* Time Complexcity -

```
count = 0
① for (i=0; i<N; i++)
  {
    for (j=i+1; j<N; j++)
      {
        count++;
      }
  }
Print (count)
```

N = 5

i = 0 → N-1 times

i = 1 → N-2 times

i = 2 → N-3 times

i = 3 → N-4 times

i = 4 → N-5 times

The deepest statement, How many times it is running.

$$= \overset{4}{(N-1)} + \overset{3}{(N-2)} + \overset{2}{(N-3)} + \overset{1}{(N-4)} + \overset{0}{(N-5)}$$

$$= 4 + 3 + 2 + 1 + 0$$

{ Sum of first (N-1) Natural Numbers)

Formula for Sum of N Natural Nos $= \frac{N(N+1)}{2}$

$$= \frac{(N-1)(N-1+1)}{2}$$

$$= \frac{N(N-1)}{2}$$

$$= \frac{1}{2}N^2 - \frac{N}{2}$$

$$= O(N^2)$$

∴ Ignore Non-dominating term & Constants

② (Floor value)

```
for( i=n; i>0; i = [i/2]) ;
{
   for (j=0; j<i; j++)
    { Print (x) }
}
```

let $n = 128$

Floor values → 128 – 64 – 32 – 16 – 8 – 4 – 2 – 1 – 0

Let $n = 343$

Floor value → 343 – 171 – 85 – 42 – 21 – 10 – 5 – 2 – 1 – 0

If

	Inner loop running till
$i = N$	N
$i = [N/2]$	$[N/2]$
$i = [N/4]$	$[N/4]$
⋮	⋮
$i = 1$	1

deepest Statement Running.

$$= N + [N/2] + [N/4] + \cdots + 1$$

$$= N + \frac{N}{2} + \frac{N}{4} + \cdots + 1 \quad (\text{G.P.})$$

Sum of N terms of a G.P. $= \frac{a(1-r^n)}{(1-r)}$ $(r<1)$

Sum of Infinite terms of a G.P. $= \frac{a}{1-r}$

(Let assume these are Infinite terms)

(for higher Input, it will work)

$$= \frac{N}{1-1/2}$$

$$= \frac{N}{1/2}$$

$$\boxed{T.C = O(N)}$$

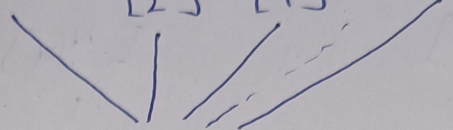
③

```
a = 0
i = n
while ( i>0)
{
   a = a + i
   i = [i/2]
}
Print ( a)
```

N = 5 ; a = 0

i = 5 → 2 → 1 → 0

- N → $\left[\frac{N}{2}\right]$, $\left[\frac{N}{4}\right]$ ---- , 1



While loop is running for single iteration

After how much time i becomes 1 Number of times, this while loop.

Let After K steps, i becomes 1

$$= \frac{N}{2^0}, \frac{N}{2^1}, \frac{N}{2^2}, \dots, \frac{N}{2^K}$$

Calculating K

$$\frac{N}{2^K} = 1$$

$$2^K = N$$

$$\log 2^K = \log N$$

$$K = \frac{\log N}{\log 2}$$

$$K = \log_2 N$$

time Complexcity = $O(\log_2(N))$

Home Work

①
```
for(i = N/2 ; i <= N; i++)
{
    for(j = 2 ; j <= N ; j = j*2)
    {
        Print(i, j)
    }
}
```

$(10) = N$

$2^k = N$

$k = \log_2(N)$

Soln.

	deepest Statement (No. of iteration)
$i = N/2$	$\log_2(N)$
$i = N/2 + 1$	$\log_2(N)$
⋮	
$i = N$	$\log_2(N)$

Time Complexity $= \log_2(N) + \cdots + \log_2(N)$ ($N/2$ times)

$= \frac{N}{2} \log_2(N)$

$$\boxed{T.C. = O(N \log_2(N))}$$

Space Complexity - Space as a resource.
- Extra space (memory)
- Extra space used ∝ Input size.

③

① a = 0
j = N
while (j > 0)
{
a = a + j
i = [i/2]
}
Print (a)

T.C. → log(N)
S.C. → O(1)

{ because when we are finding the size of Input, but space which is used is constant
i.e space is not depending on size of Input }

② ans = [ ]
for (i = 0; i < N; i++)
{
ans[i] = N + i
}
Print (ans)

T.C ⇒ O(N)
S.C ⇒ O(1)

{ In this case, the size of ans is depending on size of Input }

③

```
a = 0
j = N
ans = [ ]
while (j > 0)
{   a = a + j
    ans.push[a]
    j = [j/2]
}
Print(ans)
```

(iteration by while loop
= $\log_2(N)$

[T.C → $O(\log_2(N))$
S.C → $O(\log_2(N))$]

* Two pointer

① Detect Palindrome.

```
Function detectPalindrome(Str) {
    left = 0
    right = Str.length-1
While (left < right) {
    If (Str[left] != Str[right]) {
        return "not a Palindrome"
    }
    Left++;
    right--;
}
return "Palindrome"
}
```

T.C = $O(N/2)$	$\rightarrow O(N)$
S.C = $O(1)$	$\rightarrow O(1)$

② Two Sum

```
function twoSum(arr) {
    left = 0
    right = arr.length - 1;
  while ( left < right) {
Sum = arr [left] + arr[ right]
  if (Sum == K) {
      return { left, right}

  }
  else if (Sum > K) {
       right --;

  }
  else {
       left++;
  }
  return {-1, -1};
}
```

T.C ⇒ O(N)
S.C ⇒ O(1)

③ Reverse of array.—

```
Function reverseOfArray(arr) {
    while (left < right)
    {
        var temp = arr[left]
        arr[left] = arr[right]
        arr[right = temp.
    }
}
```

T.C ⇒ O(N)
S.C ⇒ O(1)

④ Rotate the array —

[1 2 3 4 5]

K = K % N = 9 % 5
K = 4

↓ Reverse the array.

[{5, 4 3 2} 1]

↓ Reverse the array from (0) to (k-1)

[2 3 4 5 {1}]

↓ Reverse the array from (k) to (N-1)

[2 3 4 5 1]

Time Complexity:-

Reverse — O(N)

(0, K-1) — O(K)

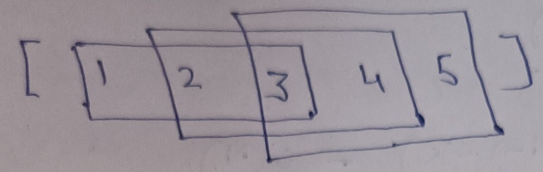
(X, N-1) — O(N-1-K) ~ O(N)

T.C ⇒ O(N) + O(K) + O(N)

T.C ⇒ O(N)

S.C ⇒ O(1)

* Sliding Window Technique




```
Function takeOutMaximum(arr[], K){
    max = 0;
    Sum = 0;
  for(i=0; i<K i++){
        Sum += arr[i];
    }
  if (Sum > max){
      max = Sum;
    }
for( i=K; i<arr.length; i++){
      Sum += arr[i];
      Sum -= arr[i-K];
    if (Sum > max){
        max = Sum;
    }
  }
  Print(max);
}
```

T.C ⇒ O(N)
S.C ⇒ O(1)