DAA ASSIGNMENT

Name : Aashi Bansal
Course : B-Tech
Semester : 5
Subject : Design & Analysis of Algorithms
Subject code : TCS 505
Branch : CSE

Q.1 → What do you understand by Asymptotic notations? Define different asymptotic notation with example?

Ans → Asymptotic notation are the mathematical notation used to describe the running time of an algorithm where the input tends towards a particular value or a limiting value. Asymptotic notation is a way to compare function that ignores constant factor & small input sizes.

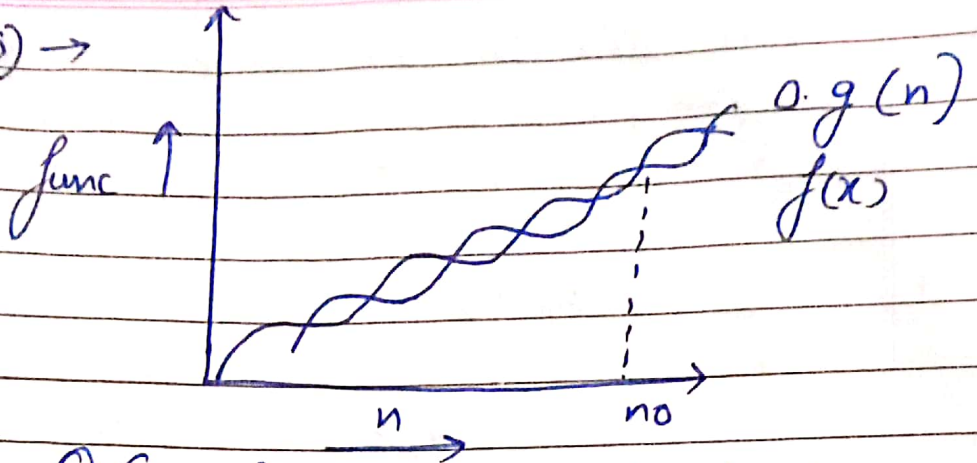
Types of Asymptotic Notations:

1). Big theta (Θ): Tight bound, complexity represented is like average value or range within with which the actual time of execution will be.

2). Big Oh (O): This is used for upper bound of algorithm or worst case of an algorithm. It tells that a function will never exceed specified time for any value of input n.

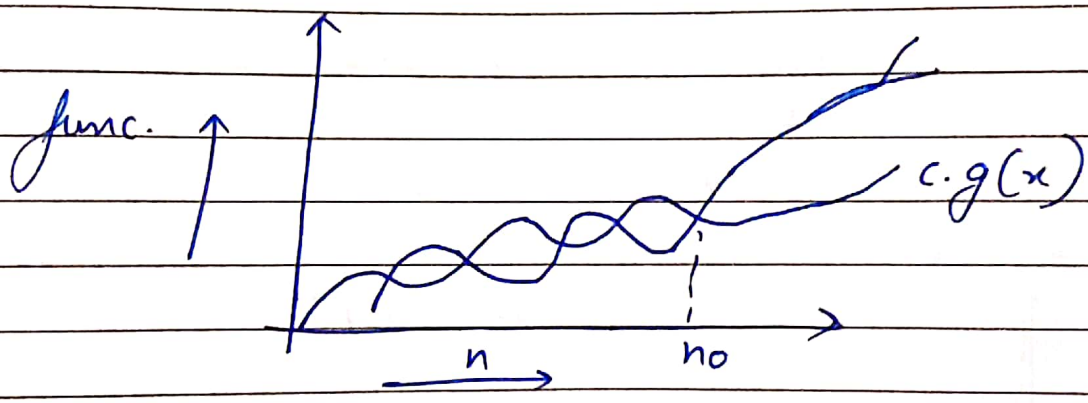
3). Big Omega (Ω): Used to define lower bound of any algorithm or best case of a algorithm.

