COL 351: Analysis and Design of Algorithms

Lecture 1

Grading Policy

1. Quizzes - 10%
   (surprised / announced)

2. Assignments - 4 x 5% = 20%
   (must be typed in word/latex)
   (group of size at most ~~two~~)

3. Exams - 30% + 35%

4. Attendance - 5%

Audit paas criteria (if audit allowed): 40%

Academic Honesty

Cheating or allowing anyone to copy in quizzes, exams, or assignments would lead to strict disciplinary action.

Reference Books

1. ***Algorithm Design*** by Jon Kleinberg and Eva Tardos
2. ***Algorithms*** by Dasgupta, Papadimitriou, and Vazirani

Tutorials

The course calendar is available at - https://web.iitd.ac.in/~keerti/calendar.html

S.No.	Date	Day	Lectures	Tutorials	Groups	Remarks
2	3.8.2022	Wednesday	L1			
3	4.8.2022	Thursday				
4	5.8.2022	Friday	L2			
5	6.8.2022	Saturday	L3			Tuesday's Timetable
6	7.8.2022	Sunday				
7	8.8.2022	Monday		1	Group 4	
8	9.8.2022	Tuesday	Muharram			
9	10.8.2022	Wednesday	L4			
10	11.8.2022	Thursday	Raksha Bandhan			
11	12.8.2022	Friday	L5	1	Group 3	
12	13.8.2022	Saturday				

COL 106 (DS)	This Course
Arrays, Link lists, Stacks	• Designing algorithms
Trees, Binary trees, AVL trees, KD trees	• Different algorithm paradigms 1. Greedy algorithms 2. Dynamic programming 3. Divide & Conquer
How to use these data structures in computational problems? • Queues are used in BFS • Stacks are used in DFS • Priority queues are used in Dijkstra's algorithm	1/5 • Hard Problems: Problems which are unlikely to have an efficient solution. poly time • How to prove that a problem is hard?

Today's Lecture

1. Asymptotic Bounds (O, Ω, Θ)

2. Examples of Time complexity

3. Merge Sort

4. Computing n^{th} Fibonacci Number efficiently

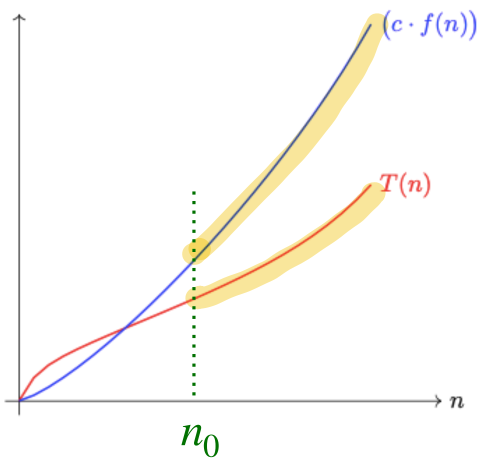
COL 106

Asymptotic Bound (Big O notation)

$T(n)$ = Number of steps taken by an algorithm on an input of size n.

Def^n $T(n) = O(f(n))$ for $T(n), f(n) \geq 0$

if $\exists\ c, n_0 > 0$ $T(n) \leq c \cdot f(n)$ $\forall n \geq n_0$

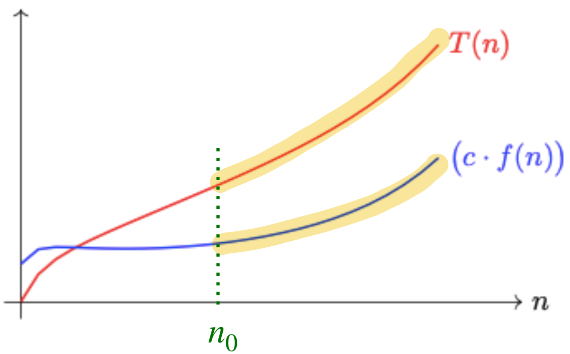


Asymptotic Bound (Ω notation)

$T(n)$ = Number of steps taken by an algorithm on an input of size n.

