Name: Ayush Chamoli

Roll no. 23 Assignment-1 Section: IAL

Ans 1. Asymptotic notations are mathematical tools used to describe the running behaviour of functions as their input size approaches infinity.

Most common Notations are:-

1. Big O notation*:- It represents the upper bound of a function. It provides an upper limit on the growth rate of a function.

2. Omega notation (Ω)*:- It represents the lower Bound of a function. It provides a lower limit on growth rate of function.

3. Theta notation (Θ)*:- It represents both the upper and lower bounds of a function.

Ans 2:- Time complexity of:-

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

So:-

```
for(i=1; i<=n; i=i*2)
{
    // loop Body
}
```

- So complexity will be O(logn).

Date / /

Ans3:- T(n) :-

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

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

Now substitute $T(n-1)$ with its relation:

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

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

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

$\Rightarrow 3^3 T(n-3)$

After K steps :-

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

continue till $n-K=0$ so

$n-K=0$

$K=n$

$T(n) = 3^n T(0)$

$T(n) = 3^n$

Ans4:- T(n) :-

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

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

Substitute $T(n-1)$:

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

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

Continue substitution:

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

After k steps:

$T(n) = 2^k T(n-k) - (2^{k-1} + 2^{k-2} + \cdots + 2 + 1)$

$n - k = 0$

$k = n$

So putting $k = n$

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

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

Ans:

```
Put i=1 , s=1;
while (s<=n) ---- (1+2+3+ ... +i)
{
    i++;
    s=s+i; ---- (i++)/2 > n
    printf("#");
}
```

$\Rightarrow T(n) = O(i^2)$

$\Rightarrow O(n^2)$ Ans.

Ans 6:-

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

loop runs till $\sqrt{n}$ times because $i = \sqrt{n}$, $i \times i = n$.

So

time complexity : $O(\sqrt{n})$ Ans.

Ans 7:-

```
void function (int n)
{
    int i, j, k, count = 0;
    for (i = n/2 ; i <= n ; i++) --> (n/2) times
        for (j = 1 ; j <= n ; j = j * 2) --> (log2 n) times
            for (k = 1 ; k <= n ; k = k * 2) (log2 n) times
                count++;
}
```

So Time complexity:

$O(n) \times O(\log n) \times O(\log n)$

$\Rightarrow O(\log^2 n)$.

8. function(int n)
{
if(n==1)
return;

for(i=1; i<=n; i++) ——— (n) times
{

for(j=1; j<=n; j++) ——— (n) times
{
printf("*");

}

}

function(n-3); → recursive call so (n-3)

}

So time complexity :

$O(n \times n \times$ recursion$(n-3))$

$\Rightarrow O\left(\frac{n}{3}\right)$ Ans.

$\Rightarrow O(n^2 \cdot n) \Rightarrow O(n^3)$

DOMS | Page No.
Date / /

9. void function (int n)
{
for (i=1 ; i<=n ; i++) ——— (n) times
{
for (j=1 ; j<=n ; j=j+i) ——→ (n/i) times
{
printf ("*");
}
}
}

So time complexity :-

$O(n) \times O(\frac{n}{i})$

loop runs i=1 to n

$O(n^2)$ Ans.

Ans:- The asymptotic notations b/w n^k and c^n is as follows:

n^k denotes polynomially with n, while c^n denotes exponentially. The exponential denotes dominates as n increases.

Especially c^n denotes faster than n^k for sufficiently large values of n.

$$c^n > n^k$$

Taking the logarithm base c on both sides.

$$n > \log_c(n^k) = k \cdot \log_c(n)$$

Now, we can solve for n in terms of c and k.

$$n > \frac{k}{\log_c(k)}$$

So the value of c and no that holds the relationship is $c > k/\log_c(k)$ and

$$n_0 = \frac{k}{\log_c(k)}$$ Ans.