1st Big OH (O) →



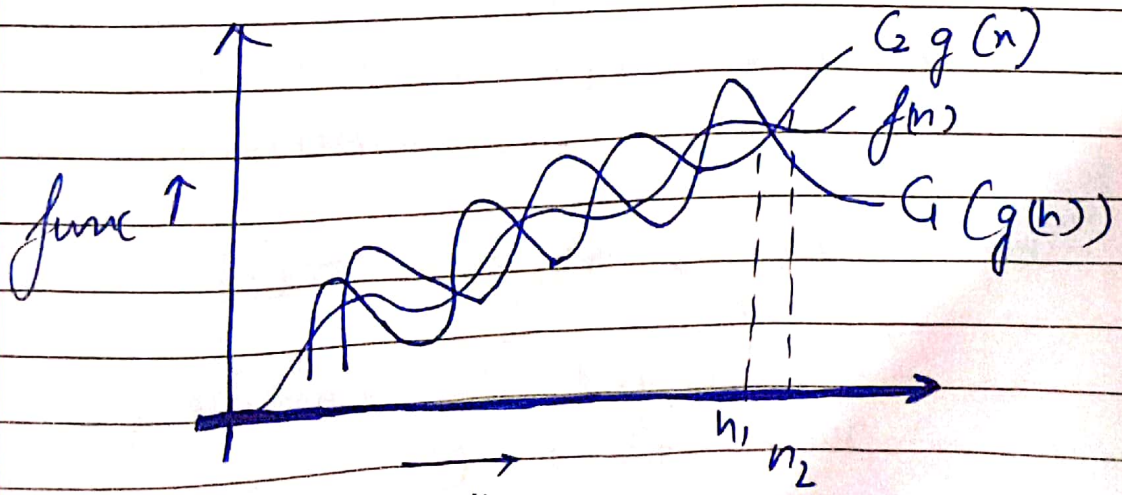
$f(n) = O(g(n))$ g(n) is tight upper bound

2nd Big Omega : (Ω)



$f(x) = \Omega(g(x))$

3rd Theta (Θ) :- Gives both upper and lower bound



$O(g(n)) = f(n)$

Q.2 What should be complexity of

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

→ $\sum_{i=1}^{n}$ Step * 2

1, 2, 4, 8 ..... n (k terms)

$\Rightarrow 2^0, 2^1, 2^2, 2^3 \ldots\ldots = n$

taking log

$\log (2^{(k-1)}) = \log n$

$k-1 = \log n$

$k = \log n + 1$

$= O(\log n)$

Q.3 → $T(N) = \{ 3T(n-1)$ if $n>0$ otherwise $1 \}$

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

$T(n) = 3T(n-1)$ if $n>0$ — (1)

put $n = (n-1)$

$T(n-1) = 3T(n-2)$ — (2)

putting (2) in (1)

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

$= 3^2 T(n-2)$ — (3)

putting $n = n-2$ in eq. (1)

$$T(n-2) = 3T(n-3) \quad \text{---(4)}$$
$$T(n) = 3^3 T(n-3) \quad \text{---(5)}$$
$$T(n) = 3^k T(n-k) \quad \text{---(6)}$$

Best case $\Rightarrow$ $T(0) = 1$

$$n-k = 0$$
$$n = k$$

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

Q.4 $\rightarrow$ $T(n) = \{2T(n-1)-1$ if $n>0$ Otherwise $1\}$

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

$$T(n) = 2T(n-1) - 1$$
$$= 2(2T(n-2)-1)-1$$
$$= 2^2(T(n-2))-1-1$$
$$= 2^2(2T(n-3)-1)-2-1$$
$$= 2^3 T(n-3) - 2^2 - 2^1 - 2^0$$
$$= 2^n T(n-n) - 2^{n-1} - 2^{n-2} \ldots\ldots 2^1 - 2^0$$
$$= 2^n - (2^n - 1)$$
$$\Rightarrow 2^n - 2^n + 1 = 1$$
$$T(n) = 1$$

Q.5 → What should be time complexity ?

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

⇒ 1, 3, 6, 10 ---- n terms

$S = 1+3+6+10+\cdots+K$

$O = 1+2+3+4+\cdots+K$

$\frac{K(K-1)}{2}$

$n \simeq K^2$

$K = \sqrt{n}$

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

Q.6 → Time complexity of :

