

# CS & IT ENGINEERING

## Algorithms

Analysis of Algorithm

Lecture No. - 10

By- Dr. Khaleel Khan  
Sir



# Recap of Previous Lecture



Topic

Problem Solving with ASNs

Topic

Analysis of Recursive Algorithms

Topic

Topic

Topic



# Topics to be Covered



## Topics

Framework for Analysing Recursive Algo

Loop Complexities







## Topic: Asymptotic Notations

Let  $f(n) = n$  and  $g(n) = n^{(1 + \sin n)}$ , where  $n$  is a positive integer. Which of the following statements is/are correct?

I:  $f(n) = O(g(n))$  ✗

II:  $f(n) = \Omega(g(n))$  ✗

[GATE-2015: 2M]

**A** Only I

**B** Only II

**C** Both I and II

☒ **D** Neither I nor II





## Topic: Asymptotic Notations

### MSQ

Let  $f$  and  $g$  be functions of natural numbers given by  $f(n) = n$  and  $g(n) = n^2$ .  
Which of the following statements is/are TRUE?

[GATE-2023-CS:1M]

- ☒ A  $f \in O(g)$  ✓
- ☐ B  $f \in \Omega(g)$
- ☒ C  $f \in o(g)$  ✓
- ☐ D  $f \in \theta(g)$





## Topic : Running Times of Program Segments with Loops:

4.  $\text{for } (i = 1; i \leq n; ++i) : n$   
     $\text{for } (j = 1; j \leq n; ++j) : n$   
         $\text{for } (k = n/2; k \leq n; k += n/2) : 2$   
             $c = c + 1;$

$K = n/2 + n/2 = n + \frac{n}{2} = \frac{3n}{2} > n$   
 $n \times n \times 2 = 2n^2 = O(n^2) \checkmark$

5.  $i = 1;$   $2^k$   $n = 16$   
     $\text{while } (i \leq n)$   
    {  
         $i = i * 2;$   
    }

$i = 1 \times 2^0, 1 \times 2^1, 1 \times 2 \times 2 (2^2), 2^2 \times 2 (2^3), \dots, 2^k$   
 $\text{Time } O(K) : O(\log_2 n)$   
 $i = 2^k = n$   
 $k = \log_2 n$   
 $\text{for } (i = 1; i \leq n; i = i * 2) \checkmark$   
     $c = c + 1;$

6.  $i = n;$   
     $\text{while } (i > 0)$   
    {  
         $i = i / 2;$   
    }

$n = 16$   
 $i = 16, 8, 4, 2, 1$   
 $2^4, 2^3, 2^2, 2^1, 2^0$   
 $O(\log n)$   
 $\text{for } (i = n; i > 0; i = i / 2) \checkmark$   
     $c = c + 1;$



$$1) \text{ for } (i=1; i \leq n; \underline{i=i+1}) : O(n)$$

$$\xrightarrow{n/1} O(n) \quad \text{Time: } O(k)$$

$$2) \text{ for } (i=1; i \leq n; \underline{i=i+2}) \quad i=1, 1+2, 1+2+2, 1+2+2+2$$

$$c=c+1; \quad \xrightarrow{n/2} \quad = \underline{1+2 \times 0}; \underline{1+2 \times 1}; \underline{1+2 \times 2}; \underline{1+2 \times 3}; \dots$$

$$; 1+2 \times k$$

$$1+2 \times k = n$$

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

$$3) \text{ for } (i=1; i \leq n; \underline{i=i+5})$$

$$c=c+1;$$

$$\xrightarrow{n/5}$$

$$= 1; 1+5; 1+5+5$$

$$= 1+5 \times 0; 1+5 \times 1; 1+5 \times 2; 1+5 \times 3; \dots$$

$$\underline{1+5 \times k} = n$$

$$k = \frac{n-1}{5} \sim \frac{n}{5}$$

Generalized form:

$$\text{for } (i=1; i \leq n; i=i+a) : n/a$$

$$1) \text{ for } (i=1; i \leq n; i=i*2) : \log_2 n$$

$$2) \text{ for } (i=1; i \leq n; i=i*3) : \log_3 n$$

$$i = 1 \times 3^0; 1 \times 3^1; 1 \times 3 \times 3; 1 \times 3^3, \dots; 1 \times 3^K$$

