LAB: O(1) => Amount of memory that is utilise by code

Find max among an array

Pseudo code: arr[] = [5, 2, 9, 1, 4, 6]

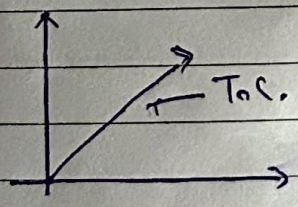
```
for i in range (len(arr))
    arr[i] = arr[i+1]

    if (arr[i] > arr[i+1])
      { return arr[i] }
    else
      { return arr[i+1] }
```

Theory: -> There are two types of complexities:

Time	Space
↳ rate of time taken to execute the code	→ space / amount of memory that has been utilised by code

Time Complexity:-



T.C. != Time taken

Problem Solving

1s ≈ 10^8 operations

n > 10^8 O(log n) O(1)

n ≤ 10^8 O(n)

n ≤ 10^6 O(n log n) sorting

n ≤ 10^4 $O(n^2)$

n ≤ 500 $O(n^3)$

n ≤ 25 $O(2^n)$

n ≤ 12 O(n!)

Notation: O

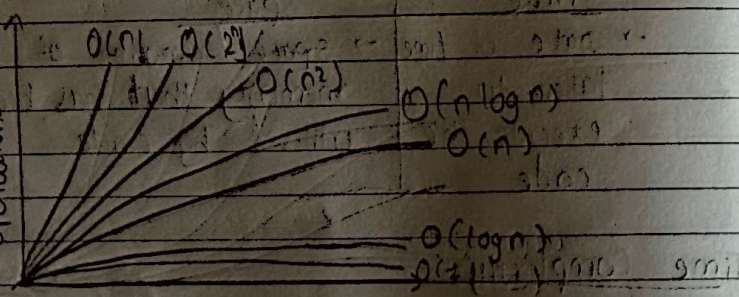
→ Time Complexity does not refer to the time taken by the machine to execute a particular code.

→ The rate at which the time required to run a code, changes w.r.t. the input size, is called time complexity.

→ Always calculate the time complexity for the worst case scenario.
→ Avoid including constant terms
→ Avoid lower values.

→ Space : Amount of extra memory utilised by your side

Big O → represents worst case
Theta θ → avg. case
Omega Ω → Best case



Constant O(1)
O(log n)
Linear O(n)
Logarithmic O(n log n)
Quadratic O(n²)
Exponential O(2^n)
Factorial O(n!)

best

⇒ $f(n) = O(g(n))$ ⇒ worst case

For any Fxn $g(n)$ to act as upper bound of $f(n)$,
∃ a positive constant C and n_0

$$0 \le f(n) \le C \cdot g(n) \quad \text{for } \forall\, n \ge n_0$$

Best case
$f(n) = \Omega(g(n))$

For any fxn $g(n)$ to act as lower bound of $f(n)$
∃ a positive constant 'c' and 'n₀'
such that $0 \le C \cdot g(n) \le f(n)$
for $\forall\, n \ge n_0$ $\quad C \ge 1$
$n_0 \ge 1$ by default

Exact case

$$f(n) = \Theta(g(n))$$

For any fxn $g(n)$ to act as exact case of $f(n)$
∃ a positive constant C_1, C_2 and n_0
such that $C_1 g(n) \le f(n) \le C_2 g(n)$
for $\forall\, n \ge n_0$
L.B. U.B.

