SCALER

Arrays

Hello Everyone

TABLE OF CONTENTS

Notes

Space Complexity: maximum space that is utilised by your program at any given point of time.

int → 4 Bytes long → 8 bytes.

```
{
  int x = 2;   → 4B
  long y = x * 3; → 8B
}
```

Total memory = 4B + 8B = 12 B → O(1)

1.

```
void fun(int N){
    int x = N;
    int y = 2*x;
    long z = x+y;
}
```

4B(x)

int x = N; → 4B

int y = 2*x; → 4B

long z = x+y; → 8B

= 4B + 4B + 8B
= 16B
= O(1)

2.

```
func(int N){ // 4 bytes
    int arr[10]; //40 bytes
    int x; // 4 bytes
    int y; // 4 bytes
    long z; // 8 bytes
    int[] arr = new int[N]; // 4*N bytes
}
```

I/P

= 40B + 4B + 4B + 8B + 4N
= 56B + 4N B
= (56 + 4N) Byte
= 4N

SC = O(N)

3.

```
void fun(int N){ // 4 bytes // -> i/p
    int x = N;   // 4 bytes
    int y = x*x; // 4 Bytes
    long z = y+y; // 8 Byte
    int[] arr = new int[N]; // 4*N Byte
    long[][] b = new long[N][N]; // 8*N*N Byte
}
```

$= (4+4+8+4N+8N^2)B$

$= (16+4N+8N^2)B$

$= 8N^2$

$SC = O(N^2)$.

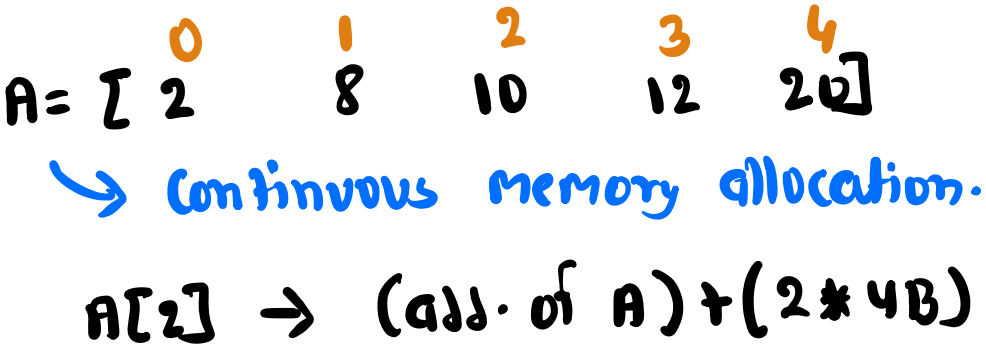
* input and output will not be considered as space complexity.

4.

```
int maxArr( int arr[], int N){   // 4*N Bytes } i/p
    int ans = arr[0];            // 4Bytes -> 4B -> o/p
    for(i=0; i<N; i++){          // 4B
        ans = Max(ans, arr[i]);
    }
    return ans;
}
```

