Day-1

*1)Finding Majority Element in an Array

3 — 2
4 — 5
2 — 2
9

n = 9, >= n/2 + 1

n = 10
min 6 (n/2 + 1)

Input : {3, 3, 4, 2, 4, 4, 2, 4, 4}
Output : 4
Explanation: The frequency of 4 is 5 which is greater than the half of the size of the array size.

Input : {3, 3, 4, 2, 4, 4, 2, 4}
Output : No Majority Element
Explanation: There is no element whose frequency is greater than the half of the size of the array size.

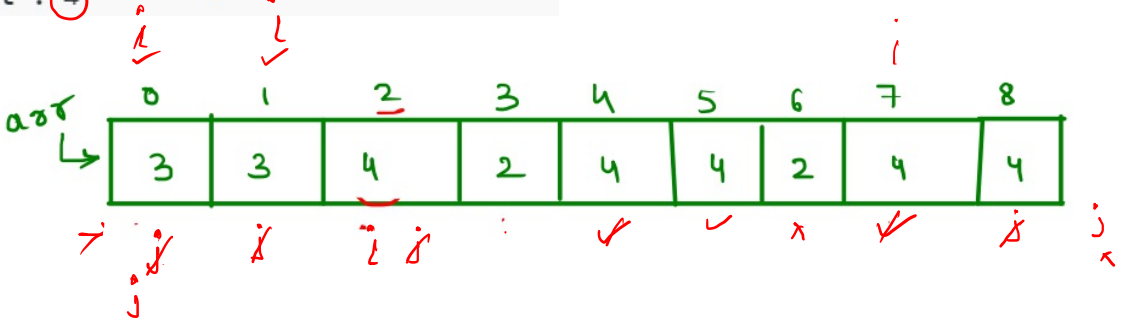
→ $O(n^2)$

→ $O(1)$

* Brute-Force

Input : {3, 3, 4, 2, 4, 4, 2, 4, 4}

Output : 4



* max_count = 0
2
5

index = 2

count = 0 1 2 3 4 5

```
function findMajority(arr[], n)
{
    maxCount = 0;
    index = -1;
    for (i = 0; i < n; i++) {
        count = 0;
        for (j = 0; j < n; j++) {
            if (arr[i] == arr[j])
                count++;
        }

        // update maxCount if count of
        // current element is greater
        if (count > maxCount) {
            maxCount = count;
            index = i;
        }
    }

    // if maxCount is greater than n/2
    // return the corresponding element
    if (maxCount > n / 2)
        print(arr[index]);

    else
        print("No Majority Element");
}
```

Nested
n * n

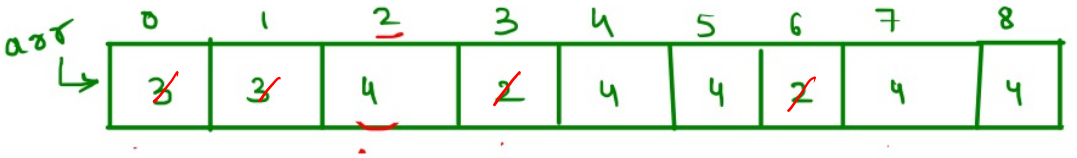
O(1)

O(1)

O(nlogn)

O(n)/O(1)

✓ Sorting Approach



i i i i i* i

0 1 2 3 4 5 6 7 8

arr → 2 2 3 3 4 4 4 4 4

n = 9

→ n/2 + 1

5 times

nlogn + n ⇒ O(nlogn)

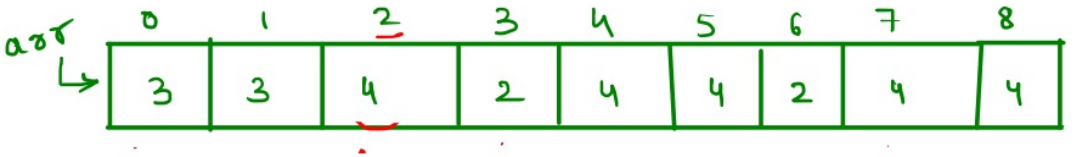
O(n) { O(1)

