Date.
Page No.

Design and Analysis of Algorithms

Tutorial - 1

1. ~~what~~ Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis.

↳ Θ notation : It bounds a function from above & below so it defines exact asymptotic behaviour.

$\Theta(g(n)) = \{f(n)$: there exists positive constants c_1, c_2 and n_0 such that $0 <= c_1 * g(n) <= f(n) <= c_2 * g(n)\ \forall\ n \geq n_0\}$

↳ Big O :- It defines an upper bound of an algorithm it bounds function only from above.

$O(g(n)) = \{f(n)$: There exists +ve constant c and n_0 such that $0 <= f(n) \leq c * g(n)\ \forall\ n >= n_0\}$

↳ Ω notation : It provides asymptotic lower ~~and~~ bound. It can be useful when we have lower bound on time complexity of an algorithm.

$\Omega(g(n)) = \{f(n)$: There exists +ve constants c and n_0 such that $0 <= c * g(n) <= f(n)\ \forall\ n >= n_0\}$

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

$i = 1, 2, 4 \ldots n \rightarrow GP$

$t_k = ar^{k-1}$ $[a = 1, r = 2]$

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

$\log_2 n = k - 1$

$k = \log_2 n + 1$

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

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

~~T(n)~~ $n = n-1$ in (1)

$T(n-1) = 3T(n-2)$ — (2)

put (2) in (1)

$T(n) = 3[3T(n-2)] = 9T(n-2)$ — (3)

put $n \to n-2$ in (3)

~~T(n)=9(T(n-2~~ $T(n-2) = 3T(n-2-1)$

$T(n-2) = 3T(n-3)$ — (4)

Put value of $T(n-2)$ in ~~T~~ (3)

$T(n) = 9[3T(n-3)] = 27(n-3) = 3^k(n-3)$

⋮

k

Assume $n-k=0$

$n = k$

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

$= 3^n T(0)$

$T(n) = 3^n$

$\boxed{T(n) = O(3^n)}$

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

$T(n) = 2T(n-1) - 1 \quad — (1)$

$T(0) = 1$

put $n \Rightarrow n-1$

$T(n-1) = 2T(n-1-1) - 1 = 2T(n-2) - 1 \quad —(2)$

put value of $T(n-1)$ in (1)

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

$= 2^2 T(n-2) - 2 - 1 \quad — (3)$

put $n \to n-2$ in (1)

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

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

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

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

$\vdots$

k

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

Assume $n-k=0 \Rightarrow n=k$

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

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

$= -\underbrace{[1 + 2 + 2^2 + \cdots 2^{n-1}]}_{GP} + 2^n$

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

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

$2^n[-2^{-1} + 1] + 1$

$T(n) = \cancel{O(2^n)} \quad O(1)$

5) int i = 1, s = 1;

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

This loop will work till $s <= n$, $S = 1$, $i = 1$

After 1 iteration, $S = 2$, $i = 2$

" 2 ", $S = 3+3=6$, $i = 3$

$S = 6+4 = 10$, $i = 4$

Let this run till k times

so $S = \frac{(k)(k+1)}{2} > n$

$k^2 > n$

$k > \sqrt{n}$

$\therefore$ Time complexity $= O(\sqrt{n})$

6)

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

Loop will run till $i*i > n$

$i^2 > n \Rightarrow i > \sqrt{n^2}$

Time complexity $= O(\sqrt{n})$

7)

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

```
for( k=1; k<=n ; k=k*2)   → logn
    count++
```

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

8.
```
void function (int n)  —— T(n)
{ if (n==1) return;    → constant Time  T(1)
  for( i = 1 to n)     ——  — n Times
  {  for (j= 1 to n) {
        printf("*");   ——  n Times
     }
     function(n-3);    —— T(n-3)
  }
}
```

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

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

9.
```
void function ( int n) {
  for( i=1 to n) {                 —— n Times
    for (j=1 ; j<=n ; j=j+i )
        printf("*");               → executes j times with
    }                                j increase by i
  }
```

Inner loop executes n/i times for each value of i
$\therefore$ $\log n$ times

Time Complexity = $O(n\log n)$

Date
Page No.

10) The asymptotic relationship b/w the functions n^k and a^n is

$$n^k = O(a^n) \qquad k >= 1, \; a > 1$$

$$n^k \leq c \cdot a^n \qquad \forall \; n \geq n_0$$

$$\frac{n^k}{a^n} \leq c$$