Assignment - 1

(1) - Asymptotic notations are languages that allow us to analyze an algorithm running time by identifying its behaviour as the input size of algorithm.

Types -

(i). Big O - commonly used for worst case, and gives upper bound for growth rate of runtime of algorithm.

Eg - Big O notation for linear search is $O(n)$

(ii) - Big Omega - It is notation used for least complexity, it provides with an asymptotic lower bound.

Eg - Big Omega for linear search is $\Omega(1)$

(iii) - Theta - used for right bound on growth rate of runtime of algo.

Eg - theta of linear search is $\Theta(n)$

(iv) - Small Omega - to denote lower bound

(2) -

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

$\Rightarrow O(\log n)$

③- $T(n) = 3T(n-1)$

$T(1) = 1$

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

$T(3) = 3T(2) = 9$

$T(4) = 3T(3) = 27$

⋮

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

Time complexity $= (n-1)^3$

④- $T(n) = 2(T(n-1)-1)$

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

$T(n) = 4T(n-2)-2-1$

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

$T(n) = 8T(n-3)-4-2-1$

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

$T(n) = 16T(n-4)-8-4-2-1$

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

$= O(1)$

⑤-

S	i
1	1
3	2
6	3
10	4

$O(\sqrt{n})$

⑥- $i * i = n$

$i^2 = n$

$i = \sqrt{n}$

$O(\sqrt{2})$.

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

(8)(9). Total $T = O(n \log n)$

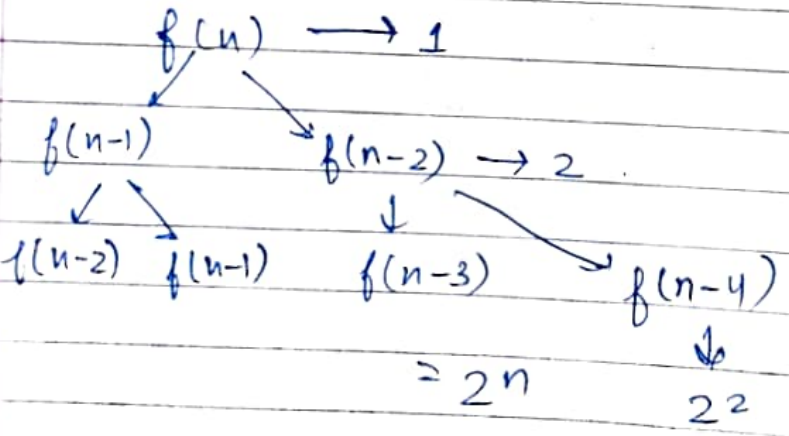
(10). n^t is $O(c^k)$ ces for example of -
When we take $n=2$, $k=2$, $c=2$
then, $2^2 \leq 2^2$ so c^k is upper limit of n^t.

(11).

j=1	i=0
1	1
2	3
3	6
4	10

Series is nearly dependent on i as 2^i so $O(2^n)$

(12). Space complexity = $O(n)$ as clear call of $(n-1)$



Time complexity = O(2

(13). $n \log n$

```
for (i=0 ; i<n ; i++)
    for (j=0 ; j<n ; j++)
        c++;
```

n^3

```
for (i=0; i<n ; i++)
    for (j=0; j<n; j++)
    for (k=0; i<n ; k++)
        c++;
```

$\log(\log n)$

```
int funct (int n)
{
    if (n == 1)
        return n;
    else
        return func(√n) + func(√n);
}
```

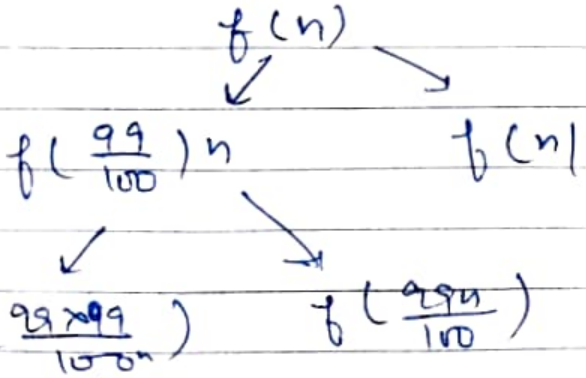
(14) $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + n^2$

$a = 2,$
$b = 2, \quad c = 1$
$f(n) > n^k \quad \& \quad n^2 > 1$
$O(n^2)$.

(15) $O(n\sqrt{n})$

(16) ~~$O(\log(\log n))$~~ $O(\log \log n)$

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



$= O(\log n)$

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

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

c) $96 < \log_2 n < \log_2 n^2 < \log 8n < \log n! < n\log_2 n <$
$n\log_2 n < 8n^2 < 7n^3 < 8^{2n} < n!$

(19) -
```
linear(arr, key)
{
  for(int i=0; i<n; i++)
    if(arr[i] == key)
      return i;
  return -1;
```

(20) -
```
int(arr, n)
{
  if(n <= 1) return;
  recursively for(n-1) elements
    insert sort(arr);
```
Iterations -
```
  insert(arr, n)
  {
    for(i=1; i<n; i++)
    {
      Pick arr[i] & insert into arr[0, ....i-1]
    }
```

	Stable	Inplace	Online
Bubble sort	✓	✓	X
Selection sort	X	✓	X
Insertion sort	✓	✓	✓

(22)

	Best	Avg	Worst
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion	$O(n)$	$O(n)$	$O(n^2)$

(23) :- Recursive -

```
Binary (arr, l, n, key)
{
 if (l<n)
 {
  mid = l+(r-l)/2;
 if (arr[mid] == key) ret -1;
 if (key < arr mid)
 Binary (l, mid-1, key);
 else
   Binary (mid+1, r, key)
 }
```

Iterative

```
While (l<r)
{
  mid = l + (r-1)/2
 if (arr[mid] == key) return 1;
   if (key < arr[mid])
     r = mid-1;

   else
       l = mid + 1;

 }
```