$= 4B$

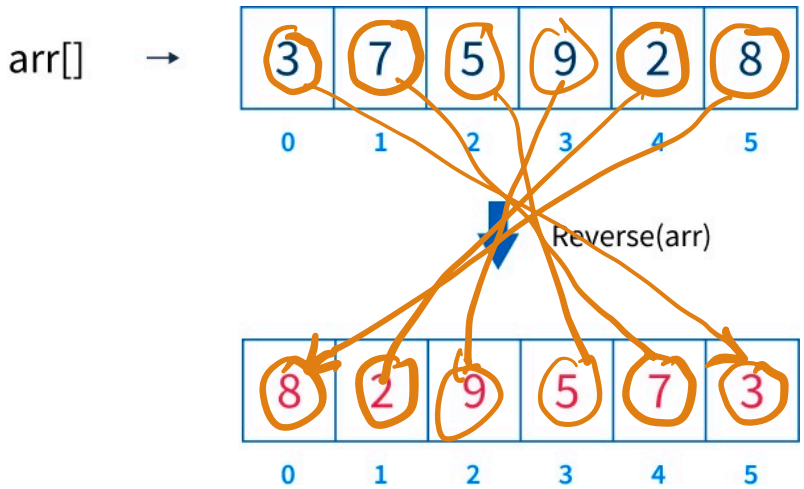
Introduction to Arrays

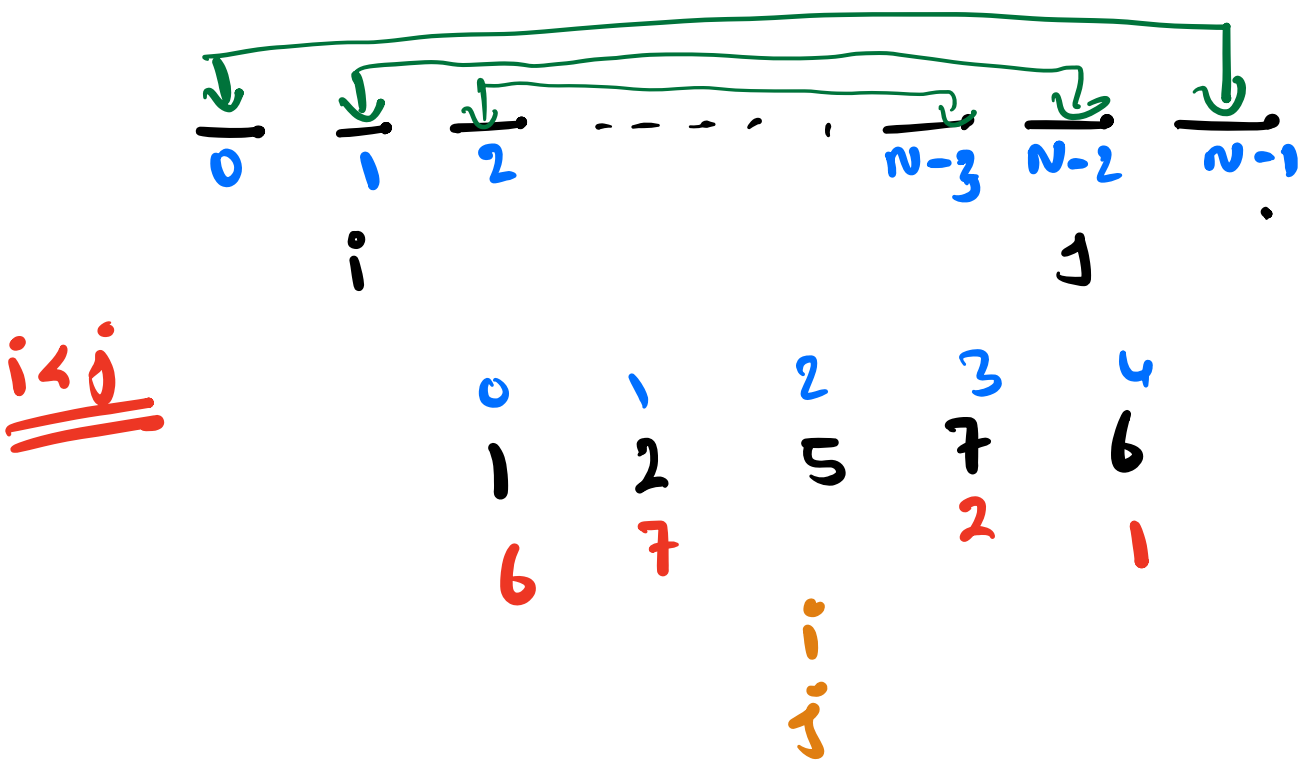


< *Question* > : Print all elements of the array

```
for i -> 0 to N-1 {
    print(A[i])
}
```

2. Reverse the given array





0	1	2	3	4	5
3	5	6	9	4	2
2	4	9	6	5	3

j

i

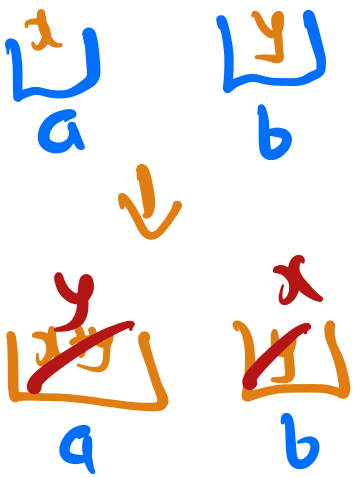
< / > Code

```
i = 0 , j = n-1
while (i < j)
{
    temp = A[i];
    A[i] = A[j];
    A[j] = temp;
    i++;
    j--;
}
```

TC: O(N)

SC: O(1)

Swap a, b without 3rd info.



