Day 5

11 November 2022 20:35

Easy 9.1K 12.6K

Companies

Given an integer array `nums` sorted in **non-decreasing order**, remove the duplicates **in-place** such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the **first part** of the array `nums`. More formally, if there are `k` elements after removing the duplicates, then the first `k` elements of `nums` should hold the final result. It does not matter what you leave beyond the first `k` elements.

Return `k` *after placing the final result in the first* `k` *slots of* `nums`.

Do **not** allocate extra space for another array. You must do this by **modifying the input array in-place** with O(1) extra memory.

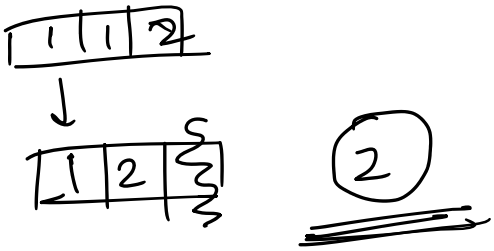
Example 1:

```
Input: nums = [1,1,2]
Output: 2, nums = [1,2,_]
Explanation: Your function should return k = 2, with the first two elements of nums being 1 and 2 respectively.
It does not matter what you leave beyond the returned k (hence they are underscores).
```

Example 2:

```
Input: nums = [0,0,1,1,1,2,2,3,3,4]
Output: 5, nums = [0,1,2,3,4,_,_,_,_,_]
Explanation: Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.
It does not matter what you leave beyond the returned k (hence they are underscores).
```

Ex1



Ex2 [0, 0, 1, 1, 1, 2, 2, 3, 3, 4]

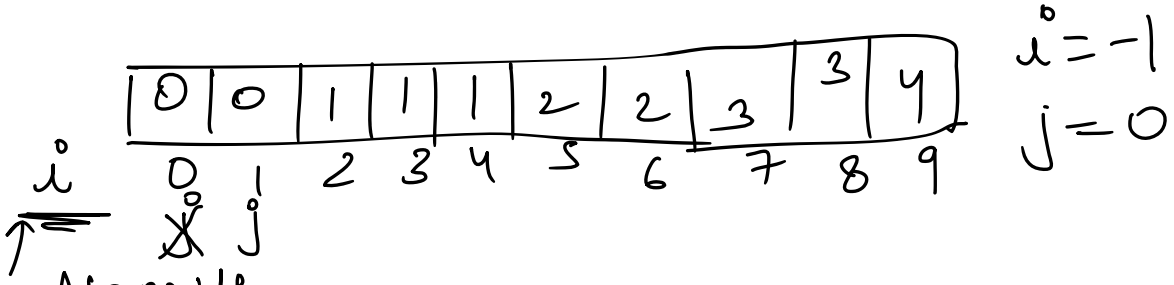
[0, 1, 2, 3, 4, _, _, _, _, _]

5

Ans

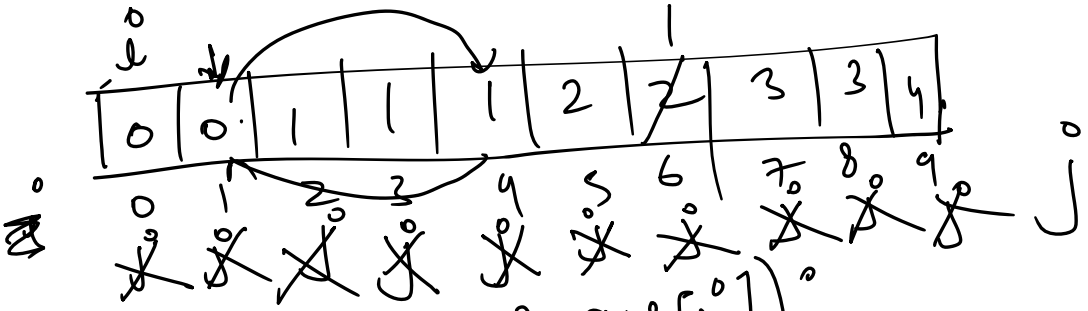
Logic

Inplace → No Extra

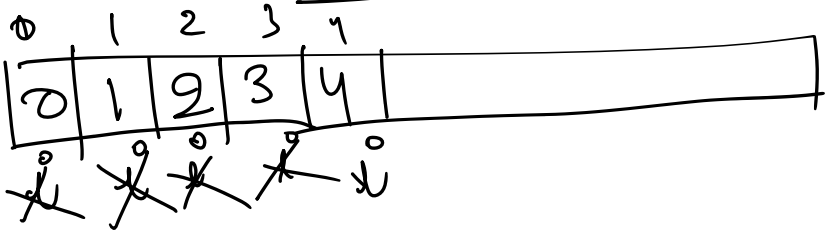


Algorithm

swap(arr[i+1], arr[j])



swap(arr[i+1], arr[j]);