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

⇒ $1, (2)^2, (3)^2, (4)^2 \cdots n$

$1, 2, 3, 4 \cdots \sqrt{n}$

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

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++;
    }}}
```

$$\Rightarrow \sum_{i=n/2}^{n} \sum_{j=1}^{j*2} \sum_{k=1}^{n} 1$$

$$\Rightarrow \sum_{i=n/2}^{n} \sum_{j=1}^{n} \log(n)$$

step $j*2$

$$\Rightarrow \sum_{i=n/2}^{n} (\log(n))^2$$

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

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

Q.8 → Time complexity of:

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

$$\Rightarrow \sum_{j=1}^{n} \sum_{j=1}^{n} 1 \Rightarrow \sum_{i=1}^{n} n = n^2$$

$$\Rightarrow T(n) = T(n-3) + n^2 \quad \text{——— (1)}$$

$T(n-3) = T(n-6) + (n-3)^2$ — (1)

$T(n-6) = T(n-9) + (n-6)^2$ — (2) $(n = n-3)$

$T(n) = T(n-6) + n^2 + (n-3)^2$ — (3)

putting (2) in eq. (1)

putting $n = n-6$ in eq. (1)

$$T(n-6) = T(n-9) + (n-6)^2 \quad \text{— (4)}$$

putting (4) in (3)

$T(n) = T(n-9) + n^2 + (n-3)^2 + (n-6)^2$

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

$T(1) = 0$

$n - 3k = 1$

$k = \frac{n-1}{3}$

$T(n) = n^2 + (n-3)^2 + \cdots + (n-k)^2$

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

Q.9 → Time complexity of:

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

Ans

$$\sum_{i=1}^{n} \sum_{j=1}^{n} = 1$$

step i

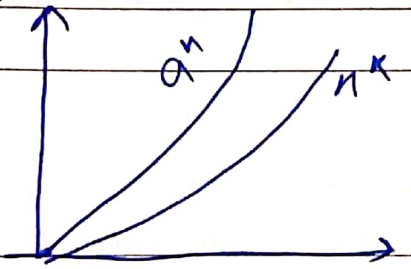
$$\sum_{i=1}^{n} \left(\frac{n-1}{i} + 1\right) \Rightarrow (n-1) \sum_{i=1}^{n} \frac{1}{i} + \sum_{i=1}^{n} 1$$

$$\Rightarrow (n-1) \sum_{i=1}^{n} \frac{1}{i} + n$$

$$(n-1) \log n + n$$

$$\boxed{T = O(n \log n)}$$

Q.10 → for function n^k and a^n what is asymptotic relationship between them? find out value of c and n_0 for which relationship holds.



$n^k = O(a^n)$

$n^k \leq a^n \cdot c$ $\forall$ $c > 0$ & $n \geq n_0$

let $n = n_0$

$n_0^k \leq c \cdot a^{n_0}$

$n_0^3 \leq c \cdot 3^{n_0}$

$K = a = 3$ (say)

$\Rightarrow c \geq 1$ & $n_0 \geq 1$

Q.11 What is time complexity —