b = a - b

= x + y - y

b = x

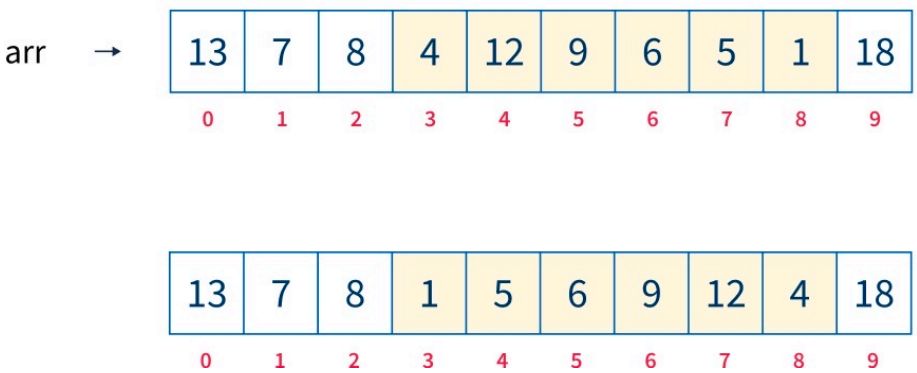
a = a - b

= x + y - x

= y

3. Reverse part of an array

l = 3, r = 8




```
void reversePart(int []arr, int l, int r){
}
```

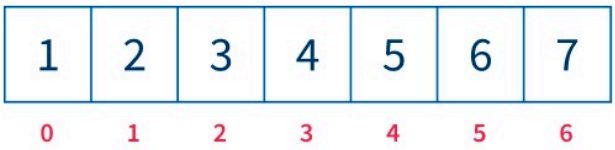
```
i = l , j = r
while(i<j)
{
    temp = A[i];
    A[i] = A[j];
    A[j] = temp;
    i++;
    j--;
}
```

TC: O(N)
SC: O(1)

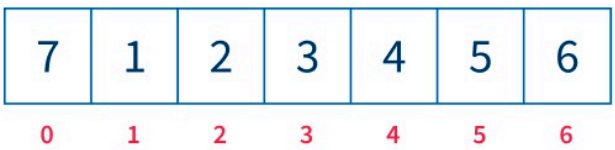
Break : 10:16 pm

* Given an array of size N, Rotate the array from right to left K times.

