CPE231RC — Algorithms — Lecture3

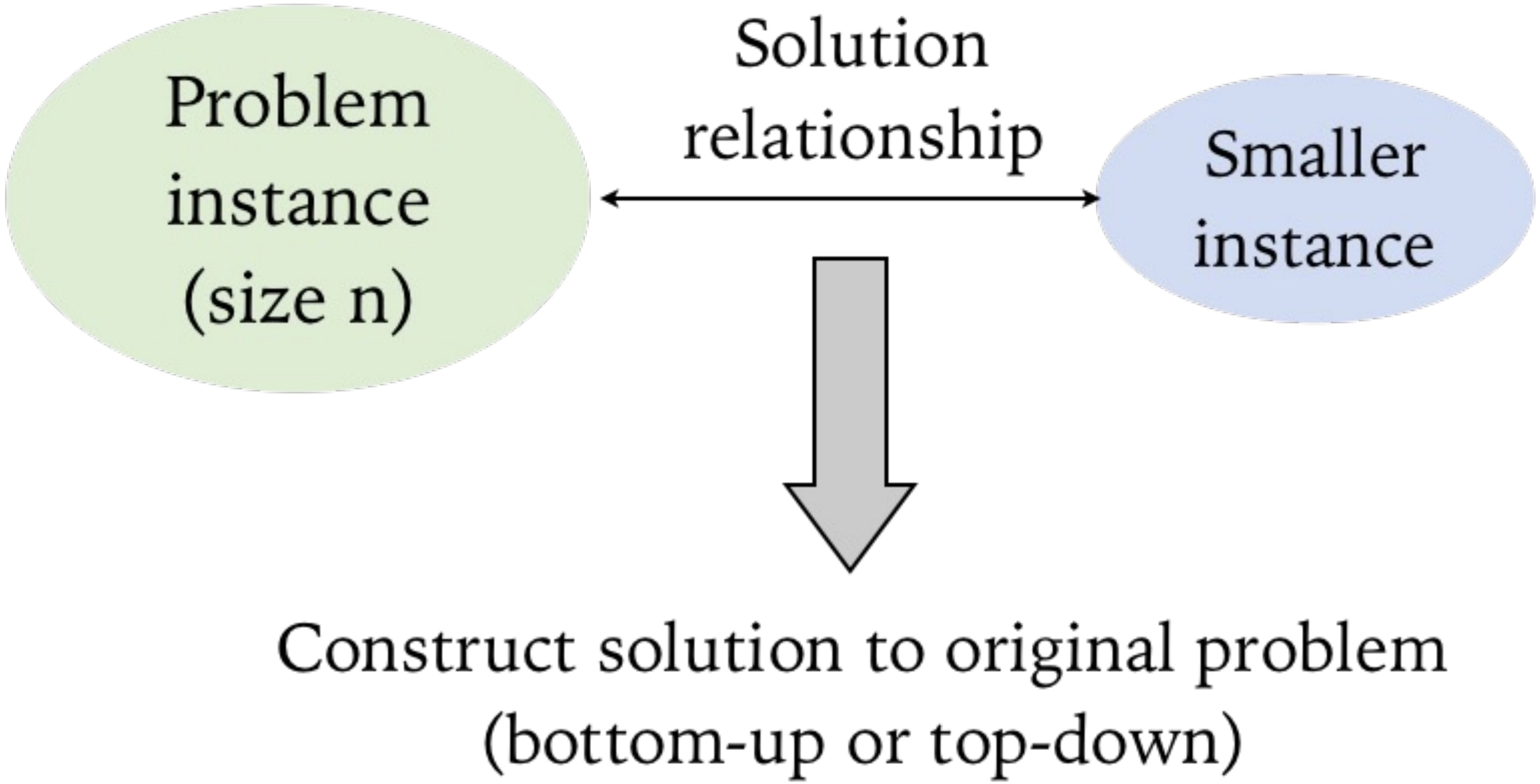
Decrease and Conquer

Dr. Prapong Prechaprapranwong

Computer Engineering

King Mongkut's University of Technology Thonburi

Concept


Problem instance (size n)
Solution relationship
Smaller instance
Construct solution to original problem (bottom-up or top-down)

Variations of Decrease and Conquer

There are 3 major variations of decrease and conquer

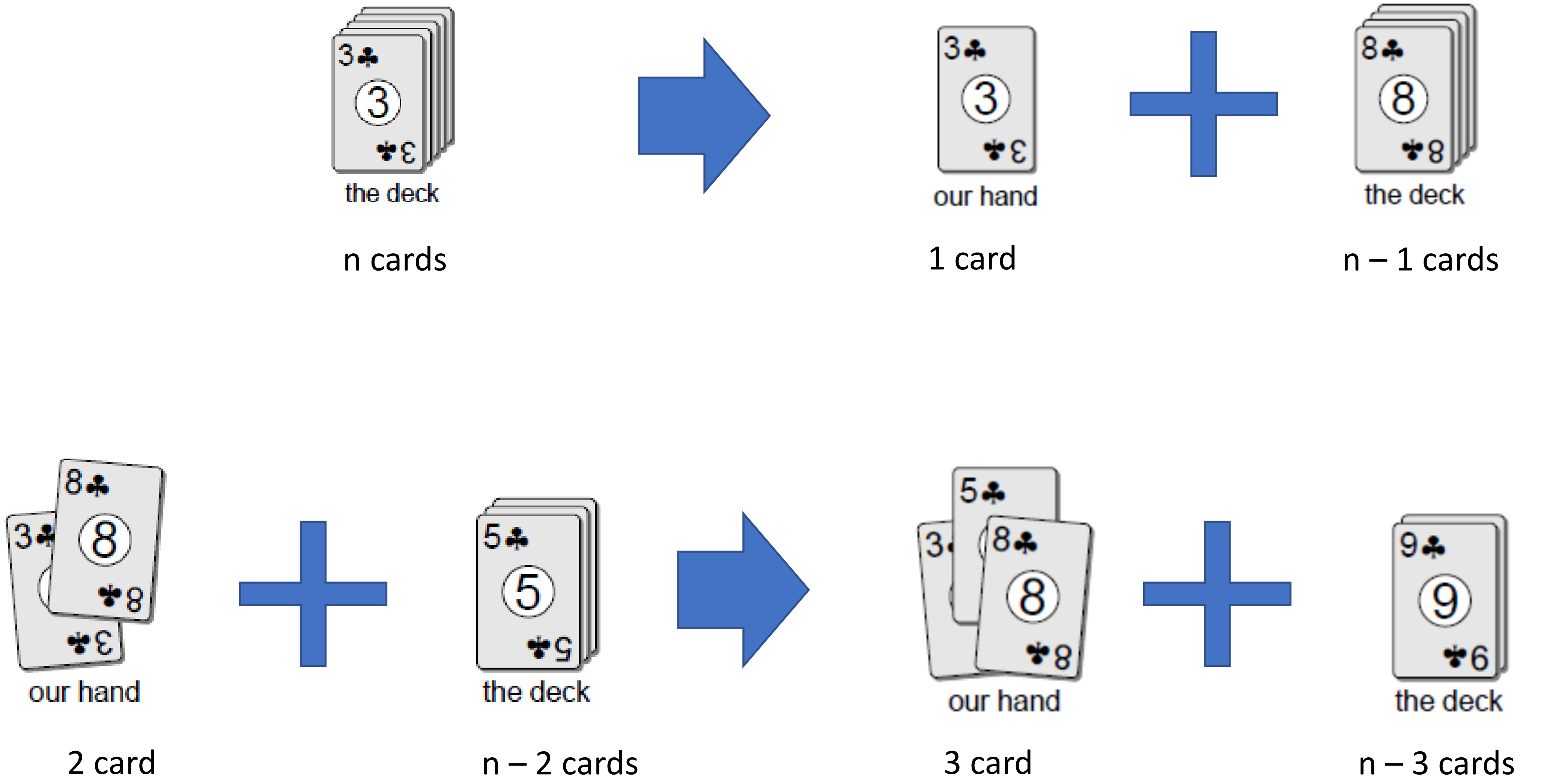
1. Decrease by a <u>constant</u>
2. Decrease by a <u>constant factor</u>
3. <u>Variable size</u> decrease