Defn $\quad T(n) = \Omega(f(n)) \quad$ for $\quad T(n), f(n) \geq 0$

if $\quad c, n_0 > 0 \quad T(n) \geq c \cdot f(n) \quad \forall\, n \geq n_0$

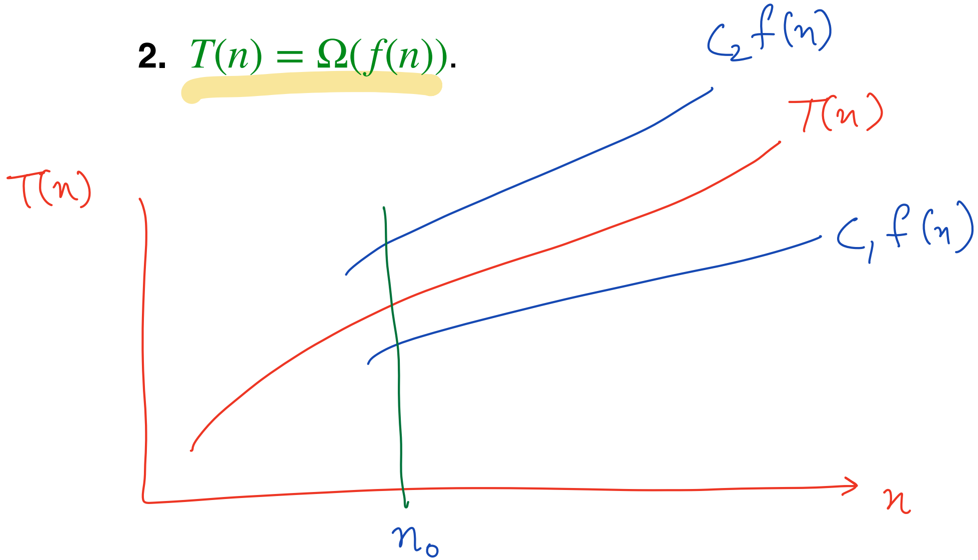


Asymptotic Bound (Θ notation)

$T(n)$ = Number of steps taken by an algorithm on an input of size n.

Definition: For any non-negative functions $T(n)$ and $f(n)$, we say $T(n) = \Theta(f(n))$ if

1. $T(n) = O(f(n))$, and
2. $T(n) = \Omega(f(n))$.



$n_0, c_1, c_2 > 0$

Example:

Problem:

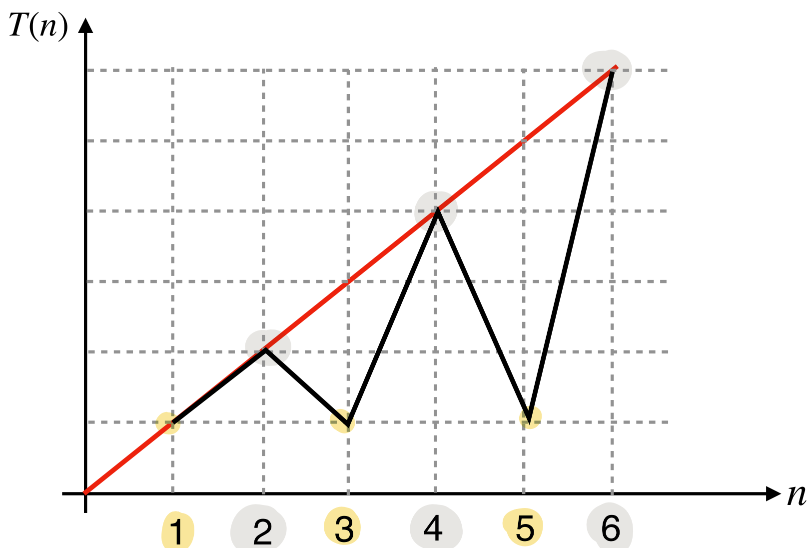
Given an array A of size n, output sum of all entries if n is even, and −1 otherwise.

$T(n)$, the number of steps is.

$$T(n) = \begin{cases} n, & \text{if } n \text{ is even} \\ 1, & \text{if } n \text{ is odd} \end{cases}$$

H.W.

