Decrease-and-Conquer

Natasha Dejdumrong.

CPE 212 Algorithm Design

Topics

- Decrase-by-a-constant
  - Insertion sort
  - Topological sorting
  - Generating permutations
- Decrease-by-a-constant-factor
  - Binary search
  - Fake-coin problem
- Variable-size decrease
  - Selection problem

Decrease by a constant	Decrease by a constant factor	Variable-size decrease
Insertion sort	Binary search	Euclid's
Topological sorting	Exponentiation	Selection by partition
Generating permutations	Multiplication	

Insertion Sort

89 | **45** 68 90 29 34 17

- Take the leftmost item as already sorted
- Successively put the leftmost item of the remaining portion to the sorted portion.
- So, size of an instance decreased by a constant in each iteration.

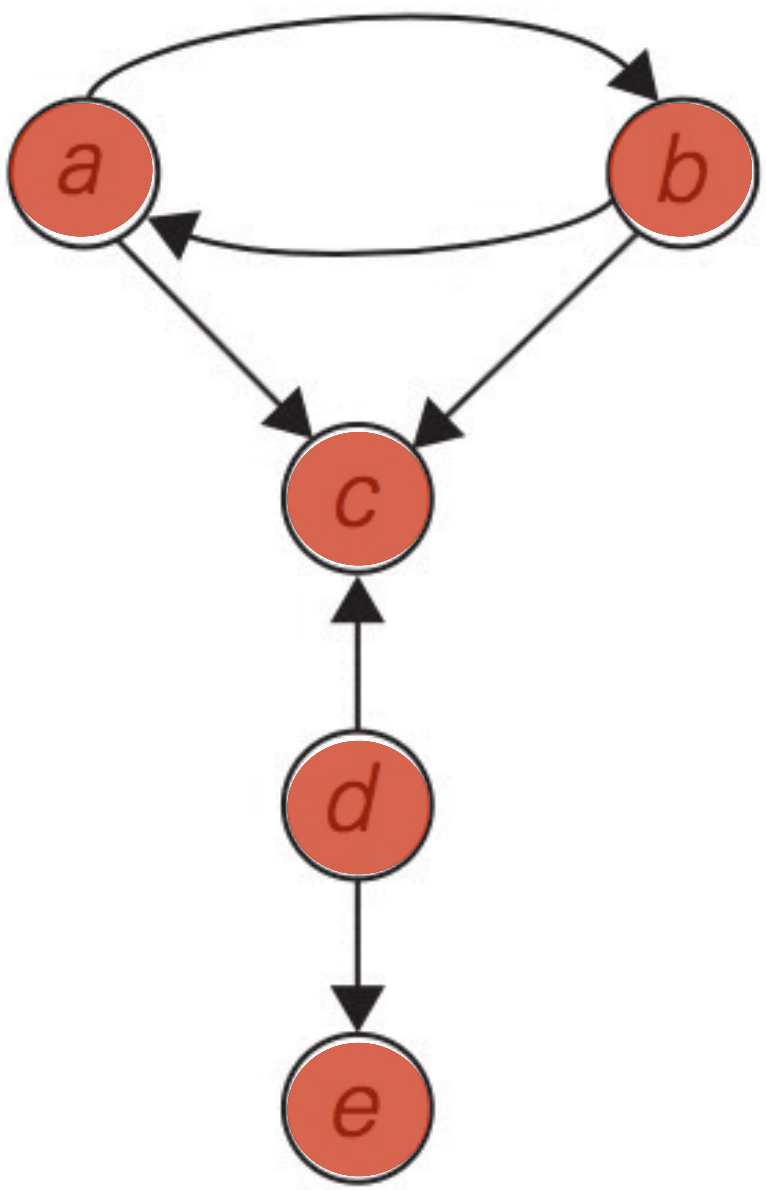
Algorithm InsertionSort

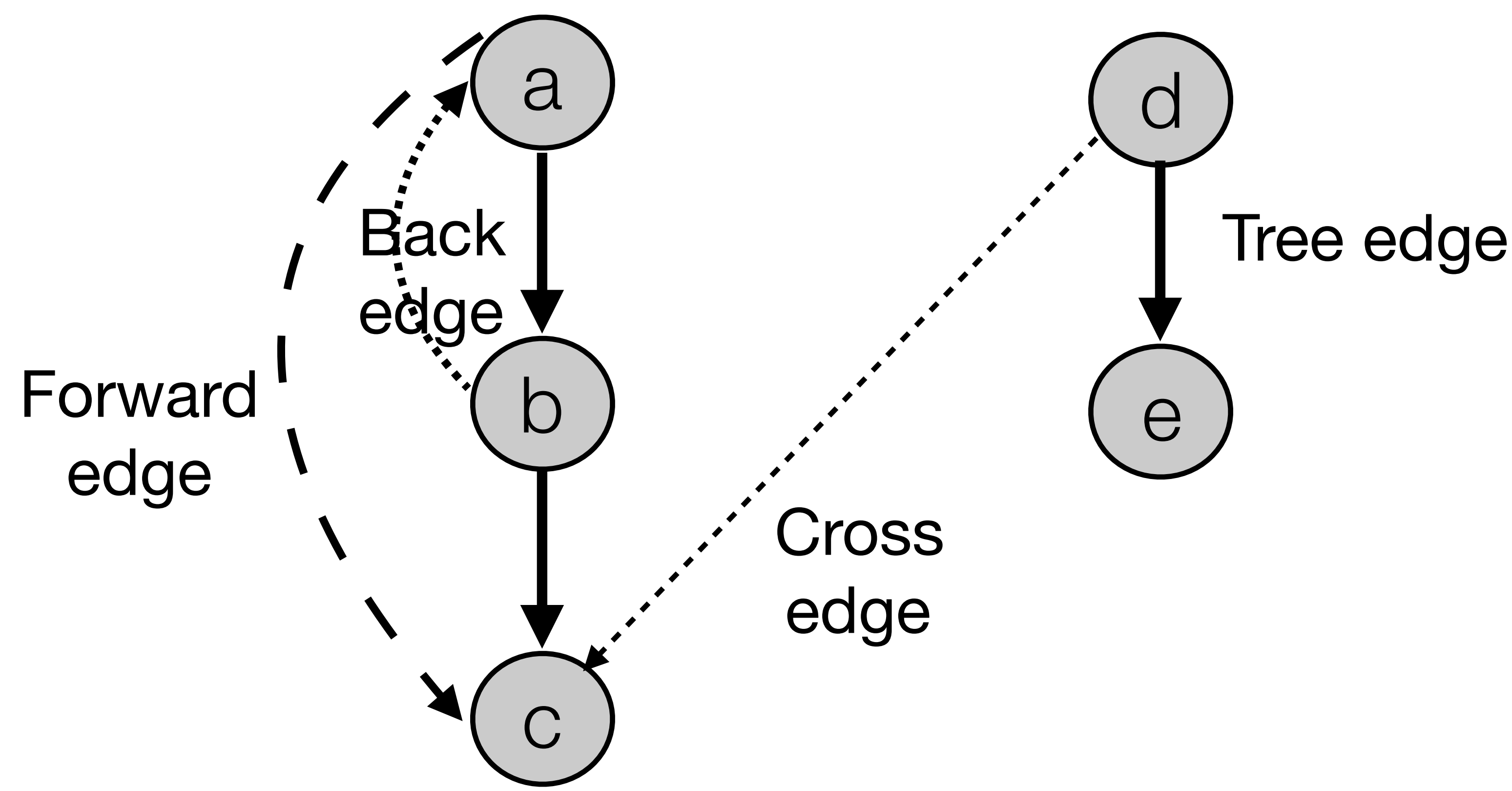
Input: An array $A[0 \dots n-1]$
Output: An array $A[0 \dots n-1]$ sorted in ascending order

for $i = 1$ to $n-1$ **do**
    $v \leftarrow A[i]$
    $j \leftarrow i - 1$
    while $j \geq 0$ and $A[j] > v$ **do**
        $A[j+1] \leftarrow A[j]$
        $j \leftarrow j - 1$
    $A[j+1] \leftarrow v$

$$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \in \Theta(n^2).$$

Directed Graph (Digraph)

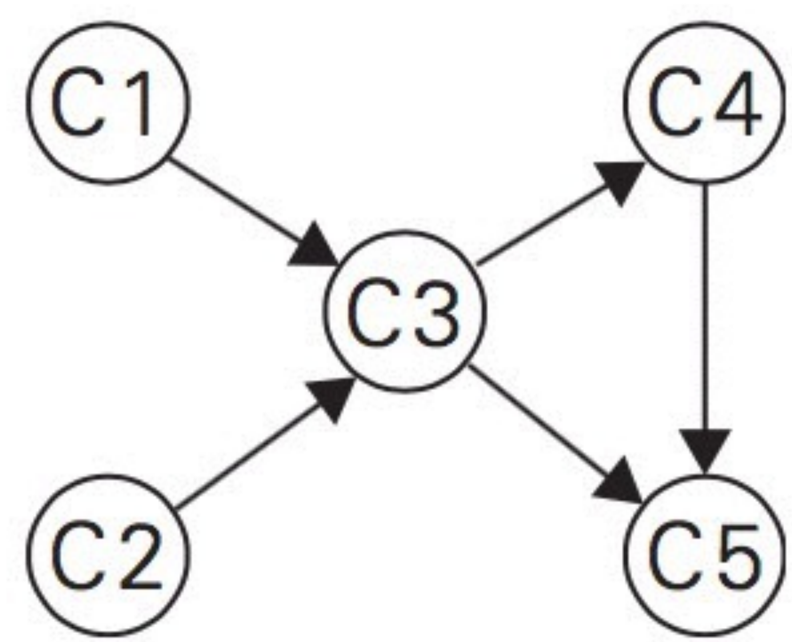




Result from DFS traversal
Not **Directed Acyclic Graph** (DAG)