1.Decrease by a Constant :

- The size of an instance is reduced by the same constant on each iteration of the algorithm
- Typically, this constant is equal to one
- Other constant size reductions can happen occasionally

  - Insertion sort
  - Graph search algorithms: DFS, BFS
  - Topological sorting
  - Algorithms for generating permutations, subsets

Insertion Sort



Algorithm InsertionSort

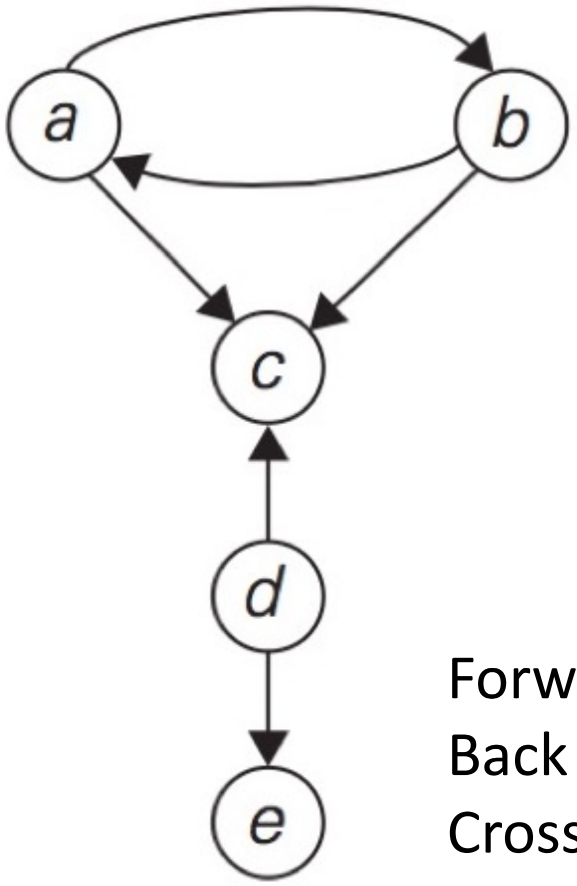
```
1: Input: An array $a_1, a_2, \ldots, a_n$
2: Output: An array $a_1, a_2, \ldots, a_n$ sorted in ascending order
3:
4: for $i = 2$ to $n$ do
5:     $v \leftarrow a_i$                          ▷ Current unsorted item
6:     $j \leftarrow i - 1$                        ▷ Starting index in the sorted portion
7:     while $j \geq 1$ and $a_j > v$ do           ▷ Find place to insert $v$
8:         $a_{j+1} \leftarrow a_j$
9:         $j \leftarrow j - 1$
10:    $a_{j+1} \leftarrow v$
```

$$\begin{aligned} C_{worst}(n) &= \sum_{i=2}^{n} \sum_{j=1}^{i-1} 2 \\ &= 2\sum_{i=2}^{n}(i-1) = 2\sum_{i=1}^{n-1} i \\ &= n(n-1) = \Theta(n^2) \end{aligned}$$

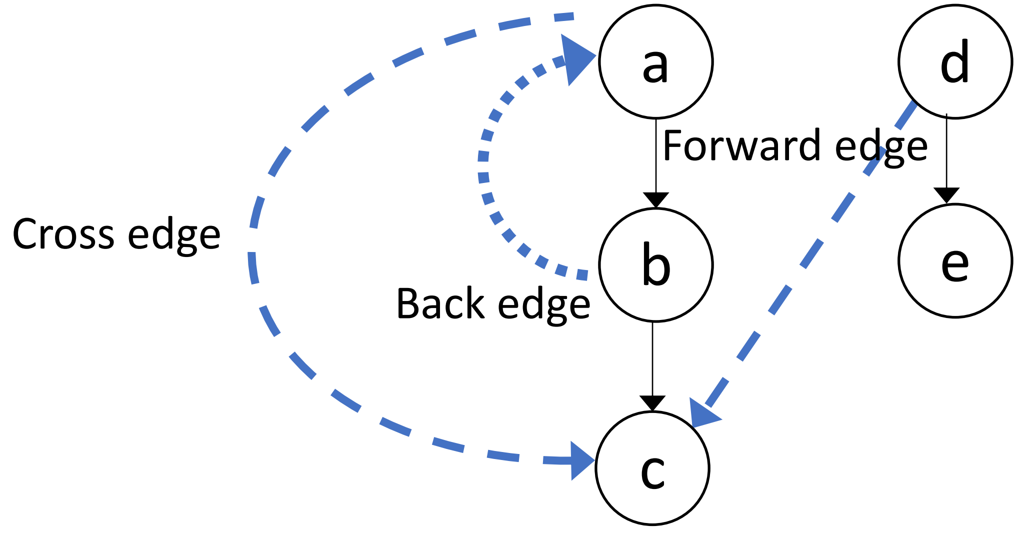
Topological sorting

(Top sort)

find an ordering of directed graph's vertices



Depth First Search(DFS) Tree



Forward edge = Edge connecting from ancestor to descendant
Back edge = Edge connecting back to one of its predecessor
Cross edge = Edge connecting between siblings

Directed Acyclic Graph(DAG) : Not every graph can have a topological ordering. A graph which contains a cycle cannot have a valid ordering → graph with directed edges and no cycles

Algorithm DFSTraversal

```
Input: Graph $G = \langle V, E \rangle$
Output: Graph $G$ with its vertices marked with consecutive integers in
the order they are first visited.

Mark all vertices with 0 (being unvisited)
$count \leftarrow 0$
**for** each vertex $v$ in $V$ **do**
    **if** $v$ is marked with 0 **then**
        $dfs(v)$

**procedure** DFS($v$)
    $count \leftarrow count{+}1$
    Mark $v$ with $count$
    **for** each vertex $w$ in $V$ adjacent to $v$ **do**
        **if** $w$ is marked with 0 **then**
            $dfs(w)$
```

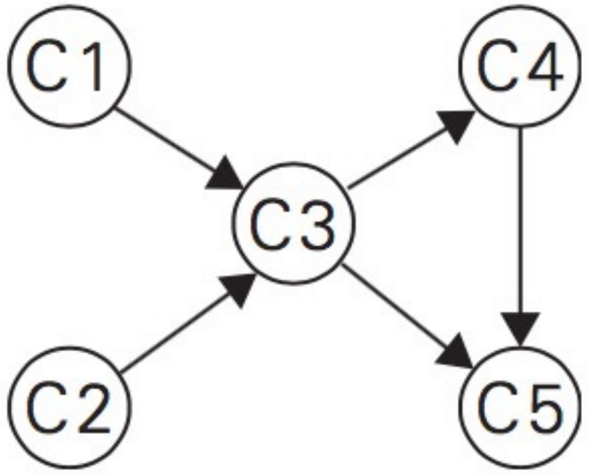
Application of Top Sort

- School class prerequisites
- Program dependencies
- Event scheduling
- Assembly instruction
- etc…

Examples: School Class Prerequisites
Five courses {C1, C2, C3, C4, C5}
C3 requires C1 and C2 as prerequisites
C4 requires C3 as prerequisite
C5 requires C3 and C4 as prerequisites
Student can only take one course per semester.

What is the order to take the courses?

