COL 351: Analysis and Design of Algorithms

Tuesday, 10 August 2021

Grading Policy

For Audit

$\geq 35\%$ each

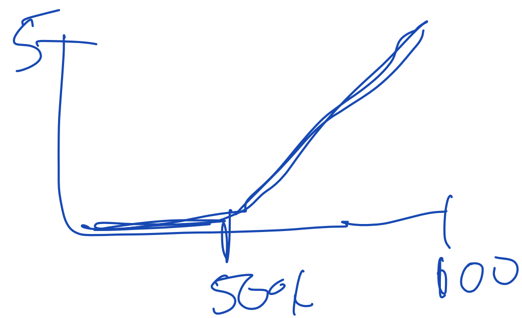
1. Quizzes - 15%
   (surprised / announced)

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

3. Exams - 30% + 30%

4. Attendance - 5%

Additional marks (Interaction)



Academic Honesty

Cheating or allowing anyone to copy in quizzes, exams, or assignments would lead to strict disciplinary action, like fetching **minus 25%** in course total.

Today's Lecture

1. Asymptotic Bounds (O, Ω, Θ)

2. Examples of Time complexity

3. Computing n^{th} Fibonacci Number efficiently

4. A tour over algorithmic problems to be studied in the course

"O" – Abhishek K.

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

$$f(n) \leq c\, g(n)$$

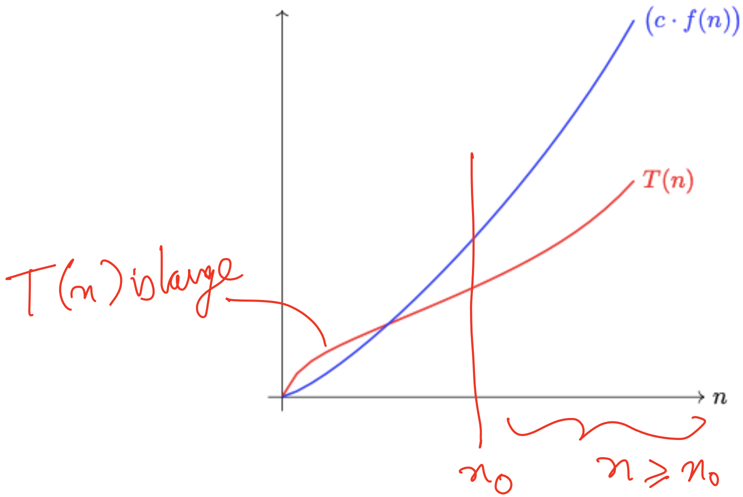
$\forall\, n \geq \underline{n_0}$

Asymptotic Bound (Big O notation)

$T(n)$ represents the number of steps taken by an algorithm on an input of size n.

Def: For any non-negative functions $T(n)$ and $f(n)$, we say $T(n) = O(f(n))$ if for large n (i.e. $n \geq n_0$ for some n_0), $T(n)$ is at most constant times $f(n)$.

$$\exists\ c, n_0 > 0 \text{ satisfying } T(n) \leqslant c\ f(n)\ ,\ \forall n \geq n_0\ .$$

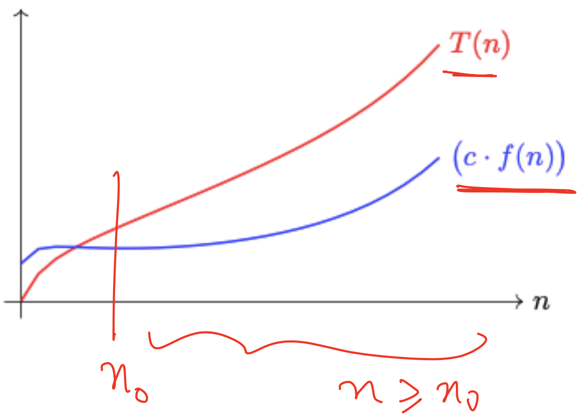


Asymptotic Bound (Ω notation)