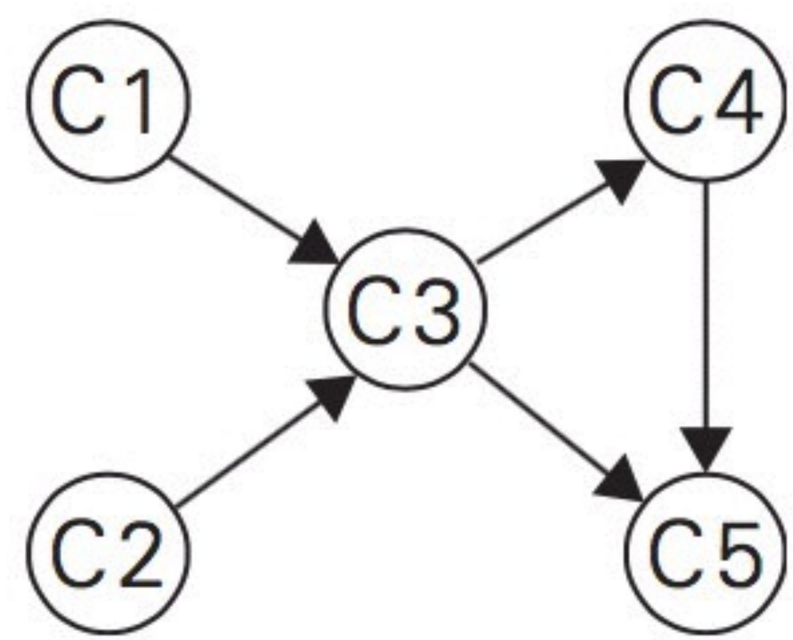
Topological Sorting Problem

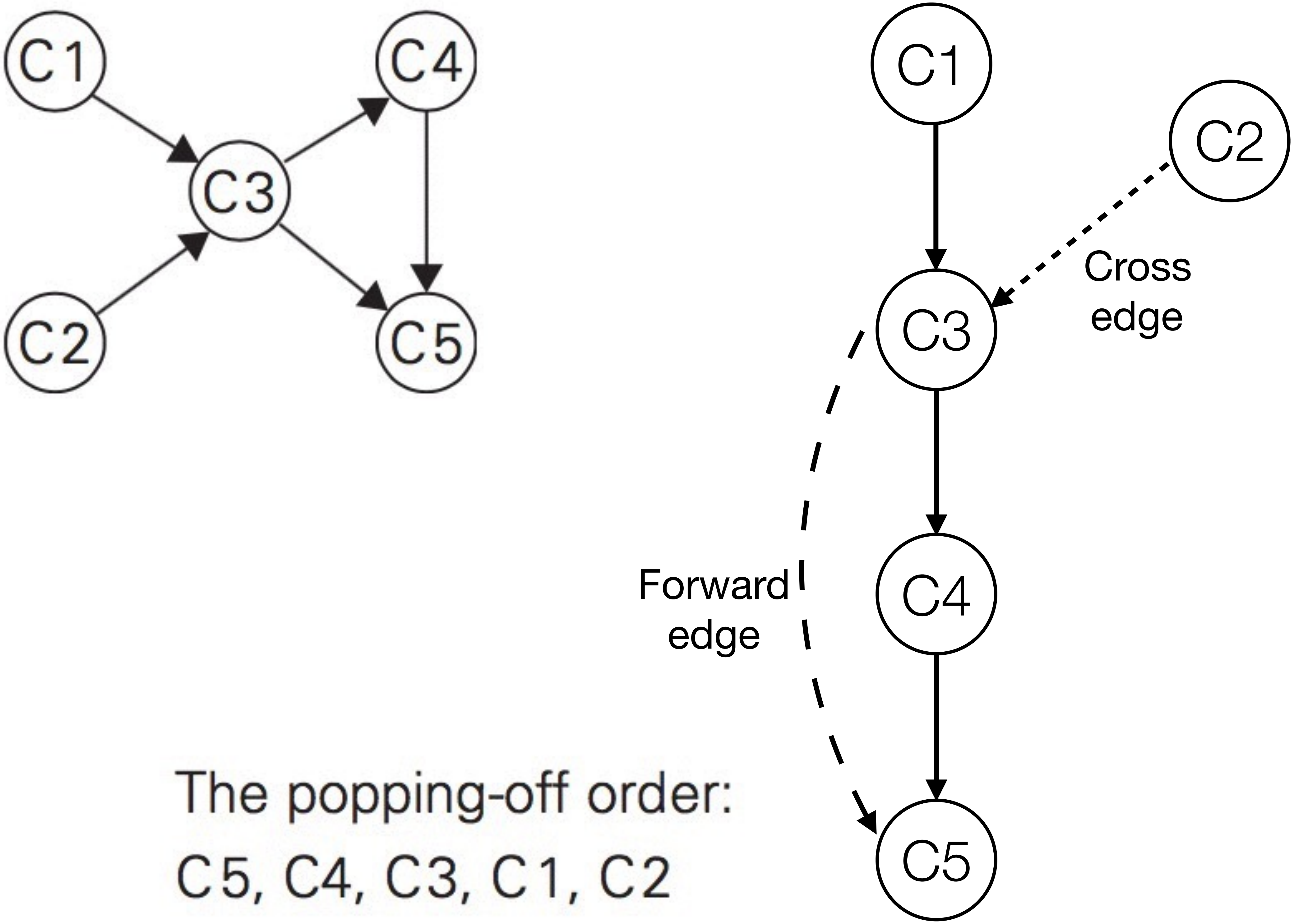
- Five courses {C1, C2, C3, C4, C5}
  - C3 requires C1 and C2 as prerequisites
  - C4 requires C3 as prerequisite
  - C5 requires C3 and C4 as prerequisites
- Student can only take one course per semester.
- What is the order to take the courses?



Solution to 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: Amount to checking if DFS traveral yields DAG (No back edge).