$0 \le 2n^2 - 4 \le C \cdot n^2$
Let $C = 2$
$\therefore 0 \le 2n^2 - 4 \le 2n^2$ ✗
Let $C = 3$ for
Let $0 \le 2n^2 + 4 \le 3n^2$

$n = 1 \quad 0 \le +6 \le 3$ ✗
$n = 2 \quad 0 \le 12 \le 12$ ✓
$n = 3 \quad 0 \le 22 \le 27$ ✗

→ tight bound : information is quite clear
→ Loose bound : information is

- $3n+8$
- $3n^3+8n-3$
- 420
- n^4+50n^2+100

$\Rightarrow$ $3n+8$

$g(n) = n$

~~$O(g(n)) = 2n$~~

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

$0 \leq 3n+8 \leq ~~c \cdot 2n~~ cn$

Let $c = 4$

$\therefore 0 \leq 3n+8 \leq 4n$

$3n+8 \leq 4n$

$8 \leq n$

$n \geq 8$

$\Rightarrow$ 420

$f(n) = 420$

$g(n) = 1$

$0 \leq 420 \leq c \cdot 1$

$420 \leq c$

$c = 420$

$n_0 = 1$

$\Rightarrow$ $3n^3+8n-3$

$f(n) = 3n^3+8n-3$

$g(n) = n^3$

$0 \leq 3n^3+8n-3 \leq (n^3) \cdot c$

Let $c = 4$

$0 \leq 3n^3+8n-3 \leq 4n^3$

$-3 \leq n^3-8n$

$n^3-8n \geq -3$

$n^2(n-8) \geq -3$

$n-8 \geq -3$

$n \geq 5$, $n^2 \geq -3$ ✗

$\Rightarrow$ n^4+50n^2+100

$f(n) = n^4+50n^2+100$

$g(n) = n^4$

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

$0 \leq n^4+50n^2+100 \leq c \cdot n^4$

Let $c = 2$

$0 \leq n^4+50n^2+100 \leq 2n^4$

$n^4+50n^2+100 \leq 2n^4$

$100 \leq n^4-50n^2$

=> $f(n) = 3n+8$
$g(n) = n$

$c \cdot g(n) \leq f(n)$
$c \cdot n \leq 3n+8$
Let $c=3$
$3n \leq 3n+8$
$0 \leq 8$
$n \geq 1$

=> Let $f(n) = 3n-8$
$g(n) = n$

$c_0 \cdot g(n) \leq f(n)$
$c \cdot n \leq 3n-8$
Let $c=2$
$2n \leq 3n-8$
$8 \leq n$
$n \geq 8$ ≠

$n \geq n_0 \Rightarrow n_0 = 8$

=> $f(n) = \theta\, g(n)$

- For any function $g(n)$ to act as exact case of $f(n)$;

$c \geq 1$
$n_0 \geq 1$ by default

∃ a positive constant c_1, c_2 and n_0
such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$
for $\forall\ n \geq n_0$

=> $3n+8$, $g(n) = n$

$f(n) = 3n+8$, $g(n) = n$

$c_1 g(n) \leq 3n+8 \leq c_2 g(n)$

(lower (Ω))
$c_1 g(n) \leq f(n)$
$c_1 g(n) \leq 3n+8$
Let $c_1 g(n) = 3n$
$\therefore 3n \leq 3n+8$
$\therefore 0 \leq 8$
$n \geq 1$
X

(Upper (Ω))
$3n+8 \leq c_2 g(n)$
Let $c_1 g(n) = 4n$
$\therefore 3n+8 \leq 4n$
$\therefore 8 \leq n$
$\therefore n \geq 8$

