Name- Abhinav Jain
Sec- DS
Roll No. - 2015142
Class Roll NO. - 03

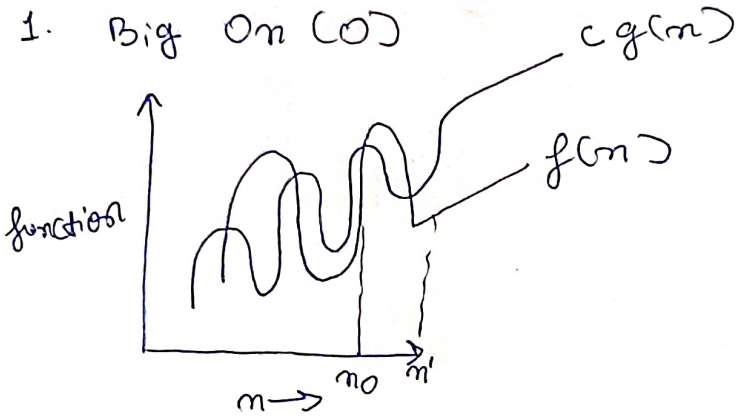
# Assignment - 1. DAA.

Q1.

Ans. These notations are used to tell the complexity of an algorithm when the input is very large.

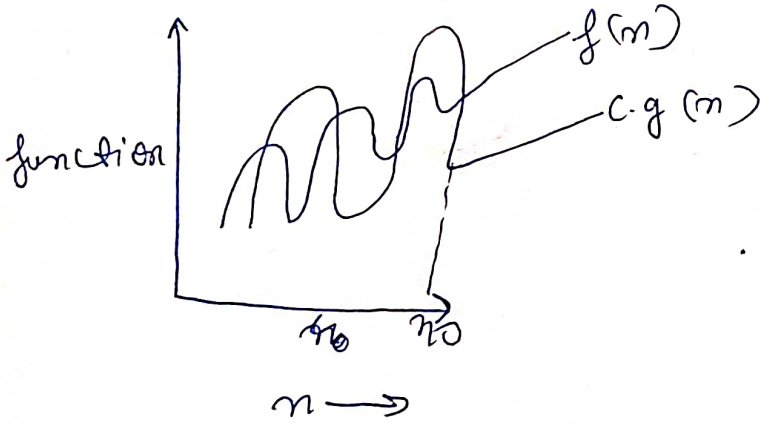
Types ->

1. Big Oh (O)




$f(n) = O(g(n))$
iff
$f(n) \leq c\, g(n)$
$\forall\ n \geq n_0$, some constant $c > 0$.

2. Big Omega ($\Omega$) :-

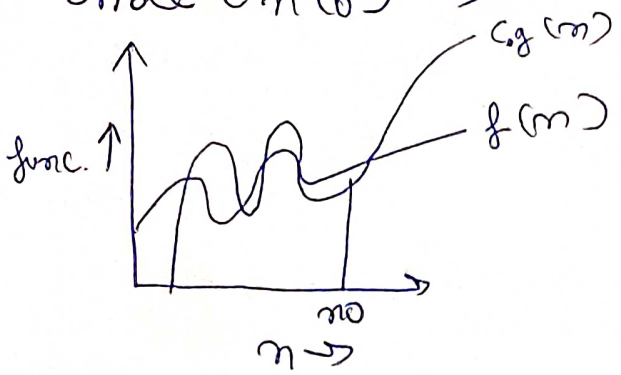



$f(n) = \Omega g(n)$ iff
$f(n) \geq c\, g(n)$
$\forall\ n \geq n_0$ & some constant $c > 0$.

