Analysis & Design of Algorithms

Date / /
Page
STUDY BUDDIES

* Why ADA??

Define Algo M.

→ Word Algorithm was named for 9th Century Persian Mathematician al-Khowarizmi.

→ Informally, an algorithm is any well defined computational procedure that takes set of values as input and produces set of values as output.

→ Formal: A set of rules for carrying out some calculation either by hand or on a machine.

→ It must not involve any subjective or intutive decisions.

* why do we study analysis of algorithm??

→ why do you write a program?

↳ To get solution by writting it on computer

→ If you know all languages (Programming) available in world

→ Also, if you write down an effic programs to find solution in all those language

→ Do you think, that your work or method of writting program will produce efficient program writting.

↳ Ans. is No.

→ So, for writing efficient programs, you need to learn basic methods that are already defined and use them to write efficient algorithms.

Example.. Amazon

↳ Search - There is a filter option Price: Low to high
Price: High to low.
Avg. Customer Review
Newest Arrivals.

Date __/__/__
Page
STUDY BUDDIES

→ Now if you select price: low to high it will sort all products in ascending order in accordance with price

→ Now, if if I tell you to write down program. N - number of products → their price from low to high.

→ For writing it down, you will definately refer which sorting algorithms are available which are designed by mathematician or scientist.

→ So, you can choose the best efficient method from available but if you write your own program, it is possible, you may not end in writing down efficient algorithm

⟹ So, that's why we need to study different algorithms and their analysis. (which gives you or provides you with best soln).

→ So, we are going to study various algorithms, their different ways of producing output and analysis of it in a very easy manner.

Date __/__/__
Page _____
STUDY BUDDIES

* Linear Search Algorithm

Algo:-

```
linear_search (A, n, v)
{
    for i ← 1 to n do
        if A[i] = v then
            return i
    return NIL
}
```

* Insertion Sort Algorithm

⇒ Working of Insertion Sort

A						
23	~~7~~	5	5	5	3	3
7	23	7	7	7	5	5
5	5	23	11	11	7	7
11	11	11	23	19	11	8
19	19	19	19	23	19	11
3	3	3	3	3	23	19
18	18	18	8	8	8	23

→ No change
Sorting is completely done.

⇒ Algorithm Insertion Sort-

```
InsertionSort(A, n)
① {
   for i ← 2 to n do ——— n                          } ①
   ② {
       key ← A[i]                  ——— (n-1)
       j ← i-1                     ——— (n-1)
       while (j > 0 and A[j] > key) do ——— Σ_{i=2}^{n} Σ_{j=1}^{(n-1)} (1)
       ③ {
           A[j+1] ← A[j]
       ③ } j ← j-1
   A[j+1] ← key ——— (n-1)                           } ②
}
```

Date __/__/__
Page _____
STUDY BUDDIES

$$\Rightarrow C_1 \cdot n + C_2(n-1) + C_3(n-1) + C_4 \sum_{j=2}^{n} t_j + C_5 \sum_{j=2}^{n} (t_j - 1) + C_6 \sum_{j=2}^{n} (t_j - 1) + C_7 (n-1)$$

Best Case Provides lower bound,

→ It occurs when data is already sorted.

→ In Best, Case, while loop will never satisfies and hence

$t_j = 1$

$$\therefore T(n) = C_1 \cdot n + C_2(n-1) + C_3(n-1) + C_4 \sum_{j=2}^{n} 1 + C_5 \sum_{j=2}^{n} (1-1) + C_6 \sum_{j=2}^{n} (1-1) + C_7 (n-1)$$

$\sum_{j=2}^{n} 1 = 1 + 1 + 1 + 1 + \dots + 1$ (n-1) times $= (n-1)$

$= C_1 \cdot n + C_2 \cdot n - C_2 + C_3 \cdot n - C_3 + C_4 \cdot n - C_4 + C_7 n - C_7$

$= (C_1 + C_2 + C_3 + C_4 + C_7) n - C_3 + C_4 + C_7$

$= O(n)$

Worst Case: $t_j = j$ in this case.

$\sum_{j=2}^{n} j = 2 + 3 + 4 + \dots + n$

$= (1 + 2 + 3 + 4 + \dots + n) - 1$

$= \sum n - 1$

$= \frac{n(n+1)}{2} - 1$

$= O(n^2)$

$\sum_{j=2}^{(n+1)} (j-1) = (2 + 3 + 4 + \dots + (n+1) - 1) + n$

$= (1 + 2 + 3 + 4 + \dots + n)$

$= \frac{n(n+1)}{2} - 1 = \frac{n(n-1)}{2}$

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

Date __/__/__
Page _____
STUDY BUDDIES

* Selection Sort Algorithm :-

[A]

1	5	2	2	2	2	2	2	2	2
2	11	11	3	3	3	3	3	3	3
3	9	9	9	5	5	5	5	5	5
4	3	3	11	11	9	9	9	9	9
5	20	20	20	20	20	11	11	11	11
6	31	31	31	31	31	31	17	17	17
7	2	5	5	9	11	20	20	20	20
8	17	17	17	17	17	17	31	31	31

No change. Sorted

NOTE:- It will exchange values with the minimum value from an array.

→ No extra exchanges occur : Minimum exchanges.

```
⇒ SelectionSort(A,n)
{
  for i ← 1 to (n-1) do
  {
    min ← A[i]
    loc ← i
    for j ← (i+1) to n do
    {
      if A[j] < min then
      {
        min ← A[j]
        loc ← j
      }
    }
  if loc ≠ i then
  {
    A[loc] = A[i]
  } A[i] = min
} } }
```

Date __/__/__
Page _____
STUDY BUDDIES

OR

```
SelectionSort(A, n)
{
    for i ← 1 to n - 1 do
    {
        minindex = i
        minvalue = j
        for(j = i+1 to n) do
        {
            if A[j] < minvalue
            {
                minval ← A[j]
                minindex ← j
            }
        temp ← A[minindex]
        A[minindex] ← A[i]
        A[i] ← temp
        }
}
```

$T(n) = T(n-1) + n.$

$\therefore \Omega(n^2), \Theta(n^2), O(n^2)$

because it also cannot detect sorted elements in an array.

Date __/__/__
Page _____
STUDY BUDDIES

i = 1 * Bubble Sort Algorithm

J-loop	1	2	3	4	5	6 - (n-i) = 6
1	12	5	5	5	5	5
2	5	12	12	12	12	12
3	21	21	21	7	7	7
4	7	7	7	21	21	21
5	29	29	29	29	13	13
6	13	13	13	13	29	11
7	11	11	11	11	11	29

i = 2

j-loop	1	2	3	4	5	(n-i) = 5
1	5	5	5	5	5	5
2	12	7	7	7	7	7
3	7	12	12	12	12	12
4	21	21	21	13	13	13
5	13	13	13	21	11	11
6	11	11	11	11	21	21
7	29	29	29	29	29	29

and so on.

Date __/__/__
Page
STUDY BUDDIES

⟹ Algo. BubbleSort(A, n)

```
{
  for i ← 1 to n-1 do
  flag ← 0
  {
    for j ← 1 to (n-i) do
    {
      if A[j] > A[j+1] then
      {
        temp ← A[j]
        A[j] ← A[j+1]
        A[j+1] ← temp
        flag ← 1
      }
      if flag = 0 then
        break
    }
  }
```

10	30	40	20

∴ (1) Best Case without Flag :-

1 2 3 4

Comp / Swap

1

∴ outer loop (n-1) comparison
Inner loop (n-1) Comparison

∴ $T(n) = n + (n-1)$

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

$T(n-2) = (n-2) + (n-3)$

∴ $T(n) = n + (n-1) + (n-2) + (n-3) + \ldots\ldots 3 + 2 + 1$

Date __/__/__
Page _____
STUDY BUDDIES

$\therefore T(n) = \frac{n(n+1)}{2}$

$= \left(\frac{n^2}{2}\right) = \Omega(n^2)$

Best Case with Flag

① $T(n) = 1 + T(n-1)$

$= \Omega(n)$ — with Flag

② Worst-Case

$O(n^2)$ as well without Flag

$O(n^2)$ With Flag

③ Average Case

$O(n^2)$ - With Flag

$O(n^2)$ - Without Flag

Date __/__/__
Page _____
STUDY BUDDIES

Asymptotic Notations

* Efficiency of Algorithms:

⇒ Instance:-
Input needed to compute a solution to the problem satisfying constraints imposed in the problem statement.

⇒ Size of Instance:
Number of bits needed to represent the instance on a computer.

⇒ Analyze efficiency of algorithm:

We analyze efficiency of algo. to decide which of the several algo. is preferable.
∴ There are two approaches to these:

(i) A priori (theoritical approach):-
Determining mathematically the quantity of resources needed by each algo. as a function of the size of instance considered. The resources are computing time and storage space.

(ii) A posteriori (Pratical) approach: (empirical)
Choosing an algorithm, writing its program & trying it on different instances with help of a computer.

