

PGCS714: Algorithm and Complexity Analysis (3 units)

Lecture 4: Decrease and Conquer

MR. M. YUSUF

Outline

- ▶ Decrease and Conquers
 - ▶ Insertion Sort and Topological Sorting
- ▶ Decrease-by-a Constantan-Factor Algorithm
 - ▶ Binary Search
- ▶ Variable-Size-Decrease Algorithms
 - ▶ Computing a median and Selection Problem
 - ▶ Search and Insertion in a Binary Search Tree

What is Decrease and Conquer?

- ▶ Decrease and Conquer approach is based on exploiting the relationship between a solution to a **given instance** of a problem and a solution to its **smaller instance**.
 - ▶ Top-down: recursive
 - ▶ Bottom-up: iterative
- ▶ 3 major types:
 - ▶ Decrease by a **constant**
 - ▶ Decrease by a **constant factor**
 - ▶ Decrease by **Variable size**

Decrease and Conquer...

- ▶ Decrease by a constant
 - ▶ Compute a^n where $a \neq 0$ and n is a nonnegative
 - ▶ $a^n = a^{n-1} \times a$
 - ▶ Top down: recursive
 - ▶
$$f(n) = \begin{cases} f(n-1) \times a & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$$
 - ▶ Bottom up: iterative
 - ▶ Multiply 1 by a , n times

Decrease and Conquer...

- Decrease by a constant factor (usually 2)

$$a^n = \begin{cases} (a^{n/2})^2 & \text{If } n \text{ is even and positive} \\ (a^{(n-1)/2})^2 \cdot a & \text{If } n \text{ is odd} \\ 1 & \text{If } n = 0 \end{cases}$$

$$5^{29} = (5^{(29-1)/2})^2 \cdot 5 = (5^{14})^2 = 5^{14} \cdot 5^{14}$$

Decrease and Conquer...

- Decrease by a constant factor (usually 2)

ALGORITHM Exponentiate(a, n)

if n = 0

return 1

tmp <- Exponentiate(a, n >> 1)

if (n & 1) = 0 // n is even

return tmp*tmp

else

return tmp*tmp*a

$$a^n = \begin{cases} (a^{n/2})^2 & \text{If } n \text{ is even and positive} \\ (a^{(n-1)/2})^2 \cdot a & \text{If } n \text{ is odd} \\ 1 & \text{If } n = 0 \end{cases}$$

What's the time complexity ? $\Theta(\log n)$

Decrease and Conquer: Insertion Sort

ALGORITHM InsertionSort($A[0..n-1]$)

for $i \leftarrow 1$ **to** $n-1$ **do**

$v \leftarrow A[i]$

$j \leftarrow i-1$

while $j \geq 0$ **and** $A[j] > v$ **do**

$A[j+1] \leftarrow A[j]$

$j \leftarrow j-1$

$A[j+1] \leftarrow v$

Input size: n

Basic op: $A[j] > v$

Why not $j \geq 0$?

$C(n)$ depends on input type ?

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

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

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

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

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

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

17 29 34 45 68 89 90

Decrease and Conquer: Insertion Sort...

ALGORITHM InsertionSort($A[0..n-1]$)

for $i \leftarrow 1$ **to** $n-1$ **do**

$v \leftarrow A[i]$

$j \leftarrow i-1$

while $j \geq 0$ **and** $A[j] > v$ **do**

$A[j+1] \leftarrow A[j]$

$j \leftarrow j-1$

$A[j+1] \leftarrow v$

For almost sorted files,
insertion sort's performance
is excellent!

Can you improve it ?
In place? Stable ?

What is the worst case scenario ?

$A[j] > v$ executes highest # of times

When does that happen ?

$A[j] > A[i]$ for $j = i-1, i-2, \dots, 0$

Worst case input:

An array of strictly decreasing values

What is the best case ?

