**Design and Analysis of Algorithms**

**Tutorial\_1**

**Q1:** **What do you understand by Asymptotic notations? Define different Asymptotic notation with examples.**

**Ans:** Asymptotic Notations. Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

Big-O Notation (O-notation) Big-O notation represents the upper bound of the running time of an algorithm. Thus, it gives the worst-case complexity of an algorithm.

f(n) = O(g(n)) iff there are two positive constants c and n0 such that |f(n)|≤ c\*|g(n)|for all n ≥ n0

Example:

n2+ n = O(n3)  
Here, we have  
f(n) = n2 + n,  
g(n) = n3

Omega Notation (Ω-notation) Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best-case complexity of an algorithm.

 f(n) = Ω(g(n)) if there are two positive constants c and n0 such that |f(n)|≥ c\*|g(n)|for all n ≥ n0.

Example:

* n3 + 4n2 = Ω(n2)
* Here, we have  
  f(n) = n3 + 4n2,  
  g(n) = n2

Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analysing the average-case complexity of an algorithm.

f(n) = Θ(g(n)) if there are three positive constants c1, c2 and n0.

Example:

n2 + 5n + 7 = Θ(n2) Where f(n) = n2 + 5n + 7  
g(n) = n2  
When n ≥ 1,  
n2 + 5n + 7 ≤ n2 + 5n2 + 7n2 ≤ 13n2  
When n ≥ 0,  
n2 ≤ n2 + 5n + 7  
Thus, when n ≥ 1  
1n2 ≤ n2 + 5n + 7 ≤ 13n2  
Thus, we have shown that n2 + 5n + 7 = Θ(n2) (by deﬁnition of Big-Θ, with n0 = 1, c1 = 1, and c2 = 13.)

**Q2. What should be time complexity of –**

**for (i=1 to n)**

**{**

**i=i\*2;**

**}**

Ans: T(n) = O(n)

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

Ans: T(n) = 3T(n-1)

= 3(3T(n-2))

= 32T(n-2)

= 33T(n-3)

...

...

= 3nT(n-n)

= 3nT(0)

= 3n

This clearly shows that the complexity

of this function is O(3n).

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

Ans: T(n) = 2T(n-1) - 1

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

= 22(T(n-2)) - 2 - 1

= 22(2T(n-3)-1) - 2 - 1

= 23T(n-3) - 22 - 21 - 20

.....

.....

= 2nT(n-n) - 2n-1 - 2n-2 - 2n-3

..... 22 - 21 - 20

= 2n - 2n-1 - 2n-2 - 2n-3

..... 22 - 21 - 20

= 2n - (2n-1)

T(n) = 1

Time Complexity is O(1).

**Q5. What should be time complexity of –**

**int i=1, s=1;**

**while(s<=n)**

**{ i++; s=s+i;**

**printf(“#”);**

**}**

Ans: I will go on i = 1,2,3-----k

s = 3,6,10,15----

while loop will terminate if : 1+2+3+ ------- +k

[k(k+1)/2] > n

So, k = O(sqrt n)

T(n) = O (sqrt n)

**Q6. Time complexity of –**

**void function(int n)**

**{**

**int i, count= 0;**

**for (i=1; i\*i<=n; i++)**

**count++**

**}**

Ans: As the statement is valid for n/10 terms

So, T(n) = O(n/10)

ie: T(n) = O(n)

**Q7. Time complexity of –**

**void function(int n){**

**int i, j, k, count=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++**

**}**

Ans: T(n) = O(nlog2n)

**Q8. Time complexity of –**

**function(int n){**

**if(n==1) return;**

**for(i=1 to n){**

**for(j=1 to n)**

**{**

**printf(“\*”);**

**}**

**}**

**function(n-3);**

**}**

Ans: T(n) = O(n)

**Q9. Time complexity of –**

**void function(int n){**

**for(i=1 to n){**

**for(j=1; j<=n; j=j+i)**

**printf(“\*”)**

**}**

**}**

Ans: T(n) =O (n2)

As n times for I loop and n times for j loop.

**Q10. For the functions, n^k and c^n, what is the asymptotic relationship between these functions? Assume that k >= 1 and c > 1 are constants. Find out the value of c and n0 for which relation holds.**

Ans: **n^k is O(c^n)**

Ie: n^k <= c^n

Taking log on both sides we get;

K logn < = n logc

In this case the relation will for values of k and c to be n.