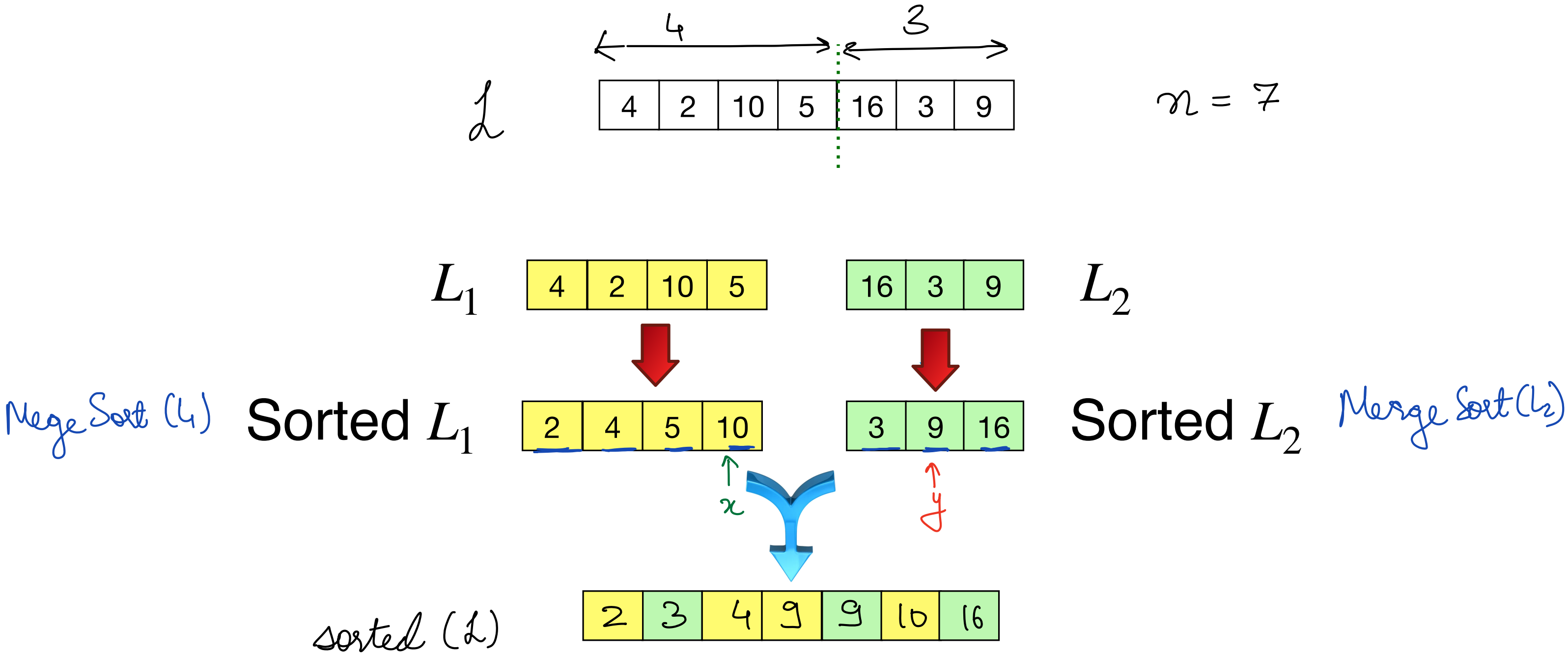
- $T(n) = O(n)$
- $T(n) \neq \Theta(n)$



Exercises

- $n + b = O(n)$
- $2\lceil \log_{10} n \rceil + c = O(\log_2 n)$
- $\dfrac{n^3}{2} + n^{1.5} = \Theta(n^3)$
- $a_d n^d + \cdots + a_1 n + a_0 = O(n^d)$, for $a_d > 0$
- $4n = O(n \log n)$
- $n^c = O(2^n)$, for each $c > 0$
- $n \neq O(\log^k n)$, for each integer $k > 0$
- $n \neq O(1)$

Example: Merge Sort



Merge time $= O(|L_1| + |L_2|)$

$= O(n)$

Example: Merge Sort

MergeSort(L)

$n = length(L)$;

If $n = 1$ then Return;

A = new list of size n;

Set $L_1 = L\left[0, \frac{n}{2}\right]$ and $L_2 = L\left[\frac{n}{2} + 1, n-1\right]$; SPLIT $O(n)$