$A[i-1] \leq A[i]$ for $i = 1, 2, \dots, n-1$

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

$$C_{\text{best}}(n) = \sum_{i=1}^{n-1} 1 = n-1 \in \Theta(n)$$

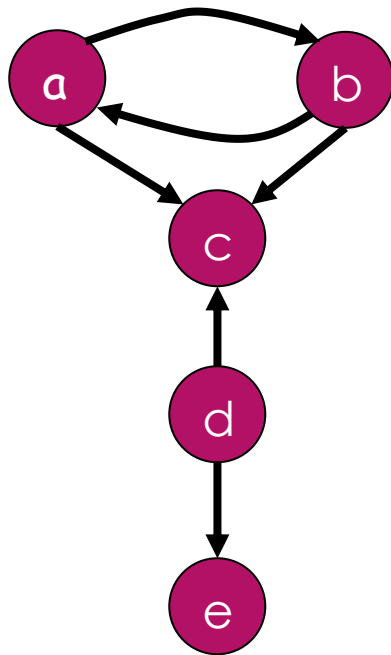
$$C_{\text{avg}}(n) \approx \frac{n^2}{4} \in \Theta(n^2)$$

Topological Sorting

- ▶ Ordering of the vertices of a directed graph such that for every edge uv , u comes before v in the ordering
- ▶ First studied in 1960s in the context of PERT (Project Evaluation and Review Technique) for scheduling in project management.
 - ▶ Jobs are vertices, there is an edge from x to y if job x must be completed before job y can be started
 - ▶ Then topological sorting gives an order in which to perform the jobs

Directed Graph or Digraph

► Directions for all edges



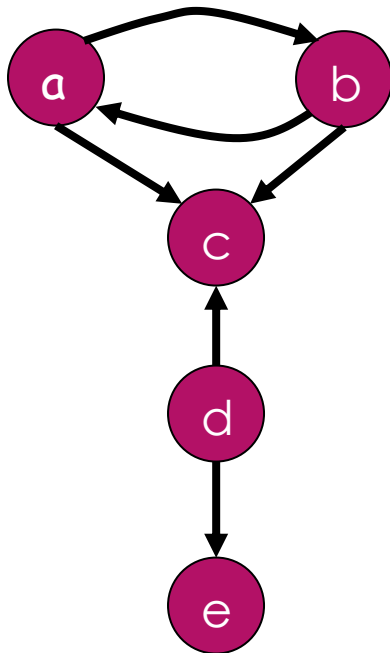
$a \longrightarrow b \longrightarrow c$
 $b \longrightarrow a \longrightarrow c$
 $c \longrightarrow$
 $d \longrightarrow c \longrightarrow e$
 $e \longrightarrow$

One node for one edge

| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | 0 | 1 | 1 | 0 | 0 |
| b | 1 | 0 | 1 | 0 | 0 |
| c | 0 | 0 | 0 | 0 | 0 |
| d | 0 | 0 | 1 | 0 | 1 |
| e | 0 | 0 | 0 | 0 | 0 |

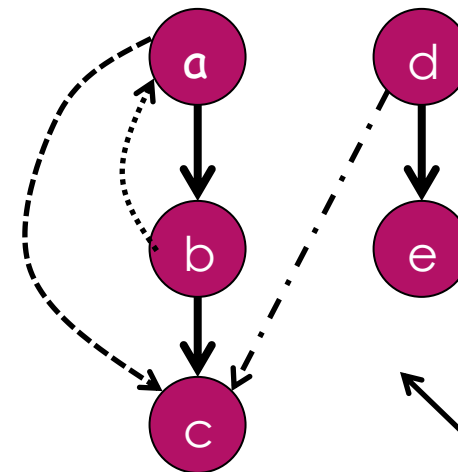
Not symmetric

Digraph



Directed cycle: a, b, a

$a \longrightarrow b \longrightarrow c$
 $b \longrightarrow a \longrightarrow c$
 $c \longrightarrow$
 $d \longrightarrow c \longrightarrow e$
 $e \longrightarrow$



