Introduction of Arrays

Content

- Space Complexity
- Arrays
- Questions on arrays.
- (HW) Dynamic Arrays.

Amreshwar	Mahesh
Anil	Manisha
Ankit Jaiswal	Manohar A N
ankit shukla	MOHAMMAD ALI
ANUBHAV RAMNANI	Mohona Paul
Arunava Basak	Naveen
Arunkumar	Nikhil D
Bhavesh Pandey	Nutan
Bhushan Ganesh Shelar	Pranjul Kesharwani
Chandra Shekhar Bhatt	Prasanna Kumar D R
Deepshikha Arora	Praveen Kumar
Dev Raj Gaur	Priyank Varshney
Dhanashree Sagane	Rahul
Dhasthagiri Reddy	Rahul
Dhruv	Rishav Ghosh
Dipika Malik	Sai Kiran JNR
Farzana kauser	Saurabh Chauhan
Harikrishnan A	Shaik Lal Jan Basha
Hemant Verma	Shivam Shiv
Hemanth T	Shivanand Patil
Induja	Srikant Kumar Pratihary
Iti	Suresh
jeevanantham	Tanya
Kanhu	Uddeepta Saikia
M S Haseeb Khan	Ved Verma

Space Complexity

Space complexity is the max amount of space used by your algorithm or function at any instant of time. Big O

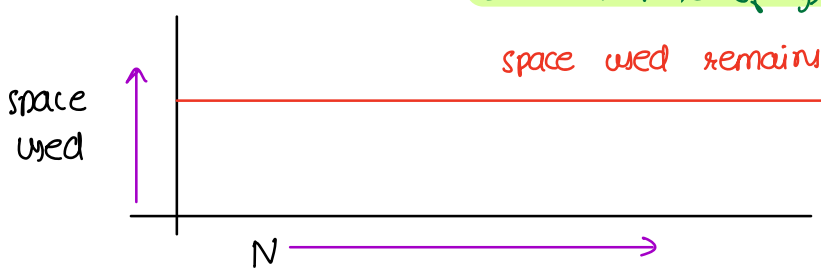
int ⟶ 4 bytes long ⟶ 8 bytes

Example

```
func (int N) {
    int x = 1 ;   // 4B
    long y = 2 ;  // 8B
}
```
12B

SC ⟶ Rate of growth of fn wrt N

space used remains constant over time.



12B ⟶ O(1)

input Algorithm Output

NOTE ⟶ Input and output are usually pre-defined only

Algorithm is in your hands

We only consider SC for algorithm ⟶ Not for Input, Not for Output

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

} C { lower order terms }

$4N + C \longrightarrow O(4N + C)$

$\longrightarrow O(N)$

```
func (int N) {
    .
    .
    .                             } lower order terms
    long[][] = new long[N][N]     // 8 * N * N
}
```

$SC = 8N^2 + \text{lower order terms}$

$O(N^2)$

```
func (int N, A[]) {
    int ans = A[0]                // 4 bytes
    for (i ⟶ 1 to N-1) {          // i ⟶ 4 bytes
        ans = max(ans, A[i])
    }
    return ans                    SC: O(1)
}
```

∵ A[] is part of input, it doesn't contribute towards SC

Array

collection of similar thing ⟶ array.