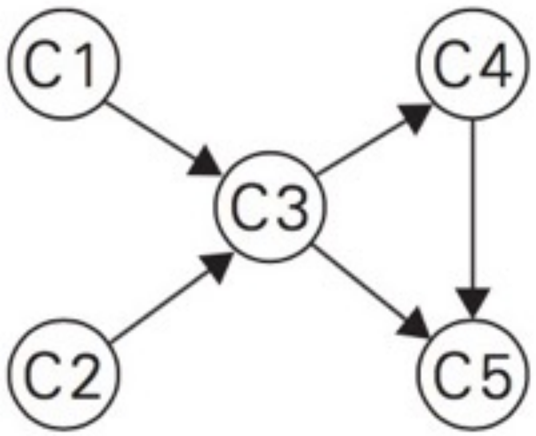




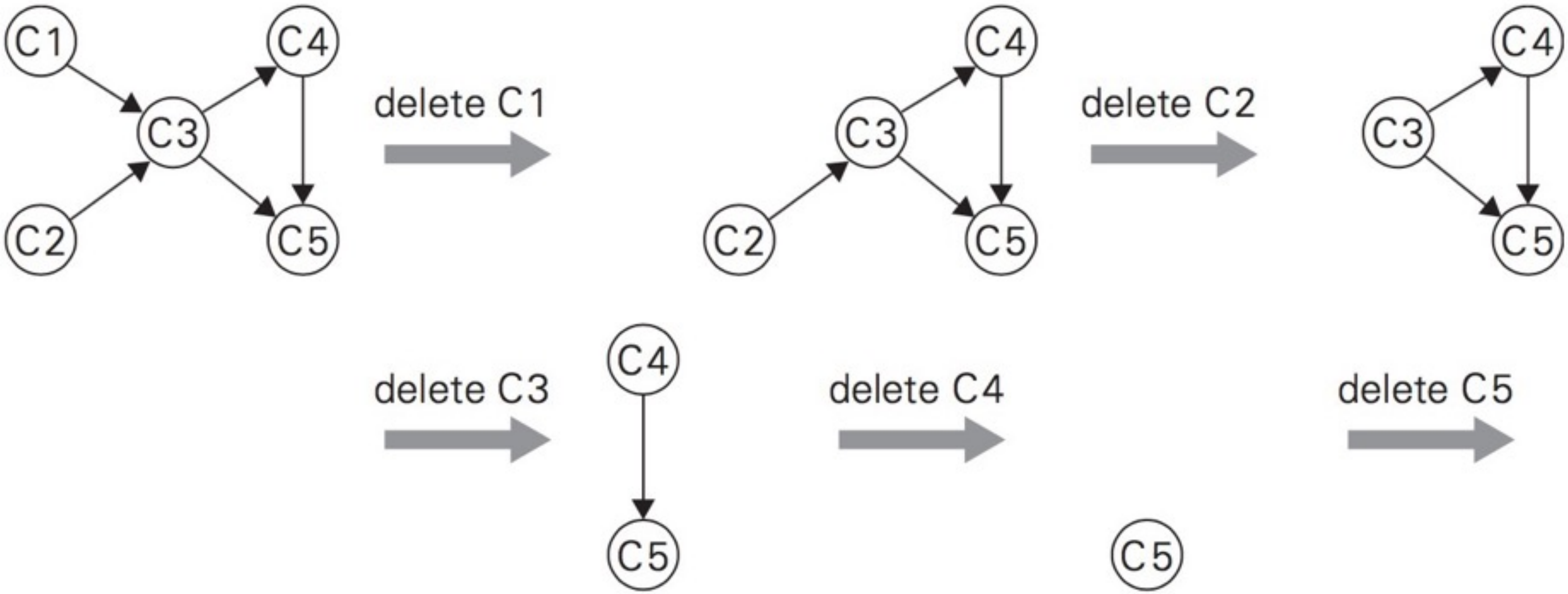
DFS traversal
Directed Acyclic Graph (DAG)

Source Removal Algorithm

- 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 than DFS.



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



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

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
  - Permutations
  - Combinations
  - Subsets

Permutation

- Ex:
  - 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.
- Called "Sampling without Replacement"

k-Permutation

- Ex:
  - Select three cards in succession from a deck of N cards. Each card is removed after being selected.
  - How many possible outcomes (a sequence of 3 distinct cards) ?
- Number of sequences of k out of n distinct objects $(1 \cdot k \cdot n)$.
- Called "Ordered Sampling with Replacement"

Examples

- A club has 25 members. The president and the secretary are to be chosen from the members. What is the total number of ways these two positions can be filled ?
- Three awards (research, teaching, service) will be given one year for a class of 30 students. Each student can receive at most one award. How many possible selections ?

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 ?

Generating Permutations

- Ex: 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.

a b c	→	<u>d</u> a b c a <u>d</u> b c a b <u>d</u> c a b c <u>d</u>
b c a		
c a b		
b a c		
a c b		
c b a	→	<u>d</u> c b a c <u>d</u> b a c b <u>d</u> a c b a <u>d</u>

- 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
- Suppose we already have a pattern of single element {1}
  - How do we generate patterns with two elements 1 and 2?
- Suppose we already have patterns of two elements {1, 2}, {2, 1}.
  - How do we generate patterns with three elements 1, 2, 3 ?

Decrease-by-One Technique

- Remove one element and generate (n-1)! permutations
- Inserting the removed element in all n possible positions of every permutations of n-1 elements.
- Total permutations = n*(n-1)! = n!
- (n-1)! permutations can be generated from (n-1)*(n-2)! and so on.

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**			
Insert 4 into 3,1,2	3,1,2,**4**			
Insert 4 into 3,2,1	2,1,3,**4**			
Insert 4 into 2,3,1	2,3,1,**4**			
Insert 4 into 2,1,3	3,2,1,**4**			

- 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

```
Input: A positive integer n
Output: All permutations of {1, 2, 3, ..., n}

Initialize the first permutation with ←1, ←2, ..., ←n
while The last permutation has a mobile element do
    Find its largest mobile element k
    Swap k with element that the arrow of k points to
    Reverse the direction of all elements larger than k
    Add the new permutation to the list
```

$$\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{\mathbf{3}}$$

Generating All Subsets (Power Set)

- Want to find all subsets of $A = \{a_1, a_2, \ldots, a_n\}$, e.g., knapsack problem.
- Subset of A = Set of whose all its members are also elements of A.
- What are the subsets of $A = \{x, y, z\}$?