\longrightarrow Tree edge
 $\cdots \longrightarrow$ Back edge
 $- - - \longrightarrow$ Forward edge
 $- \cdot - \cdot \longrightarrow$ Cross edge

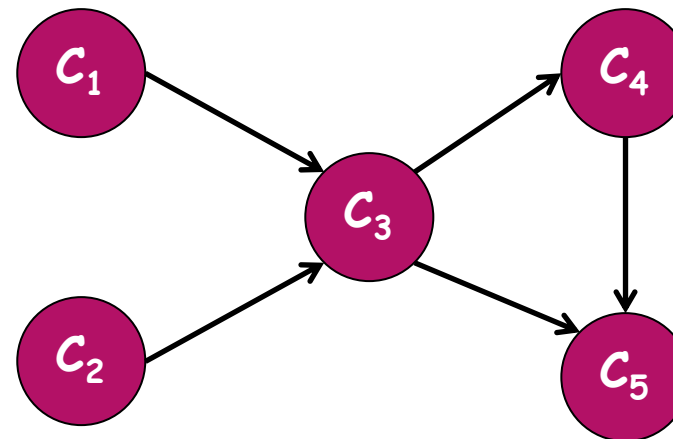
DFS forest

If no directed cycles, the digraph is called “directed acyclic graph” or “DAG”

Topological Sorting Scenario 2

- ▶ Set of 5 courses: $\{ C_1, C_2, C_3, C_4, C_5 \}$
- ▶ C_1 and C_2 have no prerequisites
- ▶ C_3 requires C_1 and C_2
- ▶ C_4 requires C_3
- ▶ C_5 requires C_3 and C_4

$C_1, C_2, C_3, C_4, C_5 !$



Topological Sorting: Usage

- ▶ A large project – e.g., in construction, research, or software development – that involves a multitude of interrelated tasks with known prerequisites
 - ▶ Schedule to minimize the total completion time
- ▶ Instruction scheduling in program compilation, resolving symbol dependencies in linkers, etc.

Outline

- ▶ Decrease and Conquers
 - ▶ Insertion Sort and Topological Sorting
- ▶ **Decrease-by-a Constantan-Factor Algorithm**
 - ▶ Binary Search
- ▶ Variable-Size-Decrease Algorithms
 - ▶ Computing a median and Selection Problem
 - ▶ Search and Insertion in a Binary Search Tree