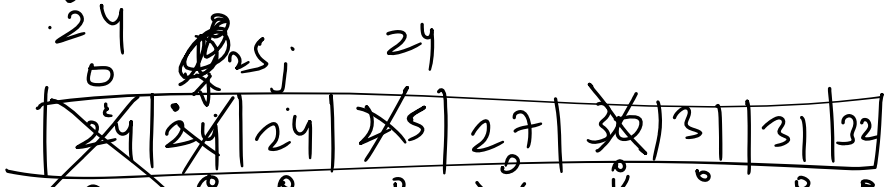
i+1 ✓

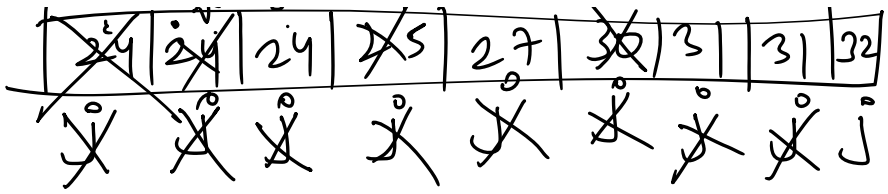
Ip 24 24 24 25 27 30 31 31 32

modified

24 25 27 30 31 32 — — —

Size = 6 ✓



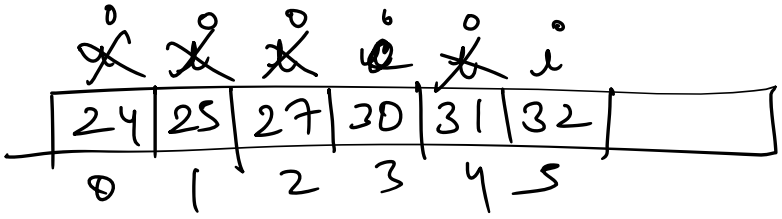


i=-1
j=0

last occurence

swap(arr[i+1], arr[j])

i++j



i+1 ⟹ 6 ✓

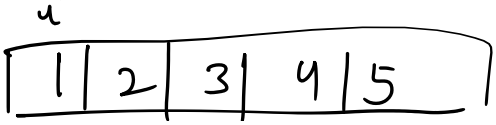
```
int removeDuplicates(vector<int>& nums) {
    int i = -1,j = 0;
    while(j < nums.size()){
        if(j + 1 < nums.size() && nums[j] == nums[j+1]){
            j++;continue;
        }
        // last occurence
        swap(nums[i+1],nums[j]);
        i++;
        j++;
    }
    return i+1;
}
```

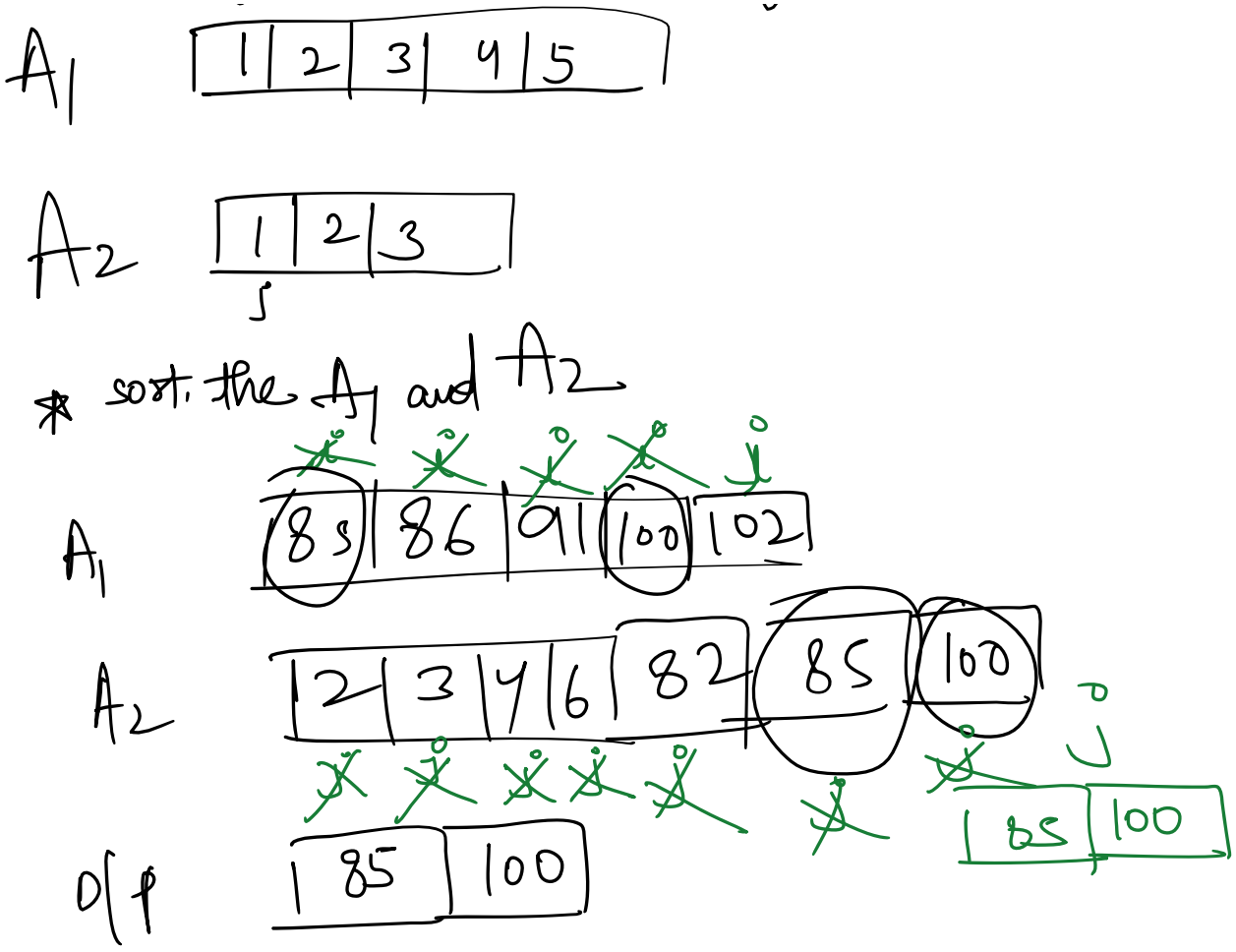
T.C - O(N)