⇒ Goal of Analysis of algorithms:-
Compare in terms of running time but also in terms of other factors (eg memory requirements, programmers effort) etc

⇒ depends on Input size

⟹ Types of analyses:

→ Best Case Analysis (Time complexity)
→ Worst Case Analysis (" " ")
→ Average Case Analysis (" " ")

Date __/__/__
Page _____
STUDY BUDDIES

⟹ Asymptotic Notations:

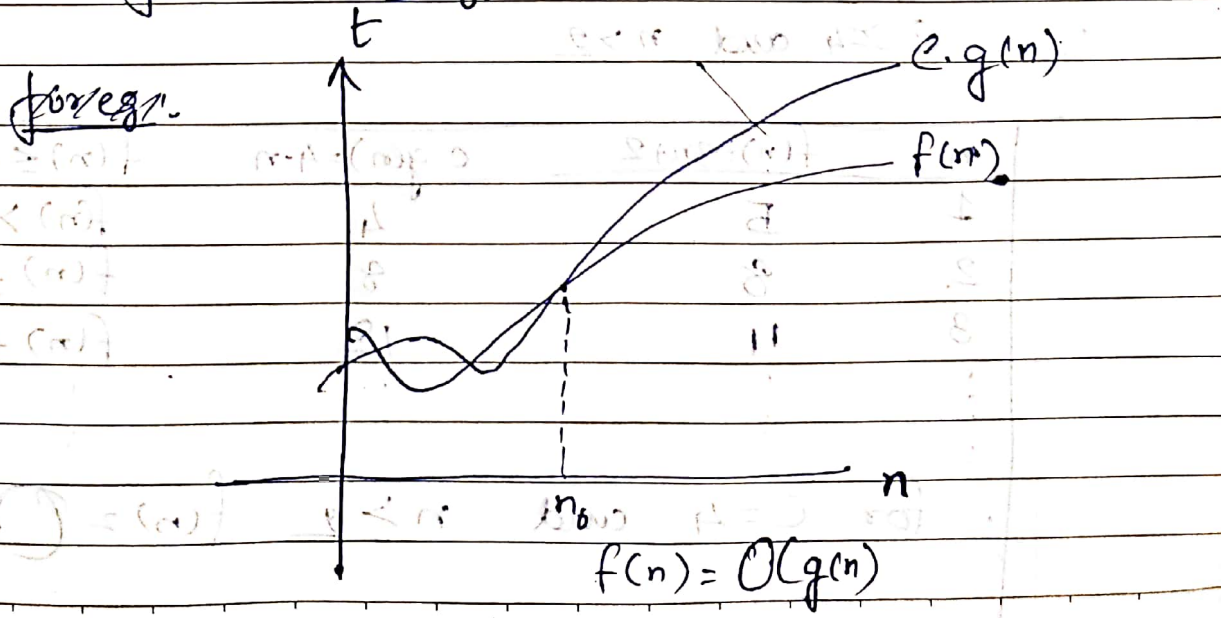
A priori analysis ignores all the machine and programming language dependent factors and considers only the frequency of execution of statements. For this kind of analysis, we use several mathematical asymptotic notations. These notations are called asymptotic bcoz. we deal with sufficiently large values of input size.

(1) Big oh (O):

is used to define the upperbound of an algorithm in terms of Time Complexity.

→ It gives maximum time required by an algorithm for all inputs values.
→ Bigoh (O) describes the worst case of an algorithm time complexity.

Defn: $f(n) = O(g(n))$ iff $\exists c \in R^+$ and a positive integer constant n_0 such that $|f(n)| \leq c|g(n)|$ for all $n \geq n_0$.



for eg:-

Date __/__/__
Page _____
STUDY BUDDIES

Example 1: $f(n) = 2n+2$, $g(n) = n^2$. Is $f(n) = O(g(n^2))$?

$\therefore \quad 2n+2 \leq c \cdot n^2$

$c = 1$

$2n+2 \leq n^2$

n	$f(n) = 2n+2$	$c \cdot g(n) = n^2$	$f(n) \leq c \cdot g(n)$
1	4	1	$f(n) > g(n)$
2	6	4	$f(n) > g(n)$
3	8	9	$f(n) \leq g(n)$
4	10	16	$f(n) < g(n)$
5	12	25	$f(n) < g(n)$

$\therefore$ for $n > 2$, and $c = 1$, we can have $f(n) < g(n)$

$\therefore f(n) = O(g(n^2))$

Example 2: Consider the following $f(n)$ and $g(n)$

$f(n) = 3n+2$

$g(n) = n$

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

$\therefore 3n+2 \leq c \cdot n$

$\therefore \quad c \geq 4$ and $n \geq 2$

n	$f(n) = 3n+2$	$c \cdot g(n) = 4 \cdot n$	$f(n) \leq c \cdot g(n)$
1	5	4	$f(n) > c \cdot g(n)$
2	8	8	$f(n) \leq g(n)$
3	11	12	$f(n) \leq g(n)$
⋮	⋮	⋮	⋮

for $c = 4$ and $n \geq 2$ $f(n) = O(g(n))$

Date __/__/__
Page _____
STUDY BUDDIES

Example 3. Show that $30n+8$ is $O(n)$

$f(n) = 30n+8$

$g(n) = n$

$\therefore f(n) \leq c \cdot g(n)$

$30n+8 \leq c \cdot n$

$\therefore$ $c = 31$ and $n \geq 8$ — uppersbound

n	$f(n) = 30n+8$	$g(n) = c \cdot n$	$f(n) \leq c \cdot g(n)$
1	38	31	$f(n) > g(n)$
2	68	62	"
3	98	93	"
4	128	124	"
5.	158	155	"
6.	188	186	"
7.	218	217	"
8.	248	248	$f(n) \leq g(n)$
9.	278	279	" "
10.	308	310	" "

Date __/__/__
Page _____
STUDY BUDDIES

Example 4 $2on^2+2n+5 = O(?)$

$f(n) = 2on^2+2n+5$

$g(n) = n^2$

$\therefore f(n) <= C \cdot g(n)$

$2on^2+2n+5 \leq C \cdot n^2$

let $C = 21$

n	$2on^2+2n+5$	$21n^2$	$f(n) \leq C \cdot g(n)$
1	27	21	$f(n) > g(n)$
2	89	84	$f(n) > g(n)$
3	191	189	$f(n) > g(n)$
4	333	336	$f(n) < g(n)$
5	515	525	$f(n) < g(n)$
6	⋮	⋮	
7			

$\therefore$ for $C = 21$ & $n \geq 4$

we have $f(n) = O(g(n))$

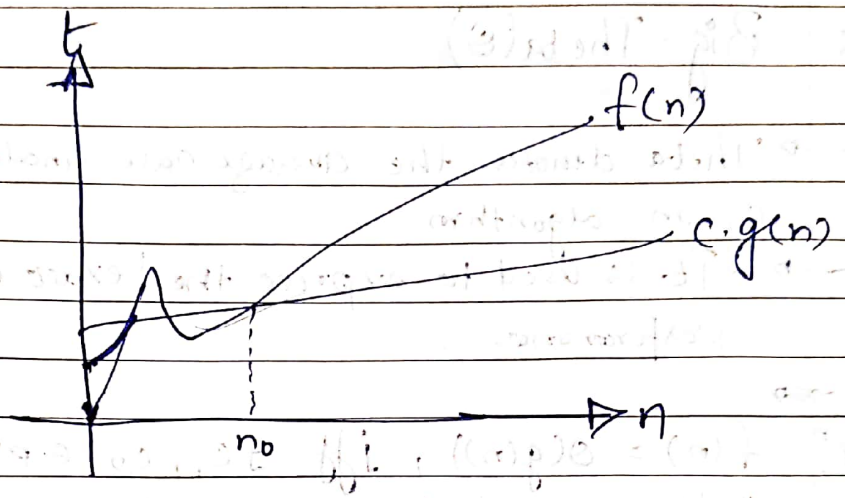
Date __/__/__
Page _____
STUDY BUDDIES

* Big-Omega (Ω)

→ It provides the lower bound of an algorithm in terms of time complexity.

→ It provides the least time required by an algorithm for all input values

→ (Ω) describes the Best case complexity of an algorithm.



Def: $f(n) = \Omega(g(n))$ iff $c \in R^+$ and a positive integer constant n_0 such that $|f(n)| \geq c \cdot g(n)$

(1) $f(n) = 2n^2+n$. Is $f(n) \in \Omega(g(n)^2)$?

$f(n) = 2n^2+n$.

$g(n) = n^2$

$f(n) \geq c \cdot g(n)$

$2n^2+2 \geq c \cdot n^2$

n	f(n)	c·g(n)	f(n) ≥ c·g(n)
1	3	2	f(n) ≥ c·g(n)
2.	10	8	f(n) ≥ c·g(n)
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.

Date __/__/__
Page
STUDY BUDDIES

Example(2) $f(n) = 5n^2$. Is $f(n) \Omega(g(n))$?

$5n^2 \geq c \cdot gn$.

$\therefore c = 1$ and $n \geq 1$

we can obtain $f(n) = \Omega(g(n))$

* Big - Theta(Θ)

→ Theta denotes the average case analysis or time complexity of an algorithm

→ It is used to express the 'exact order' of algo's performance.