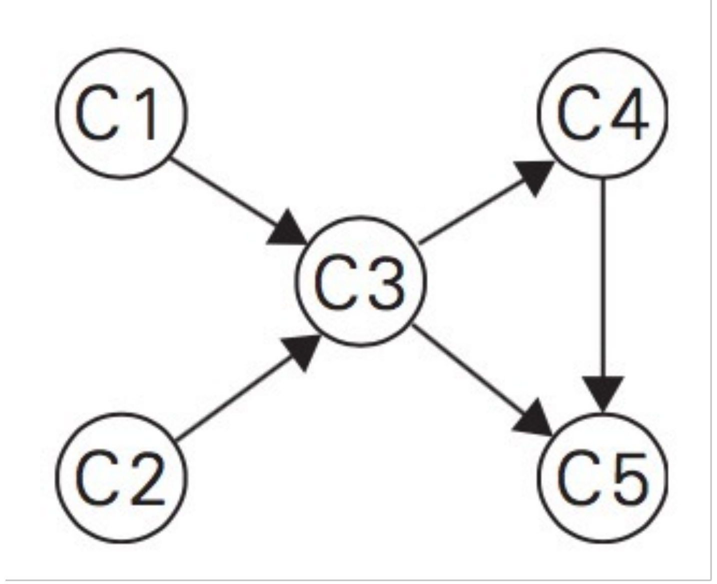
C1→C3→C4→C5→C2
C2→C3→C4→C5→C1
C1→C3→C5→C2→C4



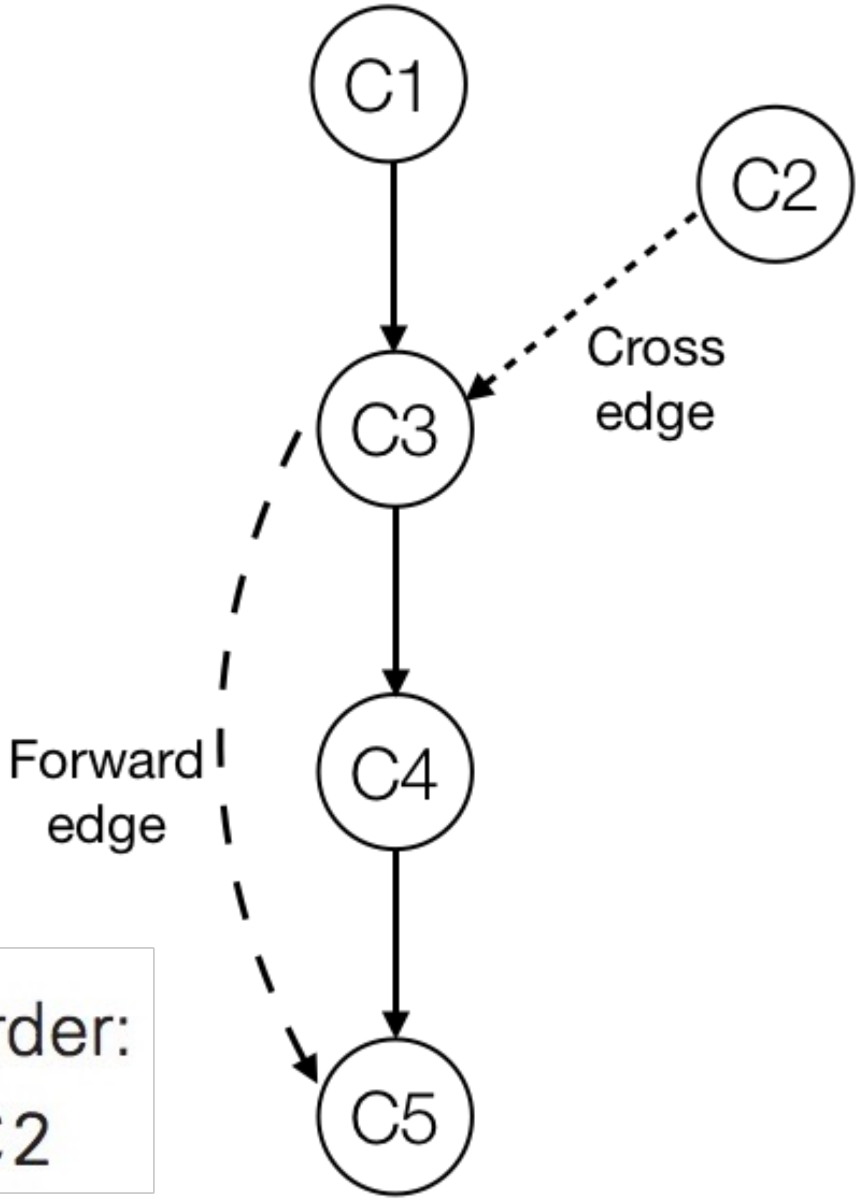
Topological sorting

Can we list all the vertices such that, for every edge, the vertex where the edge starts is listed before the vertex where the edge ends?

First solution: Check if DFS traversal yields DAG (No back edge).



The popping-off order:
C5, C4, C3, C1, C2

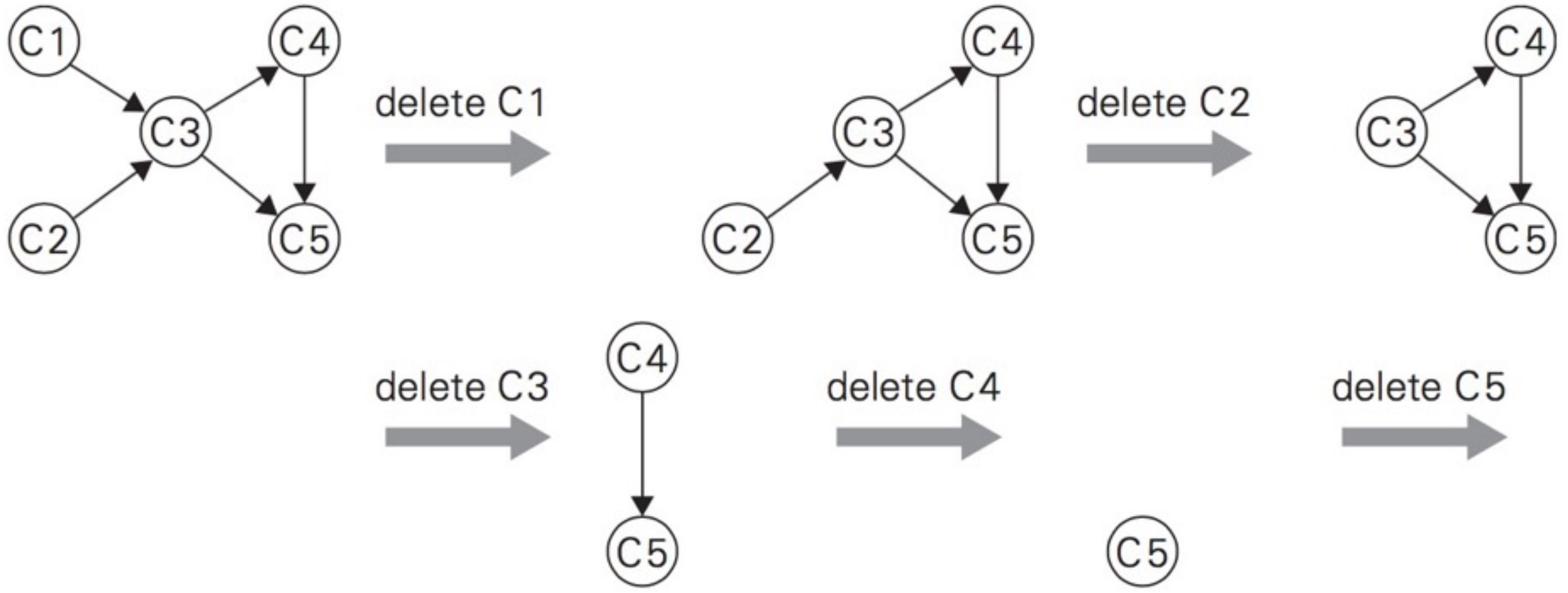


DFS traversal
Directed Acyclic Graph (DAG)

Source Removal Algorithm

The second solution:

- Decrease(by one)-and-Conquer approach
- Repeatedly delete a vertex with no incoming edge and its outgoing edges.
- The order in which the vertices are deleted yields a solution to the problem.
- May yield different solution from DFS.



