Name : Sahil Thapa
Section : Data Science :
University Roll no : 2015189

Assignment - 1

Q no 1 : Asymptotic Notation :

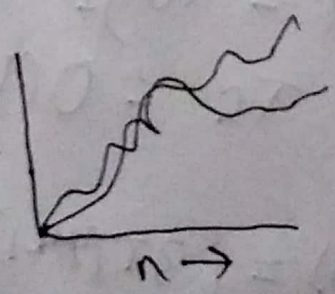
These notation are used to tell the time complexity of algorithm when input is very large :

Types :

① Big - oh (O) :

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

$g(n)$ is tight upper bound

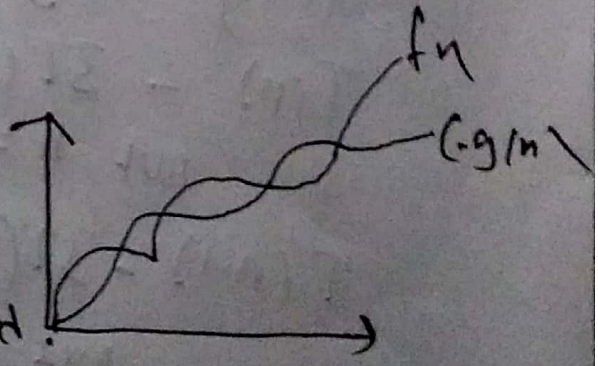


Algo 1 = $O(n^2 + 3n + 4) = O(n^2)$

Algo 2 = $O(2n^2 + 4) = O(n^2)$

② Big omega (Ω)

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



It gives tight ~~upper and~~ lower bound

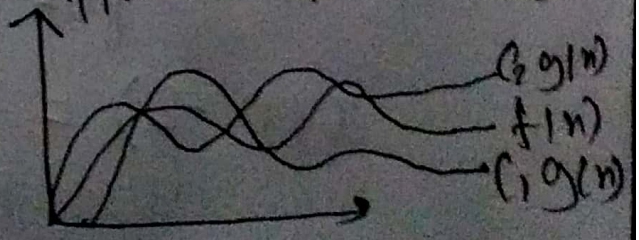
After n_1, $f(n)$ will never perform better than $g(n)$

③ Theta (Θ) : This gives tight upper and lower bound.

$f(n) = \Theta(g(n))$:

if $c_1 g(n) \leq f(n) < c_2 g(n)$

$\forall n \geq \max(n_1, n_2)$



Qno-2

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

It is GP:

$a = 1, \quad r = \frac{y}{2} = 2$

k^{th} term $= a \cdot r^{k-1}$

$n = 2^{k-1}$

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

$k = \log_2 n + 1$

$\therefore \quad TC = O(\log_2(n))$

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

Solving using backward substitution:-

$T(n) = 3T(n-1)$ ——— (1)

put $n = n-1$ in eqn (1)

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

put $T(n-1)$ in eqn (1)

$T(n) = 3 \cdot 3T(n-2)$ ——— (2)

put $n = n-2$ in eqn (1)

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

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

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

$3^3 T(n-3)$ ——— (3)

$T(n) = 3^k T(n-k)$ ——— (4)

put $n-k = 0$

$k = n$

put k in eqn (4)

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

$= 3^n T(0)$

$T(n) = 3^n$

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

Q no: 4 $T(n) = \begin{cases} 2T(n-1) - 1 & \text{if } n > 0, \text{otherwise } 1 \end{cases}$.

$T(n) = 2T(n-1) - 1$ ——— (1)

put $n = n-1$ in eqn (1)

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

put $T(n-1)$ in eq (1)

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

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

$T(n) = 2^2 T(n-2) - 1$ ——— (2)

put $n = n-2$ in eqn (1)

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

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

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

$\vdots$

$T(n) = 2^k T(n-k) - 2^{k-1} - 2^{k-2} - 2^{k-3} \ldots - 2^0$ ——— (4)

put $n-k = 0$

$k = n$ — in (4)

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

$= 2^n - 2^{n-1} - 2^{n-2} \ldots 2^1 - 2^0$

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

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

$= 1$

$T(n) = O(1)$

Qno:-5 let $n = 15$

i.e.