Defn $f(n) = \Theta(g(n))$, iff $\exists c_1, c_2 \in R^+$ and a positive integer constant n_0 such that

$c_1 |g(n)| \leq |f(n)| \leq c_2 |g(n)|$ for all all $n \geq n_0$

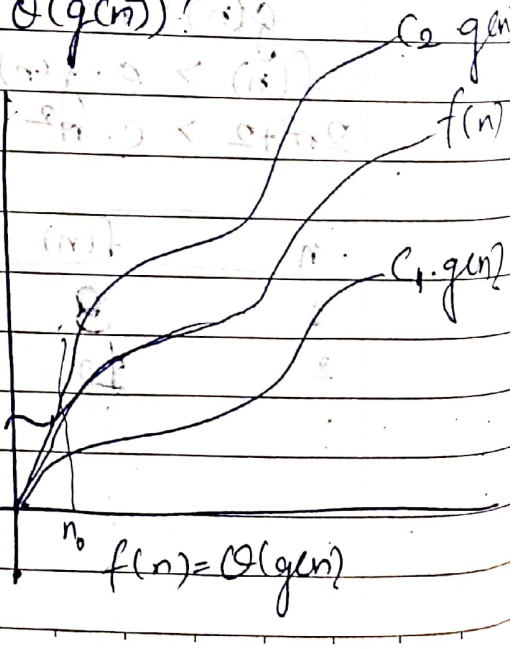
if $f(n) = \Theta(g(n))$, then $g(n)$ is both an upper and lower bound on $f(n)$.

Example: $f(n) = 2n^2 + n$, is $f(n) = \Theta(g(n^2))$?

$c_1 \cdot n^2 \leq 2n^2 + 2 \leq c_2 \cdot n^2$

$c_1 = 2$ and $c_2 = 3$

$2n^2 \leq 2n^2 + n \leq 3n^2$



Date __/__/__
Page _____
STUDY BUDDIES

* Order of Growth of Algorithm

$1 < \log n < \text{Sqrt}(n) < n < n \log n < n^2 < n^3 < 2^n < n!$

- 1 — Constant
- $\log n$ — Logarithmic
- n — Linear
- $n \log n$
- n^2 — Quadratic
- n^3 — Cubic
- 2^n — Exponential
- $n!$ — Factorial

(1) $2^n, n \log n, n^2, 1, n, \log n, n!, n^3$

→ $1, n, \log n, n \log n, n^2, n^3, 2^n, n!$

(2) $n \log n, n + n^2 + n^3, 2^n, \text{Sqrt}(n)$

→ ~~$n \log n$~~,

→ $\text{Sqrt}(n), n \log n, n + n^2 + n^3, 2n$

(3) $n^2, 2^n, n \log n, \log n, n^3$

→ $\log n, n \log n, n^2, n^3, n \log 2^n$

(4) $n!, 2^n, (n+1)!, 2^{2n}, n^n, n^{\log n}$

→ $n^{\log n}, 2^n, n!, (n+1)!, 2^{2n}, n^n$

Date __/__/__
Page _____
STUDY BUDDIES

Exclusive - very important.

* Prove that $T(n) = 1+2+3+\dots+n = \Theta(n^2)$

$\Rightarrow$ $T(n) = 1+2+3+\dots+n$

$= \sum n$

$= \frac{n(n+1)}{2}$

$= \frac{n^2+n}{2}$

$\therefore = \Theta\left(\max\left(\frac{n^2}{2}, \frac{n}{2}\right)\right) = \Theta\left(\frac{n^2}{2}\right) = \boxed{\Theta(n^2)}$

* Find upper bound of running time of cubic function.
$f(n) = 2n^3+4n+5$

$\therefore$ Since it is a cubic function. $g(n) = n^3$

$\therefore$

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

$\therefore 2n^3+4n+5 \leq 3n^3$

$\therefore c = 3$

n	$f(n) = 2n^3+4n+5$	$c \cdot g(n) = 3n^3$
1	11	3
2	29	24
3	71	81
4	149	192
⋮	⋮	⋮

$\therefore n \geq 3$ and $c = 3$, we have $f(n) = O(g(n))$

$\therefore f(n) = O(n^3)$

$g(n) = n^3$

Date __/__/__
Page _____
STUDY BUDDIES

* Find lower bound of running time of a constant function $f(n) = 23$.

$\Rightarrow 0 \leq c \cdot g(n) \leq f(n)$

$\Rightarrow 0 \leq c \cdot 1 \leq 23$

$0 \leq 23 \cdot 1 \leq 23$

$0 \leq 22 \cdot 1 \leq 23$

$0 \leq 12 \cdot 1 \leq 23$

$0 \leq 5 \cdot 1 \leq 23$

$\therefore c \leq 23$, so all such values of c are possible and $f(n)$ is constant, so it does not depend on problem-size n.

$\therefore n_0 = 1$

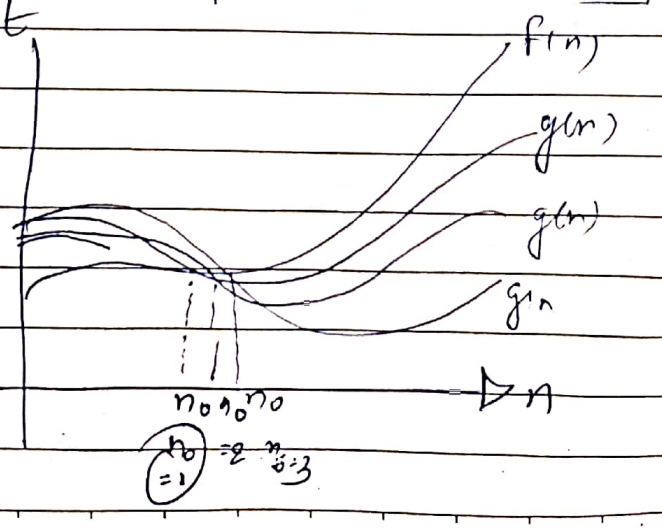
* Find the lower bound of running time of linear function $f(n) = 6n + 3$

$\Rightarrow 0 \leq c \cdot g(n) \leq f(n)$

$0 \leq 1 \cdot n \leq 6n + 3$

$0 \leq 6 \cdot n \leq 6n + 3$

$\therefore$ $c \leq 6$ and for $n \geq 1$ or $n > 1$



Date __/__/__
Page _____
STUDY BUDDIES

Tight Bound (Θ)

(1) Find the tight bound of running time of constant fn $f(n) = 23$.

$0 \le c_1 g(n) \le f(n) \le c_2 \cdot g(n)$

$\therefore 0 \le c_1 \cdot 1 \le 23 \le c_2 \cdot 1$

$\therefore 0 \le 23 \cdot 1 \le 23 < 24 \cdot 1$

$\therefore n_0 = 1$ and $c_1 \le 22$ and $c_2 \ge 24$

$f(n) = \Theta(g(n)) = \Theta(1)$ for $c_1 = 22, c_2 = 24, n_0 = 1$

$f(n) = \Theta(g(n)) = \Theta(1)$ for $c_1 = 23, c_2 = 25, n_0 = 1$

and so on.

$\therefore c_2 \le 22$ and $c_2 \ge 24$ and $n_0 = 1$

we have $f(n) = \Theta(g(n)) = \Theta(1)$

(2) Find tight Bound of running time of linear function $f(n) = 6n + 3$

$0 \le c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$

$0 \le c_1 \cdot n \le 6n + 3 \le c_2 \cdot n$

$0 \le 6 \cdot n \le 6n + 3 \le 9 \cdot n$

Date __/__/__
Page _____
STUDY BUDDIES

* Analysis of Algorithm

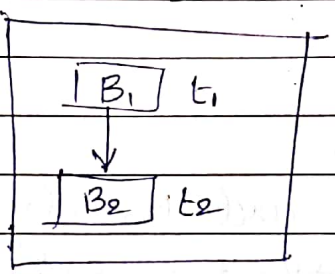
(1) Analyzing Control Statement

⇒ Sequential Execution

In sequential structure, statement or block of statement are appearing one after another.

→ Let say B_1 and B_2 are two block of instructions, which may contain single instruction, multiple instruction or set of complex instructions.