S.C - O(1) ✓

Intersection of Two Arrays

A1





Algorithm

* sort A_1 and A_2

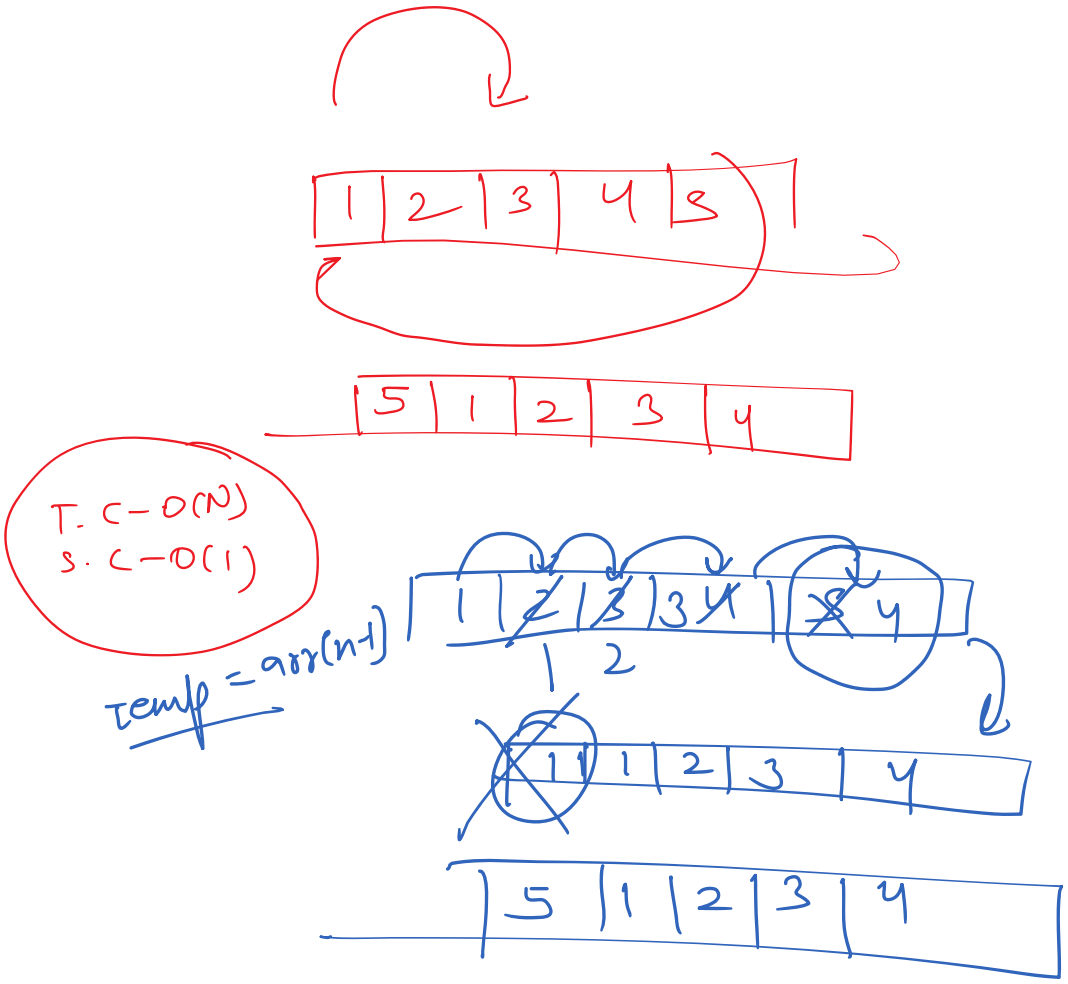
```
i=0, j=0; c=0;
while( i<n && j<m){
    if( A1[i] < A2[j]){
        i++;
    else if( A1[i] > A2[j]){
        j++;
    }else{
        c++; i++; j++;
```

} else {
cnt++; i++; j++;
}

Given an array, rotate the array by one position in clock-wise direction.

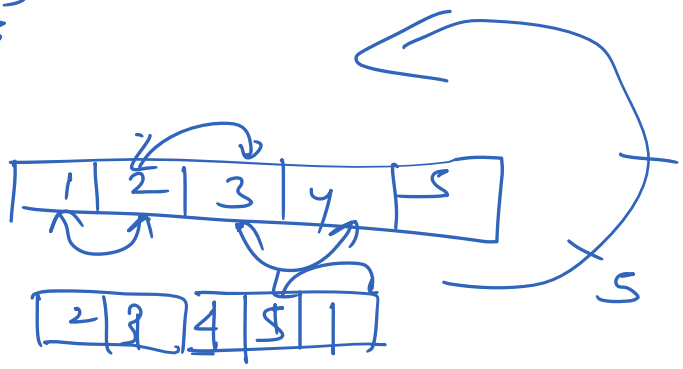
Example 1:

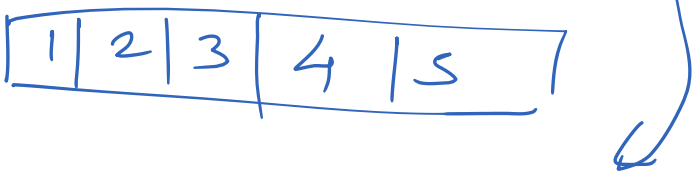
Input:
N = 5
A[] = {1, 2, 3, 4, 5}
Output:
5 1 2 3 4