Decrease-by-one approach

- All subsets of A = {those without a_n} $\cup$ {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	$\varnothing$
1	
2	
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	$\varnothing$	$\{a_3\}$	$\{a_2\}$	$\{a_2, a_3\}$	$\{a_1\}$	$\{a_1, a_3\}$	$\{a_1, a_2\}$	$\{a_1, a_2, a_3\}$

Binary Search

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

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

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

Algorithm BinarySearch

Input: A sequence of numbers $a_1, a_2, \cdots, a_n$; 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

Efficiency of Binary Search

- 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 $n = 2^k$, 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

- Inefficient solution: Compare two coins one by one.
- More efficient solution
  - Divide n coin into two halfs (leave extra one aside if not even)
  - Put two piles on the balance scale. The lighter one contains the fake coin. What if two piles weight equal?
  - Repeat the division into half like binary search.

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

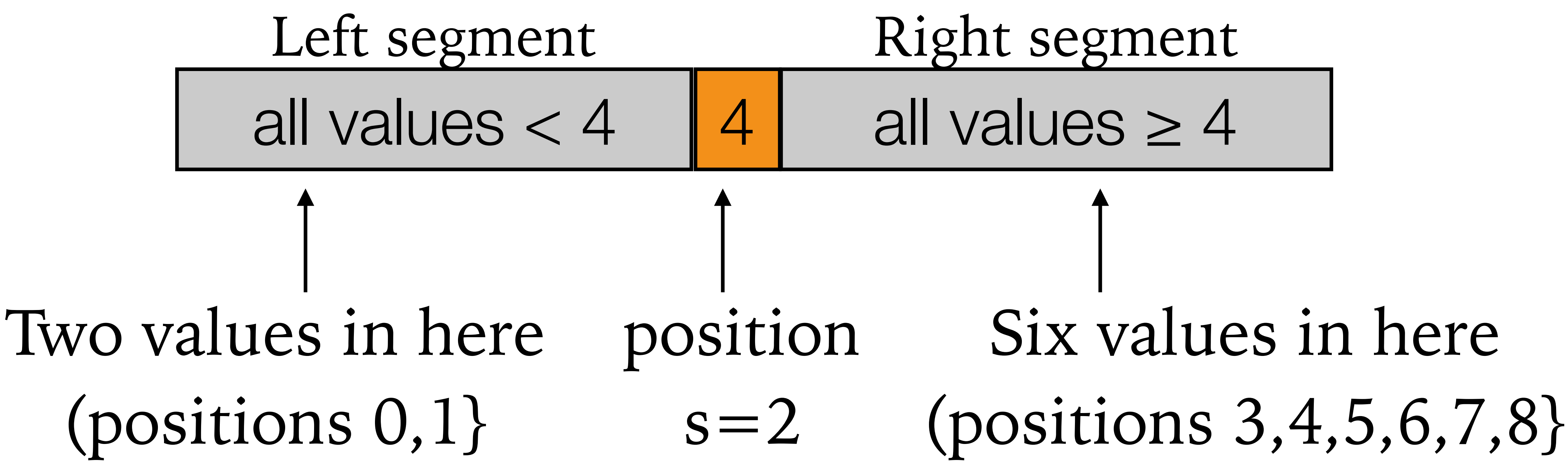
$$W(n) = W(\lfloor n/2 \rfloor) + 1, \quad \text{for } n > 1, \text{ and } W(1) = 0$$

- What is the efficiency of this algorithm?
- Can we do it by dividing into three piles instead of two
  - What if Pile 1 = Pile 2?
  - What if Pile 1 > Pile 2?
  - What if Pile 1 < Pile 2?

The Selection Problem

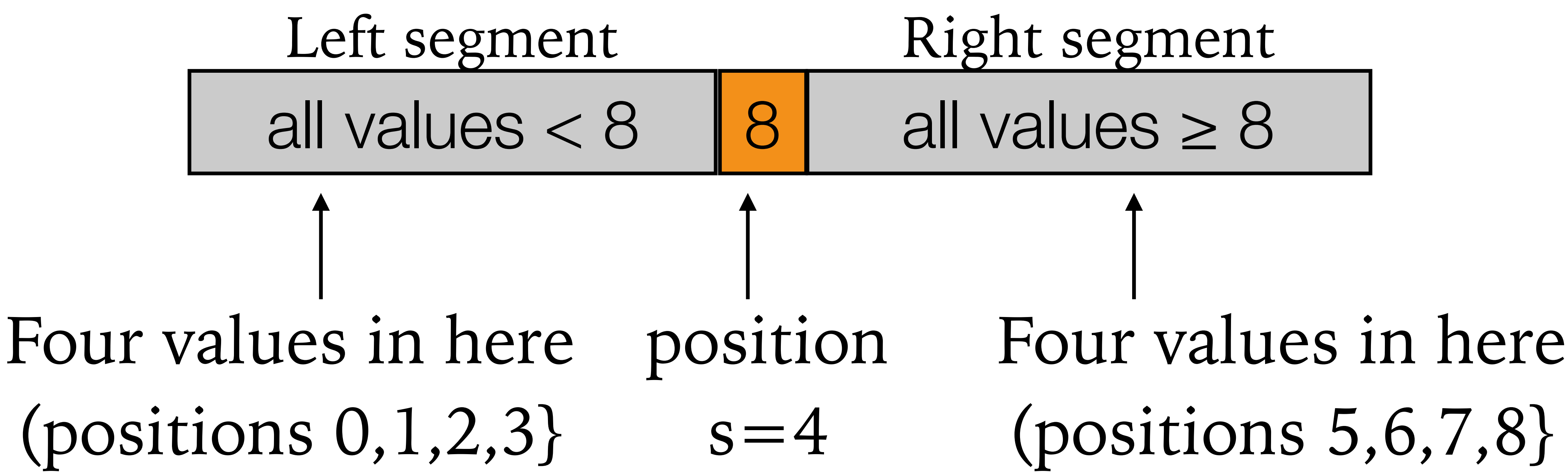
- Find k^{th} smallest element in a list of n numbers (k^{th} order statistic)
  - $k = 1$: Smallest element
  - $k = n$: Largest element
  - $k = ceil(n/2)$: Median (most of interest)
- What is the median of the list {4,1,10,8,7,12,9,2,15}?
- Straightforward approach with sorting but overkill