```
void fun (int n) {
    int j = 1, i = 0;
    while (i < n) {
        i = i + j;
        j++; i }}
```

Ans

$S = 0, 1, 3, 6, 10, 13 \ldots\ldots K$ ①

$S = 0, 1, 3, 6, 10 \ldots\ldots n$ ②

J	i
1	0
2	1
3	3
4	6
5	10
6	15
7	21

Sub ② from ①

$0 = 0, 1, 2, 3, 4, 5 \ldots\ldots K - n$

$n = \frac{K(K-1)}{2}$

$n = 0 + 1 + 2 + \ldots\ldots K$

$n \simeq K^2$

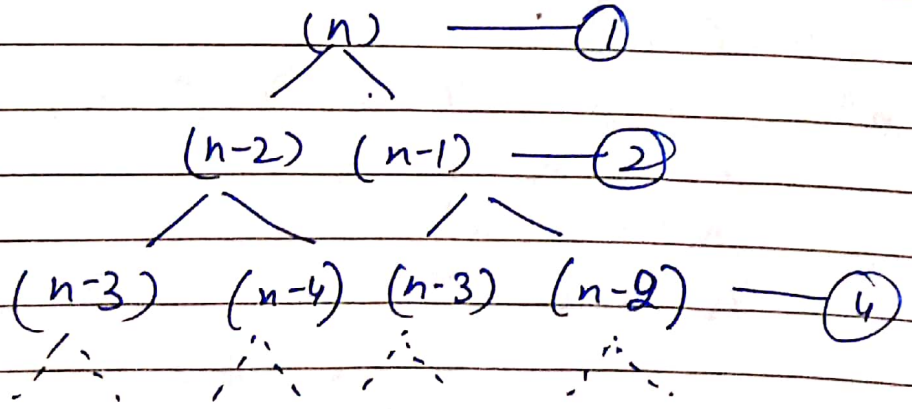
$K = \sqrt{n}$

$\boxed{T(n) = O(\sqrt{n})}$

Q.12 Write recurrence relation for recursive function that prints fibonacci series find time & space complexity.

0, 1, 1, 2, 3 ..... Tn

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



$T = 1 + 2 + 4 + 8 \ldots\ldots 2^n$

$a = 1, r = 2$

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

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

$T(n) = O(2^n)$

Space complexity = O(n) because max stack frame is same as the longest node

Q.13 → Write time & space complexity – $n(\log n)$, n^{-3}, $\log(\log n)$?

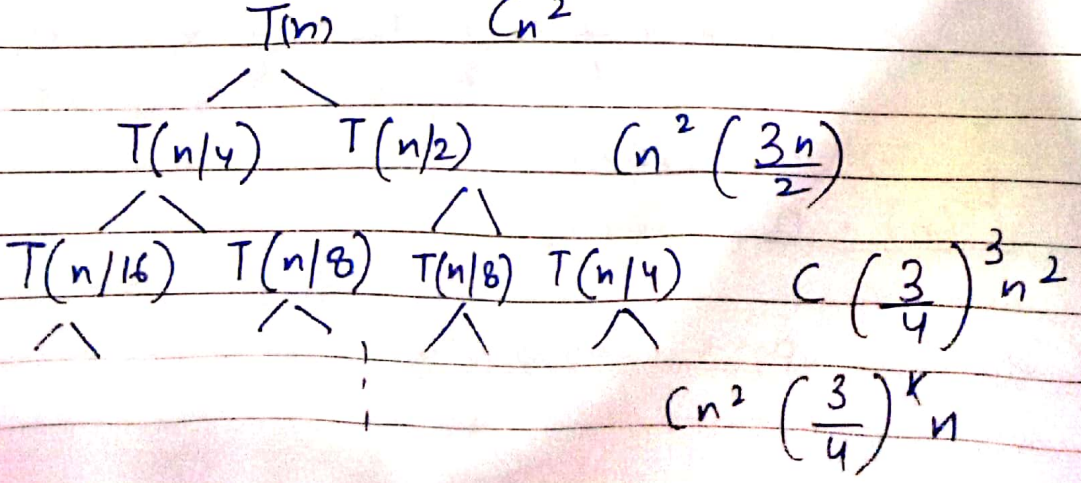
(i)
```
for (int i = 0; i <= n; i++)
{  for (int j = 1; j <= n; j *= 2)
   {  printf ("*");
   }}
```
Ans $O(n \log n)$

(ii)
```
int func (int n)
{  if (n <= 2)
      return 1;
   else
      return (func(floor (sqrt (n)) + n);
}
```
Ans $\log(\log n)$

(iii)
```
for (int i = 0; i < n; i++)
   for (int j = 0; j < n; j++)
      for (int k = 0; k < n; k++)
         print ("*");
```
Ans : n^3

Q.14 Solve following reoccurrence relation:

$$T(n) = T(n/4) + T(n/2) + cn^2$$



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

$$n = 2^k$$

$$\boxed{k = \log n}$$

$$T(n) = cn^2\left[1 + \frac{3}{4} + \left(\frac{3}{4}\right)^2 + \cdots\cdots \left(\frac{3}{4}\right)^{\log n}\right]$$

$$= cn^2(1)$$

$$= n^2$$

$$\boxed{T(n) = O(n^2)}$$

Q.15 find the time complexity of:

```
int func(int n) {
    for (int i=1; i<=n; i++) {
        for (int j=1; j<n; j+=i)
    }}
```

Ans

$$T(n) = \sum_{i=1}^{n} \sum_{j=1}^{n-1}$$

$$\Rightarrow \sum_{i=1}^{n} \frac{n-1}{i}$$

$$\Rightarrow (n-1) \sum_{i=1}^{n} \frac{1}{i}$$

$$\Rightarrow (n-1)\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots n\right)$$

$$\Rightarrow (n-1)\log n$$

$$\boxed{T(n) = O(n \log n)}$$

Q.16 find the time complexity:

