Data Structures and Algorithms

(CS210A)

Lecture 41

- Miscellaneous problems

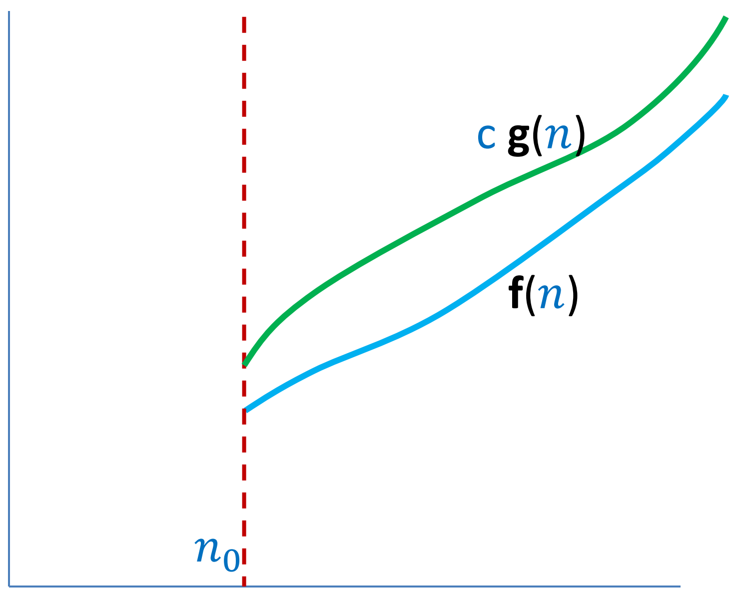
Order notation

Definition: Let **f**(n) and **g**(n) be any two increasing functions of n.

f(n) is said to be

if there exist constants c and n_0 such that

$$\mathbf{f}(n) \le c\,\mathbf{g}(n) \quad \text{for all } n > n_0$$



$\mathbf{f}(n) = O(\mathbf{g}(n))$

Order notation extended

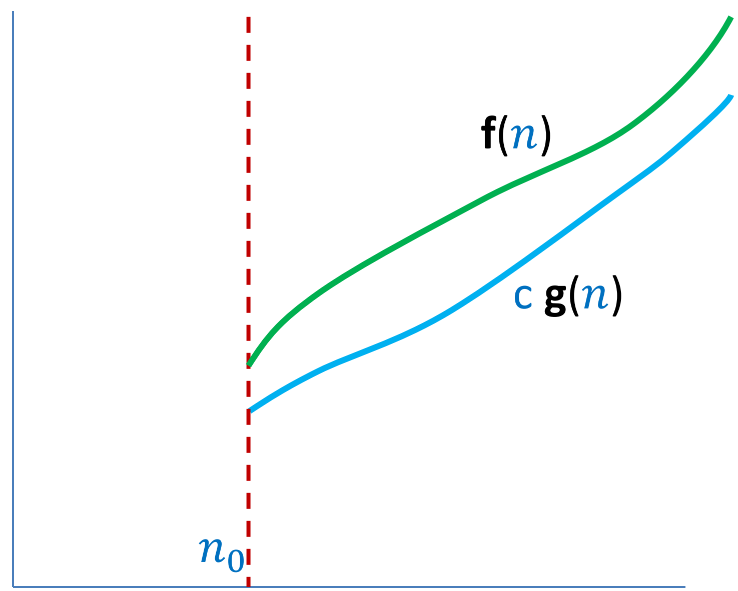
Definition: Let **f**(n) and **g**(n) be any two increasing functions of n.

f(n) is said to be

if there exist constants c and n_0 such that

$$\mathbf{f}(n) \geq c\ \mathbf{g}(n) \quad \text{for all } n > n_0$$

f(n) = $\boldsymbol{\Omega}$(**g**(n))



$$\frac{n^2}{100} = \boldsymbol{\Omega}(10000\ n \log n)$$

Order notation extended

Observations:

- **f**(n) = **O**(**g**(n)) **if and only if** **g**(n) = $\mathbf{\Omega}$(**f**(n))

One more Notation:

If **f**(n) = **O**(**g**(n)) **and** **g**(n) = **O**(**f**(n)), then

g(n) = $\mathbf{\Theta}$(**f**(n))