Declare an array of size N

int[] A = new int[size] , A = [0] * size

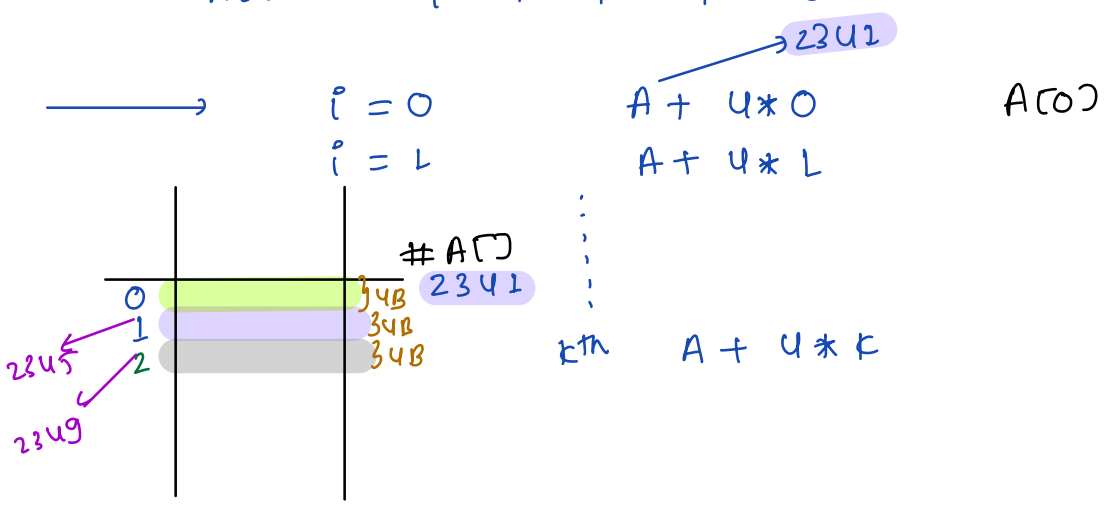
index of first element ⟶ 0

index of last element ⟶ N-1

Reason why it starts from 0

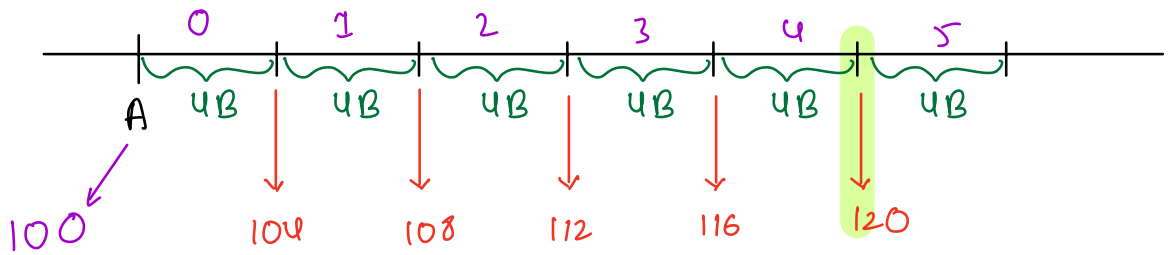
0 1 2 3

A[] = { 1 , 2 , 3 , 4 }



Print all the array elements.

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



$A[5] = A + 4 * i = A + 4 * 5 = 100 + 20$

$= 120$

Time Complexity to access $A[i] = O(1)$

⟶ $A = \{1, 2, 3, 4, 5\}$

0 1 2 3 4

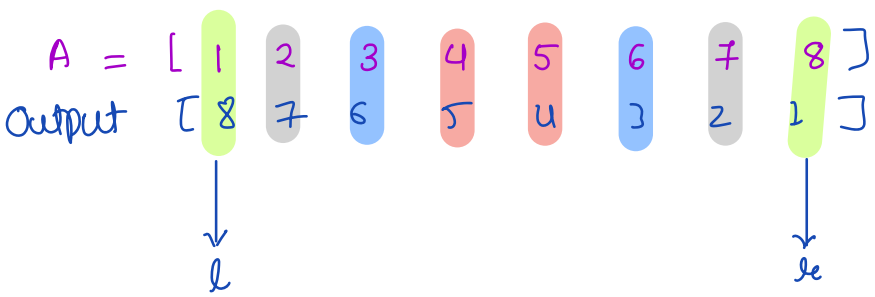
1st 2nd 3rd 4th 5th

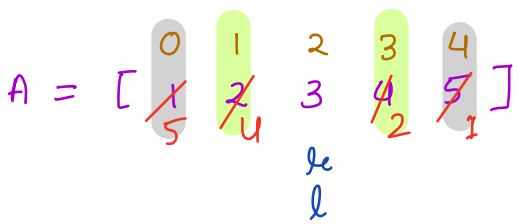
Sum first and fifth element

Q> Given an integer array. Reverse the array

NOTE ⟶ reverse the given array. Don't create new

```
          0  1  2  3  4
A = [     1  2  3  4  5 ]
Output ⟶ [ 5  4  3  2  1 ]
```