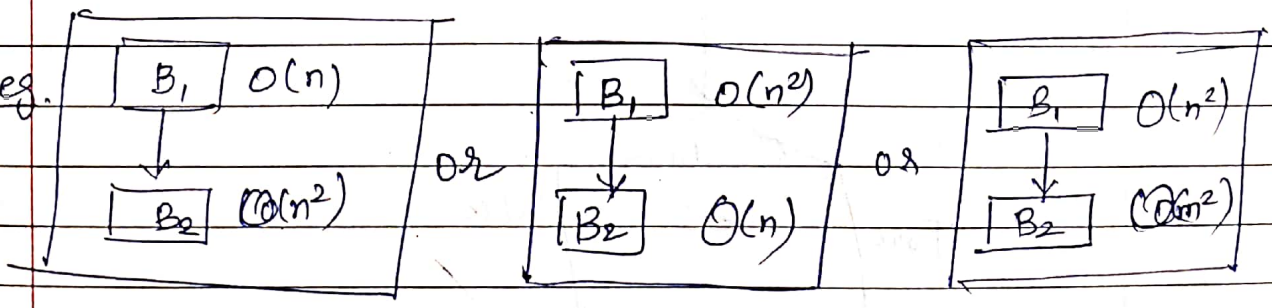
→ If they are executed in sequence i.e. first B_1 and then B_2 in which time taken by B_1 is t_1 and B_2 is t_2



∴ total complexity of entire program will be

$$T(n) = t_1 + t_2 = \max(t_1, t_2)$$

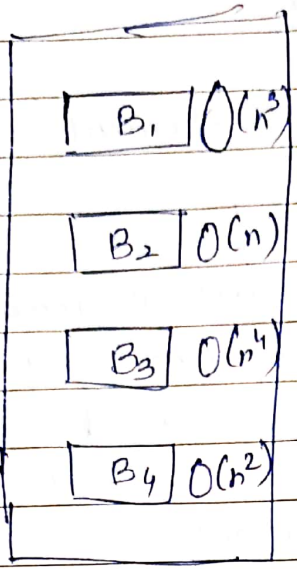
for eg.



$$\therefore T_n = \max\left(O(n), O(n^2)\right) \approx O(n^2)$$

Date __/__/__
Page _____
STUDY BUDDIES

eg 2

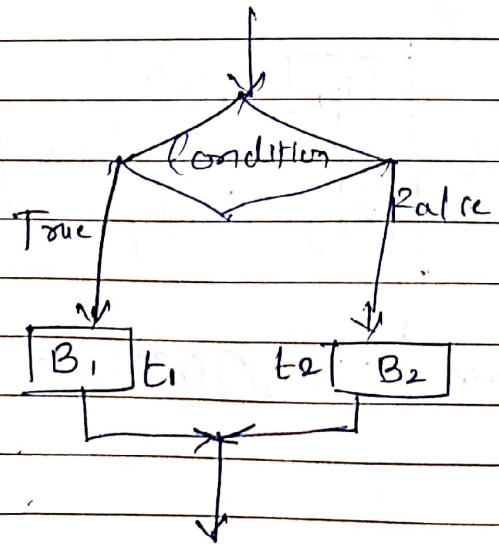


$\therefore T(n) = \max(O(n^3), O(n), O(n^4), O(n^2))$

$T_n = O(n^4)$

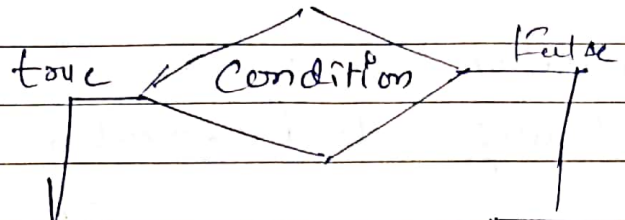
If-else

→ Execution of B_1 and B_2 depends on certain conditions. If condition is satisfied then B_1 block will be executed and if condition is false then B_2 block will be executed.



$\therefore T_n = \max(t_1, t_2)$

Date __/__/__
Page _____
STUDY BUDDIES



$\therefore$ $T(n) = \max(O(n), O(n^2)) = O(n^2)$

For Loop

Eg①
```
for(i=1 ; i<=n ; i++)
{
count = count + 1
}
```

$\therefore$ Cost of inner statement is $O(1)$, and for loop iterates n times

$$\therefore T(n) = \sum_{i=1}^{n} O(1)$$

$$= O(n)$$

Eg②
```
for(i=1 ; i<=n ; i++)          → Σ_{i=1}^{n}
{
  for(j=1 ; j<=n ; j++)        → Σ_{j=1}^{n}
  {
    count = count + 1          → O(1)
  }
}
```

$$T(n) = \sum_{i=1}^{n} \left[\sum_{j=1}^{n} O(1) \right] = \sum_{i=1}^{n} O(n)$$

$$= O(n^2)$$

Date __/__/__
Page _____
STUDY BUDDIES

eg. 3 Calculate computation time for the statement t_3 in the following code fragment

```
For i = 1 to n {                    ——— Σ_{i=1}^{n}
    For j = 1 to i {                ——— Σ_{j=1}^{i}
        c = c + 1 ..... t3          ——— O(i)
    }
}
```

$$\Rightarrow T_n = \sum_{i=1}^{n} \sum_{j=1}^{i} O(i)$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{i} O(i)$$

$$= \sum_{j=1}^{i} O(n)$$

$$= O(n^2)$$

eg. While loop

```
while(n > 0)
{
    n = n - 1
}
```

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

Date __/__/__
Page _____
STUDY BUDDIES

eg(2)

```
i = 1
while (i <= n)
{
    j = 1
    while (j <= n)
    {
        count = count + 1
        j = j + 1;
    }
    i = i + 1
}
```

$\sum_{i=1}^{n}$ — $\sum_{j=1}^{n}$ — $O(1)$

$\sum_{i=1}^{n}$ — $O(1)$

$\therefore T(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} O(1)$

$= O(n^2)$

Date __/__/__
Page _____
STUDY BUDDIES

✱ Recursion

1. Function which calls itself is called Recursion.
   → Basically Recurrences is an equation or inequality

eg Recursive algo. for fibonacci series is given as

```
Algorithm FIBONACCI(n)
    if n = 0 then
        return 0
    else if n = 1 then
        return 1
    else
        return FIBONACCI(n-1) + FIBONACCI(n-2)
end
```

⟹ Since time complexity of recursive program is defined in terms of recurrence equation.

→ Each recursive call to perform problem of size n creates two new subproblems, each of size (n-1) and (n-2) respectively, and then each call performs one addition.

∴ The recurrence equation is given by

Basically

$$T(n) = T(n-1) + T(n-2) + \Theta(1)$$

✱ Methods to Solve Recurrence equation are

(1) Substitution
(2) Master's Method
(3) Tree Method

Date __/__/__
Page _____
STUDY BUDDIES

(1) 1. Guess the solution
2. Use mathematical Induction to solve boundary conditions, and show that guess are correct

(1) $T(n) = 2T\left(\frac{n}{2}\right) + n$ ——— (1) $\quad \therefore T(n) = 1$

Replace $n \to n/2$

$\therefore T\left(\frac{n}{2}\right) = 2T(n/4) + n/2$ ——— (3)

put eq (3) in (1) we get

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

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

$= 2^2 T\left(\frac{n}{2^2}\right) + 2n$ ——— (4)

Replace equation (1) with (4) we get

$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4}$ ——— (5)

putting equation $T\left(\frac{n}{4}\right)$ (5) in (4) we get

$T(n) = 4T\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n$

$= 8T\left(\frac{n}{8}\right) + 3n$

$= 2^3 T\left(\frac{n}{2^3}\right) + 3n$ ——— (6)

Date __/__/__
Page _____
STUDY BUDDIES

So we can guess the General equation may be

$$T(n) = 2^i T\left(\frac{n}{2^i}\right) + in$$

$T(1) =$

let $\frac{n}{2^i} = 1$

$\Rightarrow n = 2^i$

$\Rightarrow \boxed{\log_2 n = i}$

$\therefore T(n) = 2^{\log_2 n} \left(\frac{n}{2^{\log_2 n}}\right) + n \log_2 n$

$T(n) = nT(1) + n\log_2 n$

$\therefore T(n) = n + n\log_2 n$

$\therefore \boxed{\text{Time Complexity} = O(n \log n)}$

Date __/__/__
Page _____
STUDY BUDDIES

(9) $T(n) = T(n-1) + n$, $T(1) = 1$

⇒ Replace n with n-1, we get

$T(n-1) = T(n-1-1) + (n-1)$

$= T(n-2) + (n-1)$ ——— (1)

putting equation 1 in Original

$T(n) = T(n-2) + (n-1) + n$ ——— (2)

Now Again n with (n-2) —in original eq.

$T(n-2) = T(n-3) + (n-2)$ ——— (3)

putting (3) in (2) we get

$T(n) = T(n-k) + (n-(k-1)) + (n-(k-2)) + \dots + n$

$T(n) = T(n-3) + (n-2) + (n-1) + n$ ——— (4)

$T(n) = T(n-k) + (n-k+1) + (n-k+2) + \dots + n$

In short the General equation is.

$= T(0) + 1 + 2 + 3 + 4 + 5 + 6$ let $n-k = 0$

$T(n) = 0 + 1 + 2 + 3 + 4 + \dots + n$

$= \frac{n(n+1)}{2}$

or let $n = k$

$T(n) = T(1) + 2 + 3 + 4 + \dots + n$

$= 1 + 2 + 3 + 4 + \dots + n$ — Sum of natural no.