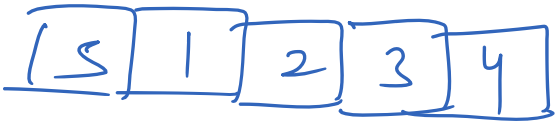


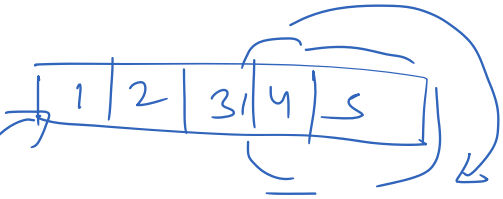
T.C-O(N)
S.C-O(1)

S.C – O(1)







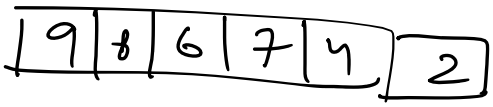


k = 2

1st time

O/p

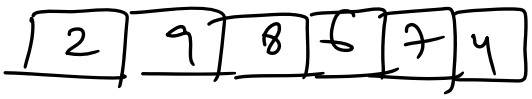
2nd time



k = 4

4 times clockwise

1st



2nd

3rd

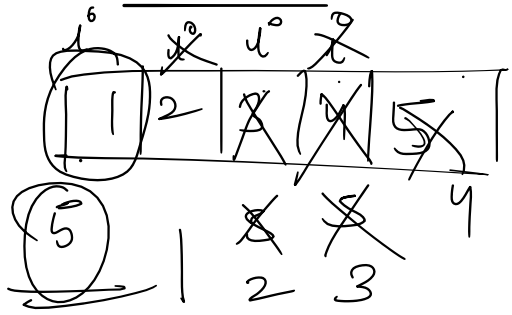
7	4	2	9	8	6

4th

6	7	4	2	9	8

O/P

O(n x k)



```
for(int i=0; i<k; i++){
    int temp = arr[n-1];
    int i = n-2;
    while(i >= 0){
        arr[i+1] = arr[i];
        i--;
    }
    arr[0] = temp;
}
```

O(N x k)

TLE

Different Approach

9	8	6	7	4	2

k = 4

O/P

6	7	4	2	9	8

Step1 last k elements ko reverse

k = 3

9	8	2	4	7	6

step2 6 से n-k elements तक reverse

k=3 तक reverse

8	9	2	4	7	6

Step3 All elements should be reversed

6	7	4	2	9	8

O/P

T.C — O(N)

Anticlockwise →

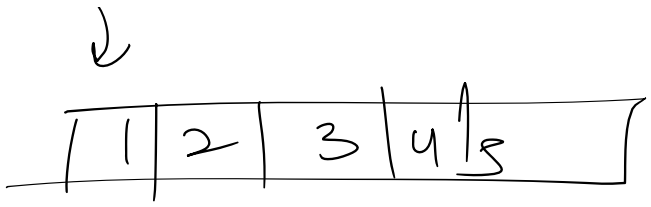
S1 starting से k elements reverse

S-2 k+1 to n-1 "