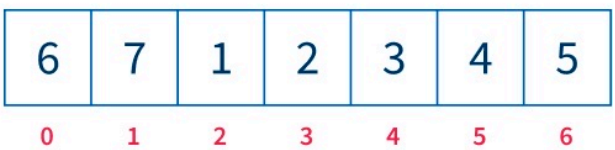
N=7



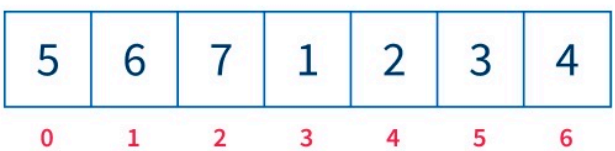
K=1



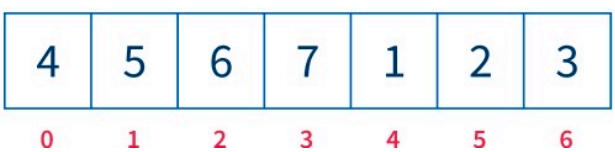
K=2



K=3



K=4



	0	1	2	3	4	5	6	7
A =	9	5	2	8	2	3	6	5

$K = 3$

	0	1	2	3	4	5	6	7
	9	4	2	8	1	3	6	5

↓ 1st rotation

	5	9	4	2	8	1	3	6

↓ 2nd

	6	5	9	4	2	8	1	3

↓ 3rd

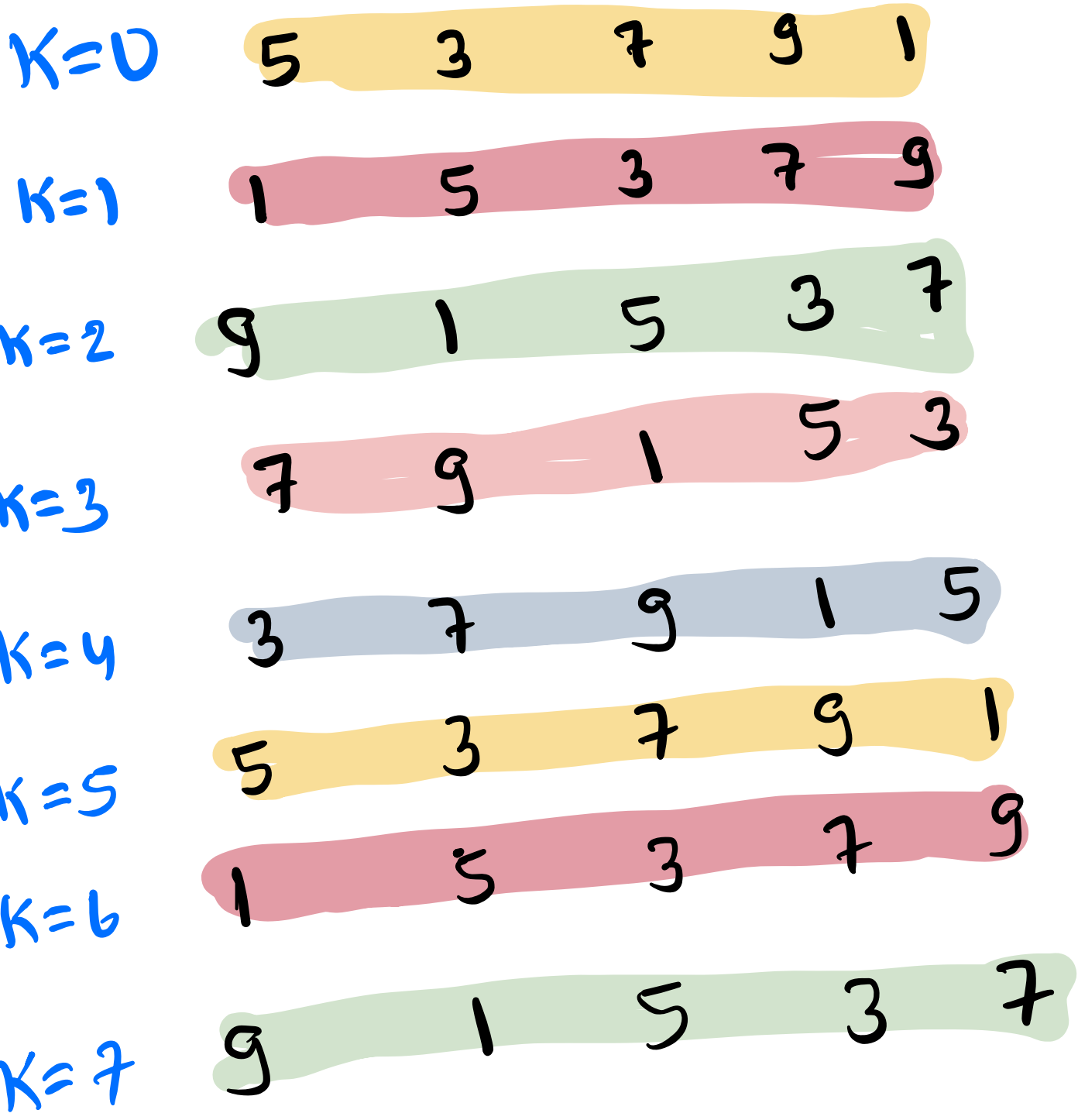
[3 6 5 9 4 2 8]

	0	1	2	3	4	5	6
A = [	9	6	8	3	4	9	1]
K=1	1	9	6	8	3	4	9
K=2	9	1	9	6	8	3	4
K=3	4	9	1	9	6	8	3

```
K = K % N        <- length of arr
for i -> 1 to K
{
    temp = A[n-1]
    for j -> N-2 to 0
    {
        A[j+1] = A[j]
    }
    A[0] = temp
}
```

TC: O(N*K)

SC: O(1)



length of arr

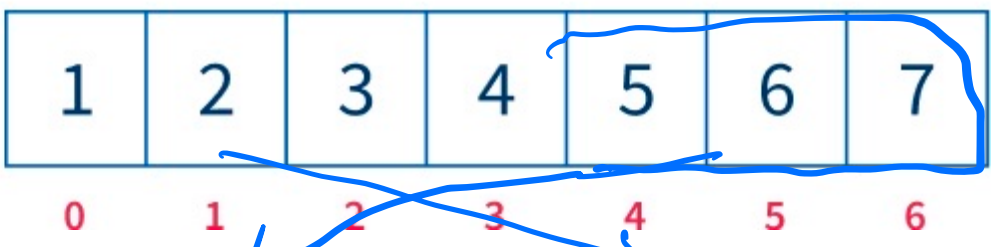
$K = K \% n$

N=5

K			
0	5	10	15
1	6	11	16
2	7	12	17
3	8	13	18
4	9	14	19

Optimisation

N=7



K=3

5 6 7 | 1 2 3 4

Reverse

7 6 5 | 4 3 2 1

Steps → $K = K \% N$

TC

1) Reverse (A, 0, N-1) → $O(N)$
2) Reverse (A, 0, K-1) → $O(K)$
3) Revene (A, K, N-1) → $O(N-K)$

