Date / /

Design and Analysis of Algorithms

Assignment - 1

Q) What do you understand by Asymptotic notations. Define different Asymptotic notation with examples.

Ans) Asymptotic Notation is used to describe the running time of an algorithm - how much time an algorithm takes with a give input, n. There are three different notation: big O, big Theta (Θ), and big Omega (Ω).

There are three different notations -

① Big-O Notation → The Big-O notation describes the worst-case running time of a program. We compute the Big-O of an algorithm by counting how many iterations an algorithm will take in the worst-case scenario with an input of N

② Big-Ω Notation → Big-Ω (omega) describes the best running time of a program. We ~~cannot~~ compute the big Ω by counting how many iterations an algorithms will take in the best-case scenario based on an input of N.

③ B-Θ Notations → Notation specifics asymptotics bound (both upper and lower) for a function f(n) and provides the average time Complexity of an algorithm.

Date / /

Q2 What should be time complexity of - for (i=1 to n) { i = i*2; }

$O(\log_2 n)$

Q3 $T(n) = \{3T(n-1)$ if $n>0$, otherwise $1\}$

$T(0) = 1$

$T(n-1) = \{3T(n-1-1)\}$

$= \{3T(n-2)\}$ — (2)

Putting equation (2) to (1)

$T(n) = \{3(3T(n-2))\}$ — (3)

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

$= \{3T(n-3)\}$ — (4)

Putting eq (4) to (3)

$T(n) = \{9(3T(n-3))\}$

$= \{3^2 T(n-3)\}$ — (5)

* Generalized formula →

$T(n) = \{3^K T(n-K)\}$

let $(n-K) = 0$

$K = n$

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

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

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

$T_n = O(3^n)$

n = no. of terms

If $r > 1$,

$a\left(\frac{r^n - 1}{r-1}\right)$

sum of G.P

Date / /

Q4 $T(n) = \{2T(n-1) - 1$ if $n>0$, otherwise $1\}$

$T(n) = \{2T(n-1) - 1\}$ — (1)

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

$= 2T(n-2) - 1$ — (2)

Putting eq (2) in (1)

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

$= (2^2 T(n-2) - 2) - 1$ — (3)

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

$= 2T(n-3) - 1$ — (4)

Putting eq (4) in (3)

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

$= 2^3 T(n-3) - 4 - 2 - 1$

Generalized formula :-

$T(n) = 2^K T(n-K) - 2^{K-1} - 2^{K-2} - 2^{K-1}$

$n - K = 0$

$n = K$

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

~~$T(n) = 0(2^n)$~~ ~~Ans.~~

~~$T(n) = 0(2^n)$~~

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

* $[2^{n-1} + 2^{n-2} + \cdots + 2^0 = 2^n - 1]$ =

$T(n) = 1$

Date / /

Qs What should be time complexity of -

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

After 1st iteration

$s = s+1$

After 2nd

$s = s+1+2$

After m iterations

$1+2+3+\cdots+m <= n$

$\frac{m \times (m+1)}{2} <= n$

$\frac{m^2+m}{2} <= n$

$m^2 <= n$

$m <= \sqrt{n}$

$T.C = O(\sqrt{n})$

Q7 Time Complexity of -

```
Void function (int n)
{
    int i, j, k, count = 0;
n/2 ——  for (i=n/2 ; i<=n ; i++)
    {
log n. n/2      for (j=1 ; j<=n ; j=j*2)
        {
log n. log n. n/2       for (k=1 ; k<=n ; k=k*2)
            {
                count++;
            }
        }
    }
}
```

$\frac{n}{2} * (\log n + 1) * (\log n + 1)$

$T.C = \#\left(\frac{n}{2} \log_2 n \cdot \log_2 n\right) = (n (\log n)^2)$ Ans

Q8 Time complexity of -

```
function (int n) —— T(n)
{
    if (n==1)
        return;
    for (i = 1 to n) {  ——> n+1
        for (j = 1 to n) {  ——> n
            printf ("*");
        }
    }
    function (n-3); —— T(n-3)
}
```

Date / /

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

Since 2 is the highest power
$\therefore T.C = O(n^2)$

Q6 Time complexity of —

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

when $i=1$, $i*i = 1 <= 5$, count = 1
let $n=5$ $i=2$, $i*i = 4 <= 5$, count = 2
$i=3$, $i*i = 9 \not<= 5$

After iterations

$i^2 <= n$

$i <= \sqrt{n}$

$\therefore T.C = O(\sqrt{n})$ Ans

Date / /

Q9 Time complexity of -

```
void function (int n)
{
    for (i = 1 to n)          ---- n
    {
        for (j = 1 ; j <= n ; j = j + i)
        {
            printf ("*");
        }
    }
}
```

for $i = 1$, loop runs n times.
$i = 2$, " " $n/2$ "
$i = 3$, " " $n/3$ "
⋮
$i = n$, " " $n/n = 1$ times.

$\therefore T.C = n + n/2 + n/3 + \cdots + n/n$

$= n (1 + 1/2 + 1/3 + 1/4 + \cdots + 1/n)$

$= n * \log (n)$

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