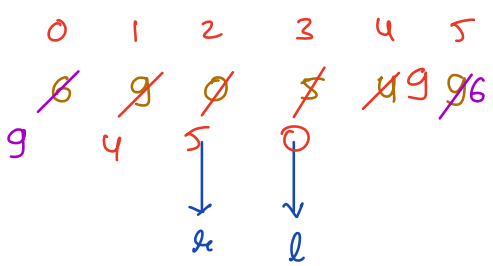




Pseudo code

```
void reverse ( int A[] ) {
    l = 0
    r = A.length - 1

    while ( l < r ) {
        temp = A[l]
        A[l] = A[r]
```

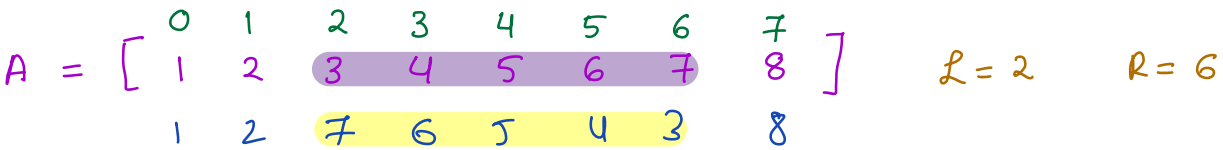


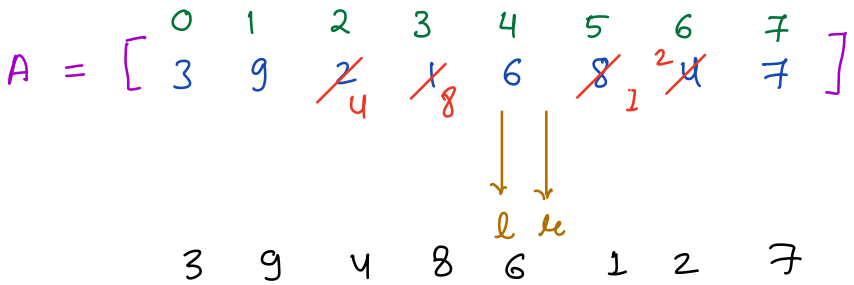
```
        A[r]  =  temp
        l  +=1
        r  -= 1
    }
}
```

TC : O(N)

SC : O(1)

Q> Reverse the array from index L to R L < R






```
void reverse (int A[], int L, int R) {
    l = L
    r = R

    while ( l < r ) {
        temp = A[l]
        A[l] = A[r]
        A[r] = temp
        l += 1
        r -= 1
    }
}
```

TC : O(N)

SC : O(1)

Q) * Given an integer array

Rotate the array from right to left K times

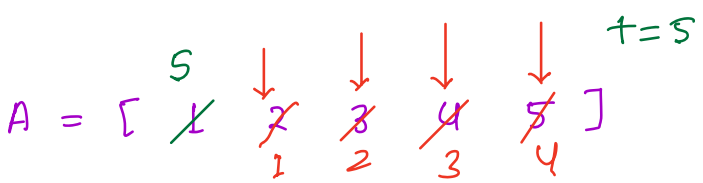
clockwise { forward rotation }

NOTE: No extra array to be used.