Examples:

- $\frac{n^2}{100} = \mathbf{\Theta}(10000\, n^2)$

- Time complexity of Quick Sort is $\mathbf{\Omega}(n \log n)$

- Time complexity of Merge sort is $\mathbf{\Theta}(n \log n)$

Time complexity of a problem

Time complexity of sorting

Example: Sorting

- **Algorithm 1** : Selection Sort with time complexity $\mathbf{O}(n^2)$

$\mathbf{O}(n^2)$ **Upper** bound

- **Algorithm 2** : Merge Sort with time complexity $\mathbf{O}(n \log n)$

$\mathbf{O}(n \log n)$

- Each comparison based sorting algorithm needs to perform $\mathbf{\Omega}(n \log n)$ comparisons in the worst case.

$\mathbf{\Omega}(n \log n)$

- Sorting must takes $\mathbf{\Omega}(n)$ time since it has to read each item at least once.

$\mathbf{\Omega}(n)$ **Lower** bound

Sorting has time complexity of $\mathbf{\Theta}(n \log n)$

Time complexity of a problem

Example: All-pairs shortest paths (APSP)

- **Algorithm 1** : Floyd Warshal Algorithm with time complexity $\mathbf{O}(n^3)$
- **Algorithm 2** : Johnson' algorithm with time complexity $\mathbf{O}(mn \log n)$
- All-pairs shortest paths must require $\boldsymbol{\Omega}(n^2)$ time

Time complexity of APSP

$\mathbf{O}(n^3)$ — **Upper** bound

$\downarrow$

$\mathbf{O}(mn \log n)$

$\boldsymbol{\Omega}(n^2)$ — **Lower** bound

There is still a gap between upper and lower bounds for APSP. ☹

Aim of theoretical computer science

For any given computational problem ***P***

- Get **smallest possible** **upper bound** on its time complexity

This requires designing better algorithm

Reduce the GAP

- Get **largest possible** **lower bound** on its time complexity.

?

How to establish lower bound

Two ways:

- **Adversarial approach**

A gentle introduction today

- **Limitation** of the model of computation

CS345

Adversarial approach

Key aspects

Algorithm:

- Algorithm does not have free access to the input.

  To access any item in the input, algorithm has to spend some time.

- The execution of an algorithm at any step is determined

  only by the (partial) input it has seen till now.

Adversary

- Adversary has access to **all** possible inputs of a problem.

- The sole aim of adversary is to make an algorithm work really hard.

  For this purpose, adversary discloses the input *cleverly*.

Locating 1 problem

Input: An array **A**$[0 \ldots n-1]$ with an unknown i s.t.

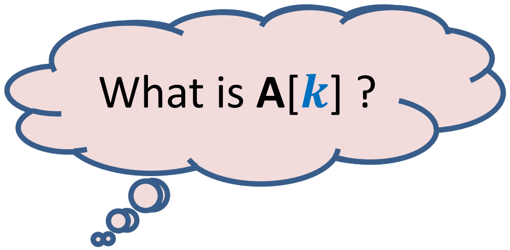
- For all $j \neq i$, **A**$[j] = 0$
- **A**$[i] = 1$

Aim: To locate/search **1** in **A**.

