Introduction of Arrays

Content

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

Space Complexity

Space complexity is the max amount of space used by your algorithm or function.

Big O

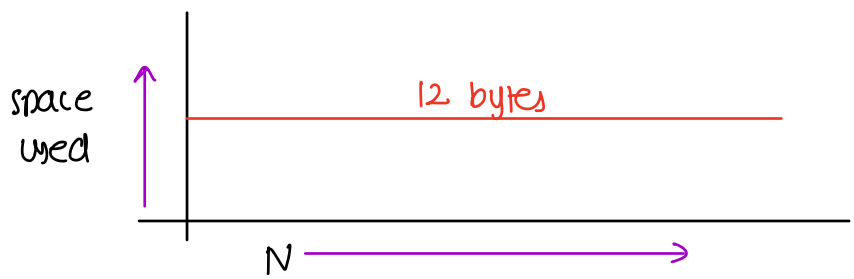
int ⟶ 4 bytes long ⟶ 8 bytes

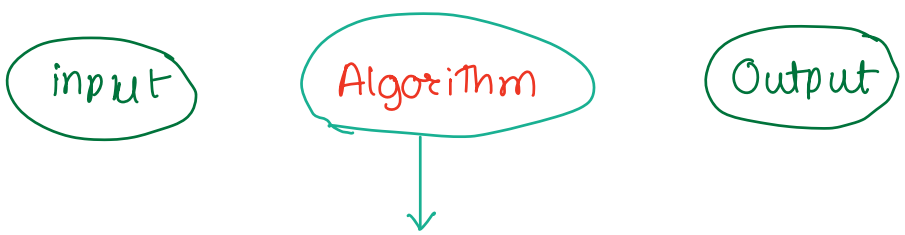
Example

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

total space utilised = 12 bytes

SC = O(1)





max space required by your algo

NOTE ⟶ Input & Output are not considered part of SC calculation.

```
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.
}
```

space used $4N + 56$

$O(4N + 56)$

$O(N)$

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

$SC = O(N^2)$

```
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
}
```

$SC = O(1)$

Array

collection of similar thing ⟶ array.

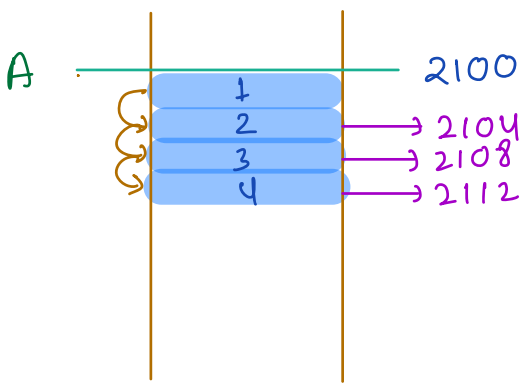
Declare an array of size N

```
int Arr = new int[N]
```

index of first element ⟶ 0

index of last element ⟶ N − 1

Reason why it starts from 0



A = [1 2 3 4]

TC to access A[i]

A[i] = A + i * 4

A[0]	2100 + 0 * 4
A[1]	2100 + 1 * 4
A[2]	2100 + 2 * 4
	⋮

Array will always be stored in contiguous block of memory

Print all the array elements.

```
        0  1  2  3
A = [ 1, 2, 3, 4 ]        N = 4
```

```
for (i = 0; i < A.length; i++) {
    print(A[i])
}
```

Time Complexity to access A[i] =

A + i * 4 ⟶ constant operation

TC: O(1)

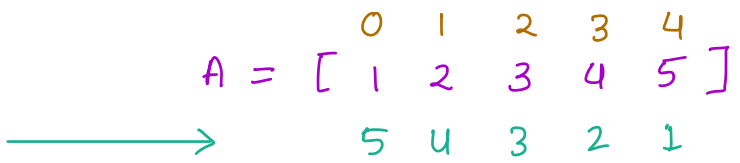
⟶

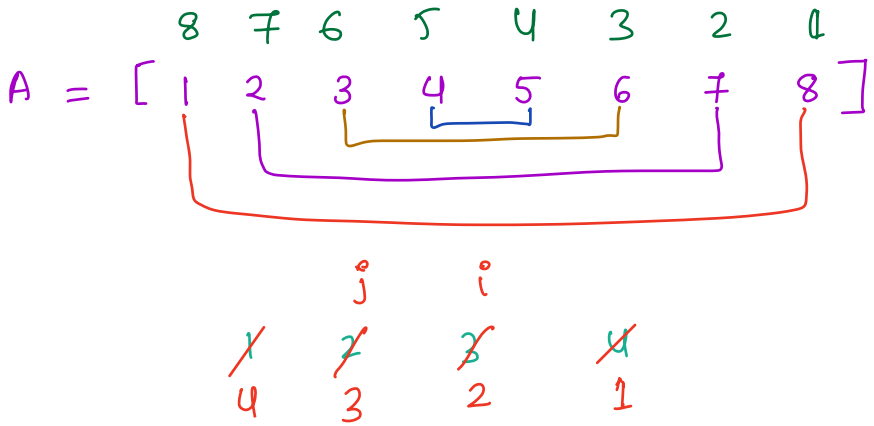
```
        0  1  2  3  4
A = { 1  2  3  4  5 }
    1st 2nd 3rd 4th 5th
```