$T(n) = \frac{n(n+1)}{2}$

$= \frac{n^2}{2} + \frac{n}{2}$

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

$n-(k-1)$
$= [n-(n-1)-1]$
$= n-n+1+1$
$= 2$

Date __/__/__
Page
STUDY BUDDIES

(1) $T(n) = T(n-1) + 1$

Example. Factorial using Recurrence

```
fact(n)
{
if n ≤ 1
  return 1          } C
else
  return n * fact(n-1);
}
```

$T(n) = \begin{cases} T(n-1) + C & n > 1 \\ 1 & n \le 1 \end{cases}$

$T(1) = 1$

f(5)

5 * fact(4)

4 * f(3)

3 * f(2)

2 * f(1)

1

C + A + T(n-1)

or

$T(n) = T(n-1) + 1 \quad n > 1$

$1 \quad n \le 1$

or

$T(n) = \begin{cases} 1 \\ T(n-1) + 1 \end{cases}$

$T(n) = 1$

Date __/__/__
Page _____
STUDY BUDDIES

$T(n) = T(n-1) + 1$ ——— (1)

Now substituting $n = n-1$ in (1) we get

$\therefore$ we get $T(n-1) = T(n-1-1) + 1$

$= T(n-2) + 1$ ——— (2)

Now Subs. (2) in (1) we get,

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

$= T(n-2) + 2$ ——— (3)

Now Subs. $n = (n-2)$ in (1) we get

$T(n-2) = T(n-3) + 1$ ——— (4)

Subs. (4) in (3) we get

$T(n) = (T(n-3)+2) + 2$

$= T(n-3) + 3$ ——— (5)

$\therefore$ The general equation is

$T(n) = T(n-k) + k$

$= T(0) + k$

$=$

put $k = n$

we get $T(n) = T(0) + n$

$T(n) = O(n)$

Or $n - k = 1$

$\therefore T(1) + n - 1$

$= 1 + n - 1$

$= O(n)$

Date __/__/__
Page
STUDY BUDDIES

(2) $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + \log n & \text{if } n > 1 \end{cases}$

$T(n) = T(n-1) + \log n$ ——— (1)

⟹ Subst. $T(n-1)$ in (1) we get

$T(n-1) = T(n-2) + \log(n-1)$ ——— (2)

Now substituting (2) in (1) we get,

$T(n) = T(n-2) + \log(n-1) + \log n$ ——— (3)

⟹ Subst. $T(n-2)$ in (1) we get,

$T(n-2) = T(n-3) + \log(n-2)$ ——— (4)

Substituting (4) in (3) we get,

$T(n) = T(n-3) + \log(n-2) + \log(n-1) + \log n.$

$T(n) = T(n-4) + \log(n-3) + \log(n-2) + \log(n-1) + \log n$

∴ Generalize eqn can be formed.

$T(n) = T(n-k) + \log(n-(k-1)) + \log(n-(k-2)) + \ldots + \log n.$

let $n-k = 1$

$\therefore k = n$

$\therefore T(n) = T(1) + \log 1 + \log 2 \ldots + \log n$

$= 1 + \log 1 + \log 2 + \ldots \log n$

$= 1 + \log(1 * 2 * \ldots * n)$ ($\because \log m + \log n = \log(mn)$)

$= 1 + \log n!$

Date __/__/__
Page _____
STUDY BUDDIES

Now $n! = n*(n-1)*(n-2)*(n-3)*\ldots\ldots(n-k)!$

$= n * n * \ldots$ (worst case: worst case)

$\therefore n! = n^n$

$\therefore T(n) = 1 + \log_2 n^n$

$T(n) = 1 + n \log n$

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

Date __/__/__
Page _____
STUDY BUDDIES

✱ Master's Method

Master's Method is quickly used to resolve any recurrences. if recurrence is of the form

$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$ ... $n \geq d$ and d is some constant

⇒ n = size of problem
a = Number of subproblems created in recursive eqn.
n/b = size of each subproblem
$f(n)$ = work done outside recursive call, This includes cost of division of problem and merging of sol^n.

⇒ Consider $f(n)$ is the polynomial of degree d.

1. $T(n) = \theta(n^d \cdot \log n)$, if $a = b^d$
2. $T(n) = \theta(n^{\log_b a})$, if $a > b^d$
3. $T(n) = \theta(n^d)$, if $a < b^d$.

Ex1 $T(n) = 4T\left(\frac{n}{2}\right) + n$

∴ Compare it with $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

Here $a = 4$, $b = 2$, and $f(n) = n$, having degree = 1

∴ Over Here $a > b^d$
$\because 4 > 2^1$

$\therefore T(n) = \theta(n^{\log_2 4})$

$= \theta(n^{\log_2 2^2})$

$\log_2 4 = \log_2 2^2$
$= 2 \log_2 2$
$= 2$

Date __/__/__
Page _____
STUDY BUDDIES

$\boxed{T(n) = \Theta(n^2)}$

(2) $T(n) = T\left(\frac{n}{2}\right) + 1$

Here comparing with $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

So, $a = 1$, $b = 2$ and $f(n) = n^0$

$\therefore a = b^d$ since $d = 0$

$\therefore 1 = 2^0$

$T(n) = \Theta(n^d \log n)$
$= \Theta(n^0 \log n)$
$= \Theta(\log n)$

(3) $T(n) = 9T\left(\frac{n}{3}\right) + n^3$

Here $a = 9$, $b = 3$, $f(n) = n^3$ and $d = 3$

$\therefore a < b^d$ as $9 = 3^3$ $\therefore 9 < 27$

$\therefore T(n) = \Theta(n^d)$
$\boxed{T(n) = \Theta(n^3)}$

Quick Sort (Partition Exchange Sort)

	0	1	2	3	4	5	6
A	10	15	1	2	9	16	11

Pivot = 10

⇒ All the elements less than Pivot is on left hand and others on Right hand Side

Partition 1	Pivot	Partition 2

Values < P. Values > Pivot

→ Note:- equal values can go (anywhere) either left or Right of Pivot.

→ Take the same fundamental of taking Pivot and Repeat the procedure

0	1	2	3	4	5	6
2	1	19	10	15	11	16

Pivot

Example

Array - a

0	1	2	3	4	5	6	7
7	6	10	6	9	2	1	15	7

lb ub

Start End

Pivot : 7 = a[0]

7	6	10	5	9	2	1	15	7

Start

Since Pivot ≥ 7, start++

7	6	10	5	9	2	1	15	7

Start

Since Pivot ≤ 6 start++

7	6	10	5	9	2	1	15	7

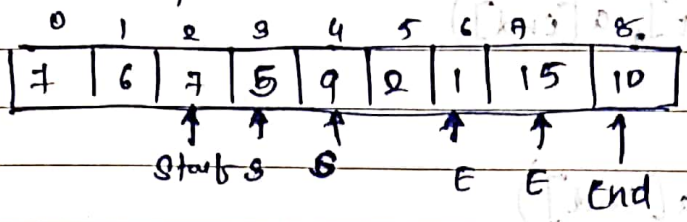
End

Not encountering End

Since Pivot not greater then end element

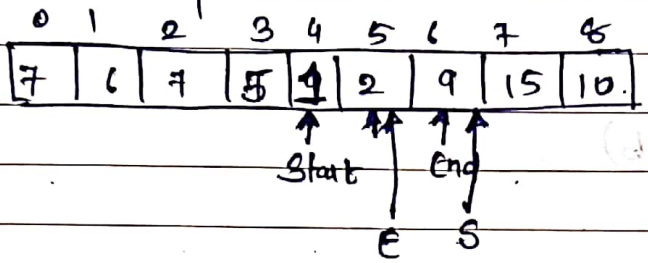
Date __/__/__
Page _____
STUDY BUDDIES

⇒ Now swap A[2] with A[8],



Repeat procedure of Start and End.

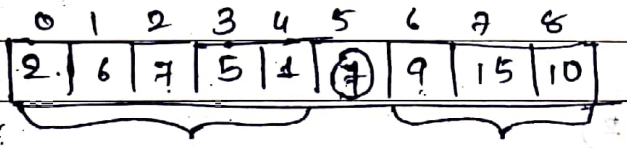
⇒ Now swap A[4] with A[6]



Repeat same fund.

Now we are not going to swap bcoz Start has crossed End.

Step.2 Now swap Pivot with "End" element



Now, you can check all the element ≤ Pivot = 7 are on left side and all elements > 7(Pivot) are on the right side.

```
.Partition(A, lb, ub)
{
   Pivot = a[lb]
   Start = lb;
   End = ub;
while(a[start] ≤ Pivot)        while(start < End)
   {                           {
    start++;
while(a[End] > Pivot)
   {
    End--;
   }
```