$1 \times 3^2$

$$\text{Time: } O(K)$$

$$\begin{aligned} &\text{for } (i=1; i \leq n; i=i*a) \\ &\quad : O(\log_a n) \end{aligned}$$

$$i = 1 \times 3^K = n$$

$$K = \log_3 n$$



```

7.  k = 1; i = 1;
    while (k <= n)
    {
        i++;
        k = k + i;
    }

```

i:	1	2	3	4	...	t
k:	1	(1+2)	(1+2+3)	(1+2+3+4)	...	(1+2+3+...+t)

$$(1+2+3+\dots+t) = n$$

$$\frac{t(t+1)}{2} \leq n$$

$$t^2 + t = 2n$$

$$t^2 \sim n \quad \therefore t = \sqrt{n}$$

$$\text{Time: } O(t)$$

$$: \underline{\underline{O(\sqrt{n})}} \checkmark$$



8. for (i = 1; i <= n; ++i) :  $n$   $O(n * \log n)$   
 for (j = 1; j < n; j = 2 \* j) :  $\log n$   
 c = c + 1;

9.  $m = 2^n$

⊗ for (i = 1; i <= n; ++i) :  $n$   
 for (j = 1; j <= m; j = 2 \* j) :  $\log_2 m$   
 c = c + 1

:  $n * \log_2 m$  :  $n * \log_2 2^n$   
 :  $n \times n = n^2$   
 ...  $O(n^2)$  ✓



$c = 0$

10. for  $i \leftarrow 1$  to  $n$

for  $j \leftarrow i$  to  $n$

$c = c + 1;$

Value of  $c =$

$i = 1 ; 2 , 3$

$c = \underline{n} ; (\underline{n} + \underline{n-1}) ; n + (n-1) + (n-2)$

$$n + (n-1) + (n-2) + \dots + 1$$

Value of  $c$  & Total Iterations

$$= \frac{n(n+1)}{2} = \underline{\underline{O(n^2)}}$$

11.  $f(n) = \sum_{i=1}^n \underline{O(n)} = \sum_{i=1}^n \sum_{j=1}^n O(1)$

$= O(n) * \sum_{i=1}^n 1$

$= O(n) * O(n)$

$= O(n^2) \checkmark$

$$f(n) = \sum_{i=1}^n 1 = O(n)$$

for  $i \leftarrow 1$  to  $n$   
for  $j \leftarrow 1$  to  $n$   
 $c = c + 1;$

for  $i \leftarrow 1$  to  $n$   
 $c = c + 1$







12.  $i = n;$   $n = 16$   
 while ( $i > 0$ ) :  $\log_2 n$

```

{
  j = 1;
  while (j <= n)
  {
    j = 2 * j;
  }
  i = i/2;
}

```

$1^{st}$ gt	$2^{nd}$ gt	$3^{rd}$	$\log_2 n$
$\log_2 n$	$\log_2 n$	$\log_2 n$	$\log_2 n$

Time :  $(\log_2 n * \log_2 n)$

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



13. int fun (int n) : 2M

P4Q {  
    int i, j, p, q = 0;  
    for (i = 1; i <= n; ++i) : n  
    {  
        1. p = 0; q = 0;  
        2. for (j = n; j > 1; j = j/2) : log n  
            ++ p;  
        3. for (k = 1; k < p; k = k \* 2)  
            ++ q;  
    }  
    return (q);  
}

Total  
Time  
n

n · log n

n · log log n

$$q = \log P + \log P$$

Q) The value of q returned by the function :  $O(n * \log \log n)$



$$q = n * \log P$$

$$P = \log n$$

$$= n * \log_2(\log_2 n)$$

Q2) Time Complexity :  $O(n \cdot \log_2 n)$

$$\text{Value of } q = O(\log \log n)$$

$$\text{Time} = O(n \cdot \log_2 n)$$



14. for (i = 1; i <= n; ++i) :  $n$

{

1. j = 1;  $\xrightarrow{\quad\quad\quad} n$

2. while (j <= n)  $\left. \begin{array}{l} j = 2 * j; \end{array} \right\} \log n \rightarrow n * \log n$

3. for (k = 1; k <= n; ++k)  $\left. \begin{array}{l} c = c + 1; \end{array} \right\} n \rightarrow \frac{n^2}{2}$