Total TC = $O(N + K + N - K)$

$= O(2N)$

$= O(N)$

SC = $O(1)$

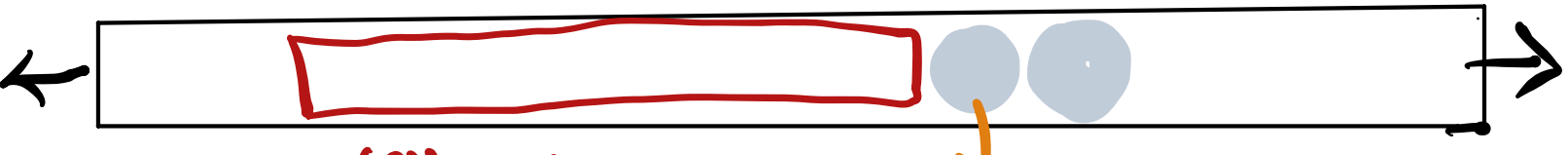
Dynamic Array:

```
int[] arr = new int[10]; // 10 length integer array will be created.
```

Sometimes defining length at the start might not be possible → dynamic array.

Eg. Java ArrayList
C++ vector
etc.

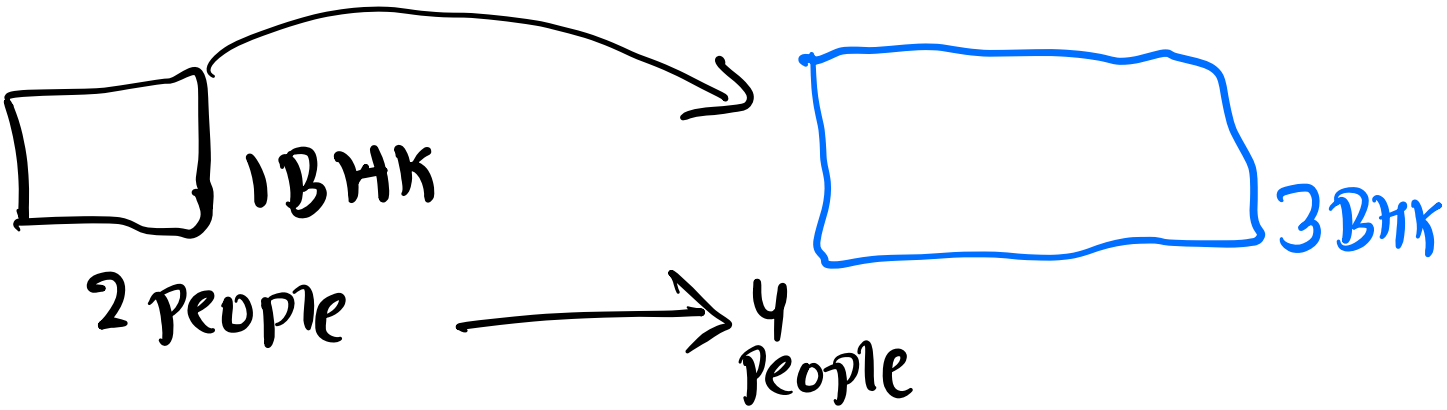
Memory:



(Allocated memory)

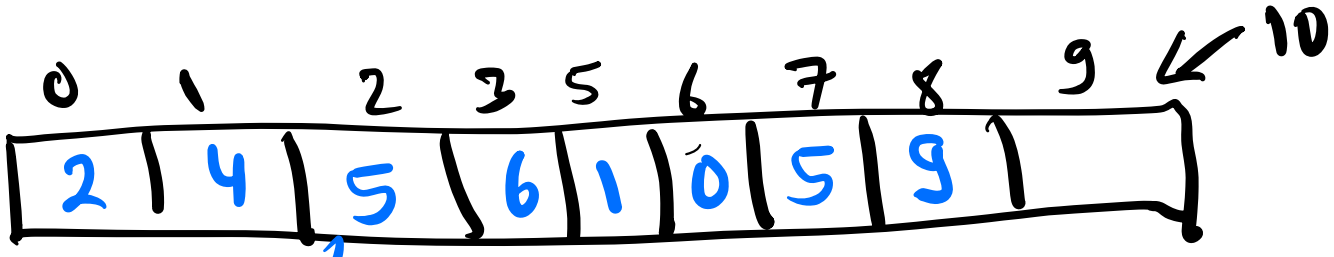
may be occupied.