Sum first and fifth element

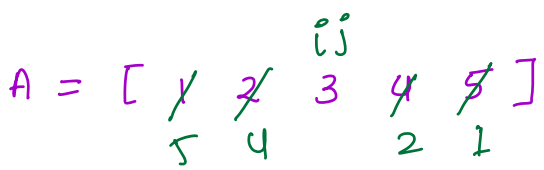
A[0] + A[4]

Q> Given an integer array. Reverse the array





we should stop swapping when $i >= j$



Pseudo code

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

    while ( i < j ) {
        // swap A[i] with A[j]
        temp = A[i]
        A[i] = A[j]
        A[j] = temp
        i = i+1
        j = j-1
    }
}
```

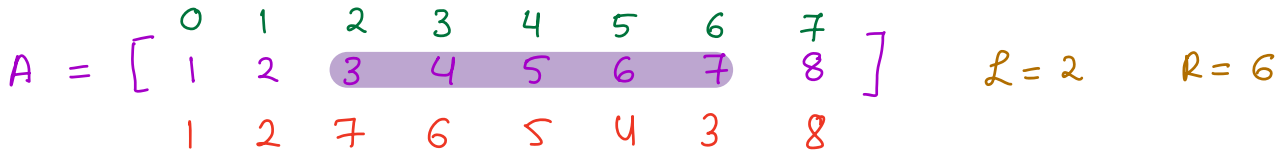
i → 5 ..... 10 ← j

t = 5

TC : O(N)

SC : O(1)

Q) Reverse the array from index L to R $L < R$



Same question as before

$i = L$

$j = R$

TC : O(N)

SC : O(1)

Break : 10:40

Q> * Given an integer array

Rotate the array from right to left { forward rotation } K times

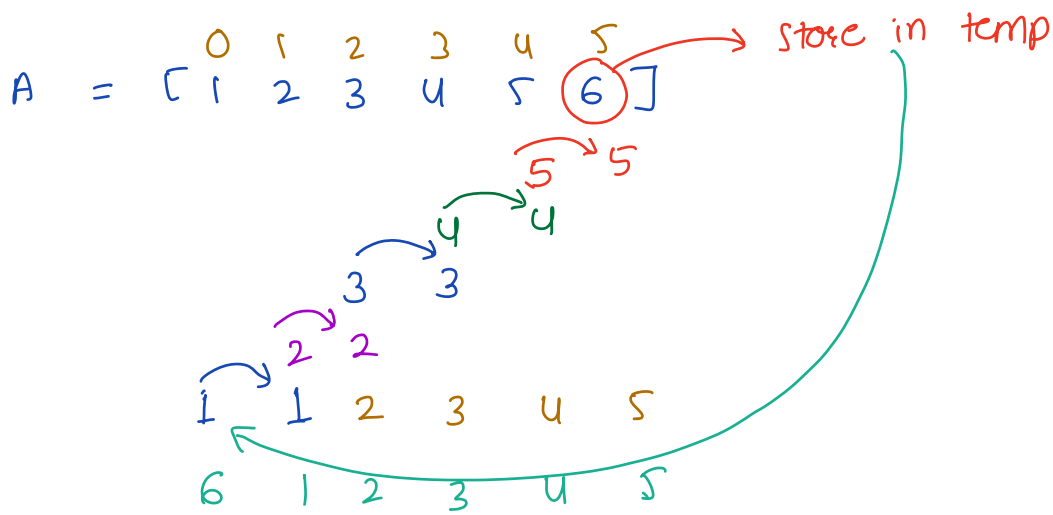
NOTE: No extra array to be used.

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



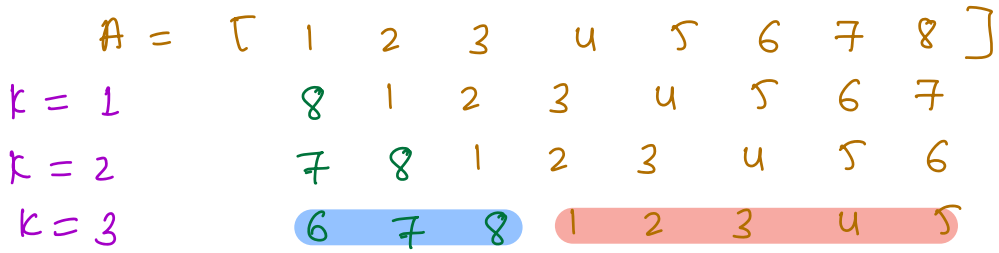
Pseudocode

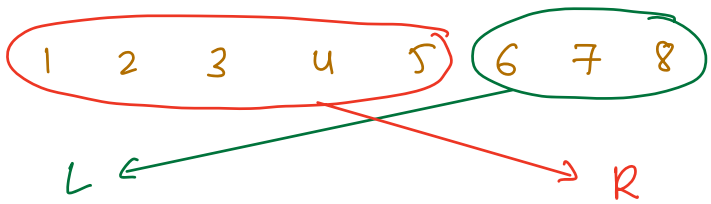
```
for ( j = 1 ; j <= K ; j++ ) {
    temp = A[N-1]
    for ( i = N-2 ; i >= 0 ; i-- ) {
        A[i+1] = A[i]
    }
    A[0] = temp
}
```