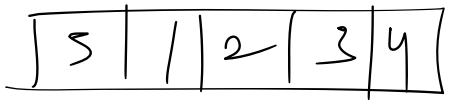
S-3 Array reverse

k=3

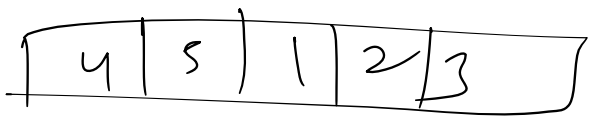
k=3



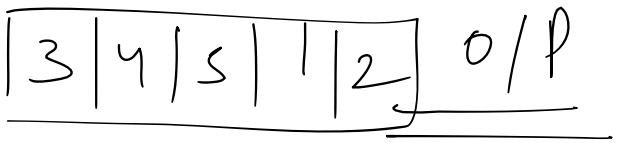
O/P 1st



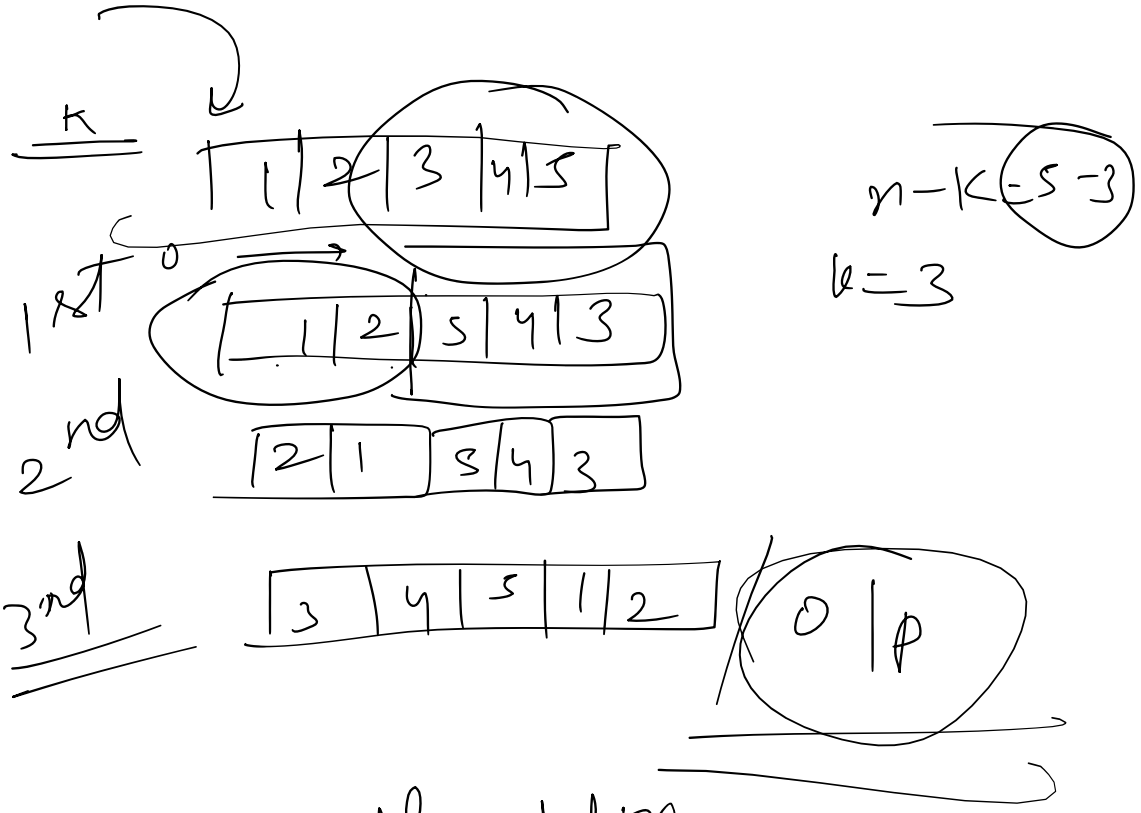
2nd



3rd



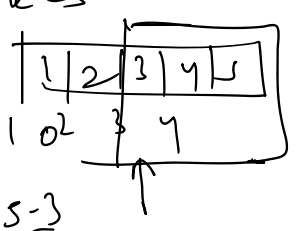
O/P



Clockwise Implementation

1st ~~ok~~ reverse(arr + (n-k), arr + n);

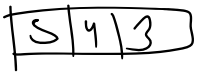
k=3



5-3 = 2

arr + (5-3), arr + 5

reverse(arr + 2, arr + 5)



=(2)

5 | 4 | 3

2nd step reverse(arr, arr + (n-k))

(4) · 5 reverse(arr, arr + 2)
arr+5

2 | 1 | 5 | 4 | 3

3rd step reverse(arr, arr + n)

