NAME: JIGYASU DEO

Univ. Roll No. : 2015245

SECTION : M.L

CLASS Roll No : 32

SUBJECT : DESIGN AND ANALYSIS OF ALGORITHM

SUBJECT CODE : TCS 505

Assignment = 1

Ans 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 don't depend on machine-specific constants and doesn't require algorithms to be implemented and time taken by the programs to be compared.

Following are the asymptotic notations that are mostly used :-

i) Θ Notation : The theta notation bounds a function from above and below, so it defines exact asymptotic behaviour.

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

iii) Ω Notation : Ω Notation provides an asymptotic lower bound.

For Example Consider Insertion Sort

It takes linear time in best case and quadratic time in worst case.

∴ We can say that Insertion sort have

$O(n^2)$

$\theta(n^2)$ for worst case

$\theta(n)$ for Best case

$\Omega(n)$

Ans 2 = $\theta(\log n)$

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

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

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

$3^2T(n-2)$

$3^3T(n-3)$

$\vdots$

$3^nT(n-n)$

$= 3^n$

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

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

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

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

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

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

$= 2^nT(n-n)-2^{n-1}-2^{n-2}-2^{n-3}\ldots 2^2-2^1-2^0$

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

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

$T(n) = 1$

Ans 5 =

$S_i = S_{i-1} + i$

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

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

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

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

Ans 7 =

j loop is executing $\log n$ times

K " " " $\log n$ times

i " " " $n/2$ times $\quad n/2 \simeq n$

$\therefore$ Time Complexity = $O(n \log^2 n)$

Ans 8 = $O(n^3)$

Ans 9/15 = Inner loop will execute $\left(n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n}\right)$

$n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right)$

$\therefore$ It is equal to $\theta(n \log n)$

Ans 10 =

n^k $\quad$ a^n

$k >= 1$ $\quad$ $a > 1$

Taking $k = a = 2$

n^2 $\quad$ 2^n

∴ We can say $n^2 = O(2^k)$

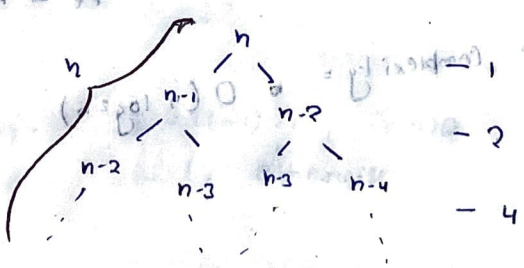
$n^k = O(a^n)$

Ans 11 = $O(\sqrt{n})$ Same logic which is given in Ques 5.

Ans 12 = Recurrence Relation

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

Making Recurrence Tree



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

$a = 1 \quad r = 2$

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

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

Space Complexity = O(n)

This is because maximum Stack frame is equal to n only as function is 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).

Ans 13:-

n log n

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

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

Ans 14 =

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

We can assume

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

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

Applying Master's Method

$$a = 2 \qquad b = 2$$

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

$$n^K = n$$

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

$$\therefore \text{It is } \theta(n^2)$$

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

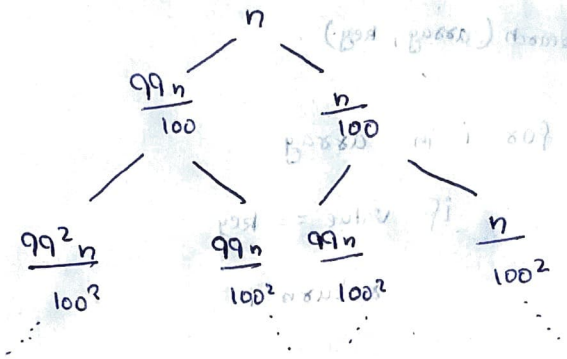
$$T(n) = O(n^2)$$

Ans 16 = if K is a constant greater than 1,

Then T.C = $O(\log \log n)$

Ans 17 =

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



If we take longer branch i.e $\frac{99n}{100}$

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

We can say that the base of ~~algori~~ log does not matter as it only a matter of constant.

Ans 18= a) 100 $\log\log n$ $\log n$ $\sqrt{n}$ n $\log n!$ $n\log n$ n^2 2^n $2^{2n}/4^n$ $n!$

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

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

Ans 19 =

```
Linear_Search (array, key)

for i in array

    if value == key

        return i
```

Ans 20 =

Iterative Insertion Sort

```
insertionSort (arr, n)

    Loop from i=1 to i=n-1

        Pick element arr[i] and insert it into
        Sorted Sequence arr[0... i-1]
```

Recursive Insertion Sort

```
insertionSort (arr, n)
{
    if n <= 1

        return

    recursively Sort n-1 element

        insertionSort (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 is called Online Sorting Algorithm

Ans 20/21/22=

Considering only 3 Sorting Algo. till now, as we get the lectures of these 3 only.

Algo	Best Case	Average Case	Worst Case	S.C	Stable	Inplace	Onli
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	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	✓	✓	✓

Ans 23=

binary. Search

A ← Sorted array

n ← Size of array

x ← Value to be Searched

while x not found

if upperbound < lowerbound

EXIT: x does not exist

Set midpoint = lowerbound + (upperbound - lowerbound)/2

if A[midpoint] < x

lowerbound = midpoint + 1

if A[midpoint] > x

upperbound = midpoint - 1

if A[midpoint] = x

EXIT: x found at midpoint

	Time Complexity	Space Complexity
Linear	O(n)	O(1)
Binary Search (Recursive)	O(logn)	O(logn)
Binary Search (Iterative)	O(logn)	O(1)

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