Decrease-by-a-Constant-Factor Algorithms

► Binary Search

- Highly efficient way to search for a key K in a sorted array $A[0..n-1]$

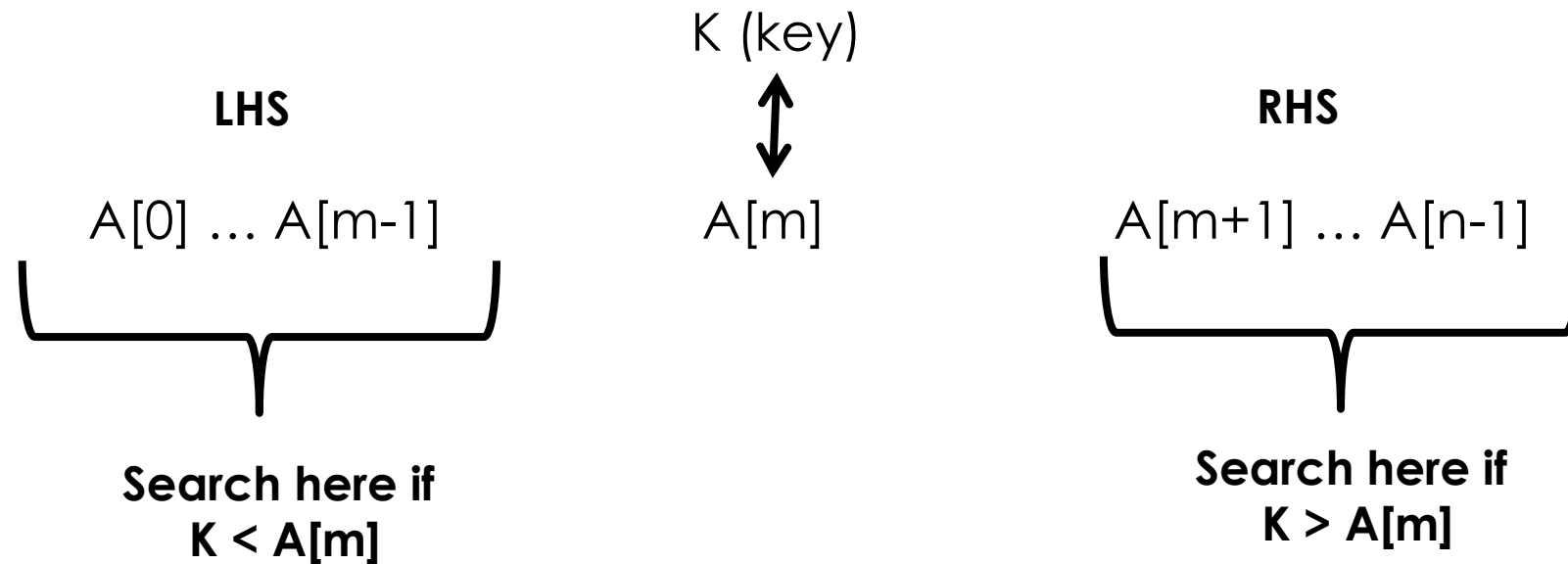
Compare K with A 's middle element $A[m]$

If they match, stop.

Else if $K < A[m]$ do the same for the first half (LHS) of A

Else if $K > A[m]$ do the same for the second half (RHS) of A

Decrease-by-a-Constant-Factor Algorithms: Binary Search



Let us apply binary search for $K = 70$ on the following array:

| | | | | | | | | | | | | |
|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|
| 3, | 14, | 27, | 31, | 39, | 42, | 55, | 70, | 74, | 81, | 85, | 93, | 98 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

Binary Search Algorithm

ALGORITHM BinarySearch($A[0..n-1]$, K)

//Input: $A[0..n-1]$ sorted in ascending order and a search key K

//Output: An index of A 's element equal to K or -1 if no such element

$l \leftarrow 0$

$r \leftarrow n-1$

while $l \leq r$ **do**

$m \leftarrow \lfloor (l + r) / 2 \rfloor$

if $K = A[m]$

return m

else if $K < A[m]$

$r \leftarrow m-1$

else

$l \leftarrow m+1$

return -1

Best-case input: K is the mid position of a sorted element ...

Worst-case input: K is absent or some K is at some special position...

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

$$C_{\text{worst}}(1) = 1$$

To simplify, assume $n = 2^k$

Then $C_{\text{worst}}(n) \in \Theta(\log n)$

Outline

- ▶ Decrease and Conquers
 - ▶ Insertion Sort and Topological Sorting
- ▶ Algorithms for Generating Combinatorial Objects
 - ▶ Generating Permutations and Generating Subsets
- ▶ Decrease-by-a Constantan-Factor Algorithm
 - ▶ Binary Search
- ▶ **Variable-Size-Decrease Algorithms**
 - ▶ Computing a median and Selection Problem
 - ▶ Search and Insertion in a Binary Search Tree

Variable-Size-Decrease

- ▶ Problem size decreases at each iteration in variable amount
- ▶ Euclid's algorithm for computing the greatest common divisor of two integers is one example

Computing Median and the Selection Problem

- ▶ Find the k -th smallest element in a list of n numbers
- ▶ For $k = 1$ or $k = n$, we could just scan the list
- ▶ More interesting case is for $k = \lceil n/2 \rceil$
 - ▶ Find an element that is not larger than one half of the list's elements and not smaller than the other half; this middle element is called the "median"
 - ▶ Important problem in statistics

Median Selection Problem

- ▶ We shall take advantage of the idea of “partitioning” a given list around some value **p** of, say the list’s first element. This element is called the “pivot”.