3 | 4 | 5 | 1 | 2 O/P

k ⇒ 0 to n

1 | 2 | 3 | 4 | 5 k = 10000

n = 5

1st 5 | 1 | 2 | 3 | 4

2nd 4 | 5 | 1 | 2 | 3

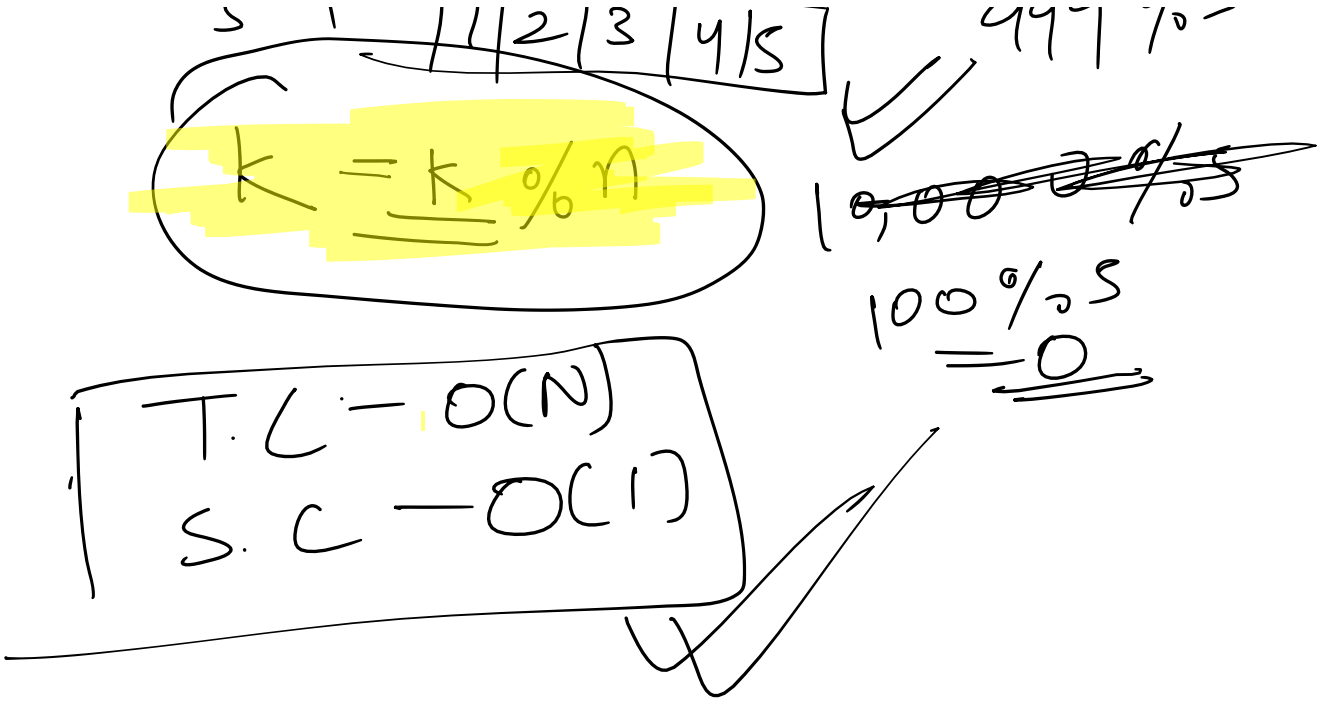
3rd 3 | 4 | 5 | 1 | 2

4th 2 | 3 | 4 | 5 | 1

5th 1 | 2 | 3 | 4 | 5

O/P

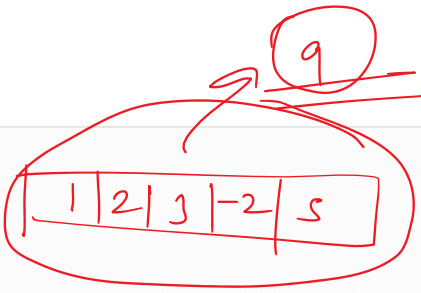
10000 % 5 = 0

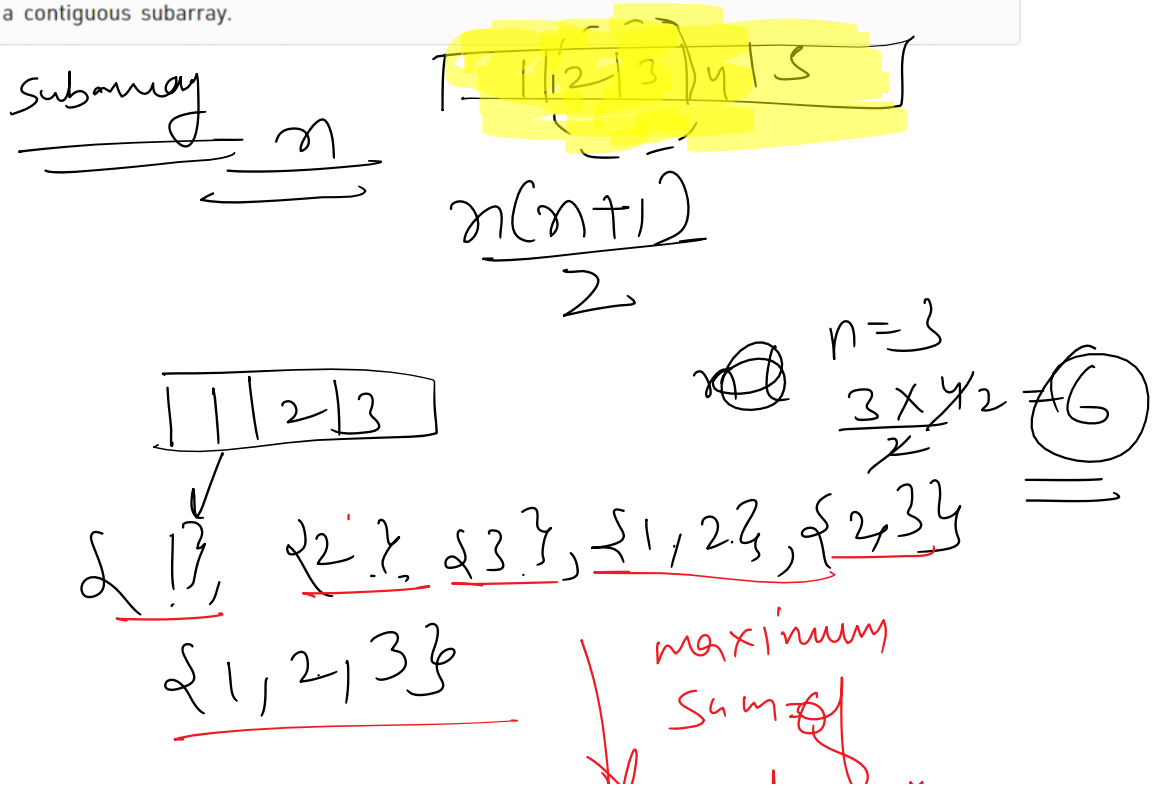


Given an array **Arr[]** of **N** integers. Find the contiguous sub-array(containing at least one number) which has the maximum sum and return its sum.

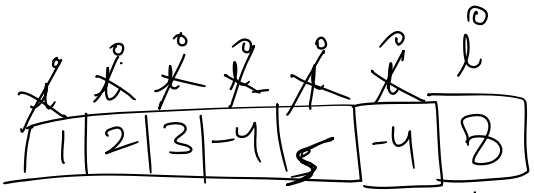
Example 1:

Input:
N = 5
Arr[] = {1,2,3,-2,5}
Output:
9
Explanation:
Max subarray sum is 9
of elements (1, 2, 3, -2, 5) which
is a contiguous subarray.