```
for (int i = 2; i <= n; i = pow(i,k))
    { some O(1) expression }
```

Ans $i = 2, 2^k, 2^{k^2}, 2^{k^3} \dots 2^{k^k}$

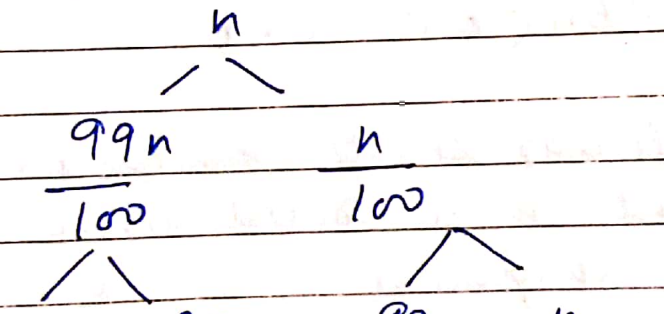
$$\log(n) = k^x \log 2$$

$$\log_k(\log(n)) = k \log k$$

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

Q.17 Write a recurrence relation for:

$$T(n) = T\left(\frac{99}{100} n\right) + T\left(\frac{n}{100}\right)$$



If we take longer branch i.e. $\frac{99n}{100}$

Time complexity $\Rightarrow \log_{\frac{100}{99}} n = \log n$

$$\left(\frac{99}{100}\right)^k n = 1$$

$$n = \left(\frac{100}{99}\right)^k \qquad K = \log_{\left(\frac{100}{99}\right)} n$$

$$T(n) = \frac{n\left(\log_{\frac{100}{99}} n\right)}{100} \qquad K = \log_{\left(\frac{100}{99}\right)} n$$

$$\Rightarrow T(n) = O(n \log n)$$

Q.18 → Increasing Order of growth:

(a) $1 < \log\log n < \log n < \sqrt{n} < n < n\log n < n^2 < 2^n < 2^{2n} < 4^n < n!$

(b) $1 < \log\log n < \sqrt{\log(n)} < \log(n) < 2n < 4n < 2(2^n) < \log(2n) < 2\log(n) < n < n\log n = \log(n!) < N!$

~~(c) $96 < \log 8^n < n\log_6 n = n\log_2 n <$~~

(c) $96 < \log_2(n) = \log_8(n) < n\log_6(n) = n\log_2 n = \log(n!) < 5n < 8n^2 < 7n < 8^{2n}$

Q.19 → Write linear search pseudocode to search an element in a sorted array with minimum comparison.

Ans

```
for( i=0 to k-1)
  { if ( arr[i] = key )
      { return i;
      }}
  return -1;
```

Q.20 a) Write pseudo code for iterative insertion sort.

```
insertion sort (arr, n)
loop from i=1 to i=n-1
pick element arr[i] & insert it into
sorted sequence
```

arr [0 ---- i-1]

Code
```
void InsertionSort(int arr[], int n)
{   Put int i, temp, j;
    for(i=1 to n-1) {
        temp = arr[i]
        j = i-1;
        while (j >=0 AND arr[j] > tmp)
        { arr[j+1] = arr[j];
          j = j-1;
        }
        arr[j+1] = tmp;
    }
}
```

(ii) Recursive Insertion Sort

```
insertionSort(arr, n)
{   if n <= 1
        return
    recursively sort n-1 element
    insertionSort(arr, n-1)
    Pick last element arr[i] and insert
    it into sorted arr[0 ---- i-1]
}
```

Code :
```
void InsertionSort(int arr[], int n)
{   if (n>2)
        return;
    InsertionSort(arr, n-1);
    last = arr[n-1], j = n-2;
    while (j >=0 AND arr[j] > tmp) {
        arr[j+1] = arr[j];
```

```
        j = j - 1;
    }
    arr[j+1] = last;
}
```

⇒ Insertion sort considers one input element per iteration & produces a partial solution without considering future elements. It is also called online sorting algorithm.

Q.21 Complexities of all sorting algorithms

	Algo	Best case	Avg. Case	Worst case
(1)	Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
(2)	Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
(3)	Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
(4)	Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
(5)	Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
(6)	Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Q.22 Divide all sorting algorithm inplace/stable/online

	Algorithms	Inplace	Stable	Online
(1)	Bubble sort	✓	✓	✗
(2)	Selection Sort	✓	✗	✗
(3)	Insertion Sort	✓	✓	✓
(4)	Merge Sort	✗	✓	✗
(5)	Quick Sort	✗	✗	✗
(6)	Heap Sort	✓	✗	✗

Q.23 Iterative Binary Search

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

Time complexity
Best = O(1)
Avg = O(log n)
Worst = O(log n)
Space ⇒ O(1)

Recursive Binary Search

```
int BinarySearch (int arr[], int l, int r, int x)
{   if (l > r)
        return -1
    int m = (l+r)/m;
    if (arr[] = n)
        return m;
    else if (arr[m] < n)
        return BinarySearch(arr, m+1, r, x);
    else
        return BinarySearch(arr, l, m-1, x)
}
```

Time complexity	Space complexity
Best → O(1)	O(1)
Average → O(log n)	O(log n)
Worst → O(log n)	O(log n)

Q.24 $T(n) = T(n/2) + 1$

$$\boxed{T(n) = O(\log n)}$$