The solution obtained is C1, C2, C3, C4, C5

Practice 1

Consider the following course prerequisites:

- Course a is a prerequisite of courses b and c.
- Course b is a prerequisite of courses g and e.
- Course c is a prerequisite of course f.
- Course d is a prerequisite of courses a, b, c, g, and f.
- Course g is a prerequisite of courses e and f.

Apply DFS and Source removal algorithms to determine the course sequence that must be registered. Show each step of your solution

Algorithms for generating combinatorial objects

Mostly used for brute-force and exhaustive search algorithms.

- All sequences of cities in TSP
- All combinations of objects in knapsack problem.

Three types of combinatorial objects

- Permutations
- Combinations
- Subsets

Permutation

Examples:

How many ways to arrange letters a, b, c ?

How many ways to arrange six books on a shelf ?

Number of orders (sequences) of a selection of n distinct objects.

Permutation is called "Sampling without Replacement"

Number of orders (sequences) of a selection of n distinct objects.

Number of permutations of n distinct objects = n!

Combination

How many ways to choose two letters from {a,b,c,d} (order doesn't matter)?
# different selections (groups) of k objects from a set of n distinct objects
Called "Unordered sampling without replacement"

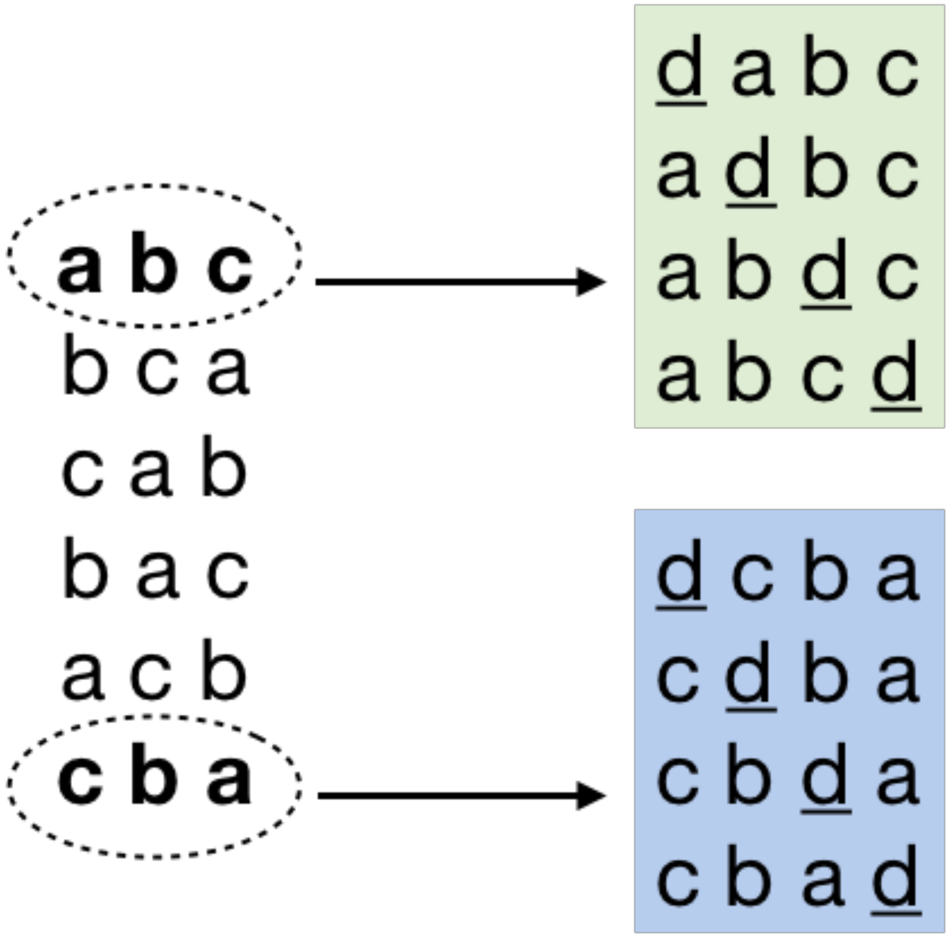
Form a group of 8 committees from 20 people. How many different groups can be formed ?

Number of combinations of $\mathbf{n}$ objects taken $\mathbf{k}$ at the time

$$c_{n,k} = \frac{P_{n,k}}{k!} = \frac{n!}{(n-k)!\,k!}$$

Generating Permutations

Consider a set of {a, b, c, d}



Generate all permutations of {a, b, c}

For each pattern, insert d at different positions to obtain the required patterns.

4! = 4*3*2*1 = 24 patterns.

Generating Permutation using Decrease-by-One

- Suppose we have permutations {1,2}, {2,1}.
- How do we generate permutations of {1, 2, 3} ?
- How do we generate permutations of {1, 2, 3, 4} ?

- Assume a set of integers {1, 2, 3, …, n}
- Can be interpreted as indices of an n-element set {a_1, a_2, …, a_n)
- Totally n! permutations

- What is the relationship between permutations of {1,2,3,..,n-1} and {1,2,3,..., n-1,n} ?
- Exploit the relationship bottom-up to generate all patterns.

Start	1			
Insert 2 into 1 right to left	1,**2**	**2**,1		
Insert 3 into 1,2	1,2,**3**	1,**3**,2	**3**,1,2	
Insert 3 into 2,1	2,1,**3**	2,**3**,1	**3**,2,1	
Insert 4 into 1,2,3	1,2,3,**4**	1,2,**4**,3	1,**4**,2,3	**4**,1,2,3
Insert 4 into 1,3,2	1,3,2,**4**	1,3,**4**,2	1,**4**,3,2	**4**,1,3,2
Insert 4 into 3,1,2	3,1,2,**4**	3,1,**4**,2	3,**4**,1,2	**4**,3,1,2
Insert 4 into 3,2,1	2,1,3,**4**	2,1,**4**,3	2,**4**,1,3	**4**,2,1,3
Insert 4 into 2,3,1	2,3,1,**4**	2,3,**4**,1	2,**4**,3,1	**4**,2,3,1
Insert 4 into 2,1,3	3,2,1,**4**	3,2,**4**,1	3,**4**,2,1	**4**,3,2,1

Johnson Trotter Algorithm

- Possible to generate n-element permutations without explicitly generating permutations for smaller values of n.
- Consider one of the 4-element permutations:

$$\overleftarrow{3}, \overrightarrow{2}, \overleftarrow{4}, \overrightarrow{1}$$

- An element said to be **"mobile"** if its arrow points to a smaller number adjacent to it.
- 4 is mobile while 3, 2, 1 are not.

```
Algorithm JohnsonTrotter
1: Input: A positive integer $n$
2: Output: All permutations of $\{1, 2, 3, \dots, n\}$
3:
4: Initialize the first permutation with $\overleftarrow{1}, \overleftarrow{2}, \dots, \overleftarrow{n}$
5: while The last permutation has a mobile element do
6:     Find its largest mobile element $k$
7:     Swap $k$ with element that the arrow of $k$ points to
8:     Reverse the direction of all elements larger than $k$
9:     Add the new permutation to the list
```

Practice II

Use Johnson Trotter’s algorithm to generate permutations of {1,2,3,4}

Generating All Subsets (Power Set)

Want to find all subsets of A = {a_1, a_2, …, a_n}, e.g., knapsack problem.