A = [	1	2	3	4	5]
k = 1	5	1	2	3	4
k = 2	4	5	1	2	3
k = 3	3	4	5	1	2

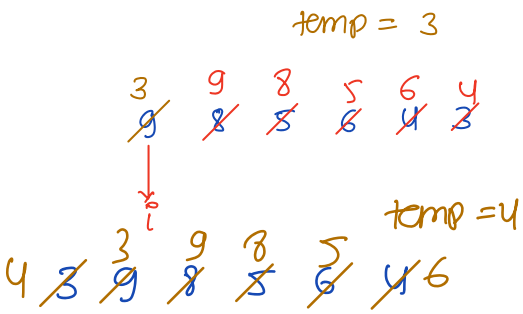
Bruteforce ⟶ Do as the question says.

Pseudocode




```
for j ⟶ 0 to k-1 {
    temp = A[N-1]
    for i ⟶ N-1 to 1 {
        A[i] = A[i-1]
    }
    A[0] = temp
}
```

k	i
0	N-1
1	N-1
⋮	
k-1	N-1



TC: O(k*(N-1))

O(k*N)

SC: O(1)

Break : 22:38

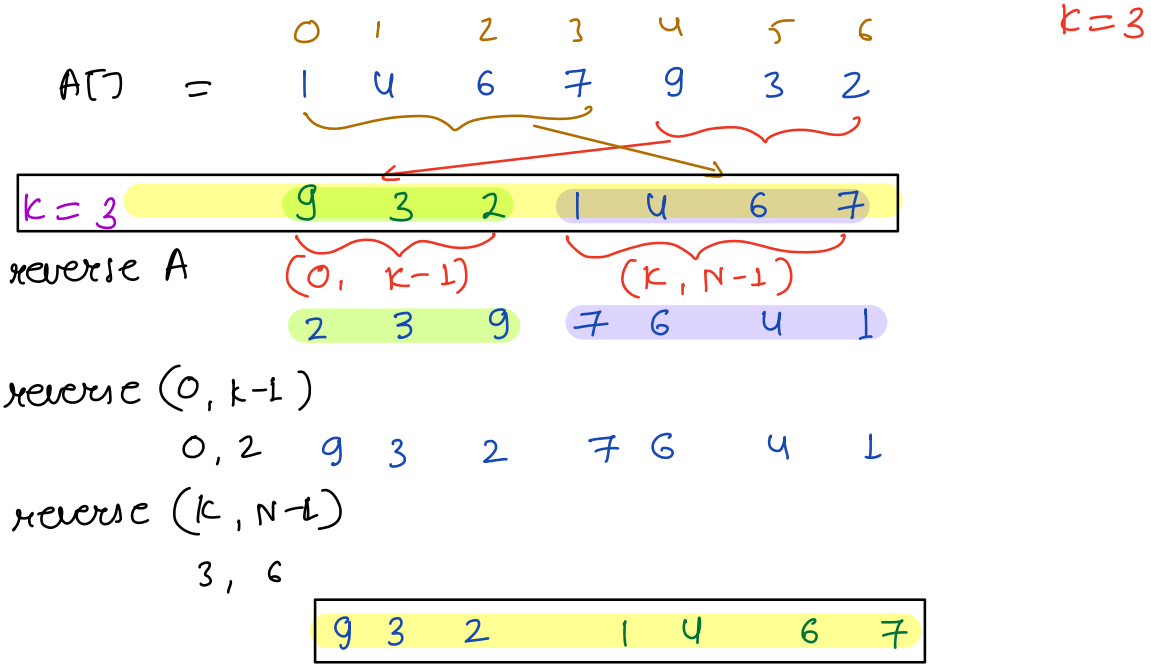
$k = k \% N$ $k = 15$

$k=0$	A = [1	2	3	4]	4%4	8%4	12%4
$k=1$	4	1	2	3	5%4	9%4	13%4
$k=2$	3	4	1	2	6%4	10%4	14%4.....
$k=3$	2	3	4	1	7%4	11%4	15%4