- 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:



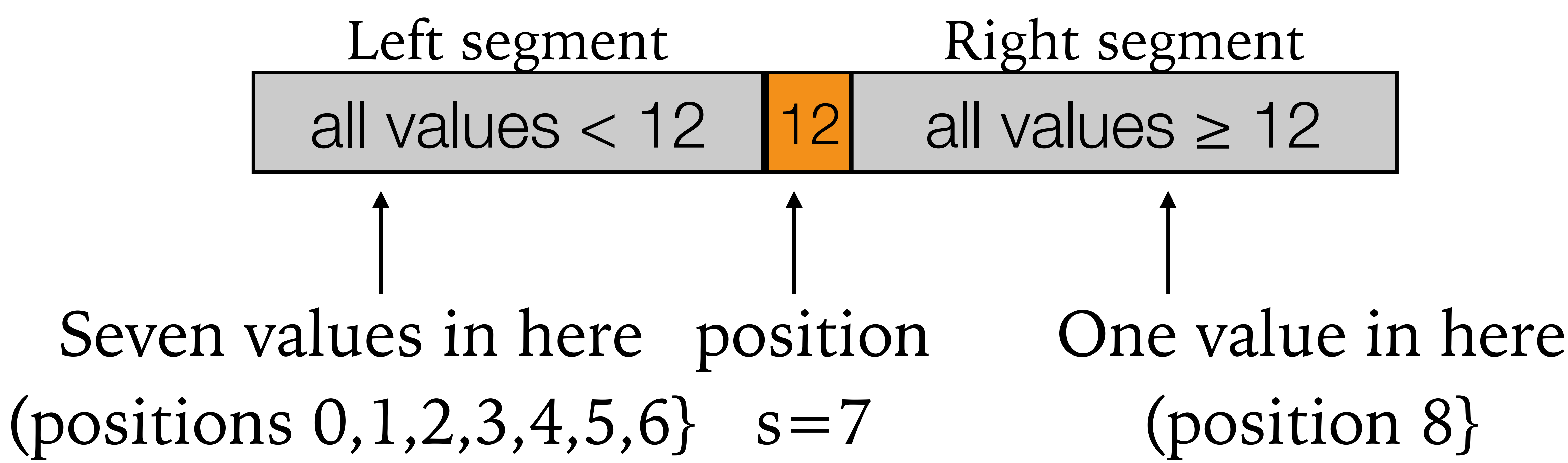
- The 5th-order statistic of the original list must be in the _________ segment as the _____ -order statistic.

- 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:



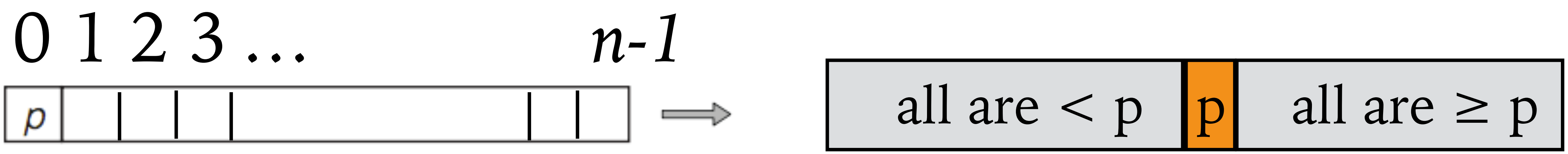
- The 5th-order statistic of the original list must be ____

- 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:

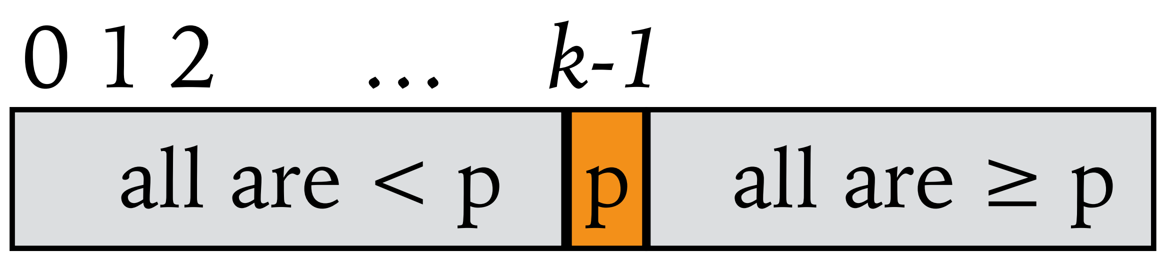


- The 5th-order statistic of the original list must in the _________ segment as the _____ -order statistic.