$\Rightarrow n \geq 8$

=> $\frac{n^2}{2} - \frac{n}{2}$

$f(n) = \frac{n^2}{2} - \frac{n}{2}$, $g(n) = n^2$

~~c.g~~ $c_1 g(n) \le f(n) \le c_2 g(n)$

$c_1 g(n) \le \frac{n^2}{2} - \frac{n}{2} \le c_2 g(n)$

$c_1 g(n) \le f(n)$

$c_1 n^2 \le \frac{n^2}{2} - \frac{n}{2}$

Let $c_1 n^2 = \frac{n^2}{2}$

$\frac{n^2}{2} \le \frac{n^2}{2} - \frac{n}{2}$

$0 \le -\frac{n}{2}$

$0 \le \frac{n}{2}$

$0 \le n$

$n \ge 0$

$f(n) \le c_2 g(n)$

$\frac{n^2}{2} - \frac{n}{2} \le c_2 n^2$

Let $c_2 n^2 = n^2$

$\therefore \frac{n^2}{2} - \frac{n}{2} \le n^2$

$\therefore \frac{n^2}{2} - n^2 \le \frac{n}{2}$

$\therefore -\frac{n^2}{2} \le \frac{n}{2}$

$\therefore -n \le 1$

$\therefore n \ge -1$

$\Rightarrow n-3$

$f(n) = n-3, \quad g(n) = n$

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

$c_1 g(n) \le n-3 \le c_2 g(n)$

$c_1 g(n) \le n-3$

$c_1 n \le n-3$

Let $c_1 n = n$

$n \le n-3$

$0 \le -3$

~~0~~ X $n \ge 1$

~~$c_2 g(n) : n-3 \le c_2 g(n)$~~

$n-3 \le c_2 n$

Let $c_2 n = 2n$

$n-3 \le 2n$

$-3 \le n$

$n \ge -3$

Hence Exact case doesn't exist

for $c_1 n \le n-3$	for $f(n) \le c_2(n)$
Let $c_1 = 1$, $n = 1$	
$1 \le n-3$	
$4 \le n$	
$n \ge 4$	
$f(n) = \Omega(1)$	$f(n) = O(n)$
⇓	
best case	

LAB :- → There are two approaches
1.) Iterative
2.) Recursive

```
T(n) —> Void ICT(int n)
        {
   1  —> if(n>0)
log22 —> [for(i->0 to n; i++)
         printf("Hello");] <— n
 T(n-1) —> ICT(n-1);
        }
```

∴ $T(n) = 2T(n-1) + \log_2 n$

→ Time Complexity = $O(n)$

$$T(n) = \begin{cases} T(n-1) + 1 ; & n > 0 \\ 1 ; & n = 0 \end{cases}$$

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

H.W. $\begin{cases} T(n) = T(n-1) + \log_2 n \\ T(n) = 2T(n-1) + \log_2 n \\ T(n) = 2T(n-1) + n \end{cases}$

→ If time complexity of function is $T(n)$ then time complexity of function $T(T(n-1))$ is $T(n-1)$

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

Substitute $n = n-1$ in eq^n (1)

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

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

Put (2) in (1)

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

$\therefore T(n) = T(n-2) + 2 \quad -(3)$

substitute $n = n-2$ in (1)

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

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

Put (4) in (3)

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

$T(n) = T(n-3) + 3 \quad -(5)$

$a = b+2$

$b = c+4$

$c = d-3$

$d = -1$

Here we can see that to get the value of a, a is dependent on b.

→ Similarly b is dependent on c & c is dependent d.

Here we have value of d.

→ So we have to go through the long process to get the value of a

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

In base case, ~~n = k~~

$\therefore T(n-k) = T(0)$

$\therefore n-k = 0$

$n = k$

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

$= T(0) + n$

$T(n) = 1 + n$

$\Rightarrow O(n)$

$\Rightarrow$ We will take the same case but we will not assume we will give base cases

$$T(n) = \begin{cases} T(n-1)+n; & n>0 \\ 1 \quad ; & n=0 \end{cases}$$

$T(n) = T(n-1) + n$ — (1)

→ substitute $n = n-1$ in eqn (1)

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

Put (2) in (1)

$T(n) = T(n-2) + n-1 + n$ — (3)

→ substitute $n = n-2$ in eqn (1)

$T(n-2) = T(n-3) + n-2$ — (4)