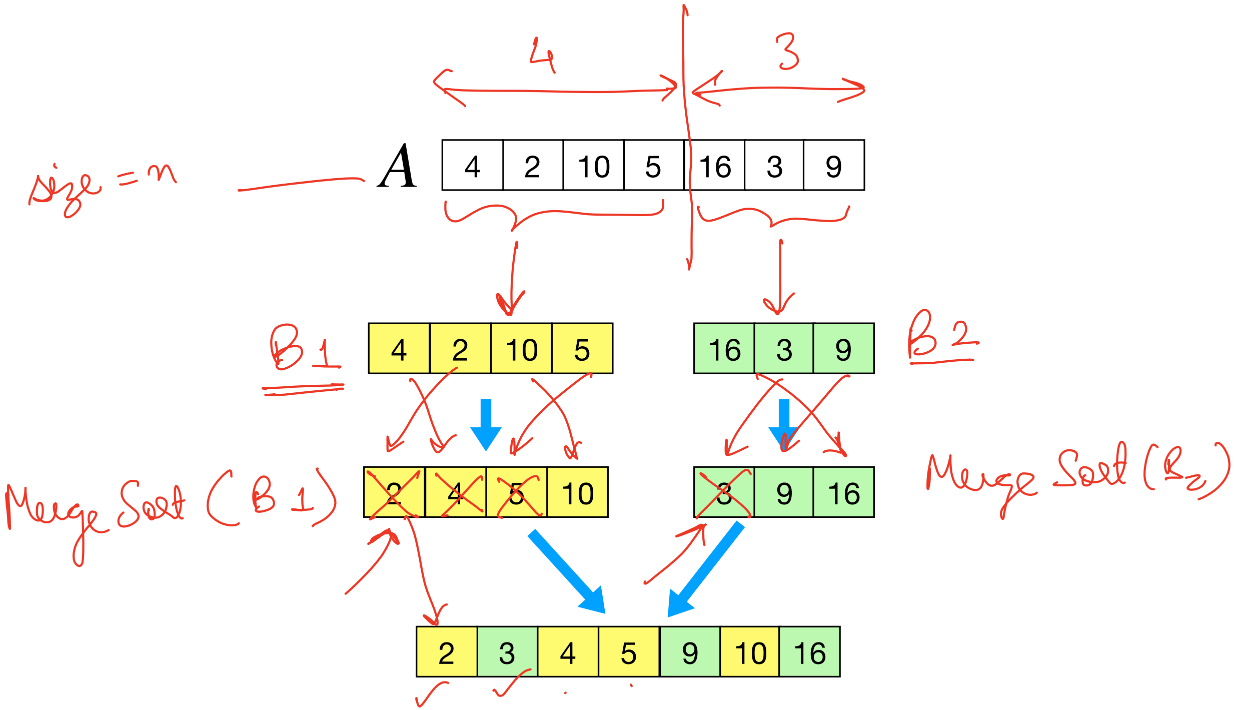
$T(n)$ represents the number of steps taken by an algorithm on an input of size n.

Def: For any non-negative functions $T(n)$ and $f(n)$, we say $T(n) = \Omega(f(n))$ if for large n (i.e. $n \geq n_0$ for some n_0), $T(n)$ is <u>at least</u> constant times $f(n)$.

$$\exists\ c, n_0 > 0 \text{ satisfying } T(n) \geqslant c\ f(n)\ ,\ \forall n \geq n_0\ .$$



Example: Merge Sort



Ravi Teja

$$\boxed{T(n) \leq 2T\left(\frac{n}{2}\right) + O(n)}$$

$O(n \log n)$

Kuldeep.

Example: Merge Sort

MergeSort(A)

Let $n = length(A)$;

If $n = 1$ then Return;

Store in B_1 the sub-array $A\left[0, \frac{n}{2}\right]$; $n/2$

Store in B_2 the sub-array $A\left[\frac{n}{2} + 1, n-1\right]$; $n/2$

MergeSort(B_1);

MergeSort(B_2);

Set x, y, pos= 0;

While $x < length(B_1)$ or $y < length(B_2)$

If $(B_1[x] \leqslant B_2[y]$ and $x < length(B_1))$ then

Set $A[pos] = B_1[x]$, and increment pos and x by 1;

Else

Set $A[pos] = B_2[x]$, and increment pos and y by 1;

Example: Merge Sort

Merge Sort (B_1)