- ▶ This method is referred to as Lomuto partitioning & Hoare’s partitioning

Median Selection Problem...



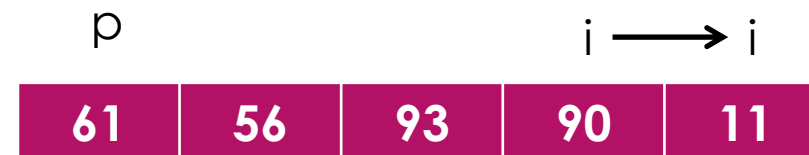
s

A[i] **not less than** p

s → s

A[i] **less than** p

s



s

A[i] **not less than** p

s → s

A[i] **less than** p

s



Lomuto Partitioning Algorithm

ALGORITHM LomutoPartition($A[L..R]$)

//Partition 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 ($L \leq R$)

//Output: Partition of $A[L..R]$ and the new position of the pivot

$p \leftarrow A[l]$

$s \leftarrow L$

for $i \leftarrow L+1$ **to** R **do**

if $A[i] < p$

$s \leftarrow s+1$

 swap($A[s]$, $A[i]$)

swap($A[l]$, $A[s]$)

return s

Median Selection Problem

- ▶ We want to use the Lomuto partitioning to efficiently find out the k -th smallest element of $A[0..n-1]$
- ▶ Let s be the partition's split position
- ▶ If $s = k-1$, pivot p is the k -th smallest
- ▶ If $s > k-1$, the k -th smallest (of entire array) is the k -th smallest of the left part of the partitioned array
- ▶ If $s < k-1$, the k -th smallest (of entire array) is the $[(k-1)-(s+1)+1]$ -th smallest of the right part of the partitioned array

Median Selection Problem: QuickSelect Algorithm

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

//Input: Subarray $A[l..r]$ of array $A[0..n-1]$ of orderable elements and integer k ($1 \leq k \leq r-l+1$)

//Output: The value of the k -th smallest element in $A[l..r]$

$s \leftarrow \text{LomutoPartition}(A[l..r])$

if $s = k-1$

return $A[s]$

else if $s > l+k-1$

 QuickSelect($A[l..s-1]$, k)

else

 QuickSelect($A[s+1..r]$, $k-1-s$)

Median Selection Problem: QuickSelect Algorithm...

$$k = \lceil 9/2 \rceil = 5$$

| | | | | | | | | |
|---|---|----|---|---|----|---|---|----|
| 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 |

| | | | | | | | | |
|---|---|---|---|---|----|---|----|----|
| | s | | | | | | | |
| 2 | 1 | 4 | 8 | 7 | 12 | 9 | 10 | 15 |

$s = 2 < k - 1 = 4$,
so we proceed
with the right part...

Median Selection Problem: QuickSelect Algorithm...

$$k = \lceil 9/2 \rceil = 5$$

| | | | | | | | | |
|---|---|---|---|---|----|---|----|----|
| 2 | 1 | 4 | 8 | 7 | 12 | 9 | 10 | 15 |
|---|---|---|---|---|----|---|----|----|

| | | | | | |
|---|---|----|---|----|----|
| s | i | | | | |
| 8 | 7 | 12 | 9 | 10 | 15 |

| | | | | | |
|---|---|----|---|----|----|
| s | i | | | | |
| 8 | 7 | 12 | 9 | 10 | 15 |

| | | | | | |
|---|---|----|---|----|----|
| s | | | | i | |
| 8 | 7 | 12 | 9 | 10 | 15 |

| | | | | | |
|---|---|----|---|----|----|
| | s | | | | |
| 7 | 8 | 12 | 9 | 10 | 15 |

Now $s (=4) = k-1$
 $(=5-1=4)$,
 we have found the
 median!

Thank You