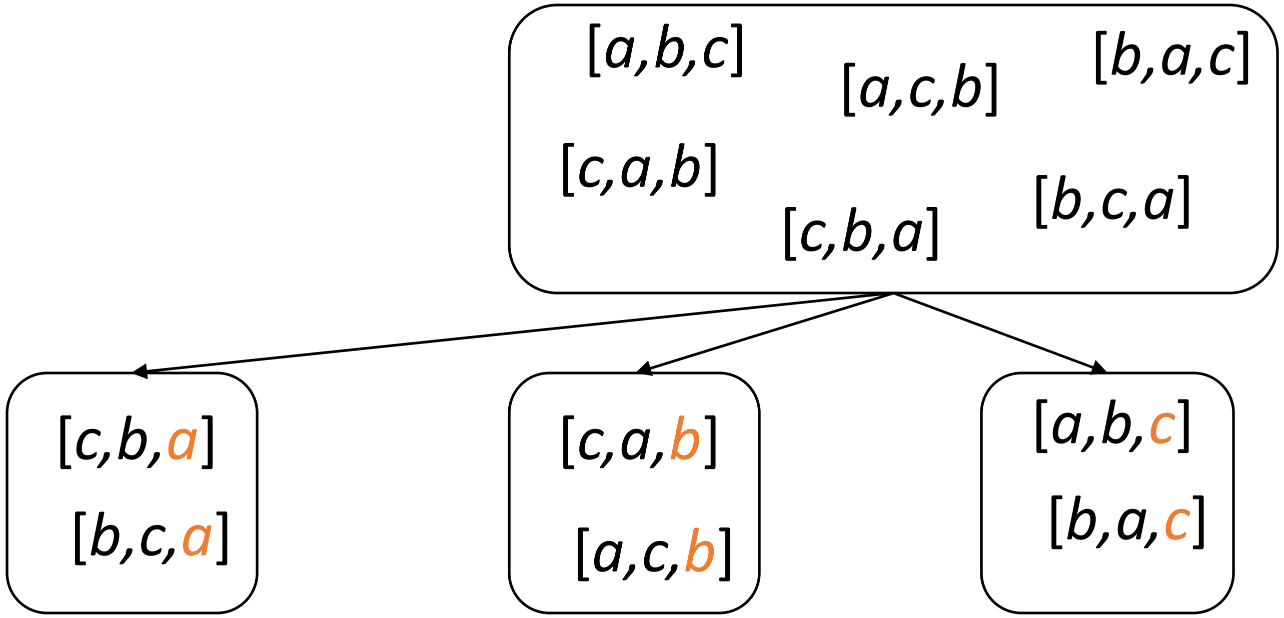
Example

We can divide them into 3 groups.



Example

Note that each group has a specific character at the end, and the group is formed by permuting the rest of the characters.



Permutation

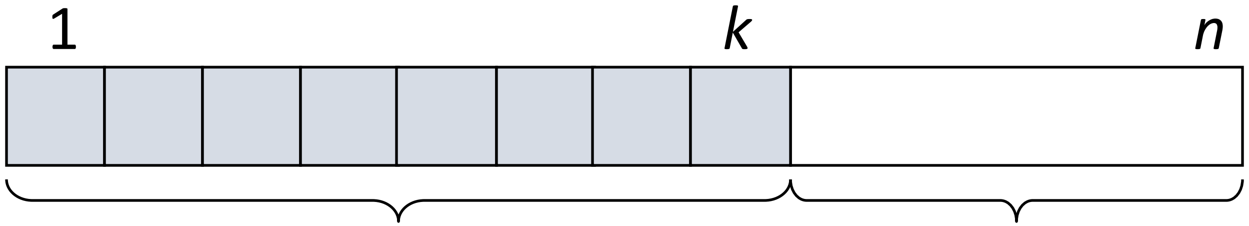
Algorithm:

```
perm(A,k,n) {

/*
Characters stored in A[1..n].
Output all permutations of A[1..k] with A[k+1..n] appended.
Chars in array A should be in the same order as it was before perm is called.
Assume k >= 1. */

  if (k==1)
    output A[1..n]
  else
    for (i=1 to k) do {
       swap(A[i],A[k]);
       perm(A, k-1, n);
       swap(A[i],A[k]); }}
```