}

$\frac{n^2 + n \log n + n}{2}$   
 $O(n^2)$  ✓



15.  $n = 2^{2^k}$   
 for ( $i = 1; i \leq n; ++i$ )  
 {  
    $j = 2;$   
   while ( $j \leq n$ )  
   {  
      $\left\{ \begin{array}{l} j = j * j; \\ \text{pf}(\text{"*"}); \end{array} \right.$   
   }  
 }

\* :  $n(k+1)$   
 :  $n \times \log \log n + n$

$$K=2$$

$$n = 2^4 = 16$$

$i=1$   
 $j=2, 4, 16$   
 (\*\*\*)

$i=2$   
 $j=2, 4, 16$   $\left[ 2^{2^0}; 2^{2^1}; 2^{2^2} \right]$   
 (\*\*\*)

$i=n$   
 $j=2, 4, 16$   
 (\*\*\*)

$$K=3$$

$$n = 2^8 = 256$$

$i=1$   
 $j=2, 4, 16, 256$   $\left[ 2^{2^0}; 2^{2^1}; 2^{2^2}; 2^{2^3} \right]$   
 (\*\*\*\*)

Per It:  $O(K)$

Total It:  $O(n \times K)$   
 :  $O(n \times \log \log n)$  ✓

$$n = 2^{2^K} \Rightarrow K = \log_2 \log n$$



1) for ( $i=2; i \leq n; \underline{i = i * i}$ )  
 $c = c + 1;$  Time:  $O(\log \log n)$

$$\underline{n = 16}$$

$$i = 2; 4; 16$$


---


$$= 2^{2^0}; 2^{2^1}; 2^{2^2}$$

$k = \text{last bit}$

2) for ( $i=1; i * i \leq n; i++$ )  
 $c = c + 1;$

Time:

$$i^2 \leq n \quad \checkmark$$

$$i^2 = n \Rightarrow i = \sqrt{n}$$

$$\text{Time: } O(i) : O(\sqrt{n})$$

$$i = 2^{2^k} = n$$

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



$$3) \text{ for } (i=n; i \geq 2; i = \text{sqrt}(i)) \Rightarrow O(\log \log n)$$

$$c=c+1;$$

$$n=256$$

$$i=256; (256)^{1/2}; \left[ (256)^{1/2} \right]^{1/2}; \left[ \left[ (256)^{1/2} \right]^{1/2} \right]^{1/2} = \left( n^{1/2^k} \right) = 2$$

$256$   
 $2^{2^3}$

$\downarrow$   
 $16$   
 $2^{2^2}$

$4$   
 $2^{2^1}$

$2$   
 $2^{2^0}$

$$\underline{\underline{K = \log \log n}}$$

$$2^{2^K} \leq n \Rightarrow 2^{2^K} = n$$

$$\Rightarrow K = \log \log n$$



16.

A: Array [1 .....n] of binary; (0/1)

 $f(m) = \theta(m)$ 

count: integer;

count = 1;

for i = 1 to n

{

if (A[i] == 1) count ++;

else

{

f(count);

~~count = 1;~~

}

}

$$\left( \frac{n}{2} + \frac{n}{2} \times \frac{n}{2} \right)$$

$$O(n^2)$$

w.c

Time-Complexity:

(i) Best Case:

(ii) Worst Case:

$O(n)$  ✓



1)  $A[i] = 1, i = 1, n$  Time:  $O(n)$   $O(n)$

2)  $A[i] = 0, i = 1, n$  Time:  $O(n)$   $O(n)$

3)  $A[i] = 1, i = 1, n-1$   
 $= 0, i = n$  A: 

1	1	1	1	1	1	1	1	0
---	---	---	---	---	---	---	---	---

Time:  $(n-1) + (n-1) + 1$   
 $= 2n-1 = O(n)$  ✓  $O(n)$

4)  $A[i] = 1, i = 1, n/2$   
 $= 0, i = (n/2 + 1), n$  A: 

1	1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---	---

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



H/w

Consider the following function:

```
int unknown (int n)
{
    int i, j, k = 0;
    for (i = n/2; i <= n; i++)
        for(j = 2; j <= n; j = j*2)
            k = k + n/2;
    return(k);
}
```

The return value of the function is \_\_\_\_.