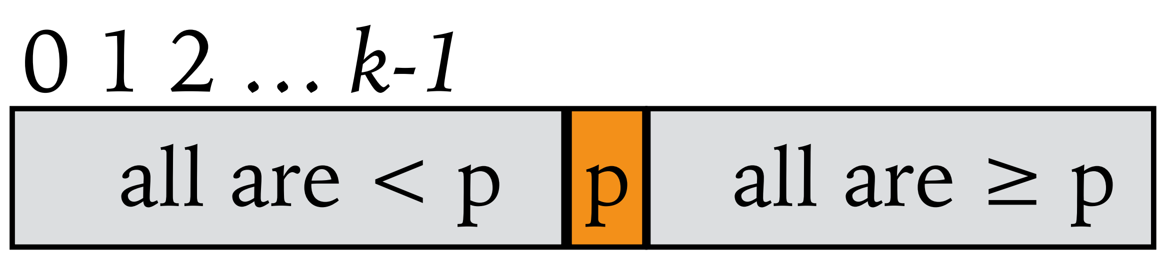
- Suppose we can partition an original list in three segments based on a pivot value *p* shown below:



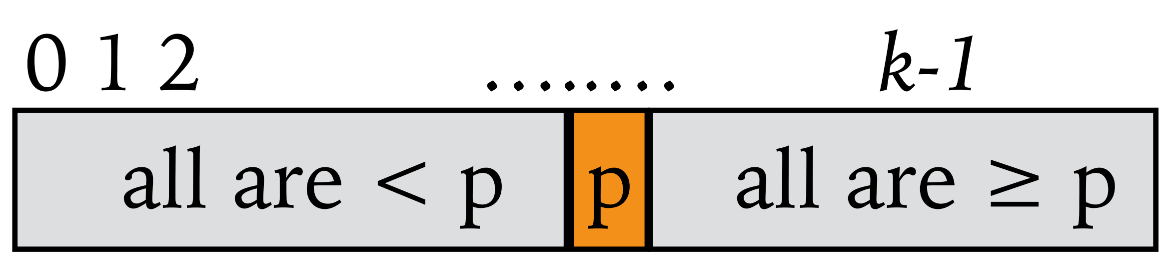
- Three scenarios can happen
  - p is at position s = k-1



  - p is in the position s > k-1



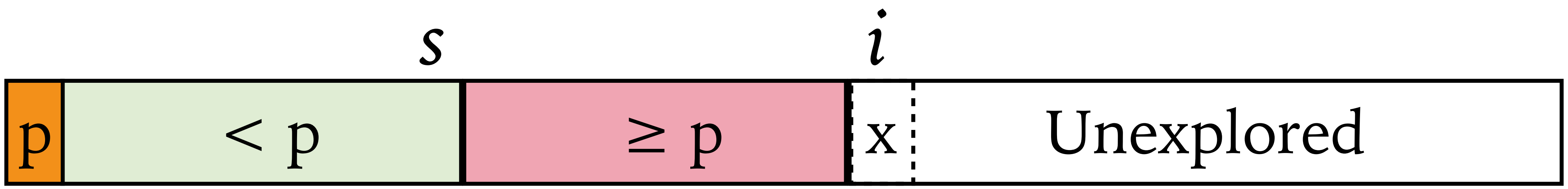
  - p is in the position s < k-1



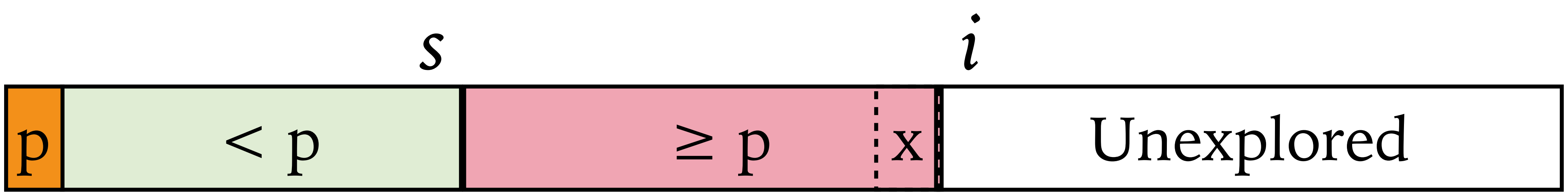
Lomuto Partitioning

- 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.

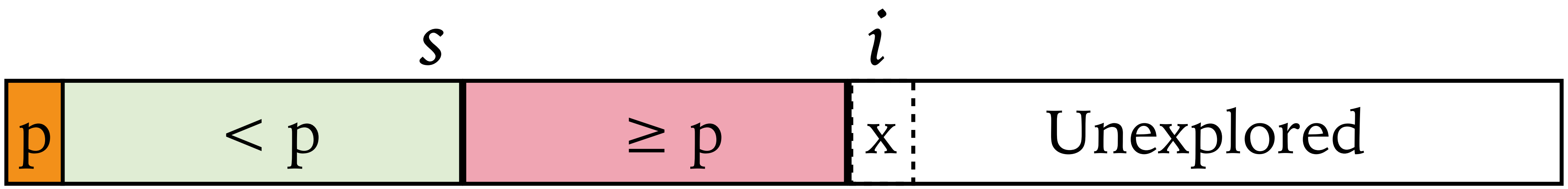
List explored from left to right



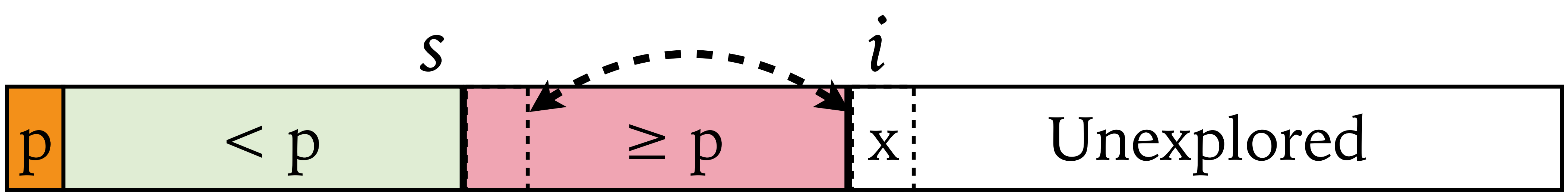
- If A[i] ≥ p, just increment i, which will expand the ≥ p segment.