Permutation

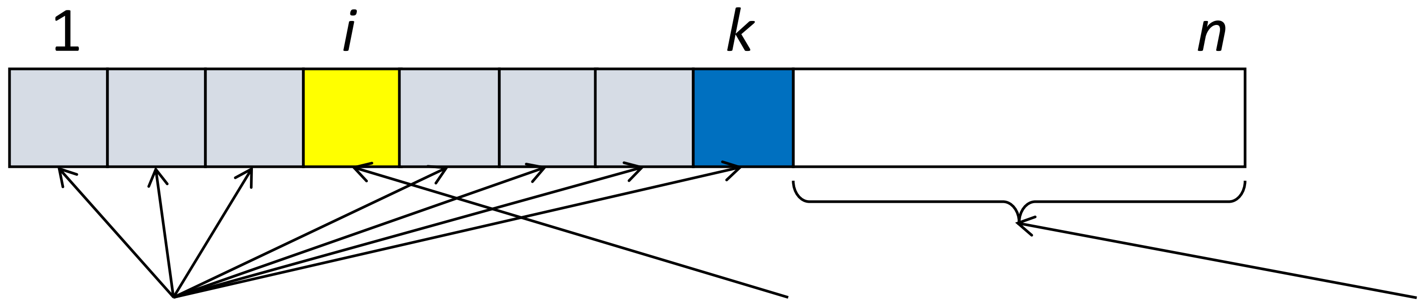


want all perms
of the first k elements.

the rest should
be appended in
each permutation.

Permutation

To generate the *i*-th permutation group, …



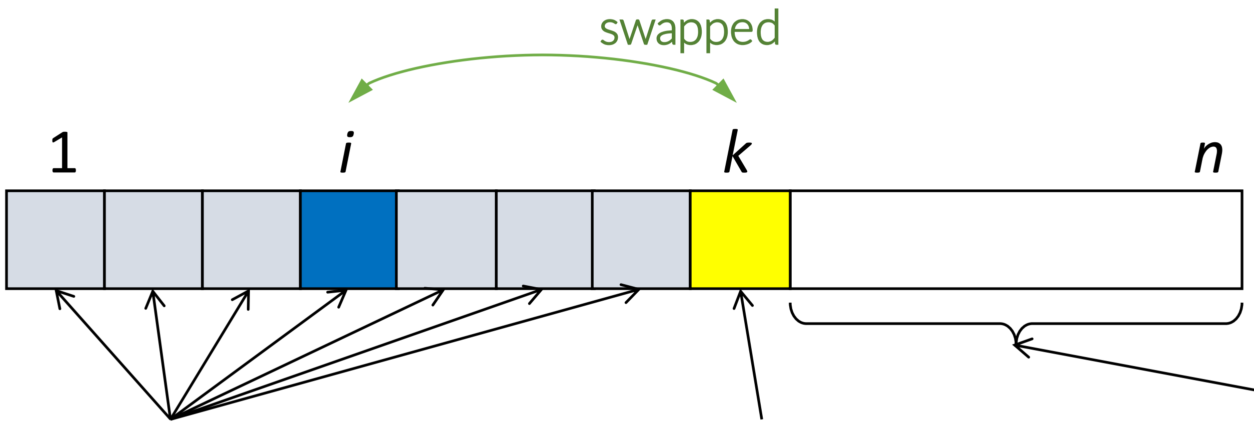
determine all the permutations that consist of these elements

with the ***i***-th (yellow) element appended at the end of each permutation

the rest should be appended at the very end.