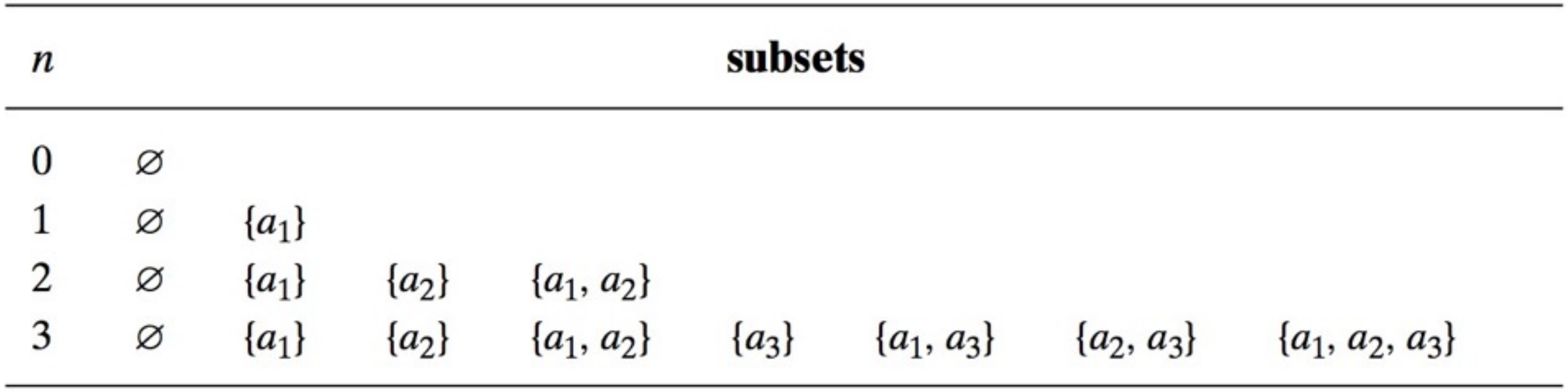
Subset of A = Set of whose all its members are also elements of A.

What are subsets of A = {x, y, z} ?

Generating subset using Decrease-by-one approach

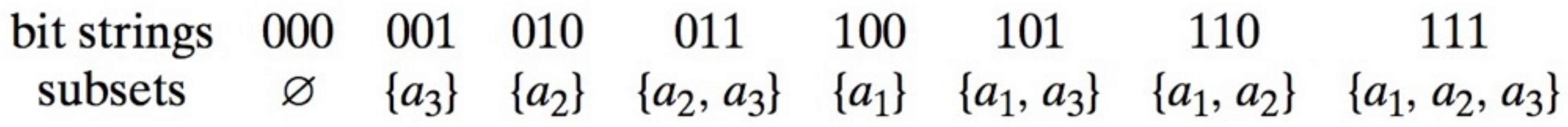
All subsets of A = {those without a_n} ∪ {those with a_n}

- The former group is all subsets of A = {a_1, a_2, …, a_{n-1}}
- The latter group is the former added by {a_n}



n	subsets							
0	∅							
1	∅	$\{a_1\}$						
2	∅	$\{a_1\}$	$\{a_2\}$	$\{a_1, a_2\}$				
3	∅	$\{a_1\}$	$\{a_2\}$	$\{a_1, a_2\}$	$\{a_3\}$	$\{a_1, a_3\}$	$\{a_2, a_3\}$	$\{a_1, a_2, a_3\}$

Easier way is to use an ***n-bit*** bit string to represent presence or absence of individual elements



bit strings	000	001	010	011	100	101	110	111
subsets	$\varnothing$	$\{a_3\}$	$\{a_2\}$	$\{a_2, a_3\}$	$\{a_1\}$	$\{a_1, a_3\}$	$\{a_1, a_2\}$	$\{a_1, a_2, a_3\}$

2. Decrease by a Constant factor:

- Reducing a problem instance by the same constant factor on each iteration of the algorithm
- In most applications, the constant factor = 2
- A reduction by a factor other than two is especially rare

Decrease by a constant factor algorithms are very efficient especially when the factor > 2 as in the fake-coin problem.

Example problems :

- Binary search
- Fake-coin problems
- Russian peasant multiplication

Binary Search

Decrease-by-a-constant factor search algorithm operated on a sorted list.

$$\underbrace{A[0]\dots A[m-1]}_{\substack{\text{search here if}\\ K<A[m]}} \quad \overset{\substack{K\\ \updownarrow}}{A[m]} \quad \underbrace{A[m+1]\dots A[n-1]}_{\substack{\text{search here if}\\ K>A[m]}}.$$

Search for K = 70

index	0	1	2	3	4	5	6	7	8	9	10	11	12
value	3	14	27	31	39	42	55	70	74	81	85	93	98
iteration 1	*l*						*m*						*r*
iteration 2								*l*		*m*			*r*
iteration 3								*l,m*	*r*				

Algorithm BinarySearch

```
Input: A sequence of numbers $A[0 \ldots n]$; A search key $K$
Output: Return the index of element if $K$ is in the sequence, and -1 otherwise.

$l \leftarrow 0$
$r \leftarrow n - 1$
while $l \leq r$ do
    $m \leftarrow \lfloor (l + r)/2 \rfloor$
    if $K == A[m]$ then
        return $m$
    else if $K < A[m]$ then
        $r \leftarrow m - 1$
    else
        $l \leftarrow m + 1$
return -1
```

Worst case when the search key is not in the list.
After one comparison, the array size to search is reduced by half. So,

$$C_{worst}(n) = C_{worst}(\lfloor n/2 \rfloor) + 1 \quad \text{for } n > 1, \quad C_{worst}(1) = 1$$

Assume that n = 2k, solving the above recurrence equation with backward substitution results in

$$C_{worst}(2^k) = k + 1 = \log_2 n + 1.$$

Fake coin problem

A set of n identical-looking coins, one of which is fake.
How to find it?

- Assume the fake one is lighter.
- Fake coin has unknown weight for the harder version

Number of weightings W(n) needed by the algorithm is

$$W(n) = W(\lfloor n/2 \rfloor) + 1, \quad \text{for } n > 1, \text{ and } W(1) = 0$$

What is the efficiency of this algorithm? $O(\log_2(n))$

Can we do it by dividing into three piles instead of two ?

- What if Pile 1 = Pile 2? $\rightarrow$ Pile 3 has a fake coin
- What if Pile 1 > Pile 2? $\rightarrow$ Pile 2 has a fake coin $O(\log_3(n))$ for 3 piles
- What if Pile 1 < Pile 2? $\rightarrow$ Pile 1 has a fake coin

Find more: What is the Fake coin problem in real world?

Assignment 1:

What is the Russian peasant multiplication?