MergeSort(L_1);

MergeSort(L_2);

Set $x, y, pos = 0$;

While $(x < length(L_1)$ **or** $y < length(L_2))$

If $(L_1[x] \leqslant L_2[y]$ and $x < length(L_1))$ **then**

Set $A[pos] = L_1[x]$, and increment pos and x by 1;

Else

Set $A[pos] = L_2[x]$, and increment pos and y by 1;

Merge. $O(n)$ time

Recurrence

$$T(n) = 2\,T\left(\frac{n}{2}\right) + O(n)$$

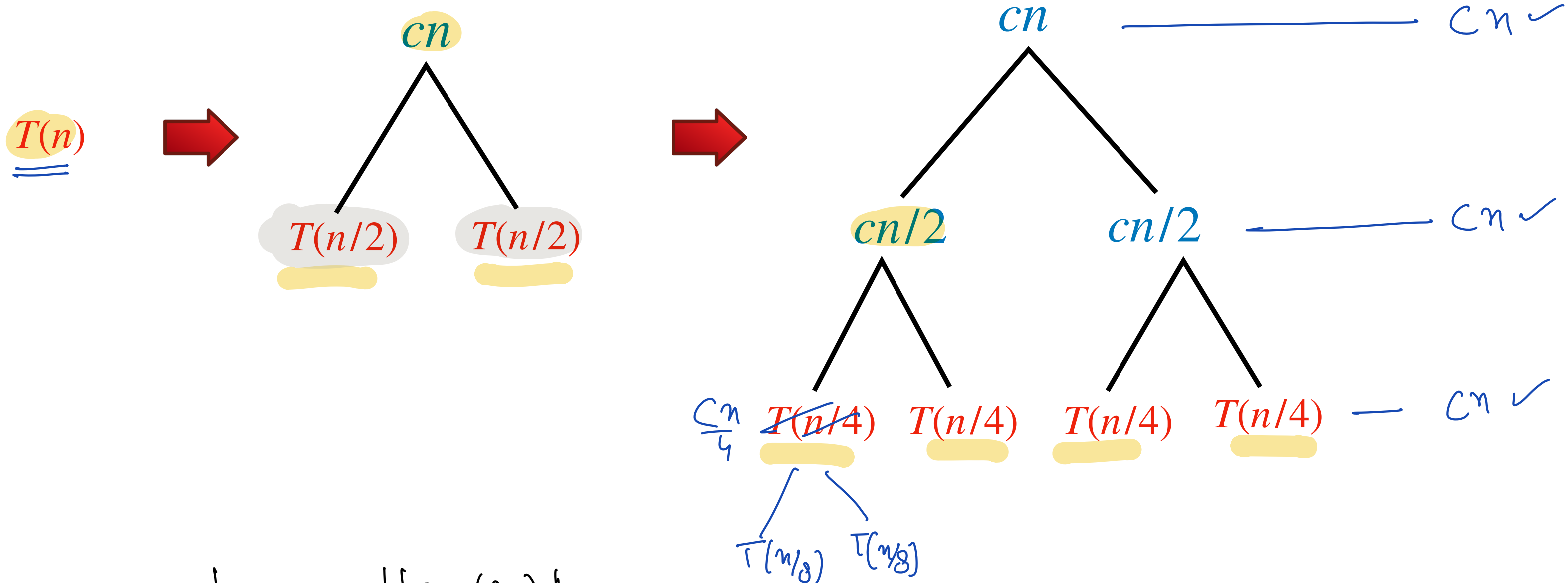
Time complexity = $O(n \log n)$

Example: Merge Sort

$T(n) \leqslant 2\,T(n/2) + cn$

Let $T(n)$ = number of steps taken by the algorithm. Then, $T(n) \;\leqslant\; 2\,T\!\left(\frac{n}{2}\right) \;+\; cn$

$n/2$



height $= \lfloor \log_2(n) \rfloor$

There will be $\Theta(n)$ terms of value $T(1)$

$\boxed{cn \log_2 n}$

Merge Sort - Unequal balance

Let $T(n)$ = number of steps taken by the algorithm.