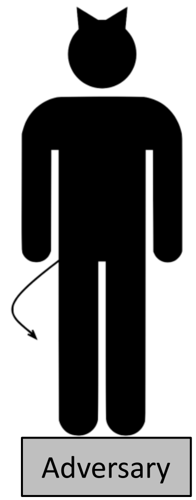
Upper bound: $\mathbf{O}(n)$

Lower bound: $\mathbf{\Omega}(n)$

Lower bound on Locating 1 problem



Algorithm



Input

0	0	0	0	0	0	1	0	0	0	**A**
		j		k			i			

Miscellaneous problems

Problem 1

Input: Given an array **A** storing n numbers,

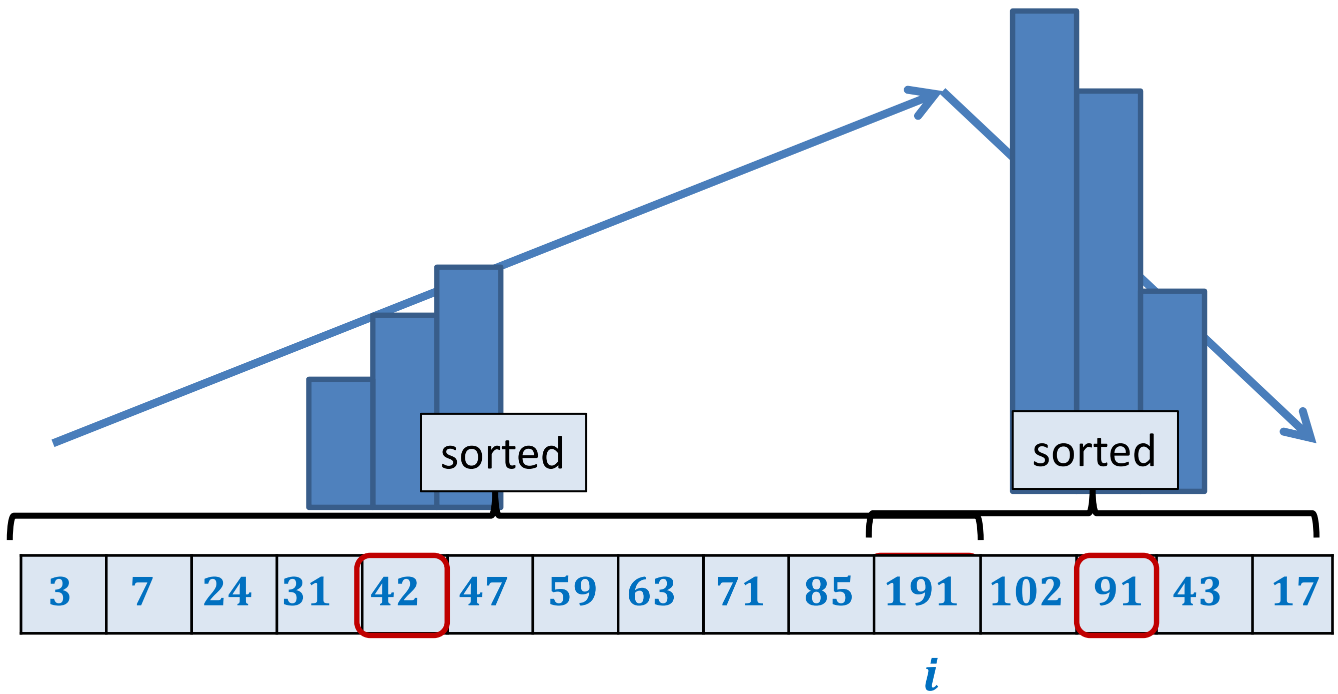
there is an $i < n$ (unknown) s.t.

A$[0] <$ **A**$[1] < \ldots <$ **A**$[i] >$ **A**$[i+1] > \ldots >$ **A**$[n-1]$

Aim:

To search efficiently

Answer : **O**$(\log n)$ is possible



Problem 2

Input:

Given an array **A** storing n numbers,

there is an $i < n$ (unknown) s.t.

A$[0] \leq$ **A**$[1] \leq \ldots \leq$ **A**$[i] \geq$ **A**$[i+1] \geq \ldots \geq$ **A**$[n-1]$

Aim:

To search efficiently

Answer : $\Omega(n)$ time complexity

Locating 1 **problem** is a special case of **Problem 2**

Problem 3

Input:

Given an array **A** storing n numbers,

there are $i < j < n$ (unknown) s.t.

A$[0] <$ **A**$[1] < \ldots <$ **A**$[i] >$ **A**$[i+1] > \ldots >$ **A**$[j] <$ **A**$[j+1] < \ldots <$ **A**$[n-1]$

Aim:

To search efficiently

Answer : $\Omega(n)$ time complexity

Locating 0 problem

Input: An array **A**$[0 \ldots n-1]$ with an unknown i s.t.