Date __/__/__
Page _____
STUDY BUDDIES

```
if (start < End)
  {
    Swap (a[start], a[End])
  }
}
Swap (a[lb], a[End])
Return End;
}

QuickSort (A, lb, ub)
{
  if (lb < ub)
  {
  loc = Partition (A, lb, ub)
    QuickSort (A, lb, loc-1)
    QuickSort (A, loc+1, ub)
    }
```

worst case = $O(n^2)$
Best case & Average Case = $O(n \log n)$

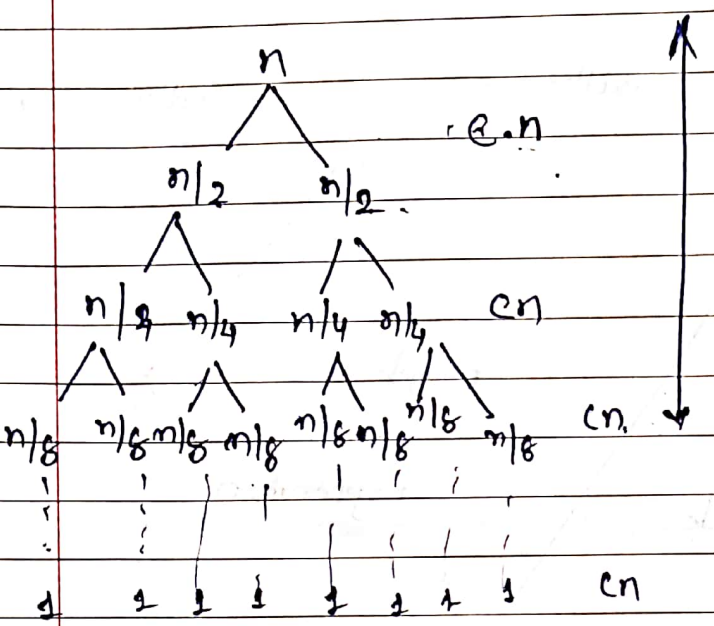
⇒ Complexity Analysis :-

Date __/__/__
Page
STUDY BUDDIES

* Recursive Tree Method :-

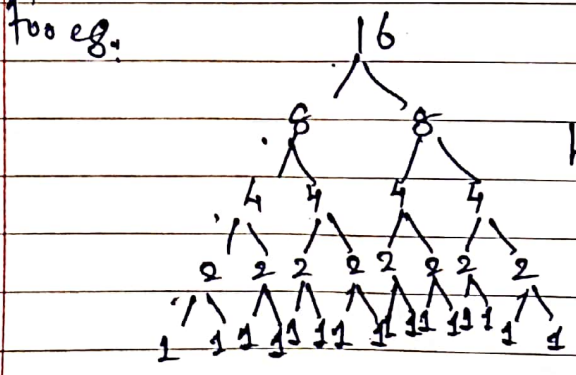
Ex 1 $T(n) = 2T\left(\frac{n}{2}\right) + cn$.

Note: Divide and conquer operation takes c·n time for dividing and conquering



Now, for eg.

⇒ Over here we have taken 3.cn, bcoz height of the tree is 3. if the height is n we need to find out what is the cost.

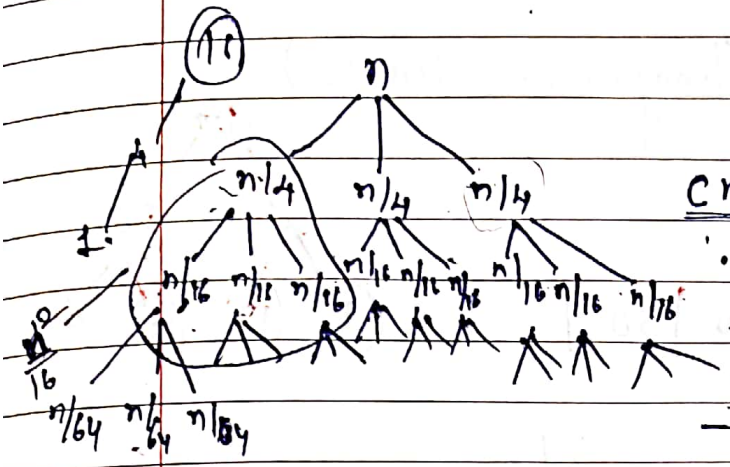


height = 4 = $\log_2 16 = 4$ = height

∴ Total Time Complexity = height * cn
= log n . cn
= n log n
= O(n log n)

Date __/__/__
Page _____
STUDY BUDDIES

Ex2 $T(n) = 3T\left(\frac{n}{4}\right) + cn^2$



cn^2

$\left(\frac{3}{16}\right)n^2$ for 3 nodes

$\left(\frac{n}{16}\right)^2$ for 1 node

$\left(\frac{n^2}{16}\right)$

$\rightarrow \left(\frac{n^2}{16}\right) = \frac{n^2}{16} = \left(\frac{n}{16}\right)^2$ for 1

$\therefore$ for single it is $\left(\frac{3}{16}\right)^2 n^2 = \frac{9}{16^2} n^2$

$\log_4 n$

$\left(\frac{n}{16}\right)^3 \left(\frac{3}{16}\right)^3 n^2$

$\therefore cn^2 + \left(\frac{3}{16}\right)cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \left(\frac{3}{16}\right)^3 cn^2 + \cdots$

$= cn^2\left[1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \left(\frac{3}{16}\right)^3 + \cdots\right]$

$= cn^2\left[1 + x + x^2 + x^3 + \cdots\right] \quad \because \frac{1}{1-x}$; if $x < 1$

$x = \frac{3}{16} < 1$

Geometric Progression formula.

$= cn^2\left(\frac{1}{1-\frac{3}{16}}\right)$

$= cn^2\left(\frac{1}{13/16}\right)$ $\boxed{= cn^2\left(\frac{16}{13}\right)}$ $\boxed{= O(n^2)}$

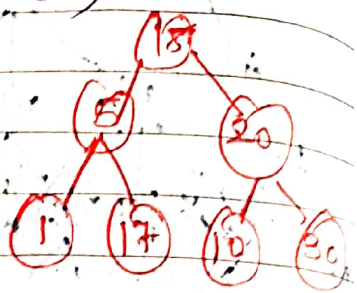
Neglecting Constants

Date __/__/__
Page
STUDY BUDDIES

* Shell Short :-

Example

Heap Sort (Jenny's lecture - Youtube)



15	5	20	1	17	10	30

$\lfloor n/2 \rfloor - 1$ to n

```
Algo MaxHeapSort(A, n)
{
    for i ← (n/2 - 1) to 0 do
    for R ← n-1 to 0 do {
        Heapify(A, n, i)
    }
    for i ← 0 to n-1 do
    {
        swap(A[i], A[0])
        Heapify(A, i, 0)
    }
}
```

```
Heapify(A, n, i)
{
    largest = i
    L = 2 * i + 1
    R = 2 * i + 2
    if L < n && a[L] > a[largest]
        largest ← L
    if R < n && a[R] > a[largest]
        largest ← R
    if (largest != i)
        swap(A[i], A[largest])
        Heapify(A, n, largest)
}
```

Analysis of algorithm

→ Since in order to create a Max Heap Binary tree, It takes a total time of O(log n)

→ Also, total swaps of (Root and last Node, it takes O(n) times

∴ total time = O(n log n)

Date __/__/__
Page _____
STUDY BUDDIES

* Insertion Sort (Complexity analysis)

(1) Best case

1 2 3 4 5 6

	Comp	Swap
1 0	0	0 (1)
	1	0
	1	0
	1	0
	1	0
n (n-1)	1	0
		0

$\therefore n + (1)$

$= O(n)$

(2) Worst Case

60 50 40 30 20 10

	Comp.	Swap
	0	0
	1	1
	2	2
	3	3
	4	4
	5	5
	(n-1)	(n-1)

$\therefore 1 - to\ n = \frac{n(n+1)}{2}$ $\frac{n(n+1)}{2}$

$= n^2 + n^2$

$= 2n^2$

$= O(n^2)$

Date __/__/__
Page
STUDY BUDDIES

~~B~~

Exponential Examples

– It uses divide and conquer approach to reduce scalar multiplications

$$\therefore a^n = \begin{cases} a & \text{if } n=1 \\ (a^{n/2})^2 & \text{if } n \text{ is even} \\ a \times a^{n-1} & \text{otherwise} \end{cases}$$

(1) $(16)^4$

∴ Normal method = $16 \times 16 \times 16 \times 16$

Using Exponential method

(1)

$$16^4$$
$$= (16^{4/2})^2$$
$$= ((16)^2)^2$$
$$= (16^2)^2$$
$$= ((16^{2/2})^2)^2$$
$$= ((16)^2)^2$$
$$= (16 \times 16)^2$$
$$= (256)^2$$
$$= 65536$$

Date __/__/__

Page _____

STUDY BUDDIES

(2) 3^5

$= 3 \times 3^{5-1}$

$= 3 \times 3^4$

$= 3 \times (3^{4/2})^2$

$= 3 \times (3^{2(2)})^2$

$= 3 \times ((3^{2/2})^2)^2$

$= 3 \times ((3)^2)^2$

$= 3 \times (9)^2$

$= 3 \times 81$

$= 243$

Date __/__/__
Page _____
STUDY BUDDIES

(5.) $2^{27} = \boxed{134217728}$

$= 2 \cdot 2^{26}$

$= 2 \cdot (2^{26/2})^2$

$= 2 \cdot (2^{13})^2$

$= 2 \cdot (2 \cdot 2^{12})^2$

$= 2 \cdot (2 \cdot (2^{12/2})^2)^2$

$= 2 \cdot (2 \cdot (2^6)^2)^2$

$= 2 \cdot (2 \cdot ((2^{6/2})^2)^2)^2$

$= 2 \cdot (2 \cdot ((2^3)^2)^2)^2$

$= 2 \cdot (2 \cdot ((2 \cdot 2^2)^2)^2)^2$

$= 2 \cdot (2 \cdot ((2 \cdot (2^{2/2})^2)^2)^2)^2$

$= 2 \cdot (2 \cdot ((2 \cdot (2)^2)^2)^2)^2$

$= 2 \cdot (2 \cdot ((2 \cdot 4)^2)^2)^2$

$= 2 \cdot (2 \cdot ((8)^2)^2)^2$

$= 2 \cdot (2 \cdot (64)^2)^2$

$= 2 \cdot (2 \cdot 4096)^2$

$= 2 \cdot (8192)^2$

$= 2 \cdot (67108864)$

$= \boxed{134217728}$

Function expo(a, n)

if n = 1 then return a

if n is even then return $[\text{expo}(a, n/2)]^2$.

else return a * expo(a, n-1)

Date __/__/__
Page
STUDY BUDDIES

Matrix Multiplication

* Strassen's Algorithm for Matrix Multiplication.

eg(1)

$$\begin{bmatrix} 1 & 3 \\ 7 & 5 \end{bmatrix} \cdot \begin{bmatrix} 6 & 8 \\ 4 & 2 \end{bmatrix}$$

Answer $\begin{bmatrix} 1\times6+3\times4 & 1\times8+3\times2 \\ 7\times6+5\times4 & 7\times8+5\times2 \end{bmatrix} = \begin{bmatrix} 18 & 14 \\ 62 & 66 \end{bmatrix}$

∴ To multiply, 2×2 matrics, total 8 (2^3) scalar multiplica are used.

