Date

~ ASSIGNMENT-1 ~

NAME : DIVYANSH SAXENA
UNIV ROLL NO : 2015242
SECTION : M6
CLASS ROLL NO : 25
SIGNATURE :

1) Asymptotic Notations :- are mathematical tools to represent the time complexity of algorithms for asymptotic analysis.
The main idea of asymptotic analysis is to have a measure of the efficiency of algorithms that depend on machines.
Most used asymptotic notations are -

(i) Θ Notation :- The theta notation bounds a func. from above & below, so it defines exact asymptotic behaviour.

(ii) Big O Notation :- It defines an upper bound of an algorithm, it bounds a func. only from above.

(iii) Ω Notation :- it provides an asymptotic lower bound.

For e.g.,
consider Insertion Sort.
It takes linear time in best case & quadratic time in worst case.
We can say, Insertion Sort have :-
$O(n^2)$ / $O(n^2)$ for worst case / $O(n)$ for best case / $\Omega(n)$

2) $O(\log n)$

4) $T(n) = \begin{cases} 2T(n-1)-1 & \text{, if } n>0 \\ 1 & \text{otherwise} \end{cases}$

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

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

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

$= 2^2(2T(n-3)-1)-2-1$

$= 2^3 T(n-3)-2^3-2^1-2^0$

$= 2^n T(n-n) - 2^{n-1} - 2^{n-2} - 2^{n-3} \cdots 2^2-2^1-2^0$

$= 2^n - (2^n-1)$

$= 2^n - 2^n + 1 = 1$

$T(n) = 1$

3) $T(n) = \begin{cases} 3T(n-1) & \text{, if } n>0 \\ 1 & \text{otherwise} \end{cases}$

$T(n) = 3T(n-1)$

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

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

$\vdots$

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

$= 3^n$

5) $S_i = S_{i-1} + i$.

If k is total no. of iterations taken by program, then while loop terminates.

$1+2+3 \cdots\cdots k = [k(k+1)/2] > n$

$\therefore k = O(\sqrt{n})$

Page No.
Date

6) $O(\sqrt{n})$

7) j loop executing $\log n$ times
k " " $\log n$ times
i " " $n/2$ times $n/2 \simeq n$
Time complexity = $O(n \log^2 n)$

8) $O(n^3)$

9/15) Inner loop will execute $\left(n + \frac{n}{2} + \frac{n}{3} + \cdots \frac{n}{n}\right)$
$= n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n}\right)$
$\Rightarrow$ It is equal to $\underline{O(n \log n)}$

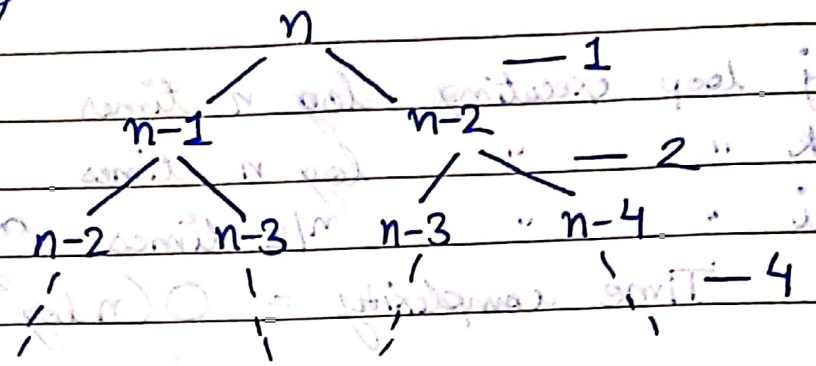
10) n^k a^n
$k >= 1$ $a > 1$
Taking $k = a = 2$
n^2 2^n
We can say $n^2 = O(2^k)$
$n^k = \underline{O(a^n)}$

11) $O(\sqrt{n})$: (NOTE: Logic same as given in (5))

12) Recurrence Relation
$T(n) = T(n-1) + T(n-2) + 1$

Page No.
Date

Making recurrence tree,



$T.C = 1+2+4+ \dots + 2^n$

$a=1, r=2$

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

$$O(2^{n+1}) = O(2+2^n) = O(2^n)$$

Space complexity = $O(n)$

This is because maximum stack frame is equal to n only as func called like this :-

$f(n-1) + f(n-2)$

$f(n-2)$ is called when we get the return value from $f(n-1)$

∴ It is equal to $O(n)$.