Page No.:
Date:
YOUVA

Put (4) in (3)

$$T(n) = T(n-3) + (n-2) + (n-1) + n \quad -(5)$$

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

$$T(n-k) = T(0)$$
$$n-k = 0$$
$$n = k \quad -1$$

Substitute $n=k$ in above eqn

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

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

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

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

$$= O(n^2)$$

H.W: $T(n) = T(n/2) + 1$

$\therefore T(n) = 2T(n/2) + 1$

$T(n) = T(n/2) + n$

$T(n) = T(n/2) + \log_2 n$

① $T(n) = T(n-1) + \log_2 n$ — ①

Substitute $n = n-1$ in eqn — ①

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

$T(n-2) = T(n-2) + \log_2 (n-1)$ — ②

Put ② in ①

$T(n) = T(n-2) + \log_2 (n-1) + \log n$ — ③

Substitute $n = n-2$ in eqn ①

$T(n-2) = T((n-2)-1)) + \log_2 (n-2)$

$= T(n-3) + \log_2 (n-2)$ — ④

Put ④ in ③

$\therefore T(n) = T(n-3) + \log_2 (n-2) + \log_2 (n-1) + \log_2 n$ — ⑤

Substitute $n = n-3$ in eqn — ①

$T(n-3) = T((n-3)-1) + \log_2 (n-3)$

$= T(n-4) + \log_2 (n-3)$ — ⑤

Put ⑤ in ⑤

$\therefore T(n) = T(n-1) + \log_2(n-3) + \log_2(n-2) + \log_2(n-1) + \log_2 n$

$\vdots$

$T(n) = T(n-k) + \log_2(n-k+1) + \log_2(n-k+2) + \ldots + \log_2(n-1) + \log_2 n$

In base cases,

$T(n-k) = T(0)$

$n-k = 0$

$n = k$

$= T(n-n) + \log_2 1 + \log_2 2 + \ldots + \log_2(n-1) + \log_2 n$

$= T(0) + \log_2(1 * 2 \times 3 \ldots (n-1) \times n)$

$= T(0) + \log_2 n!$

(base)^(exponent) => x^n

```
def power (base, exponent)
    result = 1
    for i in range (exponent)
        result = base x result
    return result
```

List of Programs:
- Linear search
- Selection sort
- Counting sort
- Bubble Sort
- Power → Iterative
- Power → Iterative
- Factorial → Recursive

Recursive approach:

```
def power (base, exponent)
```

⇒ factorial of n

n! (Let n = 2

2! = 2 x 1

3! = 3 x 2 x 1 = 6

→ So like this the logic to implement to find factorial is n! = n x (n-1) x (n-2) - - - (n-n-1)

∴ factorial (x)

x = 2

logic if x == 1
return 1

x * fact (x-1)
2 * fact (2-1)
2 * fact (1)
2 * 1
= 2

x = 5

if x == 1
return 1

else fact (x-1)

x * fact(x-1)
5 * fact (4)
5 * 4 * fact (3)
5 * 4 * 3 * fact (2)
5 * 4 * 3 * 2 * fact(1)
5 * 4 * 3 * 2 * 1
= 120

→ According it we need to implement a for loop
→ when it comes fact (1) → return 1
→ Otherwise continue fact (x-1) whatever the x comes

Substitution Method:

- Decreasive
- Dividing

→ It's a bit lengthy & Complex →
→ So we will move to some shorter Method

Master's Theorem:-

$T(n) = aT(n/b) + \theta(n^k \log^p n)$

1) If $a > b^k$, then $T(n) = \theta(n^{\log_b a})$

2) If $a = b^k$, then we follow this

- (a) If $p > -1$, then $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$
- (b) If $p = -1$, then $T(n) = \theta(n^{\log_b a} \log \log n)$
- (c) If $p < -1$, then $T(n) = \theta(n^{\log_b a})$

3) If $a < b^k$,

- (a) If $p \geq 0$, then $T(n) = \theta(n^k \log^p n)$
- (b) If $p < 0$, then $T(n) = \theta(n^k)$