⇒ In general, A and B are two matrixes to be multiplied

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\therefore C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\therefore C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} =$$
$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$$
$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$

∴ Computing each entry in the product takes n multiplications and there are n^2 entries for a total of $O(n^3)$.

⇒ Because of the problem of increase in multiplications, Strassen's method devised a better method which has same basic method as the multiplications of long integers

Date __/__/__
Page _____
STUDY BUDDIES

Strassen's Multiplication (Unit-II)

$$A = \begin{bmatrix} 5 & 6 \\ 1 & 7 \end{bmatrix} \quad B = \begin{bmatrix} 6 & 2 \\ 8 & 7 \end{bmatrix} = \begin{bmatrix} (30+48) & (10+42) \\ (6+56) & (2+49) \end{bmatrix} = \begin{bmatrix} 78 & 52 \\ 62 & 51 \end{bmatrix}$$

$P_1 = A_{11} \cdot (B_{12} - B_{22})$	$P_1 = 5\,(2-7) = -25$
$P_2 = (A_{11} + A_{12}) \cdot B_{22}$	$P_2 = (5+6) \cdot 7 = 77$
$P_3 = (A_{21} + A_{22}) \cdot B_{11}$	$P_3 = (1+7) \cdot 6 = 48$
$P_4 = A_{22} \cdot (B_{21} - B_{11})$	$P_4 = 7\,(8-6) = 14$
$P_5 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$	$P_5 = (12)(13) = 156$
$P_6 = (A_{12} - A_{22})(B_{21} + B_{22})$	$P_6 = (6-7)(14+15) = -15$
$P_7 = (A_{11} - A_{21})(B_{11} + B_{12})$	$P_7 = (4)(8) = 48$

Now $C_{11} = P_5 + P_4 - P_2 + P_6 = 156 + 14 - 77 + (-15) = 78$

$C_{12} = P_1 + P_2 = -25 + 77 = 52$ ✓

$C_{21} = P_3 + P_4 = 48 + 14 = 62$ ✓

$C_{22} = P_5 + P_1 - P_3 - P_7 = 156 + (-25) - 48 - 48 = 51$ ✓

Ex.

Date __/__/__
Page _____
STUDY BUDDIES

Unit-III

Greedy and Dynamic Programming.

→ For Optimization problem, we go through the sequence of step. with a set of choices at each step.

→ A greedy algorithm always makes the choice that looks best at that moment, i.e. It makes a locally optimal choice without worrying about the future.

→ This may not lead to a globally optimal solution.

Advantages:-

1) Easy to Design
2) Easy to Implement
3) Efficient

Disadvantage:-

Greedy algorithm do not always provides optimal solution for many problems.

General Characteristics of Greedy Algorithm

→ Greedy algo's are characterized by the following features.

① We have some problem to solve in an optimal way. To construct a solution of our problem we have a set of candidates.

② As the algorithm proceeds, we create two more sets.

(i) Contains Candidates that have been considered and chosen.

(ii) Contains Candidates that have been considered and rejected.

(3) A Solution function that checks whether a practical set of candidates provides a solution to a problem. (without thinking of optimality for the time being).

(4) A feasible Solution that checks whether a set of candidates is feasible to obtain a solution to our problem.

(5) A Selection function that selects the most promising candidate from the remaining candidates at any time. Here remaining candidates means the candidates that have neither been choosen nor rejected.

(6) An Objective function that gives the value of a solution, we have found, this is the value we are trying to optimize.

Such algorithms are called Greedy b coz

(i) At every step, it chooses the best option at that time without worrying about future

(ii) It never changes its mind. Once the candidate is included in the solution, it is there forever and once a candidate is excluded from the solution, it is never reconsidered.

Date __/__/__
Page _____
STUDY BUDDIES

```
function Greedy (C: set) : set
{
  ▷ C is set of candidates
  ▷ We construct the solution in set S.
  S ← ϕ
  while C ≠ ϕ and not Solution(S) do
    {
      x ← Select(C)
      C ← C - {x}
      if Feasible (S ∪ {x}) then
          S ← S ∪ {x}
    }
  if Solution(S) then
    return S.
  else
    {
      write "There are no solutions"
      return ϕ
    }
}
```

Unit-III

Date __/__/__
Page _____
STUDY BUDDIES

* Greedy knap Sack Algorithm

⇒ Function knapSack (w[1....n], v[1....n], W) array [1....n]

{

for i ← 1 to n do

x[i] ← 0

weight ← 0

Sort the objects into decreasing order of vi/wi

▷ Greedy Loop:

while (weight < W) do

{

i ← Select remaining object with max vi/wi;

if (weight + w[i] ≤ W) then

{ x[i] ← 1

weight ← weight + w[i]

}

else

{

x[i] ← (W - weight) / w[i]

weight ← W

}

}

return x

}

Date __/__/__
Page
STUDY BUDDIES

* Analysis

→ Sorting the objects into decreasing order of v_i/w_i takes $O(n \log n)$

→ * Greedy loop taken $O(n)$

∴ the algo. takes $O(n \log n)$.

* Possible Selection functions :-

i) Choose the remaining object with max v_i.
(to increase the value as quickly as possible)

ii) Choose the remaining object with maximum w_i
(to use the capacity as slow as possible)

iii) Choose the remaing object with maximum v_i/w_i.

* Example.

Consider the objects with following weight and value

	10	20	30	40	50
	20	30	66	40	60

Total capacity of knapsack is 100. Find the maximum profit that can be carried away by knapsack.

Table-1 [$W = 100$, $N = 5$]

Objects(i)	1	2	3	4	5
Weight(w)	10	20	30	40	50
Value(v)	20	30	66	40	60
Value/Weight(v/w)	2.0	1.5	2.2	1.0	1.2

Selection fn

Table 2

Function	x_1	x_2	x_3	x_4	x_5	Value
i) Max v_i	0	0	1	0.5	1	0+0+66+20+60 = 146
ii) Min w_i	1	1	1	1	0	20+30+66+40+0 = 156
iii) Max v_i/w_i	1	1	1	0	0.8	20+30+66+0+48 = 164

Date __/__/__
Page _____
STUDY BUDDIES

(2) For the given set of items and knapsack capacity = 60 kg. Find the optimal solution for the fractional knapsack problem using greedy approach,

item	weight	Value
1	5	30
2	10	40
3	15	45
4	22	77
5	25	90

$|N| = 60$ kg

Table-1

Objects(i)	1	2	3	4	5
Weight(w_i)	5	10	15	22	25
Value(v_i)	30	40	45	77	90
v_i/w_i	6	4	3	3.5	3.6

Table-2

Specification function(x_i)	x_1	x_2	x_3	x_4	x_5	
(i) Max v_i	0	0	0.866	1	1	38.97+77+90=205.97
(ii) Min w_i	1	1	1	1	0.32	30+40+45+77+28 = 220.
(iii) Max v_i/w_i	1	1	0	0.91	1	30+40+70.07+90 = 230.07

Hence, Max v_i/w_i = Total Profit = 230.07 (optimal Soln)

Activity Selection Problem

Date / /
Page
STUDY BUDDIES

→ Problem: "Schedule maximum number of compatible activities that need exclusive access to resources."
like

→ Aim of algorithm is to find optimal schedule with maximum number of activities to be carried out with limited resources.

→ Suppose $S = \{a_1, a_2, a_3, \ldots a_n\}$ is the set of activities that we want to schedule.

→ Scheduled activities must be compatible with each other. Start time of activities is let's say s_i and finishing time is f_i; then activities i and j are compatable if and only if $f_i < s_j$ or $f_j < s_i$. In other words, two activities are compatible their time duration do not overlap.

Greedy Approach for Activity Selection

(1) Sort the activities according to their finishing time

Initial selection (2) Select the first activity from the list of sorted array and print it.

(3) Repeat the following steps for selecting activities as follows.

The Selection of next activity will be based on the following condition:

Start time of activity(s_i) $\geq$ Finish time of previous selected Activity(f_i).

Date __/__/__
Page _____
STUDY BUDDIES

Eg ① Activity A_1 A_2 A_3 A_4 A_5 A_6

S_i	0	3	1	5	5	8
F_i	6	4	2	9	7	9

Soln → Step 1. Sort activities according to finishing time into ascending order.