What if $|L_1| = n/3$, and $|L_2| = 2n/3$?

$$\frac{|L_1|}{|L_2|} = \frac{1}{2}$$

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n)$$

H.W. $\quad T(n) = O(n \log n)$

Merge Sort - Unequal balance

Let $T(n)$ = number of steps taken by the algorithm.

What if $|L_1| = \sqrt{n}$, and $|L_2| = n - \sqrt{n}$?

$$\frac{|L_1|}{|L_2|} \approx \frac{1}{\sqrt{n}}$$

$$T(n) = T(\sqrt{n}) + T(n-\sqrt{n}) + O(n).$$

H.W. Obtain a tight bound on the $T(n)$. ← Tut 1.

Algorithm Paradigms

Merge Sort.

$n/2$, $n/2$

$T(n/2)$, $T(n/2)$

Merge - $O(n)$

1. **Divide and Conquer**
   - Divide the problem into smaller problems
   - Solve the smaller problems
   - Combine

2. **Dynamic Programming**

   Reduce the problem on an input of size n into problems of size

   $n-\alpha$

   $m-\beta$

   $n-1, n-2, n-3,..$ etc.

   Fibonchhi numbers.

   Recursion — exp time

   Memoization — $O(n^2)$

3. **Greedy Strategy**

   Build solution greedily.

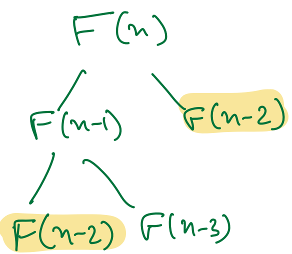
   Lec 2-3-4

$\mathbf{N^{th}}$ Fibonacci Number

0 1 1 2 3 5 8 12 ...

F_2

Defined by recurrence relation $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-2} + F_{n-1}$ for $n > 1$.



Recursion

ALGO

> If $n = 0$ then Return 0;
>
> If $n = 1$ then Return 1;
>
> x = Fibonacci($n - 1$); ✓
>
> y = Fibonacci($n - 2$); ✓
>
> Return $x + y$;

Fibonacci(n)

← BAD algo.

Time complexity is too Large

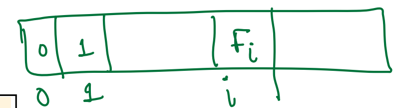
$T(n) = T(n-2) + T(n-1) + \underbrace{\text{Time-to-add-two-large-numbers}}$

$T(n) \geq F_n \geq \sqrt{2}^n \quad \Rightarrow T(n) = \Omega(\sqrt{2}^n)$

H.W. $F_n \geq \sqrt{2}^n, \ \forall n \geq 2$

N^{th} Fibonacci Number - Improved algorithm

Defined by recurrence relation $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-2} + F_{n-1}$ for $n > 1$.



Storing all sol^n

$O(n)$ iterations

$\boxed{\text{Memoization}}$

Allocate an array A of size $n+1$;

Set $A[0] = 0$ and $A[1] = 1$;

For ($i = 2$ to n) do

$$A[i] = \underbrace{A[i-1] + A[i-2]}_{};$$

Return $A[n]$;

Fibonacci-Improved(n)