- A  $\Theta(n^2)$
- B  $\Theta(n^2 \log n)$
- C  $\Theta(n^3)$
- D  $\Theta(n^3 \log n)$

[GATE-2013: 2M]



Consider the following C function.

```
int fun (int n)
{ int i, j;
  for (i = 1; i <= n; i++) {
    for (j = 1; j < n; j += i)
    {
      printf("%d %d", i, j);
    }
  }
}
```

H/w

- A  $\Theta(n\sqrt{n})$
- B  $\Theta(n^2)$
- C  $\Theta(n \log n)$
- D  $\Theta(n^2 \log n)$

Time complexity of fun in terms of  $\Theta$  notation is [GATE-2017: 2M]



# MSQ



Consider functions Function\_1 and Function\_2 expressed in pseudocode as follows:

<A; D>

Function\_1

```
while n > 1 do
  for i = 1 to n do
    x = x + 1;
  end for
  n = ⌊n/2⌋;
end while
```

Function\_2

```
for i = 1 to 100 * n
do
  x = x + 1;
end for
```

Time:  $100 * n = O(n)$

$T(n) = 100 * n$

$$\begin{array}{cccc} 1 & 2 & 3 & \dots & k \\ \hline \frac{n}{2^0} & \frac{n}{2} & \frac{n}{4} & & \frac{n}{2^k} \end{array}$$

$O(n)$

Let  $f_1(n)$  and  $f_2(n)$  denote the number of times the statement “ $x = x + 1$ ” is executed in Function\_1 and Function\_2, respectively.

Which of the following statement is/are TRUE?

[GATE-2023-CS:2M]

$$n = 2^k \\ k = \log n$$

☒ A  $f_1(n) \in \Theta(f_2(n))$

☒ B  $f_1(n) \in o(f_2(n))$

☒ C  $f_1(n) \in \omega(f_2(n))$

☒ D  $f_1(n) \in O(n)$

$$\begin{aligned} \text{Total time} &= n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots + \frac{n}{2^k} \\ &= n \left[ \sum_{i=1}^k \frac{1}{2^i} \right] = n \left( 1 - \frac{1}{2^k} \right) \\ &= n - 1 = O(n) \end{aligned}$$



# SPACE - Complexity

↓  
Memory

Program / Algorithm

Instructions

Data

↓  
Input

Space Complexity

→ Additional Space ✓

Auxiliary Space ✓

void SWAP(int a, int b) Inputs  
{

int temp;

1. temp = a;

2. a = b;

3. b = temp;

}

$2B = \text{Constant} = O(1)$



Algorithm Sum(A, n)

{

integer n, A[n], i, Sum

1. Sum = 0;

2. for  $i \leftarrow 1$  to n

Sum = Sum + A[i];

}

Input :  $n ; A[n]$

Aux  
workspace : 4 Bytes

---

$O(1)$

1) Time :  $O(n)$  ✓

2) Space :  $O(1)$





## Topic : Space Complexity

We define the space used by an algorithm to be the number of memory calls (or words) needed to carry out the computational steps required to solve an instance of the problem excluding the space allocated to hold the input. In other words, it is only the work space required by the algorithm.





## Topic : Space Complexity

All definitions of order of growths and asymptotic bounds pertaining to time complexity carry over to space complexity. It is clear that the work space cannot exceed the running time of an algorithm, as writing into each memory call requires at least a constant amount of time.

Thus, if we let  $T(n)$  and  $S(n)$  denote, respectively, the time and space complexities of an algorithm, then  $S(n) = O(T(n))$ .





## Topic : Space Complexity

**Example:** In Algorithm LINEARSEARCH, only one memory cell is used to hold the result of the search. If we add local variables, e.g. for looping, we conclude that the amount of space needed is  $\Theta(1)$ . This is also the case in algorithms BINARYSEARCH, SELECTION SORT and INSERTION SORT.

```
Algo LS(A, n, x)
{
  int n, A[n], x, i;
  for i ← 1 to n
  {
    if (x = A[i]) {
      print(i);
      return;
    }
  }
  return(0);
}
```

Time:  $O(n)$   
Space:  $O(1)$





## Topic : Space Complexity

**Example:** In Algorithm MERGE for merging two sorted arrays, we need an auxiliary amount of storage whose size is exactly that of the input, namely  $n$  (Recall that  $n$  is the size of the array  $A[p..r]$ ). Consequently, its space complexity is  $\Theta(n)$ .



integer  $n, A[n]$ ;  
 Algorithm RSum( $A, n$ )  
 {  
   if ( $n=1$ ) Return ( $A[n]$ );  
   else  
   {  
     Return ( $A[n] + \text{RSum}(A, n-1)$ );  
   }  
 }