3.Variable-Size-Decrease :

- The size-reduction pattern varies from one iteration of an algorithm to another

As Euclid’s algorithm problem that solved the problem of finding **gcd** of two number though the value of the second argument is always smaller on the right-handside than on the left-hand side, it decreases neither by a constant nor by a constant factor

Example problems :

- Euclid’s algorithm
- Computing median and selection problem
- Interpolation Search

The Selection Problem

Find kth smallest element in a list of n numbers (kth order statistic)

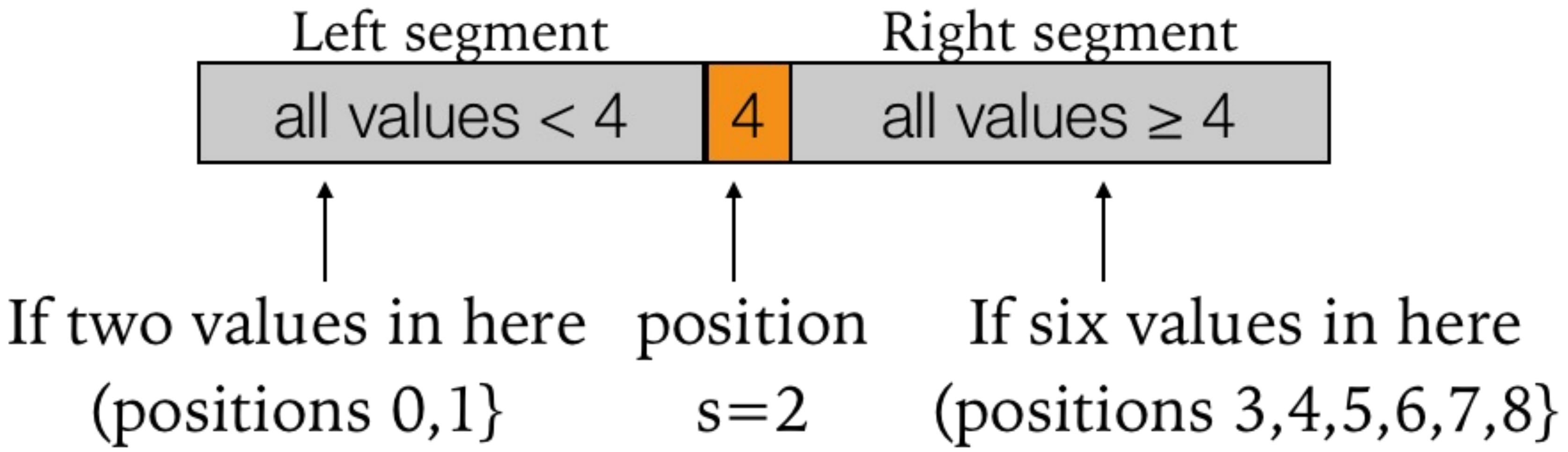
- k = 1: Smallest element
- k = n: Largest element
- k = ceil(n/2): Median (most of interest)

Consider the list {4, 1, 10, 8, 7, 12, 9, 2, 15}

- What is the median ?
- What are the 2nd and 7th order statistics?

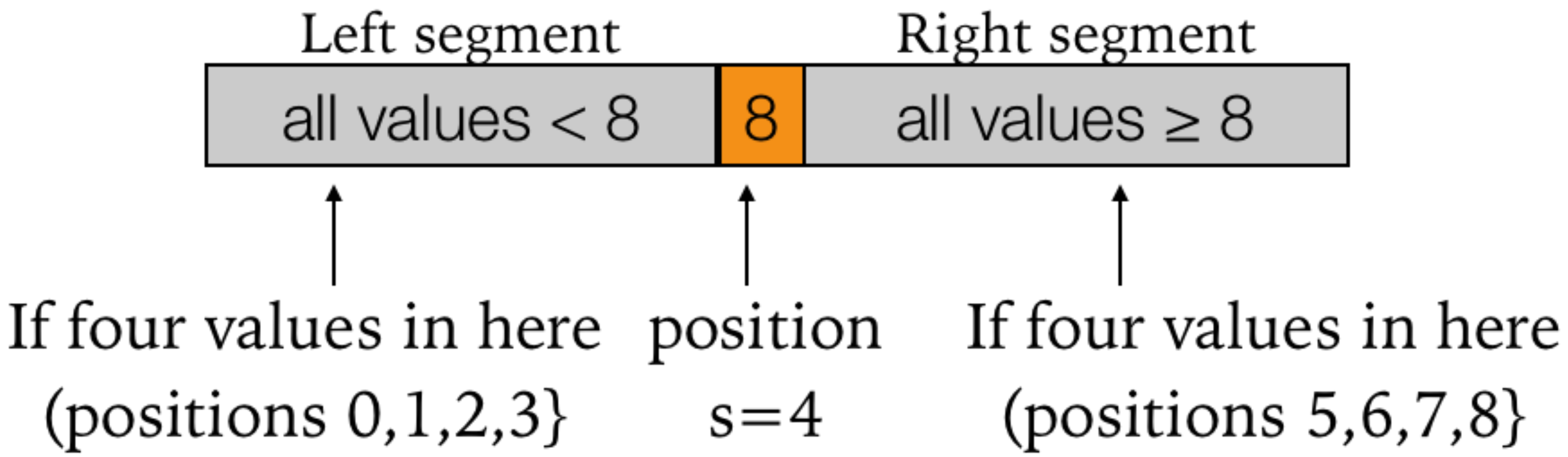
Straightforward approach with sorting but overkill

Suppose we have a list {4, 1, 10, 8, 7, 12, 9, 2, 15} that we want to find the 5th-order statistic. Choose "4" as the pivot element to partition the list into three segments:


Left segment
Right segment
all values < 4
4
all values ≥ 4
If two values in here (positions 0,1}
position s=2
If six values in here (positions 3,4,5,6,7,8}

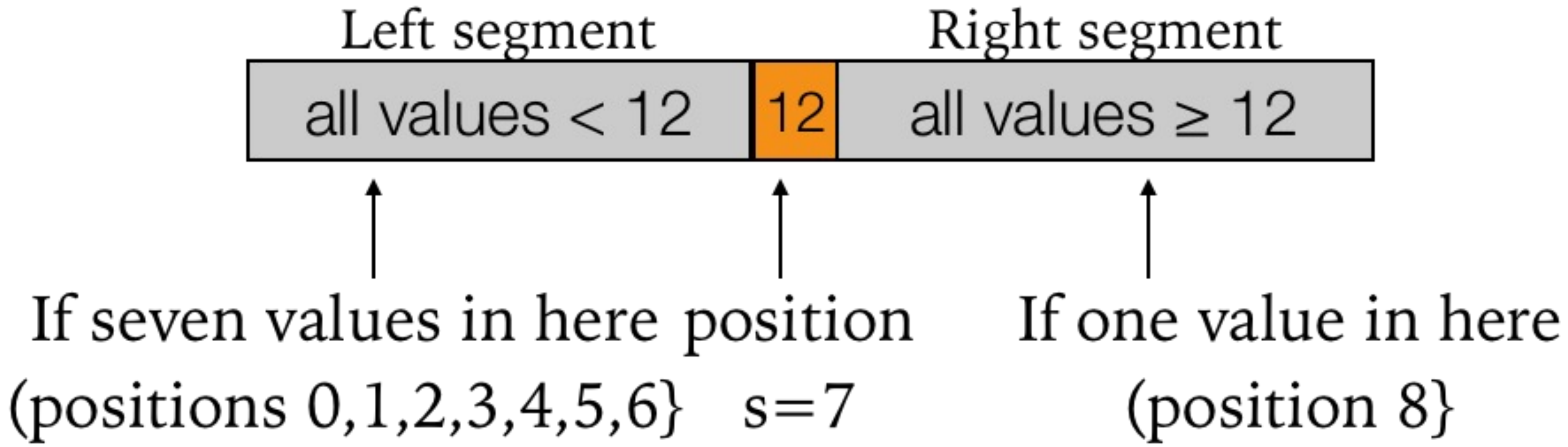
The 5th-order statistic of the original list must be in the ________ segment

Suppose we have a list {8, 1, 10, 4, 7, 12, 9, 2, 15} that we want to find the 5th-order statistic. Choose "8" as the pivot element to partition the list into three segments:


Left segment
Right segment
all values < 8
8
all values ≥ 8
If four values in here (positions 0,1,2,3}
position s=4
If four values in here (positions 5,6,7,8}