$n = 1 + 3 + 6 + \ldots + k$

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

$k = \sqrt{n}$

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

$i = 1, S = 1$

$i = 2, S = 3$

$i = 3, S = 6$

$i = 4, S = 10$

$i = 5, S = 15$

The increment is linear, but in S it is sum of first 5

Qno-6

let n = 15 :

The value of i is increasing with its square so for value of K

i = 1 <= n
i = 4 <= n
i = 9 <= n
i = 16 <= n (x)

The last point will be mean to k^2

i.e. $1^2\ 2^2\ 3^2\ 4^2 \dots k^2$.

$k^2 = n$

$k = \sqrt{n}$

$\therefore TC = O(\sqrt{n})$

Qno-7

The 1th loop will run n/2 times.
The jth loop will run $\log_2(n)$ times
The kth loop will run $\log_2(n)$ times.

$T.C = O(\frac{n}{2}) + O(\log_2(n) * O(\log_2 n)$

$= \frac{1}{2} O(\frac{n}{2} * (\log_2(n))^2$

$= O(n (\log_2(n))^2)$

Qno-8

Recurrence relation $T(n) = T(n-3) + n^2$

$T(1) = 1$

Q no-9

$T(n) = T(n-3) + n^2$ ——— (1)

put $n = n-3$ in eqn (1)

$T(n-3) = T(n-6) + (n-3)^2$ ——— (2)

put eqn (2) in (1)

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

put $n = n-6$ in eqn (1)

$T(n-6) = T(n-9) + (n-6)^2$ ——— (4)

put eqn (4) in eqn (3)

$T(n) = T(n-9) + (n-6)^2 + (n-3)^2 + n^2$ ——— (5)

⋮

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

$n-k = 1$

$k = n-1$

$T(n) = T(n-(n-1)) + (n-(n-1-3))^2 + (n-(n-1-6))^2 + n^2$.

$T(n) = T(1) + 4^2 + 7^2 + \cdots n^2$.

$= 1 + 4^2 + 7^2 + \cdots n^2$

$T(n) = 4n^2 + 1 + 4n$

$T.C = O(4n^2 + 4n + 1)$

$= O(n^2)$

Qno-9

The ith loop run 'n' times.

The jth loop value depends on 'i'

i=1, j=1,2,3,4,5

i=2, j=1,3,5

i=3, j=1,4

i=4, j=1

i=5, j=1

We can see that for each value of 'i', j loop is running n/i times:

for a value of k.

$k = \frac{n}{1}, \frac{n}{2}, \frac{n}{3}, \frac{n}{4} \cdots \frac{n}{n}$

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

$k = n(\log n)$

T.C $= n^2 \log(n)$: ie Tc ith loop * Tc jth loop.

$= 16T(n-8) + 15C$

Qno 11 The increment of i depends on j values:

let $n = 16$

$j = 1, \; i = 1 < n$

$j = 2, \; i = 3 < n$

$j = 3, \; i = 6 < n$

$j = 5 \quad i = 15 < n$

we can observe 'i' is the sum of first n natural no:

$n = 1 + 3 + 6 + \cdots + k$

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

$k = \sqrt{n}$

$\therefore TC = O(\sqrt{n})$

Qno :- 12'

Recursive relation for fibonacci Series:

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

$T(0) = 1$

$T(1) = 1$

let $T(n-1) \approx T(n-2)$

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

$= 2 \cdot \{2T(n-4) + C) + C$

$= 4T(n-4) + 3C$

$= 8T(n-6) + 7C$

$= 16T(n-8) + 15C$

$T(n) = 2^k T(n-2k) + (2^k - 1)C$

$n - 2k = 0 \Rightarrow k = \frac{n}{2}$

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

$= 2^{n/2}(2^{n/2} - 1)C$

$= 2^{n/2} + C2^{n/2} - C)$

$T(n) \propto 2^{n/2}$ (lower bound)

let $T(n-2) \approx T(n-1)$

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

$= 4T(n-2) + 3C$

$= 8T(n-3) + 7C$

$\vdots$

$2^k T(n-k) + (2^k - 1)C$

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

$T(n) = 2^n T(0) + (2^n - 1)C$

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

$T(n) \propto 2^n$ (upper bound)