> Space:  $O(n)$

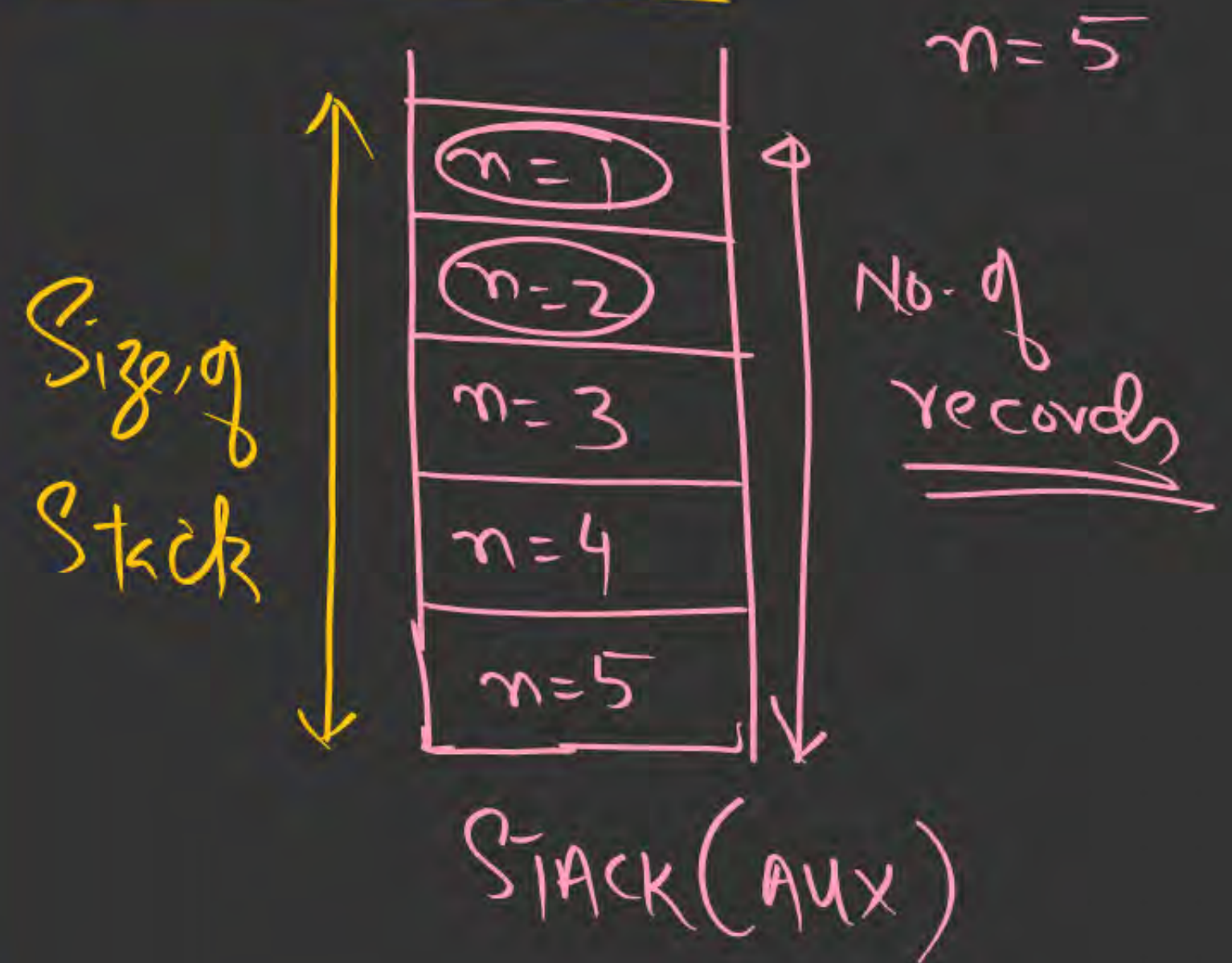
$n$  records  
Aux  
W.S

Sum of elems of Array

Time :  $O(n)$

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

