Assignment 1 1910029

Date.

Ishika Jain Page No.

(1.) What do you mean by Asymptotic notations. Define different asymptotic notation with examples.

Asymptotic notations are methods using which we can define running time of the algorithm based on the input size.

Asymptotic notations are following:

i.) Big-O: It is for the worst case or the ceiling of growth for a given function.
ex. $f(n) = 3\log n + 100$, $g(n) = \log n$

ii) Big-Ω: It is for the best case or a floor growth rate for a given function.

iii-) Small-o :- It is used to denote upper bound on the growth rate of run time of algo.

iv.) Small-omega: It is used to denote lower bound on the growth rate of runtime of algo.

v.) Theta: It is used to denote the asymp-totically tight bound on growth rate of runtime of an algorithm.

Date.
Page No.

(2) What is TC of:

```
for(i=1 to n)
  { i=i*2; }
```

1	2	4	8	.....	n
i					

in GP,
$a=1, r=2$

$n = 1 \cdot 2^{k-1}$

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

$2n = 2^k$

$\log_2(2n) = k$

$k = \log_2 n$

$\boxed{TC = O(\log_2 n)}$

(3) $T(n) = 3T(n-1)$ if $n>0$, otherwise 1

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

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

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

$= 9T(n-2)$

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

$T(n) = 27T(n-3)$

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

$= 3^n T(0)$

$= 3^n$

$\boxed{TC = 3^n}$

Date.
Page No.

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

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

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

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

$T(n) = 4T(n-2) - 2 - 1$

$= 4T(n-2) - 3$

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

$T(n) = 4[2T(n-3) - 1] - 2 - 1$

$= 8T(n-3) - 2^2 - 2^1 - 2^0$

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

$= 2^n - 2^{n-1} - 2^{n-2} - \dots - 2^0$

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

$= 1$

$TC = O(1)$

(5.) TC of :

```
int i=1, s=1;
while (s<=n)
{ i++;
  s=s+i;
  printf("#");
}
```

n = 8

i = 1, 2, 3, 4

s = 1, 3, 6, 8

1 <= 8

i 1, 2, 3, 4

s = 1, 3, 6, 8

$TC = O(\sqrt{n})$

Date.
Page No.

6] TC ?

```
void function (int n)
{ int i, count=0;
  for(i=1; i*i<=n; i++)
    count++;
}
```

TC : $O\sqrt{n}$

(7.) TC ?

```
void function(int n)
{
  int i, j, k, c=0;
  for(i=n/2; i<=n; i++)
    for(j=1; j<=n; j=j*2)
      for(k=1; k<=n; k=k*2)
        count++;
}
```

i	j	k
n/2	$\log_2 n$	$\log_2 n$
n/2+1	2	2
	4	4
n	$\log n$	$\log n$

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

Date.
Page No.

(21.) Complexity of all sorting algo that has been discussed in class.

algorithm name	Time Complexity
Binary Search using Recurrence Relation	$T\left(\frac{n}{2}\right)+1$
Linear search	Worst case: $O(n)$ Best case: $O(1)$ Avg. case $O(n)$
Iterative Binary Search	Best Case: $O(1)$ Worst case: $O(n)$ Avg. case $O(n)$
Recursive Binary Search	Best case: $O(1)$ Avg. case: $O(\log(n))$ Worst case: $O(\log n)$

Date.
Page No.

(23) Write Recursive / Iterative pseudo code for binary search. What is TC & SC of linear & binary search (Recursive & iterative)

i) Recursive :

```
int bs( int a[], int l, int r, int x)
 {   if (r >= l)
       int mid = (l+r)/2;
     if (a[mid] == x)
       return mid;
     else if (a[mid] > x)
       return bs( a, l, mid-1, x);
     else
       return bs(a, mid+1, r, x);
 } return -1; }
```

Best case = O(1)
Worst case = O(log n)
Avg case = O(log n)

ii) Iterative :

```
int bs(int a[], int l, int r, int x)
{  while (l <= r)
   {  int mid = (l+r)/2;
   if (a[mid] == x) return mid;
    if (a[mid] < x)  l = m+1;
   else   r = m-1;
   } return -1;
   }
```

Best case = O(1)
Worst case = O(log n)
Avg case = O(log n)

Date.
Page No.

(24) Write Recurrence relation for binary Recursive Search.

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

$$T(1) = 1$$

$\Rightarrow T(n) = T\left(\frac{n}{2}\right) + 1$ — (I) ; $T(1) = 1$

putting $n = \frac{n}{2}$ in eq (I)

$$T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + 1 \quad \text{— (II)}$$

putting value of $T\left(\frac{n}{2}\right)$ of eq (II) in eq (I)

$$T(n) = T\left(\frac{n}{4}\right) + 1 + 1 \quad \text{— (III)}$$

putting $n = n/4$ in eq (I),

$$T\left(\frac{n}{4}\right) = T\left(\frac{n}{8}\right) + 1 \quad \text{— (IV)}$$

putting value of $T(n/4)$ of eq (IV) in eq (III)

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

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

Date.
Page No.

$$\frac{n}{2^k} = 1$$

$$n = 2^k$$

taking log both sides,

$$\log_2 n = k$$

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

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

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

$$= 1 + \log_2 n$$

$$\boxed{TC = \Theta(\log_2 n)}$$