$\nparallel (B_2)$

Let $T(n)$ be the number of steps taken by the algorithm. Then, $\boxed{T(n) \leq 2\,T\left(\frac{n}{2}\right) + 4n}$

cn

$T(n) = O(n \log n)$

$$T(n) \leq 2\,T\left(\frac{n}{2}\right) + cn$$

$$\leq 2\left[2T\left(\frac{n}{4}\right) + c\,\frac{n}{2}\right] + cn$$

$$= 4\,T\left(\frac{n}{4}\right) + 2cn$$

$$\vdots$$

$$= \boxed{2^k\,T\left(\frac{n}{2^k}\right)} + cn(k) = O(n \log n)$$

$k = \log(n)$

Example: Merge Sort (unequal split) $\left(\frac{n}{5}, \frac{4n}{5}\right)$

$\boxed{(10, n-10)}$

Let $T(n)$ be the number of steps taken by the algorithm. Then, $\cancel{T(n) \leq 2\,T\left(\frac{n}{2}\right) + 4n}$

A

B1 $\boxed{n/3}$ B2 $\boxed{2n/3}$

MergeSort(B_1) MergeSort(B_2)

Merge the two arrays

Sorted array A

$\Theta(n)$ $\Theta(n)$

$$\boxed{T(n) \leq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn}$$

Satyam – $O(n \log n)$

H. W. – Prove this bound

(Tutorial next week)

H. W. – $(\sqrt{n}), \; (n-\sqrt{n})$

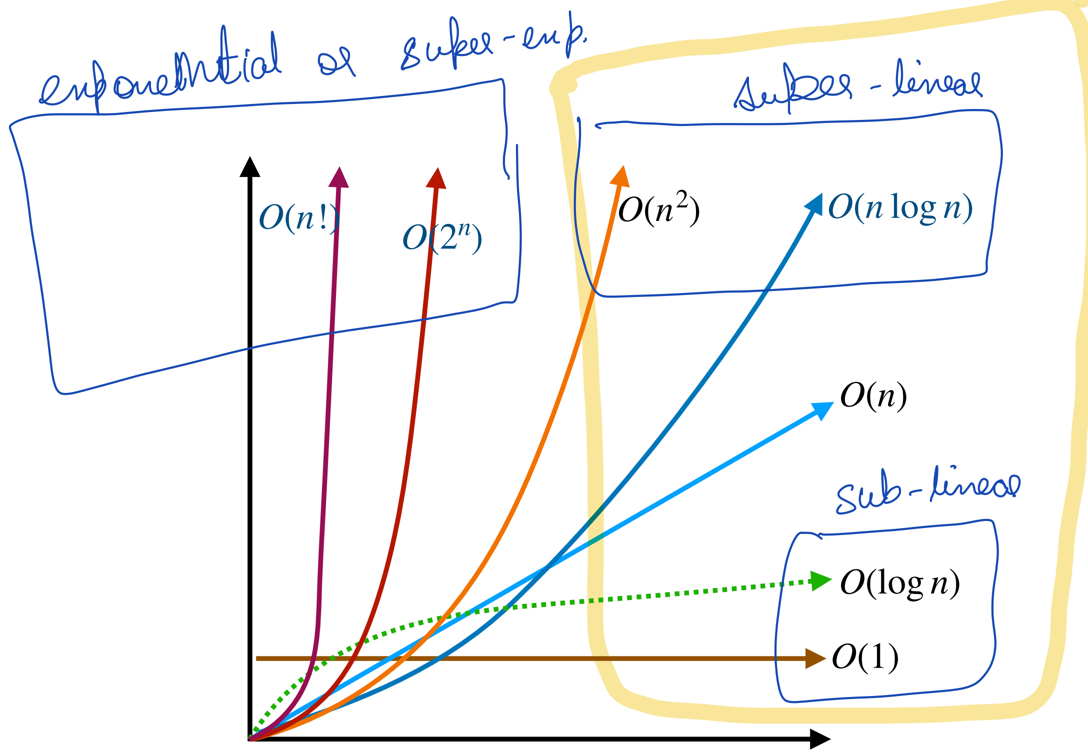
Time Complexity

$O(n^2)$?

$O(n^{1.5})$?

log n

Plots of different time complexities



Poly-time complexity

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

for some $d > 0$

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

$T(n) = O(n^3)$

Time Merge Sort $= O(n \log n)$

$n \log n = O(n^{1.000001})$

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

Asymptotic Bound (Θ notation)

O Big-Oh

Ω Omega

$T(n)$ represents the number of steps taken by an algorithm on an input of size n.