Activity	A_3	A_2	A_1	A_5	A_4	A_6
S_i	1	3	0	5	5	8
F_i	2	4	6	7	9	9

Step 2 Select First activity
ie. A_3 ∴ Solution set = $\{A_3\}$

→ Selection of Next activity
ie.. A_2 bcoz $S_i \geq F_i$ (Previous)
i.e. $3 \geq 2$
∴ Solution set = $\{A_3, A_2\}$

→ Next selection of activity
ie.. A_1 → Cannot be added to soln set
bcoz. $S_i \geq F_i$ (p) IS FALSE
ie. $0 \geq 4$ → False.
∴ This activity is not added to soln set

→ Next selection of activity
ie A_5 → bcoz. $S_i \geq F_i$ (previous)
i.e. $5 \geq 4$.
∴ Solution set = $\{A_3, A_2, A_5\}$

→ Next Selection of activity
ie. A_4 → False Not added to soln set.
bcoz $5 \geq 7$ → FALSE

→ Next Selection of activity
ie. A_6 → bcoz $S_i \geq F_i$ (Previous)
ie $8 \geq 7$
∴ Solution set = $\{A_3, A_2, A_5, A_6\}$

Date / /
Page
STUDY BUDDIES

Algorithm Act_Selection_Greedy(A, S, F)

{

// A is the set of activities sorted by finishing time.

S ← A[1];

i ← 1;

for j ← 2 to n do

{

if ($S_j \geq F_i$) then

{

S ← S ∪ {A_j}

i ← j

}

}

return S;

}

* Complexity Analysis :-

O(n log n) if the list is not sorted.

O(n), if the list is already sorted

Sorting takes time of O(n log n)

Selection takes time of n is O(n)

∴ Total time = O(n log n) + O(n)

= O(n log n)

Date __/__/__
Page _____
STUDY BUDDIES

Eg(2)

i	0	1	2	3	4	5	6	7	8	9	10	11
S_i		1	3	0	5	3	5	6	8	8	2	14
F_i		4	5	6	7	8	9	10	11	12	15	16

Activity Sel'n set $S = \{1, 4, 8, 11\}$

Now ⇒ Select first activity ∴ $S = \{1\}$

⇒ Selection of Next activity

$i = 2$, $S_i \geq f_i$ ⇒ $3 \geq 4$ → False

∴ $S = \{1\}$

⇒ Selection of Next

$i = 3$, $S_i \geq f_i$ ⇒ $0 \geq 4$ → False, $S = \{1\}$

⇒ Selection of Next

$i = 4$, $S_i \geq f_i$ ⇒ $5 \geq 4$ ⇒ True, $S = \{1, 4\}$

⇒ Selection of Next

$i = 5$, $S_i \geq f_i$ ⇒ $3 \geq 7$ ⇒ False, $S = \{1, 4\}$

⇒ Selection of Next

$i = 6$, $S_i \geq f_i$ ⇒ $6 \geq 7$ ⇒ False, $S = \{1, 4\}$

⇒ Selection of Next

$i = 7$, $S_i \geq f_i$ ⇒ $6 \geq 7$ ⇒ False, $S = \{1, 4\}$

⇒ Selection of Next

$i = 8$, $S_i \geq f_i$ ⇒ $8 \geq 7$ ⇒ True, $S = \{1, 4, 8\}$

⇒ Selection of Next

$i = 9$, $S_i \geq f_i$ ⇒ $8 \geq 11$ ⇒ False, $S = \{1, 4, 8\}$

⇒ Selection of Next

$i = 10$, $S_i \geq f_i$ ⇒ $2 \geq 11$ ⇒ False, $S = \{1, 4, 8\}$

⇒ Selection of Next

$i = 11$ ⇒ $S_i \geq f_i$ ⇒ $14 \geq 11$ ⇒ True, $S = \{1, 4, 8, 11\}$

Date / /
Page
STUDY BUDDIES

* Job Scheduling Problem

Problem: Schedule some J jobs out of set of N Jobs on single processor which can maximize profit as much as possible.

⇒ We have N jobs, each taking unit time.
→ Each job is having some profit and deadline associated with it.
→ We can earn the profit only if job is completed before its deadline.
→ Each job has deadline $d_i \geq 1$ and profit $P_i \geq 0$.
Only one job can be active on processor.
→ Feasible solution is subset of N Jobs, such that each job can be completed on or before its deadline.
→ An Optimal solution is the solution with maximum profit.

Ex(1) Solve the job scheduling problem for given data. Let $n = 4$, $J = (J_1, J_2, J_3, J_4)$, $P = (50, 30, 25, 45)$, $D = (2, 1, 2, 1)$ using brute force and greedy approach.

⇒ **Brute Force Approach**

→ Brute force approach generates all possible solution schedules and finds optimal schedule from that.

	J_1	J_2	J_3	J_4
Profit	50	30	25	45
Deadline	2	1	2	1

Max-Min Problem (Unit-II)

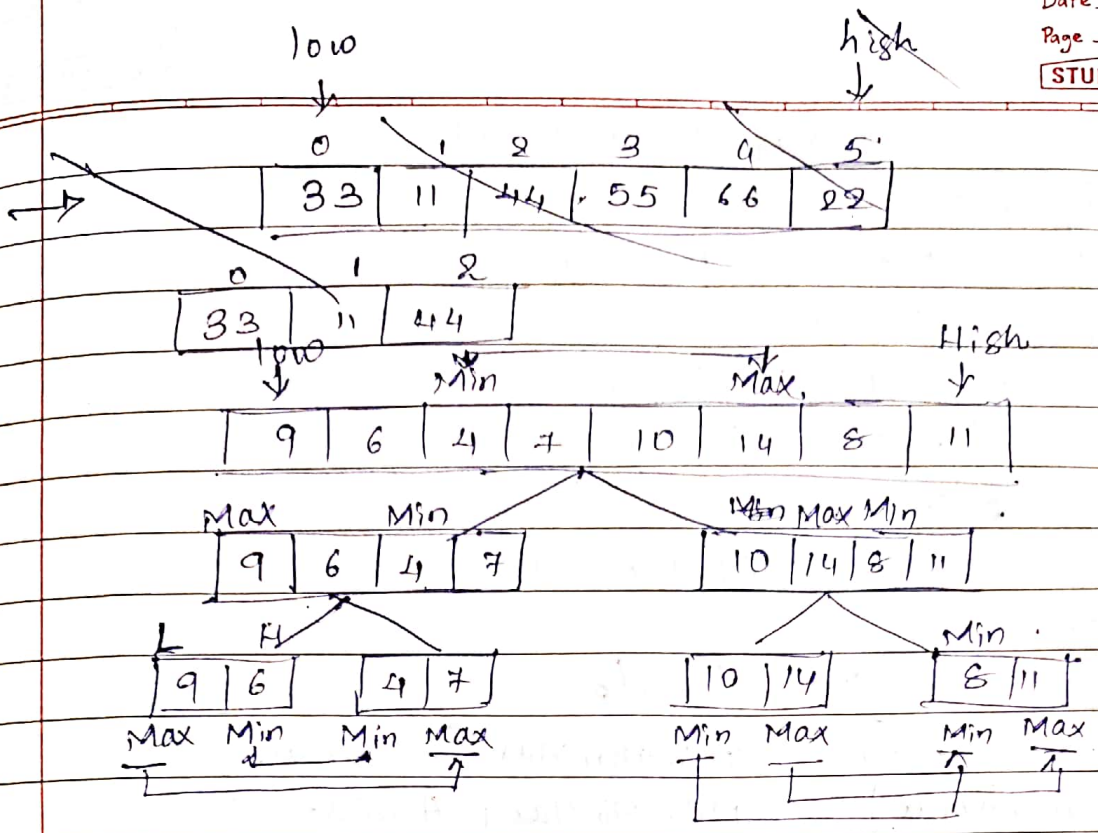
Date __/__/__
Page
STUDY BUDDIES

- Max-min problem is to find a maximum and minimum from the given array.
- We can effectively solve it using divide and conquer approach.
- In traditional method, the maximum and minimum can be found by comparing each element and updating Max and Min values as and when required.
- This approach is simple but it does (n-1) comparisions for finding max and the same number of comparisions for find the min.
- It results in a total of 2(n-1) comparisions.
- Using divide and conquer approach, we can reduce the number of comparisions.

~~Algo. MaxMin~~

Algo. MaxMin(a, n, Max, min)

```
{
    max = min = a[0]
    for i = 0 to n-1 do
    {
        if a[i] > max then
            max = a[i];
        if a[i] < min then
            min = a[i]
    }
    Return (max, min);
}
```



Recurrence eq^n of $T(n) = 2T(n/2) + 2$ - Solve using back substitution.
Min-Max problem $= O(n)$ $\rightarrow \left(\frac{3n}{2} - 2\right)$

Master's Method

$a = 2 \quad b = 2, \quad n^0 = d = 0$

$\therefore \quad a > b^d$

$2 > 2^0$

$= O(n^{\log_b a})$

$= O(n^{\log_2 2})$ $\quad \because \log_2 2 = 1$

$= O(n)$ ✓

~~$O(\log_2 2)$~~

~~$= O(1)^{\log_2 2}$~~

Complexity of Min-Max Algo. using Divide and Conquer