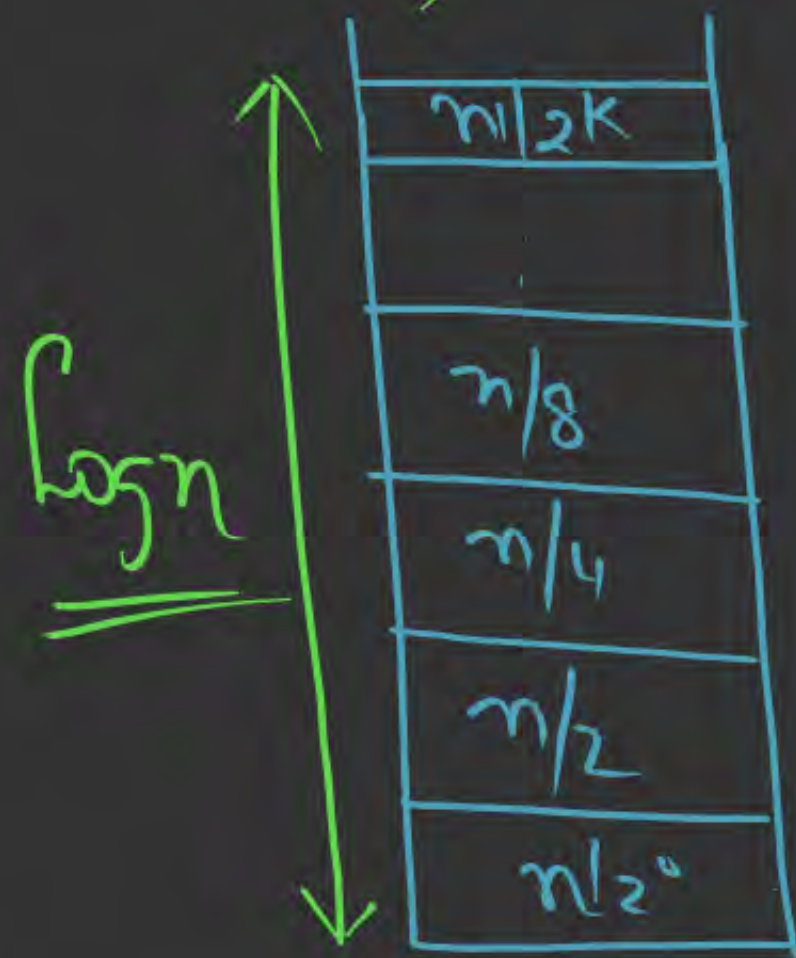
Space - Complexity :





Algorithm DO-it( $n$ )

```
{  
  if ( $n=1$ ) return,  
  else  
  {  
    DO-it( $n/2$ );  
  }  
}
```