T.C will be somewhere $2^{n/2}$ to 2^n

$TC = O(2^n)$

Space complexity will be $O(n)$

Qno : 3 = n(logn) , n^3, log(logn)

(i) n logn

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

(ii) n^3

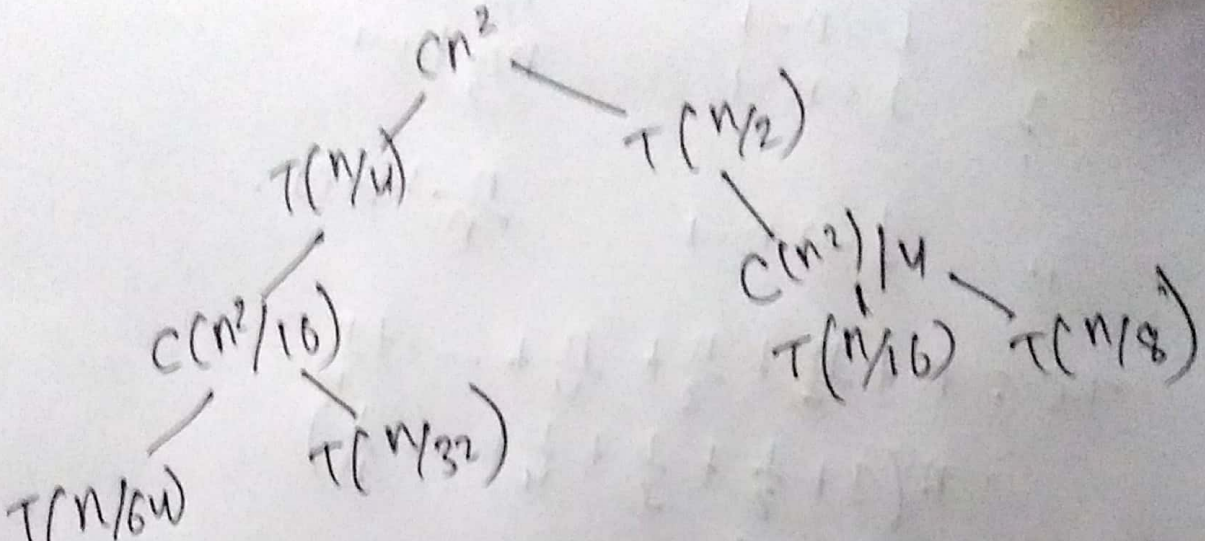
```
for (int i=0; i<n ; i++)
{ for (int j=0 ; j<n ; j++)
  {
    for int k=0 ; j<n ; k++)
    { O(1)
    }
}.}
```

(iii) log (log n)

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

Q no 14 $T(n) = T(n/4) + T(n/2) + cn^2$

Solving using recurrence :



To find $T(n)$: we calculate sum of tree node by level by level :

$$T(n) = \left(n^2 + \frac{5n^2}{16} + 25\frac{n^2}{256} + \cdots\right)$$

$$T(n) = cn^2\left(1 + \frac{5}{16} + \frac{25}{256} + \cdots\right)$$

$a = 1$, $r = 5/16$ | Sum of GP $= \frac{a}{1-r}$ if $r < 1$ [illegible]

$$= cn^2\left(\frac{a}{1-r}\right)$$

$$= cn^2\left(\frac{1}{1 - 5/16}\right)$$

$$= \frac{cn^2}{1 - 5/16}$$

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

Qno:15

i	j	times
1	j = n-1	n
2	j = n-1	n/2 times
3	j = n-1	n/3 times
:		
n	j → n-1	$\frac{n}{n}$ times.

$$T.C = n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \cdots + \frac{n}{n}$$

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

$$= n(\log n)$$

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

Qno-16

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

last term must be equal to n

$$n = 2^{\log(n)}$$

So, there are total $\log_k(\log n)$ iteration.

