Assignment – 01

Name : DOMAN SARKAR

Sec : B

Roll no. : 23

univ. roll no. : 20146150

Q1 Asymptotic notations are mathematics 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 dont depend on machine. Specific constants and dosent requiret 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 & below, so its defines exact asympto behaviour.

(ii) Big O Notation : It defines and 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
$\Omega(n)$

Q2 $\Theta(\log n)$

Q3 $T(n) = \begin{cases} 3T(n-1) & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases}$

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

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

$3^2 T(n-2)$

$3^3 T(n-3)$

$\vdots$

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

Q4 $T(n) = \begin{cases} 2T(n-1)-1 & \text{, if } n \to 0 \\ 1 & \text{, otherwise} \end{cases}$

$T(n) = 2T_1(n-1) - 1$

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

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

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

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

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

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

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

$T(n) = 1$

Q5 $S = S + i$

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

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

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

Q6 $O(\sqrt{n})$

Q7 j loop is executing $\log n$ times

k " " " " "

i " " " $n/2$ " i.e $n/2 \simeq n$

Time complexity $= O(n \log 2n)$

Q8 $O(n^2)$

Q9 Inner loop will execute $(n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n})$

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

$\therefore$ its is equal to $O(n \log n)$

Q10

n^k	a^n
$k=1$	$a>1$
taking	$k=a=2$
n^2	2^n

$\therefore$ we can say $n^2 = O(2^k)$

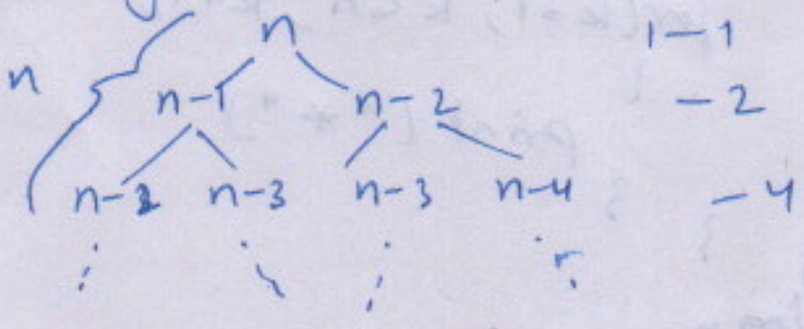
$\therefore n^k = O(a^n)$.

Q11 $O(\sqrt{n})$ (ans 5)

Q12 Recurrence Relation :-

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

making Recurrence tree



$= 1 + 2 + 4 + \cdots + 2^n$

$a = 1 \quad r = 2$

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

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

$\therefore$ 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)

Q13 n log n

```
for(i=1 ; i<n ; i++)
  {
    for( j=1 ; j<=n ; j=j+1)
      {
        print(" *")
      }
  }
```

n^3

```
for (i=1 ; i <n ; i++)
 { for(j=1 ; j<n; j++)
    {
     for(k=1; k <n ; k++)
       {
         print (" * ")
       }
    } }
```

log log n

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

Q14 $T(n) = T(n/4) + T(n/2)$

we can assume

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

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

applying masters Method

$a = 2 \quad b = 2$

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

$n^k = n$

$f(n) = n^2$

$\therefore$ if it is $\theta(n^2)$

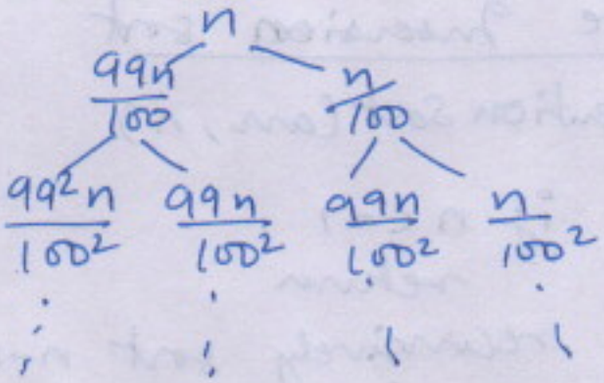
But as $T(n) <= \theta(n^2)$

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

Q16 If k is a constant greater than 1.

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

Q17 $T(n) = T\left(\frac{99n}{100}\right) + T\left(\frac{n}{100}\right)$



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

T.C. = $\log_{\frac{100}{99}} n \simeq \log n$.

we can say that base of log does not matter as it only a matter of constant

Q18 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, $\log^2 n$, n, 2^n, 4n, log n!, n log n, n^2, $2(2^n)$, n!

(c) 96, $\log 8^n$, $\log 2^n$, 5n, log n!, n $\log_{60} n$, n $\log_2 n$, $8n^2$, $7n^3$, 8^{2n}, n!

Q19 Linear search (array, key)

```
for i in array
    if value == key
        return i
return -1
```

Q20 Iterative insertion sort

```
insertion sort (array, 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

```
insertion Sort (arr, n)
{
    if n <=1
        return
    recursively sort n-1 element
        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 sol^n without considering future elements.

∴ it is called online sorting algorithm.

Q 20/21/22

considering only 3 sorting algo now, as we got got lectures

Algo Best case Avg case worst case S.C. stable

Algo	Best case	Avg case	worst case	SC.	stable	In place	online
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	✓	✓	X
selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	X	✓	X
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	✓	✓	✓

Q 23 Binary search

A ← sorted array
n ← size of array
X ← value to be searched.

while x not found
if upper bound < lower bound
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 complex	space complex
Linear	$O(n)$	$O(1)$
Binary search (recur.)	$O(\log n)$	$O(\log n)$
Binary Search (iterative)	$O(\log n)$	$O(1)$

(B)

Q24 $T(n) = T(n/2) + c$