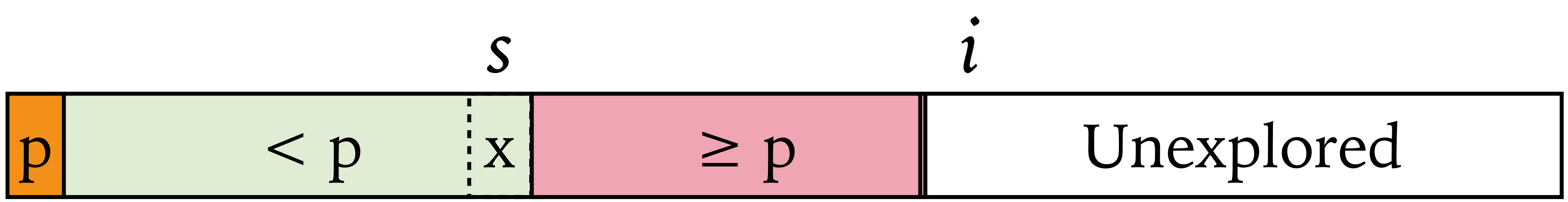


List explored from left to right

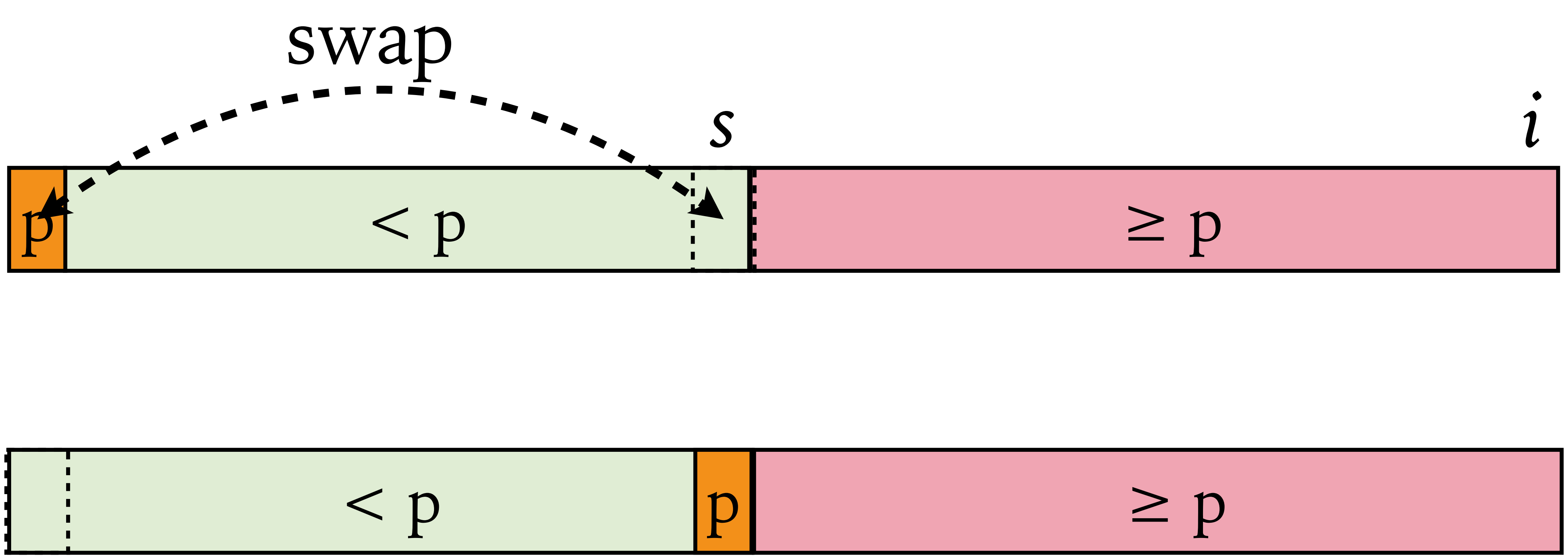


- If A[i] < p,
  - increment s and swap A[s] with A[i], which will expand the <p segment.
  - increment i





- When all elements are explored,



ALGORITHM *LomutoPartition*$(A[l..r])$

```
//Partitions subarray by Lomuto's algorithm using first element as pivot
//Input: A subarray A[l..r] of array A[0..n − 1], defined by its left and right
//         indices l and r (l ≤ r)
//Output: Partition of A[l..r] and the new position of the pivot
p ← A[l]
s ← l
for i ← l + 1 to r do
    if A[i] < p
        s ← s + 1;   swap(A[s], A[i])
swap(A[l], A[s])
return s
```

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
4								
4								
4								
4								
2								

New order is k - (s+1) = 5 - (2+1) = 2

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

ALGORITHM *Quickselect*($A[l..r]$, k)

//Solves the selection problem by recursive partition-based algorithm
//Input: Subarray $A[l..r]$ of array $A[0..n-1]$ of orderable elements and
// integer k ($1 \le k \le r - l + 1$)
//Output: The value of the kth smallest element in $A[l..r]$
$s \leftarrow$ *LomutoPartition*($A[l..r]$) //or another partition algorithm
if $s = k - 1$ **return** $A[s]$
else if $s > l + k - 1$ *Quickselect*($A[l..s-1]$, k)
else *Quickselect*($A[s+1..r]$, $k - 1 - s$)

Also identify k smallest
and n-k largest elements as by product.

Efficiency of Quick Select

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

Summary

- Reduce problem instance to smaller instance of the same problem.
- Solve smaller instance
- Extend solution of smaller instance to obtain solution to original instance

Summary

- Find relationship between a solution to a problem instance and that of a smaller instance.
- Exploit the relationship top-down or bottom-up
- Three variations
  - Decrease-by-one
  - Decrease-by-constant-factor
  - Variable-size-decrease