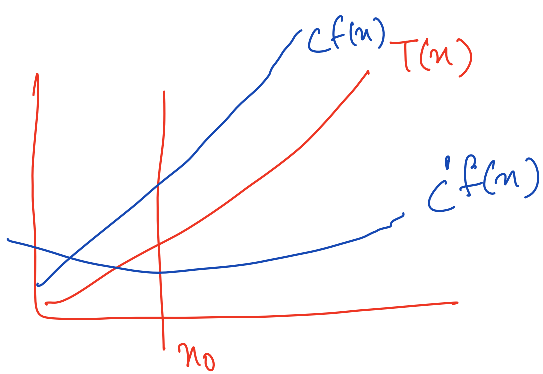
Def: For any non-negative functions $T(n)$ and $f(n)$, we say $T(n) = \Theta(f(n))$ if

(i) $T(n) = O(f(n))$, and

(ii) $T(n) = \Omega(f(n))$.

$\exists\, c', c$

$$c' f(n) \le T(n) \le c f(n), \quad \forall\, n \ge n_0$$



Example:

Sum(A)

Consider the problem:

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

$n = \text{odd}$

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

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

Ques. Is $T(n) = O(n)$?

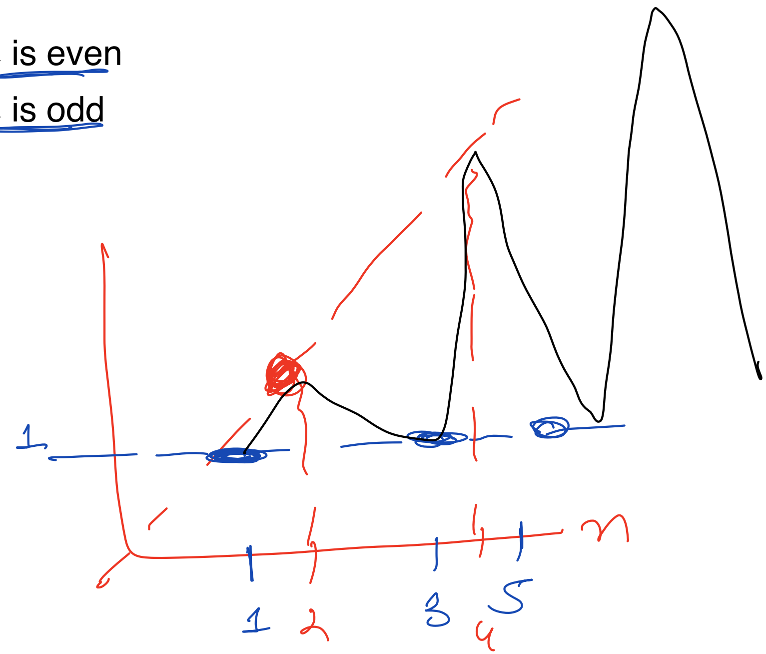
Ques. Is $T(n) = \Omega(n)$?