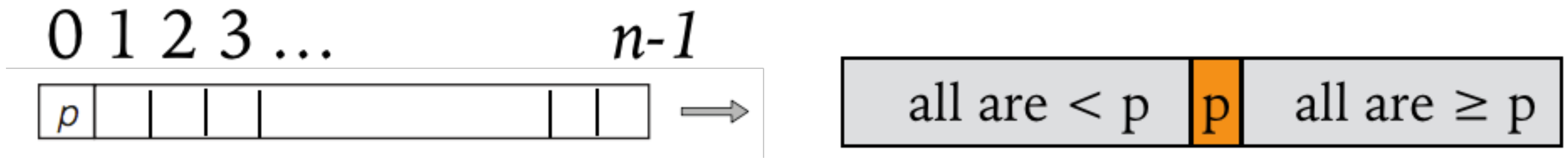
The 5th-order statistic of the original list must be ________

Suppose we have a list {12, 1, 10, 8, 7, 4, 9, 2, 15} that we want to find the 5th-order statistic. Choose "12" as the pivot element to partition the list into three segments:



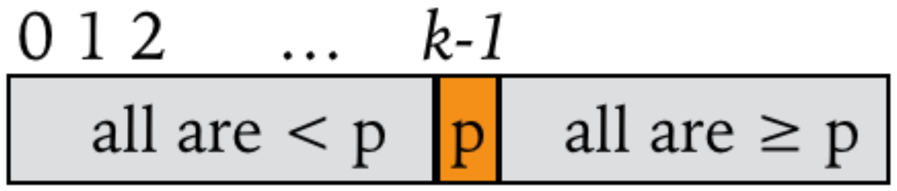
The 5th-order statistic of the original list must be in the _________ segment.

Suppose we can partition an original list in three segments based on a pivot value p shown below:

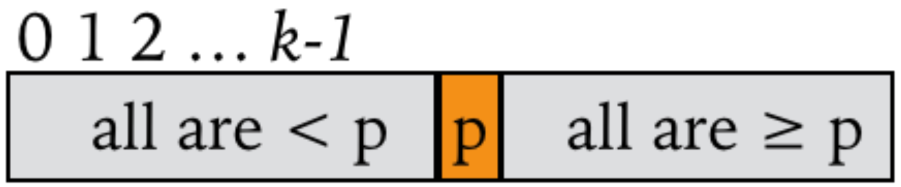


Three scenarios can happen

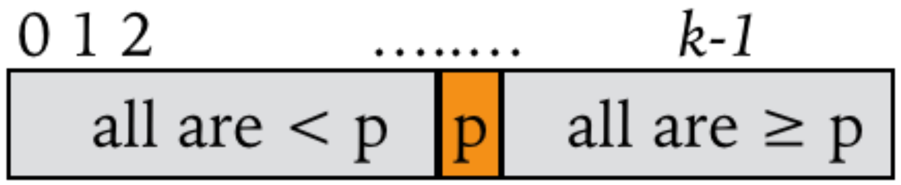
p is at position s = k-1



p is in the position s > k-1

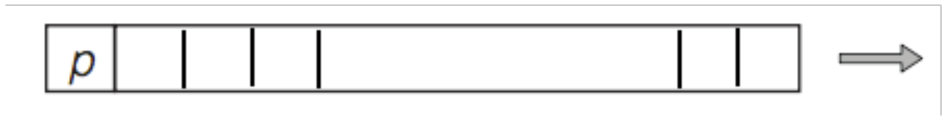


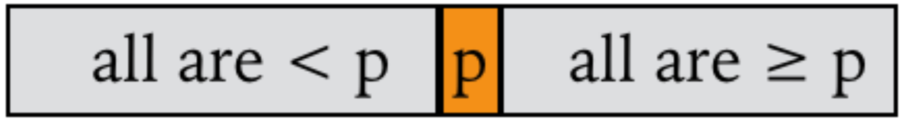
p is in the position s < k-1



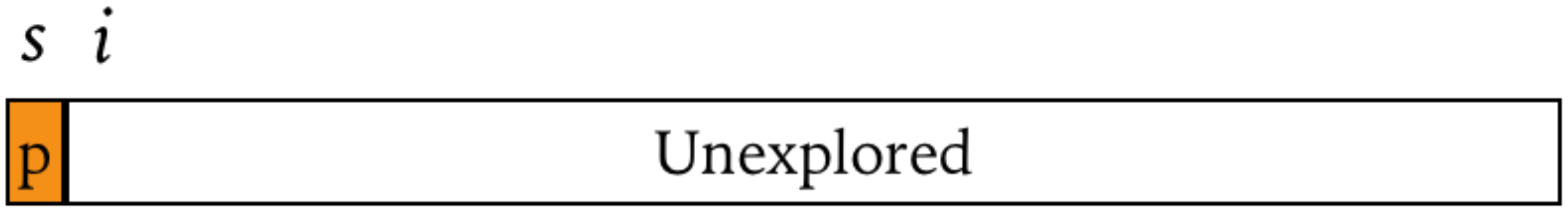
Lomuto Partitioning

Take a pivot value p
Partition the list so that
Left part contains all elements less than p.
Right part contains all elements greater than equal to p.

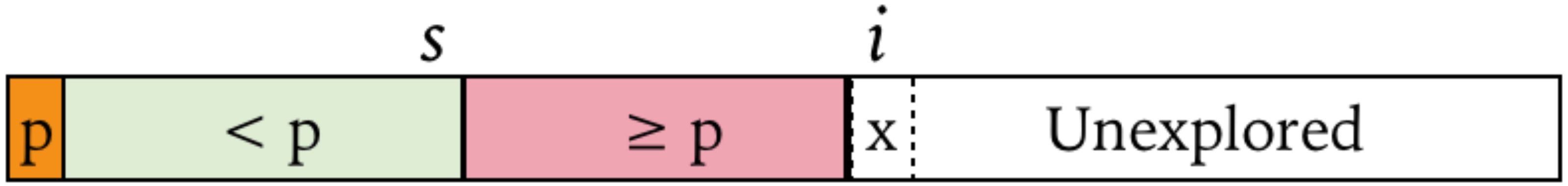




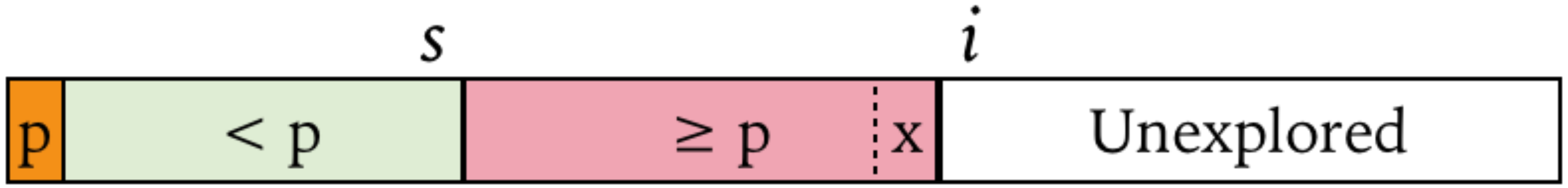
Initialization



List explored from left to right

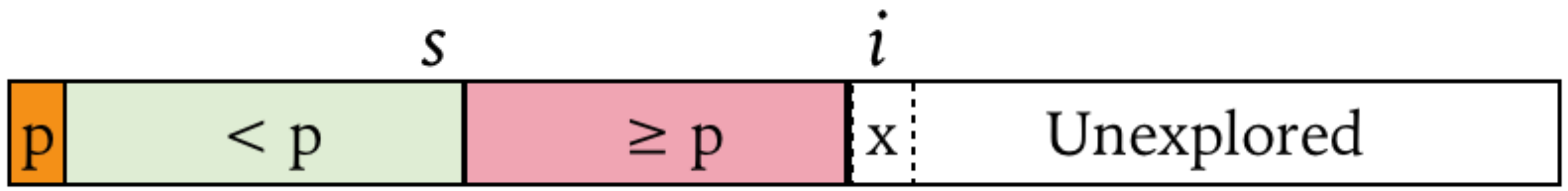


If A[i] ≥ p, just increment i, which will expand the ≥ p segment.