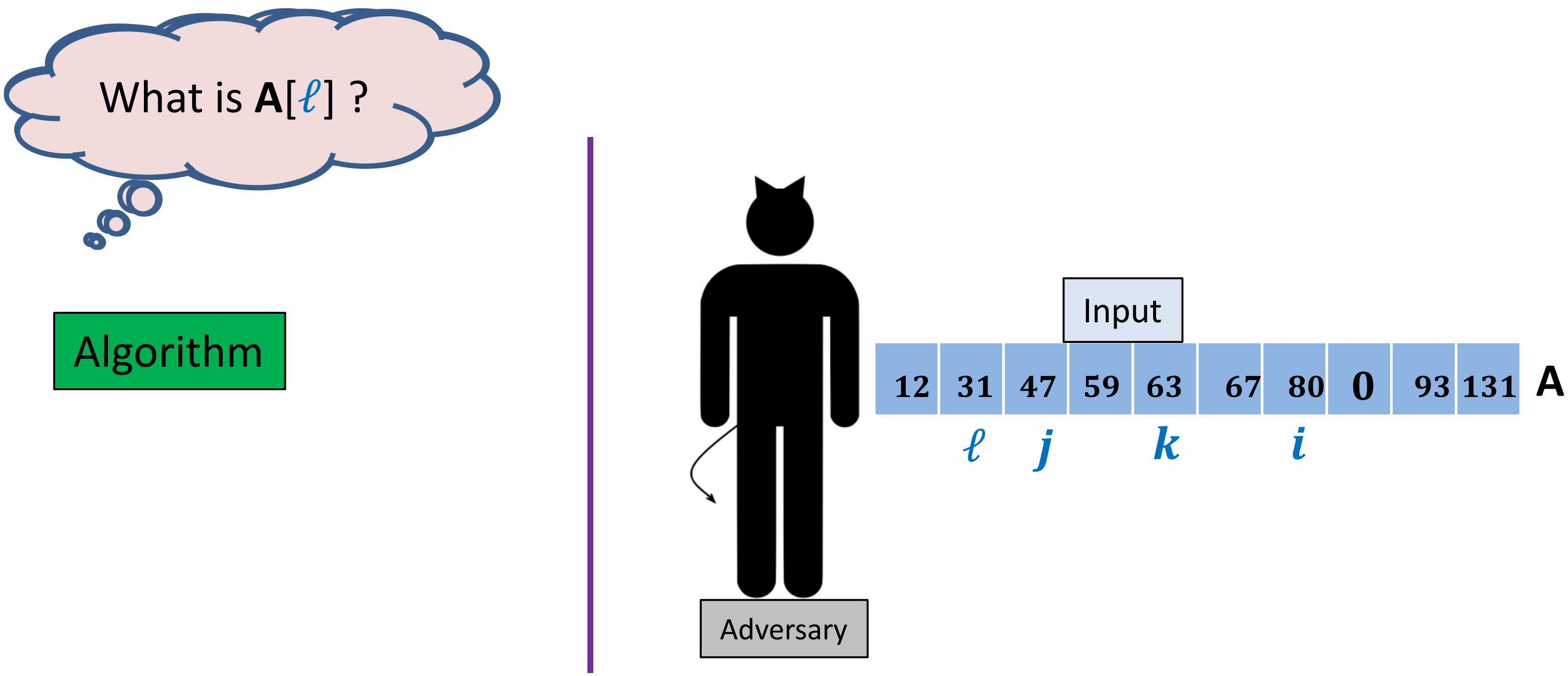
- For all $j < i$, **A**$[j] > 0$ **and** **A**$[j-1] <$ **A**$[j]$
- **A**$[i]$ = 0
- For all $j > i$, **A**$[j] > 0$ **and** **A**$[j] <$ **A**$[j+1]$

Aim: To locate/search **0** in **A.**

Upper bound: **O**(n)

Lower bound: $\Omega(n)$

Lower bound on Locating 0 problem



Problem 3

Input:

Given an array **A** storing n numbers,

there are $i < j < n$ (unknown) s.t.