(1) $T(n) = 2 \cdot T(n/2) + 1$

Here $a = 2, \quad b = 2, \quad k = 0, \quad p = 0$

$a = 2, \quad b^k = 2^0 = 1$

$\Rightarrow a > b^k$

$\therefore$

→ So we apply case-1 from Masters Theorem

→ Now we will check value of $p = a$

$p > -1$

$\Rightarrow T(n) = \theta(n^{\log_b a} \log^{p+1} n)$

$= \theta(n^{\log_2 1} 1$

$\Rightarrow T(n) = \theta(n^{\log_b a})$

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

$= \theta(n)$

$\Rightarrow O(n)$

2.) $T(n) = 4T(n/2) + 1$

3.) $T(n) = 4T(n/4) + n$

4.) $T(n) = 8T(n/2) + n^2$

5.) $T(n) = 8T(n/4) + n$

6.) $T(n) = T(n/2) + n$

7.) $T(n) = 2T(n/2) + n^2 \log n$

8.) $T(n) = 4T(n/2) + n^3 \log^2 n$

9.) $T(n) = 2T(n/2) + n^2/\log n$

10.) $T(n) = 4T(n/2) + n$

11.) $T(n) = 16T(n/2) + n^2$

12.) $T(n) = T(n/2) + 1$

13.) $T(n) = 2T(n/2) + n$

14.) $T(n) = 2T(n/2) + n^{n \log n}$

15.) $T(n) = 4T(n/2) + n^2$

16.) $T(n) = 4T(n/2) + n^2 \log^2 n$

17.) $T(n) = 2T(n/2) + n/\log n$

Divide & Conquer :-

→ Iterative approach has Time complexity : O(n)
→ Recursive approach has Time complexity : O(n)

```
for result = 1
    for (i → 0 to n)
        result = result * n
→ output = result
```

} Iterative approach

→ A recursive function will make a stack at backend

Recursive approach {

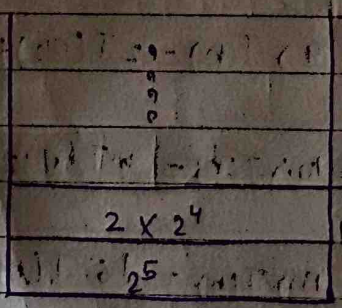
T(n) → function power (x, n)
    I → if (n = 0)
        return 1
    else
T(n-1) → return x* power (x, n-1)

∴ T(n) = T(n-1) + 1

⇒ O(n) : Time complexity

→ Space complexity : O(n)

→ When we provide input Let x = 2 & n = 5 then



→ It will store the values in stack like this & it will go untill n=0

→ A stack is build with space complexity of O(n) as it will store the data upto 2^0.

Code: → function Power (x, n)
if (n == 0)
{return 1}

else if (n%2 == 0)
{ return power (x, n/2) * power (x, n/2)}

else
{ return x* power (x, n/2)* power (x, n/2)}

→ power (x, n/2) * power (x, n/2) is only for even value of n
→ But for odd values of n we will take a case i.e. x* power (x, n/2) * power (x, n/2).

Here DSC (I) O(n) → Time Complexity
→ space complexity

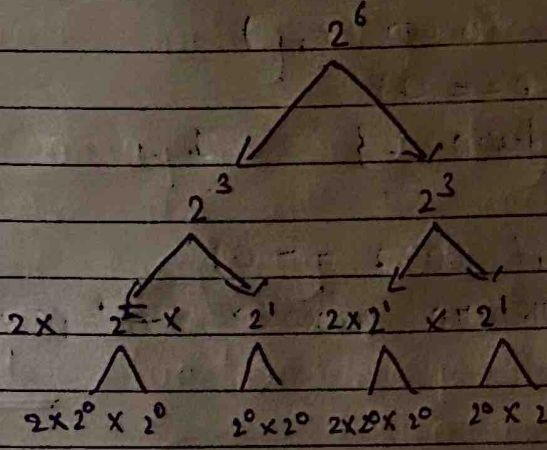
DSC (II) O(log n) → Time complexity
→ space complexity

Code: function power (x, n)
if (n == 0)
return 1

temp = power (x, n/2)
if (n%2 == 0)
return temp * temp

else
return x * temp * temp

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



Q.1 A = [1, 3, 5, 6, 8]
B = [2, 4, 5, 9, 11, 12]

A & B are sorted, Find result list which is combination of A & B and it needs to be sorted.