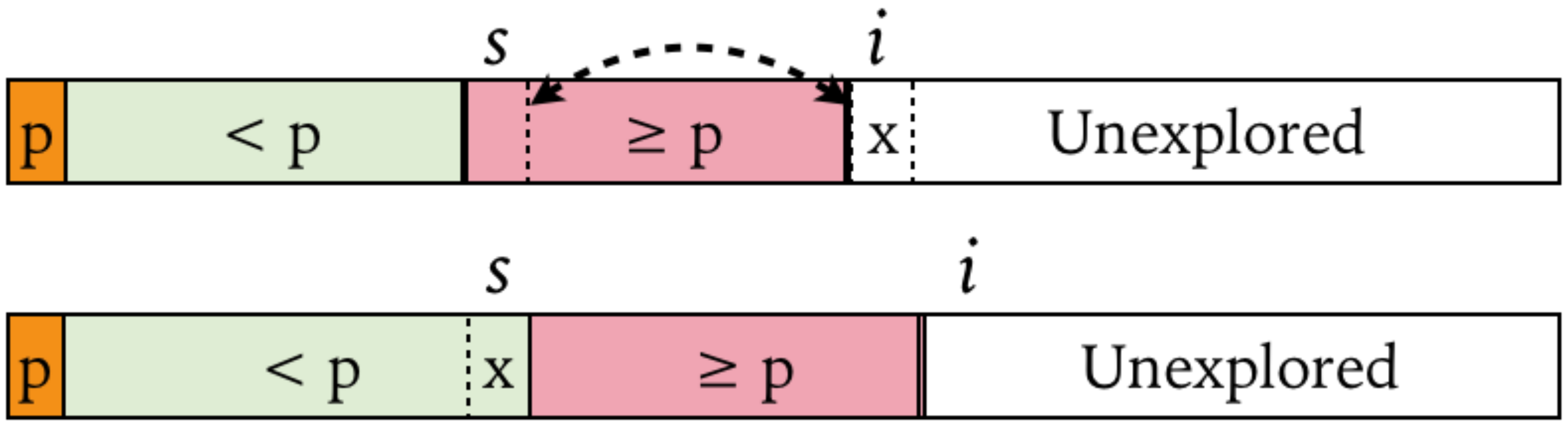


s = last position containing element < p

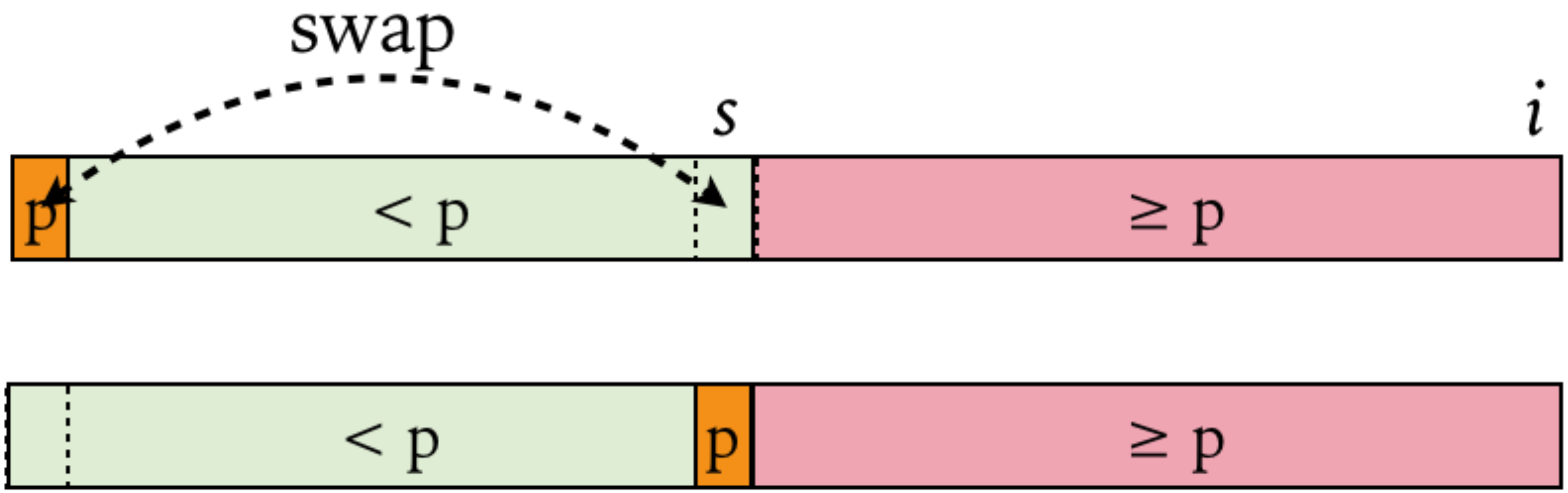
List explored from left to right



If A[i] < p,
increment s and swap A[s] with A[i], which will expand the <p segment.
increment i



When all elements are explored,



Algorithm LomutoPartition($A[l..r]$)

Input: Subarray $A[l..r]$

Output: Partition of $A[l..r]$ and the new pivot position

```
$p \leftarrow A[l]$
$s \leftarrow l$
**for** $i \leftarrow l + 1$ to $r$ **do**
    **if** $A[i] < p$ **then**
        $s \leftarrow s + 1$
        swap($A[s], A[i]$)
swap($A[l], A[s]$)
**return** $s$
```

Algorithm QuickSelect($A[l..r]$,k)

```
Input: Subarray $A[l..r]$ and integer $k$ $(1 \leq k \leq r - l + 1)$
Output: $k$th-smallest element in $A[l..r]$

$s \leftarrow$ LomutoPartition($A[l..r]$)
**if** $s == k - 1$ **then**
    **return** $A[s]$
**else if** $s > k - 1$ **then**
    QuickSelect($A[l\ ..\ s-1], k$)
**else**
    QuickSelect($A[s+1\ ..\ r], k$)
```

.

Find the 5th order statistic

0	1	2	3	4	5	6	7	8
s	*i*							
4	1	10	8	7	12	9	2	15
	s	*i*						
4	1	10	8	7	12	9	2	15
	s						*i*	
4	1	10	8	7	12	9	2	15
		s					*i*	
4	1	2	8	7	12	9	10	15
		s						*i*
4	1	2	8	7	12	9	10	15
2	1	**4**	8	7	12	9	10	15

l = s+1 = 3, r = 8

0	1	2	3	4	5	6	7	8
			s	*i*				
			8	7	12	9	10	15
				s	*i*			
			8	7	12	9	10	15
				s				*i*
			8	7	12	9	10	15
			7	**8**	12	9	10	15

s == 4 == k-1, stop

Efficiency of QuickSelect()

- Partition an n-element array requires n-1 key comparisons.
- **Best case** if the split solves the problem or $C_{best}(n) = n-1 \in \Theta(n)$
- Worst-case if k = n and the array is strictly increasing

  Example: Finding 9th-order statistic in {1,2,4,7,8,9,10,12,15} needs n-1 partitions

$$C_{worst}(n) = (n-1) + (n-2) + \cdots + 1 = (n-1)n/2 \in \Theta(n^2),$$

- Average case about $\log_2(n)$ like binary search.

Assignment 2:

What is the Interpolation Search?

Summary

Decrease and Conquer

a problem-solving technique - breaking down a problem into smaller instances or subproblems in each step , solving these smaller instances, combining their solutions to solve the original problem.

- Decrease by a constant
- Decrease by a constant factor (nomally 2)
- Variable size decrease

Assignment1: What is the Russian peasant multiplication?

Assignment2: What is the Interpolation Search?