3. Theta ($\Theta$) -> Theta gives the tight upper & lower bound both.


$f(n) = \Theta(g(n))$
iff
$c_1 g(n) \leq f(n) \leq c_2 g(n)$
$\forall\ n \geq \max(n_1, n_2)$
some constant $c_1$ & $c_2 > 0$

4. Small O notation (O) →

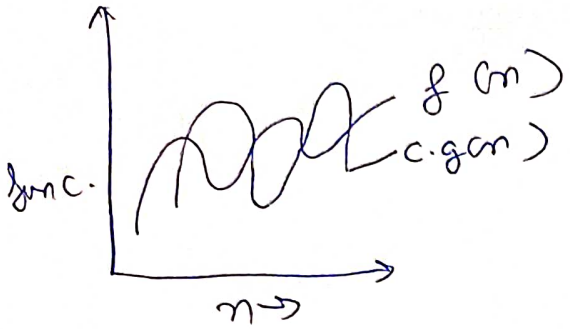



O gives upper bound

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

$f(n) < c.g(n)$ $\forall$ $n > n_0$ & $\forall$ $c > 0$

5. Small omega (ω) →




Lower bound

$f(n) = \omega . g(n)$

$f(n) > c.g(n)$

$\forall n > n_0$ & $\forall$ $c > 0$

$n^2 = \omega(n)$

Q2.

Ans.

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

$i = 1, 2, 4, 8, \ldots n$

$2^0\ 2^1\ 2^2\ 2^3 \ldots 2^k$ GP

$a = 1$

$r = 2$

$t_k = a r^{k-1}$

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

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

$2^k = 2n$

$k = \log_2(2n) = \log_2(n) + \log_2(2)$

$k = \log_2 n + 1$

Time complexity = $O(\log n)$

Q3.

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

$T(n) = 3T(n-1)$ —eq(1)

$T(1) = 1$

Put $n = n-1$

$T(n-1) = 3T(n-2)$ —(2).

Put value of $T(n-1)$ from (2) to (1).

$T(n) = 3.3T(n-2)$

$T(n) = 9T(n-2)$ —(3)

Put $n = n-2$ in eq 1

$T(n-2) = 3T(n-3)$ —(4)

Put value of $T(n-2)$ from eq (4) to eq (3).

$T(n) = 9 . 3T(n-3)$

$T(n) = 27T(n-3)$.

$T(n) = 3^k T(n-k)$ —(5).

$T(1) = 1$

$n - k = 1$

$k = n - 1$ —(6)

from (5) & (6)

$k = n-1$

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

$T(n) = \frac{3^n}{3} \times 1 =$ **T.C. $= O(3^n)$**

Q4.

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

$T(n) = 2T(n-1)-1$ —①

$T(1)=1$

put $n=n-1$ in eq①

$T(n-1) = 2T(n-2)-1$ —eq②.

put $T(n-1)$ in eq① from eq②

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

$T(n) = 4T(n-2)-3$ —③.

Put $n=n-2$ in eq①.

$T(n-2) = 2T(n-3)-1$ —④.

Put $T(n-2)$ in eq②.

Put value of $T(n-2)$ from eq④ to eq③.

$T(n) = 4.[2T(n-3)-1]-3$

$= 8T(n-3)-4-3$

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

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

$T(1)=1$

$n-k=1$

$k=n-1$ —⑤

from ④ & ⑤

$T(n) = 2^{n-1}[T(n-(n-1)] - 2^{n-2} - n^{-3} \cdots 2^0$

$= 2^{n-2} - 2^{n-1} - 2^{n-3} - 1$

$= \frac{1}{2}[2^n - [2^n - 1]] = \frac{1}{2} \times 1 = \frac{1}{2}$

T.C. $= O(1)$

Q5.

Ans.

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

1, 3, 6, 10 --- K
2 3 4

A.P

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

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

Q6.

Ans.

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

$1 + 1 + n^2 + n + n$

$n^2 + 2n + 2$

T.C $O(n^2)$

Q7.

Sol.

```
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++
}
```

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

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

Q8.

Sol.

```
function(int n) {
    if(n==1) return;          1
    for(i=1 to n) {           n
        for(j=1 to n) {       n
            printf("*");      1
        }
    }
    function(n-3);            n*n^2
}
```

$1 + n^2 + 1 + n^3$

$O(n^3)$

Q9.

Sol.