Permutation

which is equivalent to …



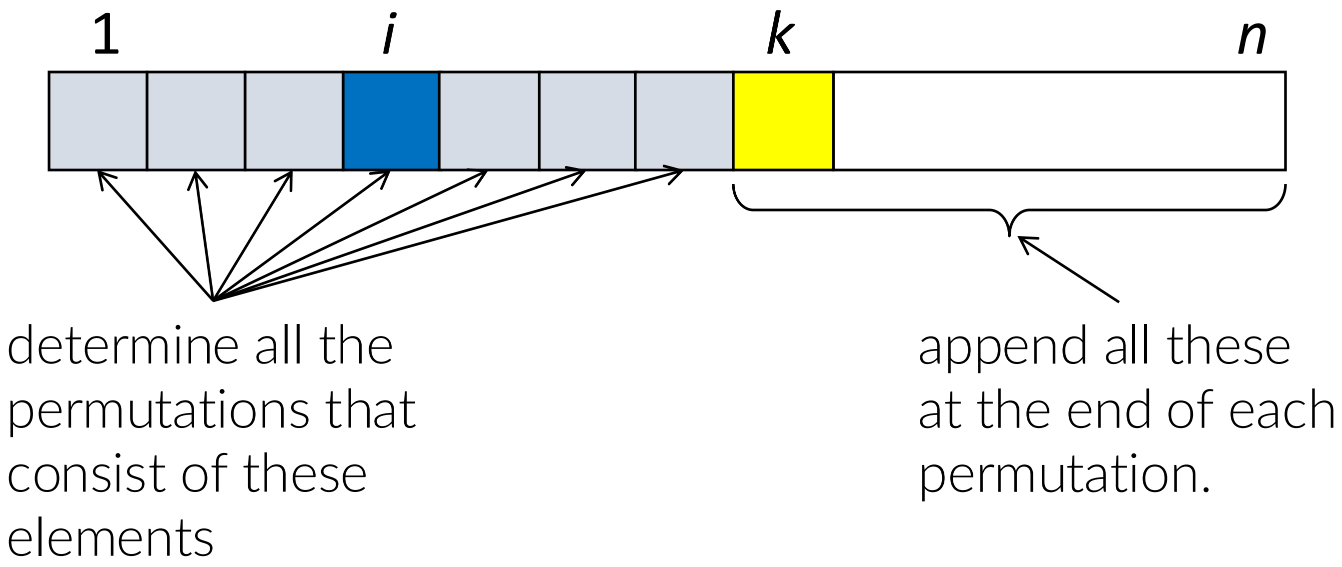
determine all the permutations that consist of these elements

with the yellow element appended at the end of each permutation

the rest should be appended at the very end.

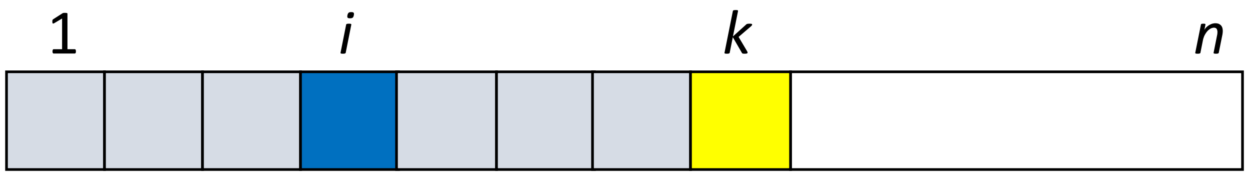
Permutation

which is equivalent to …



Permutation

which is equivalent to calling perm(A, k−1, n)



Permutation

To output all permutations, we simply call:

perm(A, n, n)

Recurrence equation, recursive function, and M.I.

To analyze an algorithm, we perform 2 tasks:

- Prove its *correctness*
- Estimate its *running time*.

Use M.I.

Formulate it as a recursive function

Recurrence equation, recursive function, and M.I.

E.g., we can

- prove the correctness of perm(A,k,n) using M.I.
- show that perm(A,k,n) executes the swap statement

$$f(k) = \begin{cases} 0, & k = 1 \\ 2[k + k(k-1) + \ldots + k(k-1)\ldots 2], & k > 1 \end{cases}$$

  times. Hence, for $k > 1$,

$$2(k!) \leq f(k) \leq 2e(k!)$$

e is the base of the natural log = 2.71828….

- Q: How many times is "output" executed?

Complication

Note that there could be more than 1 variable/parameter in a recurrence equation / recursive function.

Example: Ackermann's function

$$
\begin{aligned}
A(0,n) &= n+1 && \text{for } n \geq 0, \\
A(m,0) &= A(m-1,1) && \text{for } m > 0, \\
A(m,n) &= A(m-1,A(m,n-1)) && \text{for } m,n > 0.
\end{aligned}
$$

Exercises

Based on the recurrence equation of the triangular numbers $\mathtt{T}(n)$, give a closed-form formula of $\mathtt{T}(n)$.

Let $\mathtt{L}(n)$ be the max. number of regions formed by n lines in a plane.

- Give a recurrence equation of $\mathtt{L}(n)$.
- What is the relation between $\mathtt{L}(n)$ and $\mathtt{T}(n)$?

Exercises

You are given the following partial sequence of numbers:

n	1	2	3	4	5	6	7	8	…
***f*(*n*)**	5	12	25	44	69	100	137	180	…

Guess a closed-form formula of $f(n)$.
Verify your formula against the following numbers in the sequence. Did you guess it right?

n	9	10	11	12	13	14	15	16	…
***f*(*n*)**	229	284	345	412	485	564	649	740	…