13) $n \log n$

```
for(i = 1; i<n; i++)
    for(j=1; j<=n; j = j+i)
        printf("#");
```

Page No.
Date

n^3

```
for(i=1; i<n; i++)
    for(j=1; j<n; j++)
        for(k=1; k<n; k++)
            printf("#");
```

log log n

```
int func(int n)
{   if(n<=2)
        return 1;
    else
        return(func(func(sqrt(n)))+n);
}
```

14) $T_n = T(n/4) + T(n/2) + cn^2$

assuming,

$T(n/2) >= T(n/4)$

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

Applying masters method,

$a = 2, \quad b = 2$

$k = \log_b a = \log_2 2 = 1$

$n^k = n$

$f(n) = n^2$

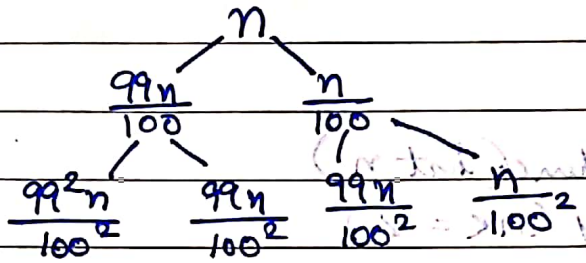
It is $\Theta(n^2)$

But as $T(n) <= \Theta(n^2)$

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

16) If k is a constant > 1
Then T.C $= O(\log \log n)$

17) $T(n) = T\left(\frac{99n}{100}\right) + T\left(\frac{n}{100}\right)$



If we take longer branch, i.e., $99n/100$

T.C $= \log_{(100/99)} n \approx \log n$
base

We can say, the base of log doesn't matter as it's only a matter of constant.

18) (a) $100 \quad \log \log n \quad \sqrt{n} \quad n \quad \log n! \quad n \log n \quad n^2 \quad 2^n \quad 2^{2n} \quad 4^n \quad n!$

(b) $1 \quad \log \log n \quad \sqrt{\log n} \quad \log n \quad 2 \log n \quad \log 2n \quad n \quad 2n \quad 4n \quad \log n! \quad n \log n \quad n^2 \quad 2(2^n) \quad n!$

(c) $96 \quad \log_8 n \quad 5n \quad \log n! \quad n \log_6 n \quad n \log_2 n \quad 8n^2 \quad 7n^3 \quad 8^{2n} \quad n!$

~~19)~~

Page No.
Date

19) linear search (array, key)
for i in array
if value == key
return i

20) Iterative Insertion Sort

insertion_sort (arr, n)
loop from i=1 to i=n-1
pick element arr[i] & insert it into sorted sequence arr[0....i-1]

Recursive Insertion Sort

insertion_sort (arr, n)
{ if n <= 1
return
recursively sort n-1 elements
insertion_sort (arr, n-1)
pick last element arr[i] and insert it into sorted sequence arr[0.....i-1]
}

Insertion sort considers one input element per iteration and produces a partial solution without considering future elements.
It's called online sorting algorithm.

Page No.
Date

20/21/22 > Considering the 3 sorting algorithms taught,

Algorithm	Best case	Avg Case	Worst Case	SC	Stable	Inplace	On
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	✓	✓	×
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	×	✓	×
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	✓	✓	✓

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

23) Binary Search

A ← Sorted array
n ← Size of array
X ← value to be sorted
while X not found
  if upperbound < lowerbound
    EXIT : X does not exit
  set midpt = lowerbound + (upperb. − lowerb)/2
  if A[midpt] < X
    lowerbound = midpoint + 1
  if A[midpt] > X
    upperbound = midpoint − 1
  if A[midpoint] = X
    EXIT : X found at midpt

	Time complexity	Space Complexity
Linear Search	$O(n)$	$O(1)$
Binary Search (recursive)	$O(\log n)$	$O(\log n)$
Bin. Search (Iterative)	$O(\log n)$	$O(1)$

X ——— X ——— X