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

| i | j | times |
|---|---|---|
| 1 | $1 \to n$ | $\frac{n+1}{2}$ |
| 2 | $1 \to n$ | $\frac{n+1}{2}$ |
| ⋮ | | |
| n | $1 \to n$ | $\frac{n+1}{2}$ |

T.C. = $\log n\left(\frac{n+1}{2}\right)$

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

Q10.

Ans $n^k \leq c a^n$

$a^n + n^k \leq c a^n \rightarrow a^n$

$a^n + n^k \leq a^n (c-1)$

$\frac{a^n + n^k}{a^n} \leq (c-1)$

$c \geq 1 + \frac{n^k}{a^{n_0}} + 1$

$c \geq 2 + \frac{n_0^k}{a^n}$

$c \geq 2 + \frac{n_0^1}{1.5^n}$

$n_0 = 1$

$c \geq 2 + \frac{1}{1.5}$

$c \geq 3.0 + 1$

$c \geq 4$

Q11.

Ans. Time complexity $= O(n)$

The execution of diff. code lines here are →

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

2) $i = i + j \Rightarrow (n)$

3) $j++ \Rightarrow n$

T.C. $\simeq n + n + n - 1$

$= 3n - 1$

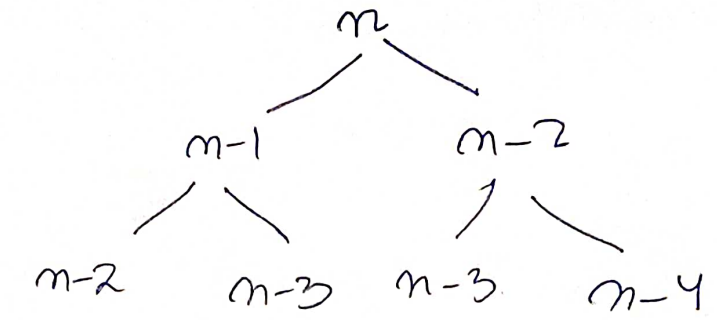
T.C. $= O(3n - 1)$

T.C. $= O(n)$

Q12.

Ans. The main working of fibonacci series is:

$f(x) = f(n-1) + f(n-2)$




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

$a=1, r=2$

$$\frac{a(r-1)}{r-1} = \frac{1(2^{n+1}-1)}{2-1} = 2^{n+1}$$

$T(n) = O(2^{n+1}) = O(2^n * 2^1) = O(2^n)$

Q13.

Ans. $O(n(\log n))$

```
int n;
for(int i=0; i<n; i++) {
    for(int j=n; j>0; j/=2){
        printf("*");
    }
}
```

$O(n^3)$

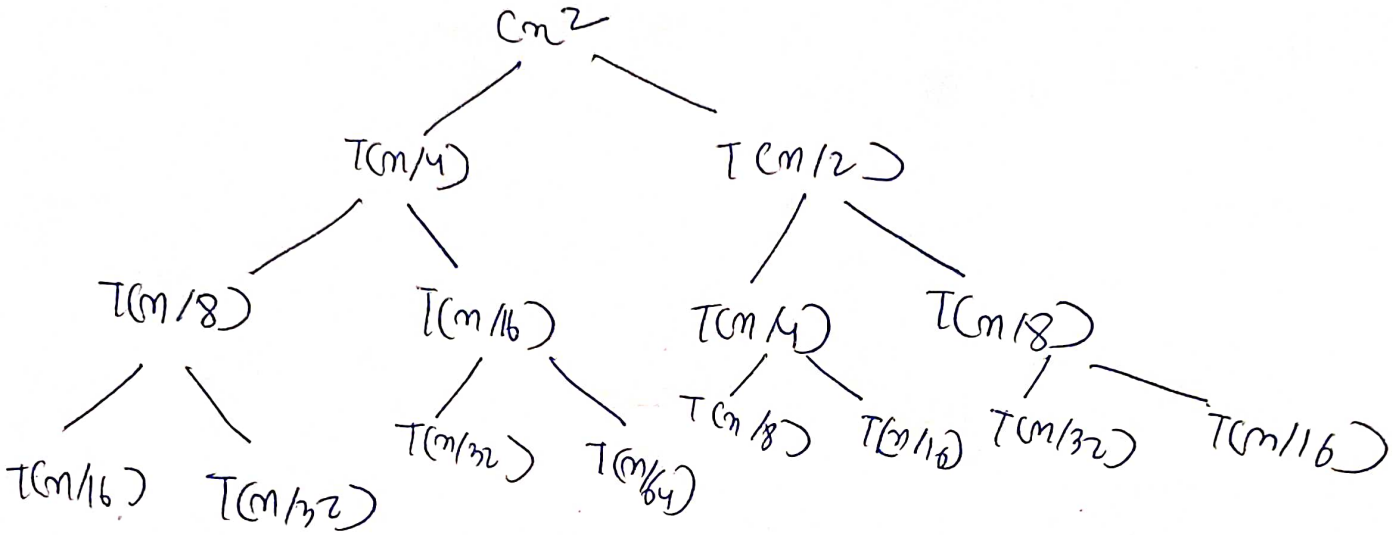
```
int i, j, k;
for(i=1; i<=n; i++) {
    for(j=1; j<=n; j++) {
        for(k=1; k<=n; k++){
            printf("*");
        }
    }
}
```