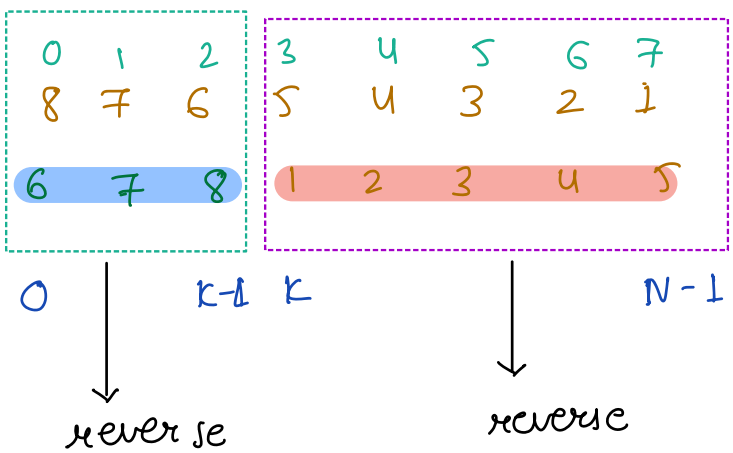
TC : O(K * N)

SC : O(1)

Brute force







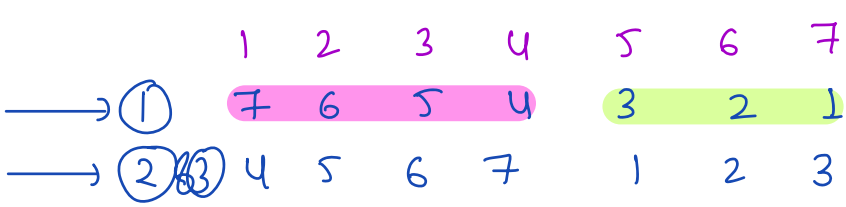
k = 3

Steps → 0

$k = k \% N$

1. reverse (A[], 0, N-1)
2. reverse (A[], 0, k-1)
3. reverse (A[0], k, N-1)

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



k = 4

A =	1	2	3	4
	4	1	2	3
	3	4	1	2
	2	3	4	1

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

From observation rotating 0 times is same as rotating 4, 8, 12 times.

$\Rightarrow$ $4 \% 4 = 8 \% 4 = 12 \% 4 = 0$

$5 \% 4 = 9 \% 4 = 13 \% 4 = 1$

$6 \% 4 = 10 \% 4 = 14 \% 4 = 2$

$7 \% 4 = 11 \% 4 = 15 \% 4 = 3$

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

Doubt session

```
for (i ⟶ 0 to 2^N - 1) {
        j = i
        while (j > 0) {
            j--
        }
}
```

i	j	# iterations
0	0	0
1	1	1
2	2	2
3	3	3
⋮	⋮	⋮
$2^N - 1$	$2^N - 1$	$2^N - 1$

t

0 to t

$\frac{t(t+1)}{2}$ $\quad \frac{t^2 + t}{2}$ $\quad O(t^2)$

$O((2^N - 1)^2)$

$(2^N - 1)^2$ $\quad (a-b)^2$

$a^2 + b^2 - 2ab$

$(2^N)^2 + 1 - 2(2^N)$ $\quad 2^N * 2^N$

$(2^N)^2$ $\quad 2^{2N}$ $\quad 4^N$

TC : $O(4^N)$