Proof

Infinitely many values

$T(n) = 1$

$\boxed{\text{NOT True} \mid T(n) \geq n}$

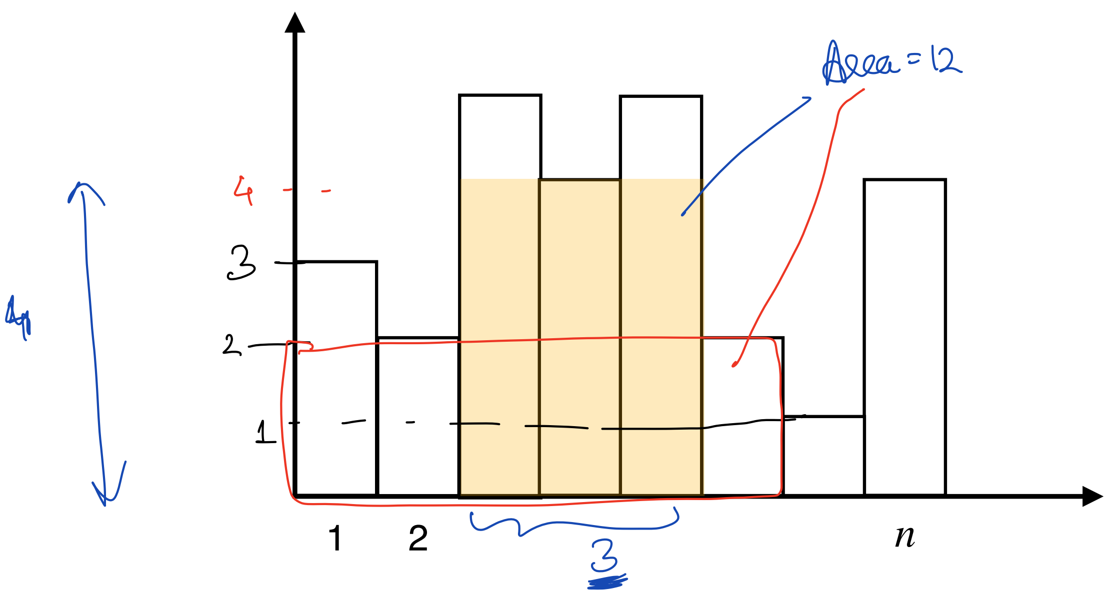


Challenge Problem

h(1) | h(2) | h(n)
3 2 4

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

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



$O(n^2)$

OPTIMIZE Time

Best time

What is the best possible time complexity?

Fibonacci Sequence

$F(0)=0$, $F(1)$, $F(n) = F(n-1) + F(n-2)$

Ravi – $\boxed{T(n) = Fib(n)}$

$0, 1, 1, 2, 3, 5, 8 \ldots$ # n^{th} value

$\exp\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$

Algo 1

> 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)

Fib(n-1)

Fib(n-2)

Fib(n-3)

Two calls

Algo 2

> Allocate an array A of size $n + 1$;
>
> Set $A[0] = 0$ and $A[1] = 1$;
>
> For ($i = 2$ to n) do
>
> $\quad A[i] = A[i-1] + A[i-2]$;
>
> Return $A[n]$;

Fibonacci-New(n)

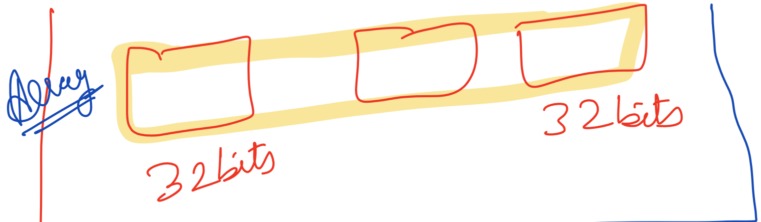
O(n) steps

Better Algo (Ishika)

Ques. What is time-complexity of above algorithms, and which is more efficient?

Word

64-bits



$T(n) \neq O(\phi(n))$

$$\boxed{T(n) = O(n \cdot \phi(n))}$$

Libraries

$n >>>> 64$

Very small numbers

$\log n \le 64$ bits

Assumption

O(1) ops

Fib(n)

x = Fib($n-1$)

y = Fib($n-2$)

Return $\underline{x+y}$.

No of iterations = Fib(n) = c^n, for some c

Each add^n = $\underline{O(n)}$ bits

$$O(n * c^n) = O(c^{2n})$$

Remains exponential.

H.W. $\phi(n) \geq (\sqrt{2})^n$

Tutorial - $\phi(n)$

$$\left(\frac{\sqrt{5}+1}{2}\right)^n + \left(\frac{\sqrt{5}-1}{2}\right)^n$$

$$\phi(n) = c^n$$

$k \approx 2^n$

$\log(\phi(n)) = O(n)$

No of bits : $\log(k) = n$

$T(n) = O(n)$

Array A

$A[0] = 0 \quad A[1] = 1$

$\boxed{\text{for loop } i = 2 \text{ to } n}$

$A[i] = A[i-1] + A[i-2]$

O(n) time

$T(n) = O(n * \underbrace{\text{no. of bits}}_{\Theta(n)})$

$Fib(n) = Fib(n-1) + Fib(n-2)$

H.W.: $(\sqrt{2})^n \leq Fib(n) \leq 3^n$

Number of bits $= \log(Fib(n)) = \Theta(n)$

Algorithm Paradigm

1. Divide and Conquer (Example?) Merge-Sort

2. Greedy Strategy

3. Dynamic Programming (Example?)

There are several problems whose solution are based on one of the above paradigms.

Miscellaneous

1. Depth-First-Search Trees, Strong connectivity
2. Maximum Flows
3. String Matching

Towards the end of Semester

1. NP-completeness — exponential / poly?

2. Polynomial time reductions

Exercises

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