⇒ Cannot expand.

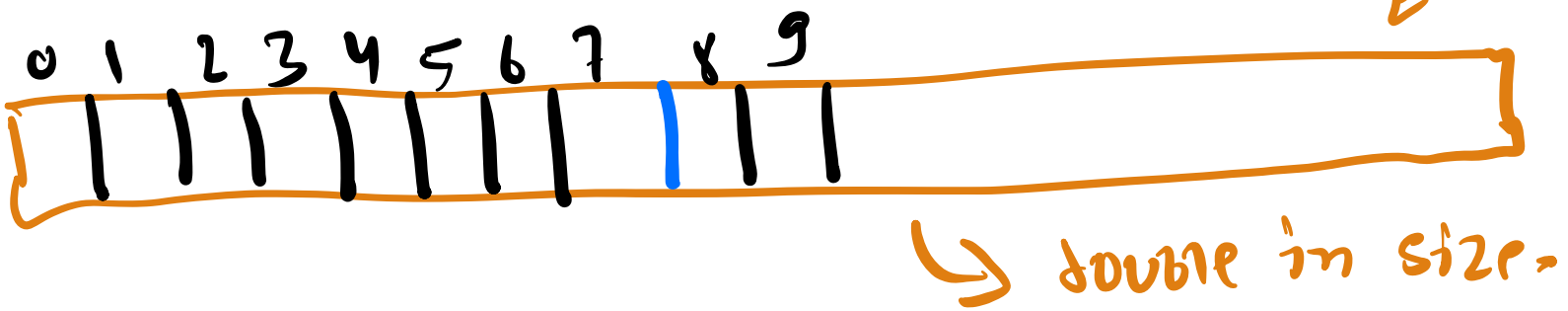


Load factor: (0 1) = 0.8 (eg.)

Whenever 80% memory is occupied it double the size of a memory.



move



double in size.

if (current size * load factor) is full)

find another location with double size.

Doubt Session

Tausif

fun (int[][] arr, n)
{

↓ ↙ O(1)

return ans;

}

Vishal

for (int [] [], int n) arr

{

// int[] arr = computation.

→ O(n)

return ans;

}

SC: $O(n^2)$

SC: $O(1)$

SC: $O(n^2)$.

SC: $O(n)$

4B

```
int a = 10;
for (i=0 → n)
{
    a++        → 4B
}
```

i=0 → a → 103

i=1 → a → 103

i=2 → a → 103

logn vs nlogn

```
for (i=0; i<=2^N; i++)
{
    int j = i
    while (j >=0)
    {
        j--;
    }
}
```

i	j	iteration
0	0	0
1	1	1
2	2[2 1]	2
3	3	3
4	4	4
50	50	50

2^N 2^N 2^N ← rough off

$= 1 + 2 + 3 + \dots + 2^N$

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

$= \frac{4^n + 2^n}{2}$ $= \frac{4^n}{2} + \frac{2^n}{2}$

0 1 2 3 4

1 3 5 7 9

3 1 5 7 9

i = 0

j = 1

0 < 1

↓ ↓

1 < 0 X

```
for(i=1; i<=n; i*=2)
{
    for(j=1; j<=n; j++)
    {

    }
}
```

i	j
1	[1 n]
2	[1 n]

iteration

n iterations

n iter

$$= 1 + 2 + 3 + 4 + \cdots\cdots + 2^N$$

$$\Rightarrow x$$

$$\frac{x * (x+1)}{2}$$

$$= \frac{x^2 + x}{2}$$

$$= \frac{x^2}{2} + \frac{x}{2} \rightarrow \text{lower order}$$

$$= x^2 * \frac{1}{2} \rightarrow \text{constat}$$

$$= x^2$$

$$= (2^N)^2$$

$= (2^N)^2$

$= 4^N$

$= (2^3)^2$

$= (2*2*2)^2$

$= 2*2*2*2*2*2$

$= 4*4*4$