Sum of subarray



-1	-2	-3	-4

cs = 0 1 2 3 4 5 6 7 8 11

ls = INT_MIN 1 2 3 4 5 6 7 11

Ans 11

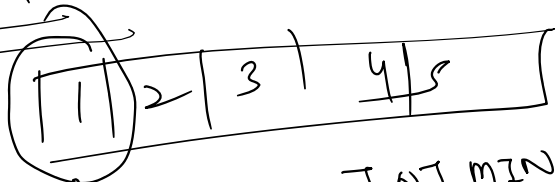
T.C — O(N)

S.C — O(1)

Naive Algo

- array ? subarray generate करो
- Sum Compare max sum से return करो

Implement




```
int ms = INT_MIN;
for(int i=0; i<n; i++) {
    for(int j=i; j<n; j++) { int s=0;
        for(int k=i; k<=j; k++) {
            s += arr[k];
        }
        ms = max(ms, s);
```

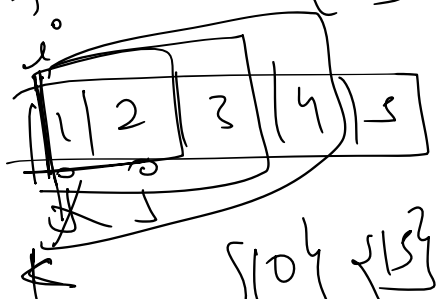
{1} {2} {3} {4}

Classes Page 12

ms = max(ms, s);

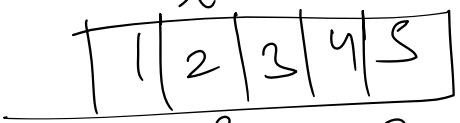
}

return ms;



{6}

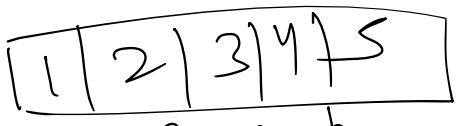
{10} {15}



{2}, {5}, {9} {14}

T.C - O(N^3)
S.C - O(1)

TLE



{1} {3} {6} {10} {15}
{2} {5} {9} {14}
{3} {7} {12}
{4} {9}

```
int ms = INT_MIN;
for(int i=0; i<n; i++){
    int s=0;
    for(int j=i; j<n; j++){
        s += a[j];
        ms = max(ms, s);
```

ms = max(ms, s);

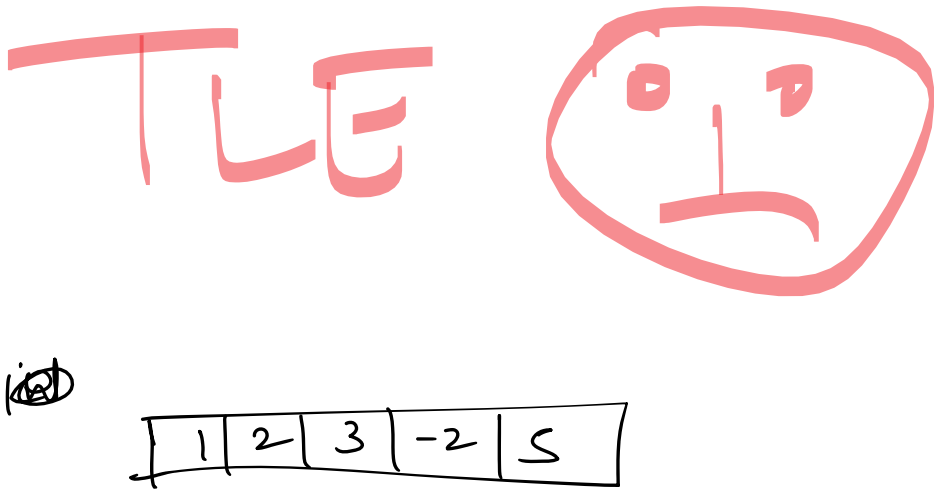
}

return ms;

T.C - O(N²)

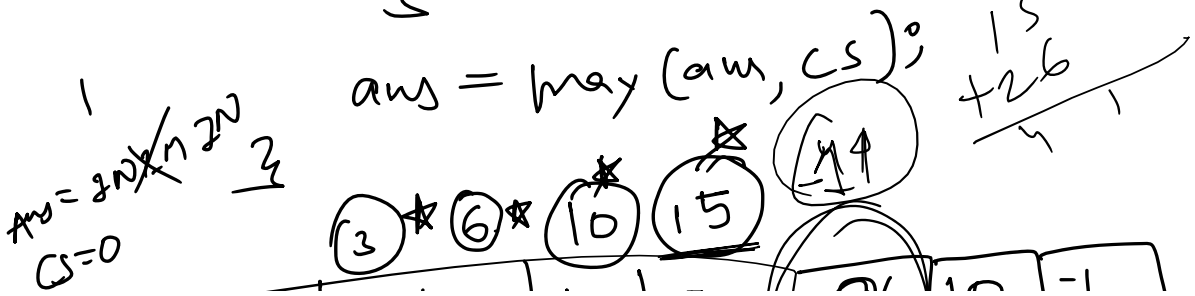
S.C - O(1)