$$\boxed{6^n \geq F_n \geq \sqrt{2}^n}$$

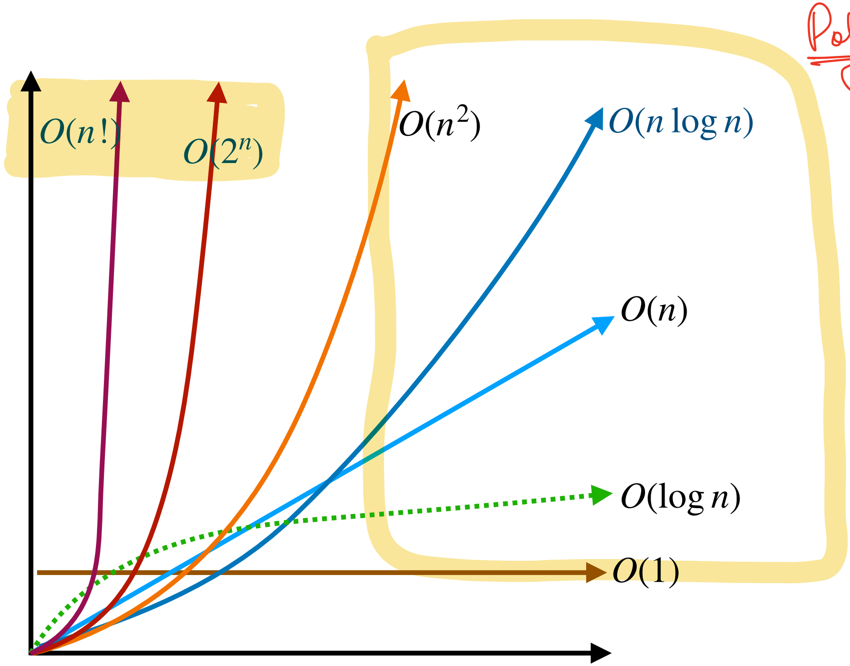
* No of bits needed to express F_n will be $\Theta(n)$

$$T(n) = n \times \underbrace{\text{Time-to-add-two-large-numbers}}_{O(n)}$$

$$T(n) = O(n^2)$$

Plots of different time complexities

Exp



Poly time

$O(n^d)$

$d \geq 0$

Merge Sort: $O(n \log n) = O(n^{1.0000001})$

Homework: Prove that $n \log n = O(n^{1+\epsilon})$, for each constant $\epsilon > 0$

Challenge Problem

– $O(n^2)$ ← easy

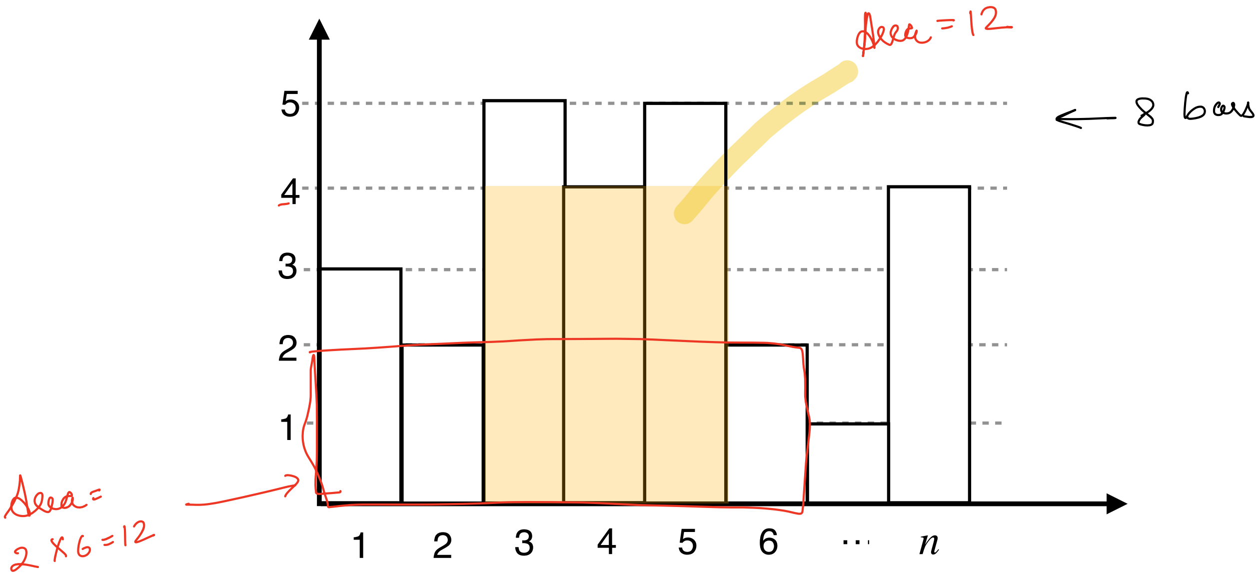
– subquadratic? $O(n^{2-\varepsilon})$

– $O(n \log n)$?

– $O(n)$?

Given : A histogram consisting of n bars of unit length.

Find : The axis-parallel rectangle of maximum area which is covered by the histogram.



What is the best possible time complexity?