Ans: We need the Result ~~array~~ is [1, 2, 3, 4, 5, 6, 9, 11, 12]

~~X~~ for (int i=0; i<6 ; i++)
for (int j=0; j<7; j++)
if (A[i] > B[j]

Logic A: 1 Exceed B:

1	3	5	6	8		2	4	5	9	11	12	Exceed
i↑	i↑	i↑	i↑	i↑	it	j↑	j↑	j↑	j↑	j↑	j↑	j↑

Result: k↑

1	2	3	4	5	6	8	9	11	12
k↑	k++	k++	k++	k++	k++	k++	k++	k++	k++

→ Here Lets say A is left array & B is right array & K is used to store Elements in array for result.

→ First we will run the while loop where we take length of both arrays.

→ Then we will check the first elements of both arrays & compare them, whosoever is less than other it will be stored as first Element in result array.

→ Now we will compare second ~~err~~ element of A & first element of B, whichever is less it will be stored in result array

→ In the similar way all the elements will be stored & they will be in sorted manner.

Pseudo code:

```
function merge (left(i), right(j)) {
  result = []
  i = j = 0

  while (i < len(left) && j < len(right)
  { if (left[i] < right[j])
      {result.append(left[i]);
       i++;
       k++; }
  else
      {result.append(right[j]
       j++;
       k++; }} return result; }
```

∴ TC = O(n)

Greedy Approach :-

Greedy	Dynamic
- Makes local optimal choices	- Guarantees global optimal solution as it consider all possibilities
- Speed of Execution is faster (simple logic / single pass)	- Slower as requires solving Sub problems, overlapping computation
- Very low memory consumption (stores few variables)	- High memory (stores solution of subproblems in tables).
- Sub-optimal solution	- Optimal solution

→ 100 students wants to appear for interview so then gone through first ~~aptitute test~~ they gave interview 50 is selected for aptitute

selected
- 50 (1) in Aptitude
- appears → 20 (2) in Coding
- appears → 08 (3) in Interview
- appears → 05 (4) in HR.

8 appears for coding 8 is selected for Interview then final 5 for HR round

→ In Dynamic we consider all possible cases so we get optimal answer.

LAB

Quick Sort:-

→ Choose a pivot element
→ Partition the array around pivot
→ on le left side < pivot & on right side > pivot
→ Applied recursively on left & right subarrays.
→ Faster than Merge sort
→ Not stable (relative order of equal elements may change)
→ Best case : $O(n \log n)$
→ Worst case : $O(n^2)$
→ Space C. : $O(\log n)$

Theory :-

Greedy Algorithm :-

□ Activity selection Problem :

Here

Activity No.	0	1	2	3	4
Start time	5	6	0	4	10
End time	8	10	5	7	12

2 0 4 ← these activity Nos can be performed
Stage

⇒ We need to design an algorithm to schedule the maximum number of activities on a stage such that no two activities overlaps.
i.e. No two activities should overlap & max activities is to be performed.

- **Steps :** ① Sort activities in ascending order based on end time
  ② Consider 1st activity
  ③ if finish time of ≤ start time of ith activity (i+1)th activity
  then consider i^{th} activity
  else
  move to next activity

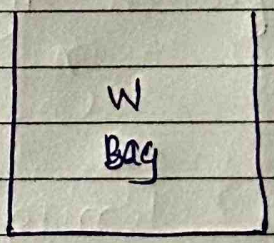
knapsack :-

→ meaning is "Bag":

Mall =>

P_1	P_2	P_3	- -	- -	-	P_n
w_1	w_2	w_3				w_n

Theif has a bag =>



→ A thief enters a mall to steal products. He has a bag with capacity of Wkg to put the stolen products. We need to help him choose items such that he maximises his profit without exceeding the weight of bag.

let W = 7 kg

Product No	1	2	3	4	5	6	7
Product Price	10	5	3	2	8	7	11
Product wt.(kg)	2	3	1	4	3	2	7