Time:  $O(\log n)$

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

Space:  $O(\log n)$

$$1, 2, 4, 8, \dots, 2^k$$

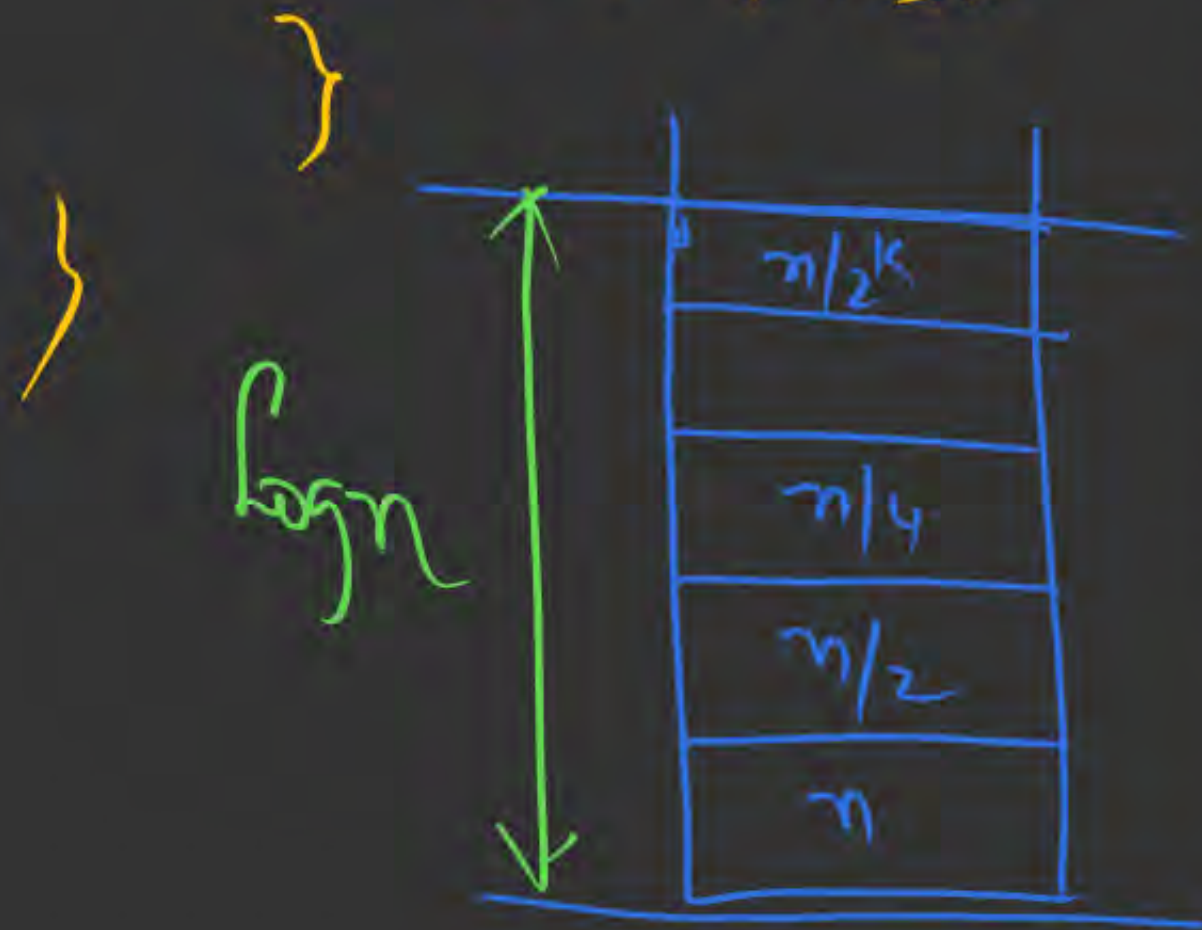
$$2^k = n$$
$$\Rightarrow k = \underline{\underline{\log n}}$$



```

Algorithm Test(n)
{
    if (n = 1) return,
    else
    {
        Test(n/2);
        Test(n/2);
    }
}

```



Time :  $O(n)$

$$T(n) = 2T(n/2) + c$$

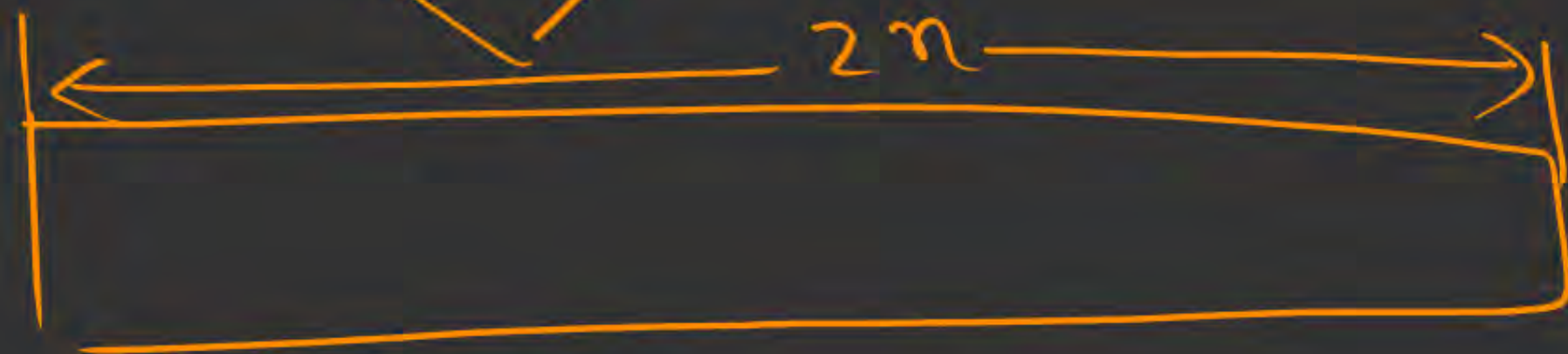
Space :  $O(\log n)$



Q) Given Two Sorted arrays of Size 'n' each, The Space Complexity to merge the Two arrays is  $O(\quad)$

	1	2	3	-	-	n	
A	8	10	15	22	30	...	50

	1	2	3	-	-	n
B	3	7	9	15	...	25





**THANK - YOU**