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

Qno:17 $T(n) = 2T(N/2) + N$

$T(0) = T(1) = 0$

Using master method compare with

$T(n) = aT(n/b) \rightarrow f(n)$

$a = 2, b = 2$

$c = \log_b a = \log_2 2 = 1$

$f(n) = n$

$n^c = n^1$

case 2 : $f(n) = n^c$

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

Qno:18 a) 100, $\log(\log n)$, $\log(n)$, $\log(n))$, root(n), $n \log n$, n, n^2, 2^n, $n!$, 4^n, 2^{2n}

b) 1, $\log(\log n)$, $\sqrt{\log(n)}$, $\log(n)$, $\log(2n)$, $2 * \log(n)$, $\log(n!)$, n, $n \log(n)$, $2n$, $4n$, n^2, $2(2^n)$, $n!$

c) 96, $\log_2(n)$, $\log_2(n)$, $\log(n!)$, $n \log_2(n)$, $n \log(n)$, $5n$, $8n^2$, $7n^2$, $n!$, 8^{2n}

Qno-19

```
int LinearSearch(int arr[], int n, int x)
{
    int l=0, r=n-1;
    int mid=(l+r)/2
    if (arr[mid]==x)
        return mid;
    else if (arr[mid]>x)
        r=mid-1;
    else l=mid+1;
    for (int i=l; i<=r; i++)
    {
        if (arr[i]==x)
            return i;
    }
    return -1;
}
```

Qno:-20 Iterative insertion sort:-

```
Void InsertionSort(int arr[], int n)
{
    int j;
    for (int i=1; i<n-1; i++)
    {
        int key=arr[i];
        j=i-1;
        while(j>=0 && arr[j]>key)
        {
            arr[j+1]=arr[j];
            j--;
        }
```

```
arr[j+1] = key;
}
```

Recursive

```
Void insertionSort(int arr[], int n)
{
    if (n<=1)
        return.
    insertionsort(arr, n-1);
    int last = arr[n-1];
    int j = n-2;
    while (j>=0 && arr[j]>last)
    {
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = last
}.
```

Insertion sort is called online sorting because it consider one input element per iteration and produces partial solution without considering future elements.

The other algorithm repeatedly select the min element from unsorted remainder and places it at the front which require access to entire input.

Qno 21

Time Complexity

	Bubble sort	Selection sort	Insertion sort	quick sort	Merge sort
Best Case	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n \log n)$	$O(n \log n)$
Worst Case	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \log n)$
Avg Case	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \log n)$	$O(n \log n)$
Space Complexity:	$O(1)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$.

Qno 22

	In place	Stable	online sorting.
Bubble Sort	✓	✓	✗
Insertion Sort	✓	✓	✓
Selection Sort	✓	✗	✗
quick Sort	✓	✗	✗
Merge Sort	✗	✓	✗
Heap Sort.	✓	✗	✗

Q no: 25 Recursive pseudo code for Binary Search:-

```
int binarySearch (int arr[], int l, int r, int n)
{   if (l >= r) return -1;
    int mid = l + (r-l)/2
    if (arr[mid] == n)
        return mid;
    else if (arr[mid] > n)
        return binarySearch (arr, l, mid-1, n);
    else
    return binarySearch (arr, mid+1, hi n)
}
```

Iterative Binary Search:

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

Time Complexity	Linear	Binary Recursive	Binary Iterative
Best Case	O(1)	O(1)	O(1)
Worst Case	O(n)	O(logn)	O(logn)
Avg case	O(n)	O(logn)	O(logn)
Space Complexity	O(1)	O(logn)	O(1)

Q no: 24

```
int binarySearch(int arr[], int l, int r, int x)   T(n)
{
    if (l >= r) return -1;                          O(1)
    int mid = l + (r-l)/2                           O(1)
    if (arr[mid] == x)                              O(1)
        return mid;                                 O(1)
    else if (arr[mid] > x)                          T(n/2)
        return binarySearch(arr, l, mid-1, x);
    else
        return binarySearch(arr, mid+1, r, x);      T(n/2)
}
```

Recurrence Relation : $T(n) = T(n/2) + 1$

$T(n) = 1$