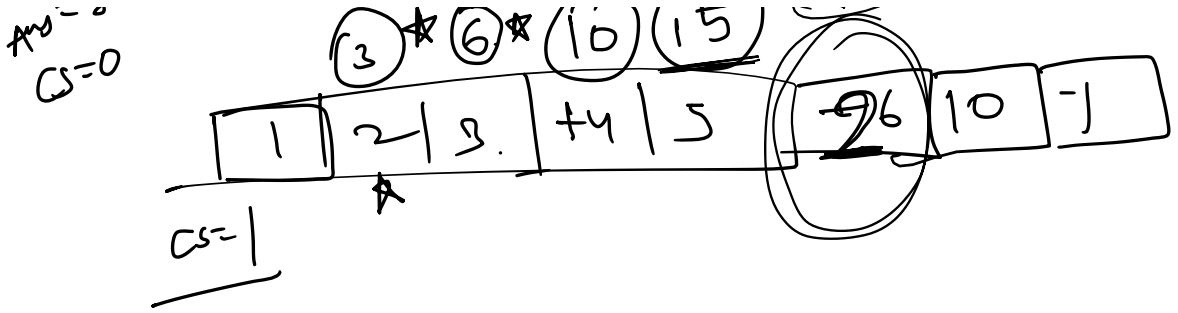
TLE

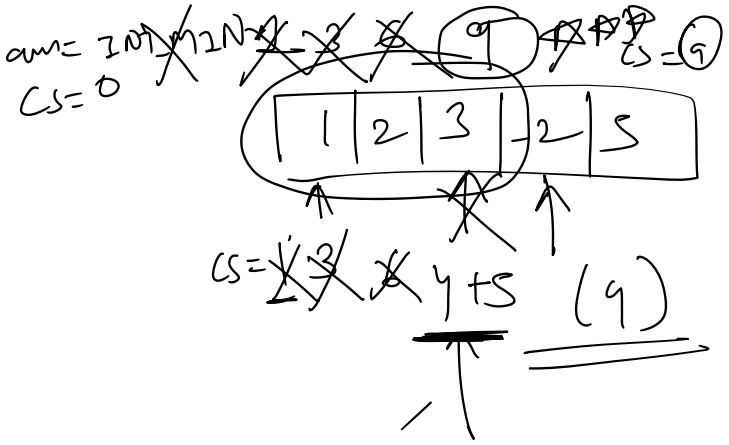


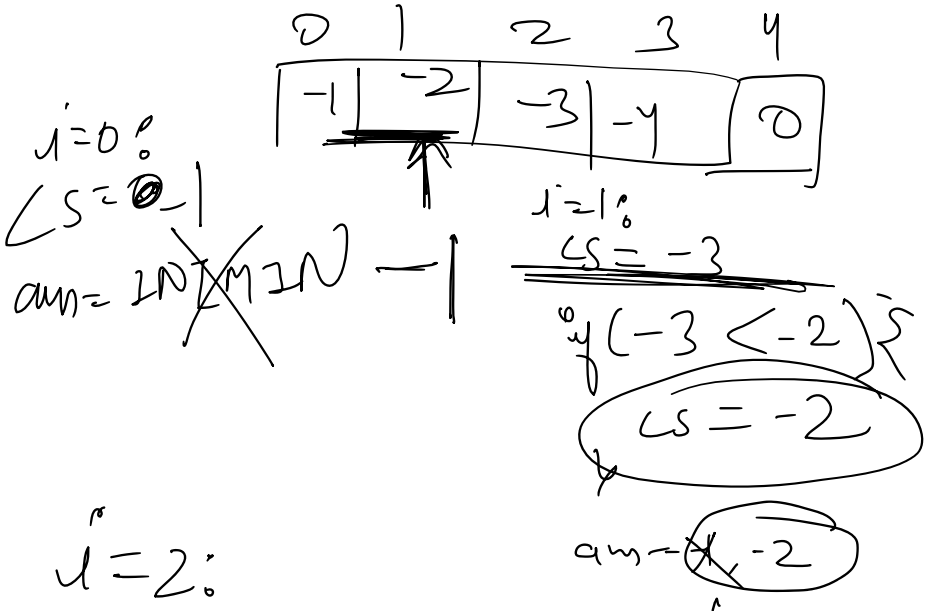
int ans = INT_MIN;
int cs = 0;

for(int i=0; i<n; i++){
    cs += a[i];
    if(cs < a[i]){
        cs = a[i];
    }
    ans = max(ans, cs);
}




cs=0
3
6
10
15
1
2
3
4
5
6
10
-1
cs=1


cs=0
1
2
3
-2
5
cs=4+5 (9)


0
1
2
3
4
-1
-2
-3
-4
0
i=0:
cs=0-1
ans=INT_MIN-1
i=1:
cs=-3
if(-3<-2){
cs=-2
}
ans=-2
i=2:
cs=-5
if(-5<-3){
cs=-3
}
ans=-1,-3
-1
i=3:
cs=-3+-4=-7
if(-7<-4)
cs=-4

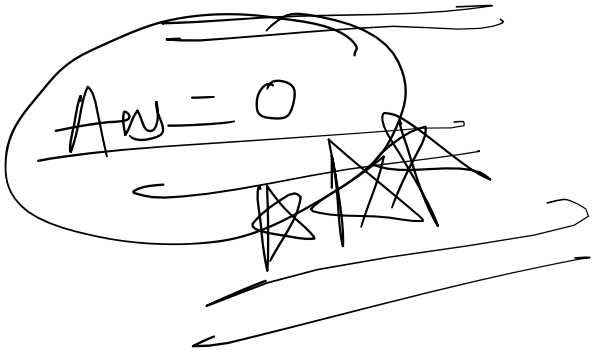
i = 4;

cs = -4 + 0 = -4

if(-4 < 0){

cs = 0;

Ans = 0



* Algorithm

(1) Cumulative Sum Calculate करने का

If(cs < a[i]){

cs = a[i]

}

ans = max(ans, cs)

T.C - O(N)

S.C - O(1)

```cpp
long long maxSubarraySum(int arr[], int n){
    long long ans = INT_MIN;
    long long cs = 0;
    for(int i = 0;i < n;i++){
        cs += arr[i];
        if(cs < arr[i]){
            cs = arr[i];
        }
        ans = max(ans,cs);
    }
    return ans;
}
```

T.C – O(N)

S.C – O(1)