log(log(n))

```
for (i=2; i<=n; i*=i)
    {
    }
```

Q14.

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




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

$$\text{ratio} = \frac{5}{16}$$

$$= \frac{n^2}{1 - 5/16} = O(n^2)$$

Q15.

Ans.

| i | j | times |
|---|---|---|
| 1 | 1→n | n |
| 2 | 1→n | n/2 |
| ⋮ | | |
| n | | n/n |

$$T(n) = n + \frac{n}{2} + \frac{n}{3} + \cdots \frac{n}{n}$$

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

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

Q16.

Ans. $i = 2, 2^c, 2^{c^2}, 2^{c^3} \text{ ---- } 2^{c^{\log_c \log(n)}}$

The last term has to be $<= n$.

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

There are in total $\log_c(\log(n))$ iterations each take a constant amount of time to run.

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

Q.18.

Ans.

a) $100 < \log\log n < \log n < \sqrt{n} < n < n\log n <= \lg(n!)$
$< n^2 < 2^n < 2^2 < 4^n < n!$

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

c) $96 < \lg_2(n) = \log_8(n) < n\log_6(n) = n\lg_2(n) < \lg(n!)$
$< 5n < 8n^2 < 7n^3 < 8^{2n}$

Q.19.

Ans.

```
for (i=0 to n-1) {
    if (A[i] = key) {
        return i; }
}
retur -1;
}
```

Q.20.

Ans. a) Iterative Insertion sort

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

Recursive Insertion Sort →

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

Q21. Algo

| Algo | Best Case | Average Case | Worst Case |
|---|---|---|---|
| Bubble | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ |
| Selection | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ |
| Insertion | $O(n)$ | $O(n^2)$ | $O(n^2)$ |
| Merge | $O(n \log n)$ | $O(n \log n)$ | $O(n \log n)$ |
| Quick | $O(n \log n)$ | $O(n \log n)$ | $O(n^2)$ |
| Heap | $O(n \log n)$ | $O(n \log n)$ | $O(n \log n)$ |

Q22.

| Algorithm | Inplace | Stable | Online |
|---|---|---|---|
| Bubble | ✓ | ✓ | ✗ |
| Selection | ✓ | ✗ | ✗ |
| Insertion | ✓ | ✓ | ✓ |
| Merge | ✗ | ✓ | ✗ |
| Quick | ✗ | ✗ | ✗ |
| Heap | ✓ | ✗ | ✗ |

Q23.

Ans. Iterative

```
int BinarySearch( int arr[], int l, int r, int x)
  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;
}
```

Recursive Binary Search →

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

Time complexity →

| | Space Comp. |
|---|---|
| Linear (Recursive) → O(n) | O(1) |
| Binary (Recursive) → O(n) | O(logn) |
| Linear (Ite.) → O(1) | O(1) |
| Binary (Ite.) → O(1) | O(1) |

Q24.

Ans

Recurrence relation → $T(n) = T(n/2) + 1$.