$k = 3$

A = [	1	4	6	7	9	3	2]
$k=1$	2	1	4	6	7	9	3
$k=2$	3	2	1	4	6	7	9
$k=3$	9	3	2	1	4	6	7

$k = 3$



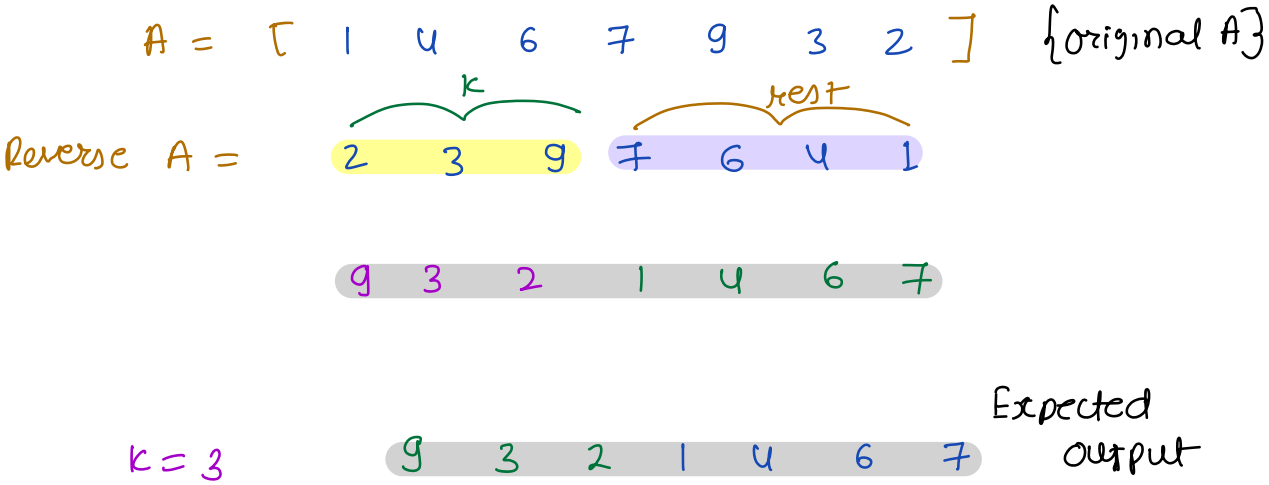
```
void reverse (int A[], int l, int r) {
    l = l
    r = r

    while (l < r) {
        temp = A[l]
        A[l] = A[r]
        A[r] = temp
        l += 1
        r -= 1
    }
}
```

```
void rotateKTimes (int[] A, int k) {        // N length
    k = k % N
    // step 1 reverse entire Aseray
    reverse (A, 0, N-1)      // O(N)
    // step 2  reverse first k element
    reverse (A, 0, k-1)      // O(k)
    // step 3  reverse remaining elements
    reverse (A, k, N-1)      // O(N-k)
}
```