A$[0]<$ **A**$[1]<$ …$<$ **A**$[i]$ $>$ **A**$[i+1]$ $>$**…** $>$**A**$[j]$ $<$ **A**$[j+1]$ $<$**…** $<$**A**$[n-1]$

Aim:

To search efficiently

Answer : $\Omega(n)$ time complexity

Locating 0 **problem** is a special case of **Problem 3**

Problem 4

Input:

Given a sorted array **A** storing n numbers,

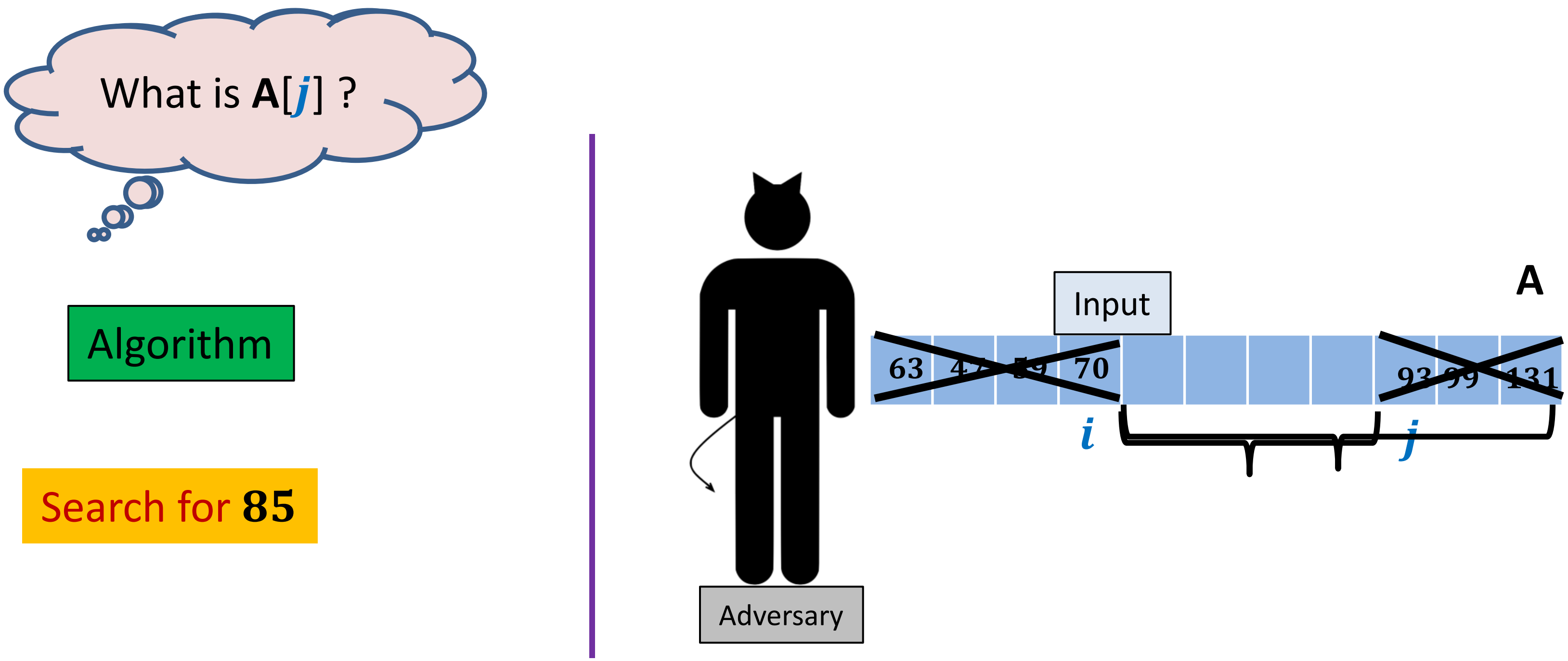
Aim:

To search efficiently

Answer : $\mathbf{O}(\log n)$ time complexity (binary search)

$\mathbf{\Omega}(\log n)$ time complexity

Lower bound on Problem 4



Problem 5

Input:

Given a 2-dimensional square grid storing n^2 distinct numbers,

Aim:

To find a Local minima efficiently

Answer :

$\mathbf{O}(n)$ time complexity (in the second week of the course)

$\boldsymbol{\Omega}(n)$ time complexity

Prove it using Adversarial arguments during summer vacations

Final slide

That’s all. I hope you enjoyed this lecture.