TC : $O(N+k+N-k) = O(2N) = O(N)$

SC : $O(1)$



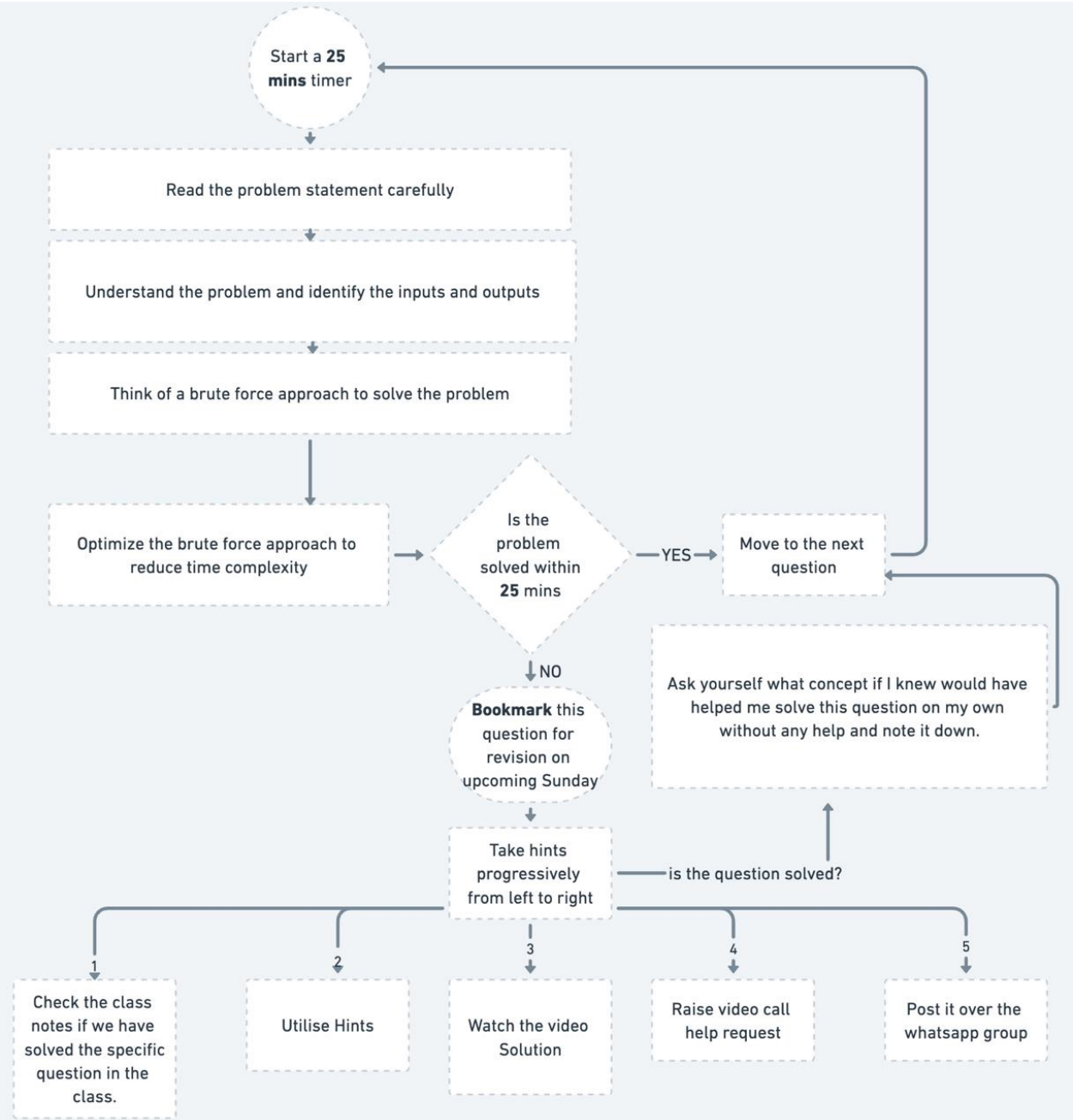
Dynamic Array. {H.W.}

An array will have fixed size.
Dynamic array doesn't have a fixed size

Java ⟶ ArrayList and basic operation

Python ⟶ List and its operations

C++ ⟶ vector


Start a **25 mins** timer
Read the problem statement carefully
Understand the problem and identify the inputs and outputs
Think of a brute force approach to solve the problem
Optimize the brute force approach to reduce time complexity
Is the problem solved within **25** mins
YES
Move to the next question
NO
Bookmark this question for revision on upcoming Sunday
Ask yourself what concept if I knew would have helped me solve this question on my own without any help and note it down.
Take hints progressively from left to right
is the question solved?
1
2
3
4
5
Check the class notes if we have solved the specific question in the class.
Utilise Hints
Watch the video Solution
Raise video call help request
Post it over the whatsapp group