2 2

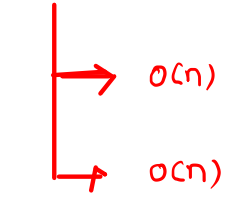
✓ ✓

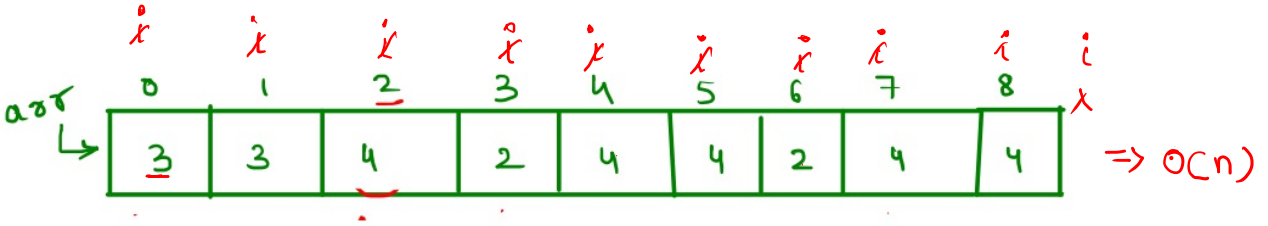
```
index = -1;
for(i=0; i<n/2; i++)
{
    if(arr[i] == arr[i+n/2])
    {
        index = i;
        break;
    }
}
```

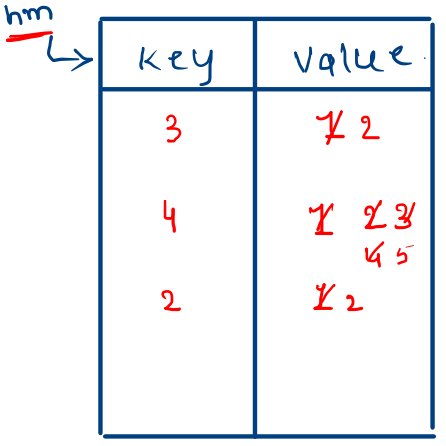
Sorting Approach



HashMap :- / object







all elements are distinct

O(n)

	AP1	AP2	AP3	AP4
T.C:	$O(n^2)$ ↓	$O(n\log n)$ ↓	$O(n)$	$O(n)$
S.C:	$O(1)$	$O(1)$ ↑	$O(n)$	$O(1)$ ✓

$n^2 \rightarrow n\log n$

$128 \times 128 \quad 128 \times 7$

$n = 128 \quad \log_2^{128} = 7$

time ↓ vs space ↑

✓ Moore-Voting Algo

arr

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

C = 2

→ O(n)

1. Returns some element as majority element

3 2

2. our responsibility to check, whether is maj / not

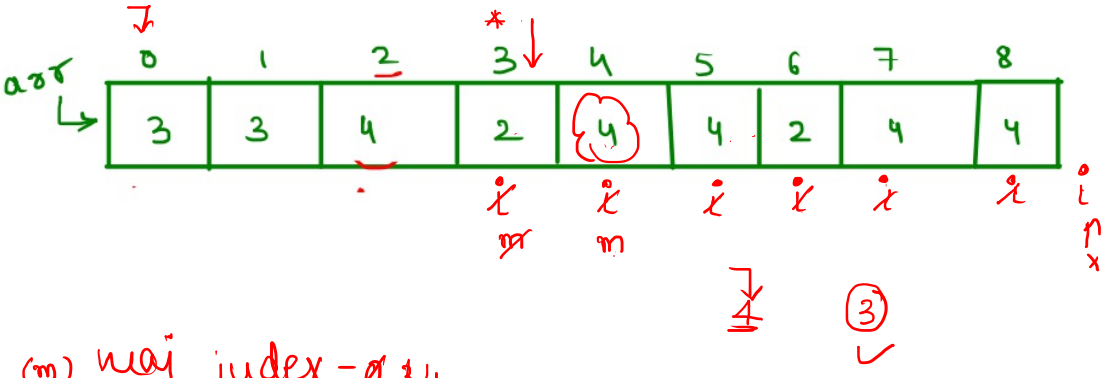
↳ ? ↳ O(n)

> $\frac{n}{2}$

O(n) + O(n) ⇒ O(n)

Moore's Voting Algorithm

```
int findCandidate(int a[], int size)
    {
        int maj_index = 0, count = 1;
        int i;
        for (i = 1; i < size; i++)
        {
            if (a[maj_index] == a[i])
                count++;
            else
                count--;
            if (count == 0)
            {
                maj_index = i;
                count = 1;
            }
        }
        return a[maj_index];
    }
```



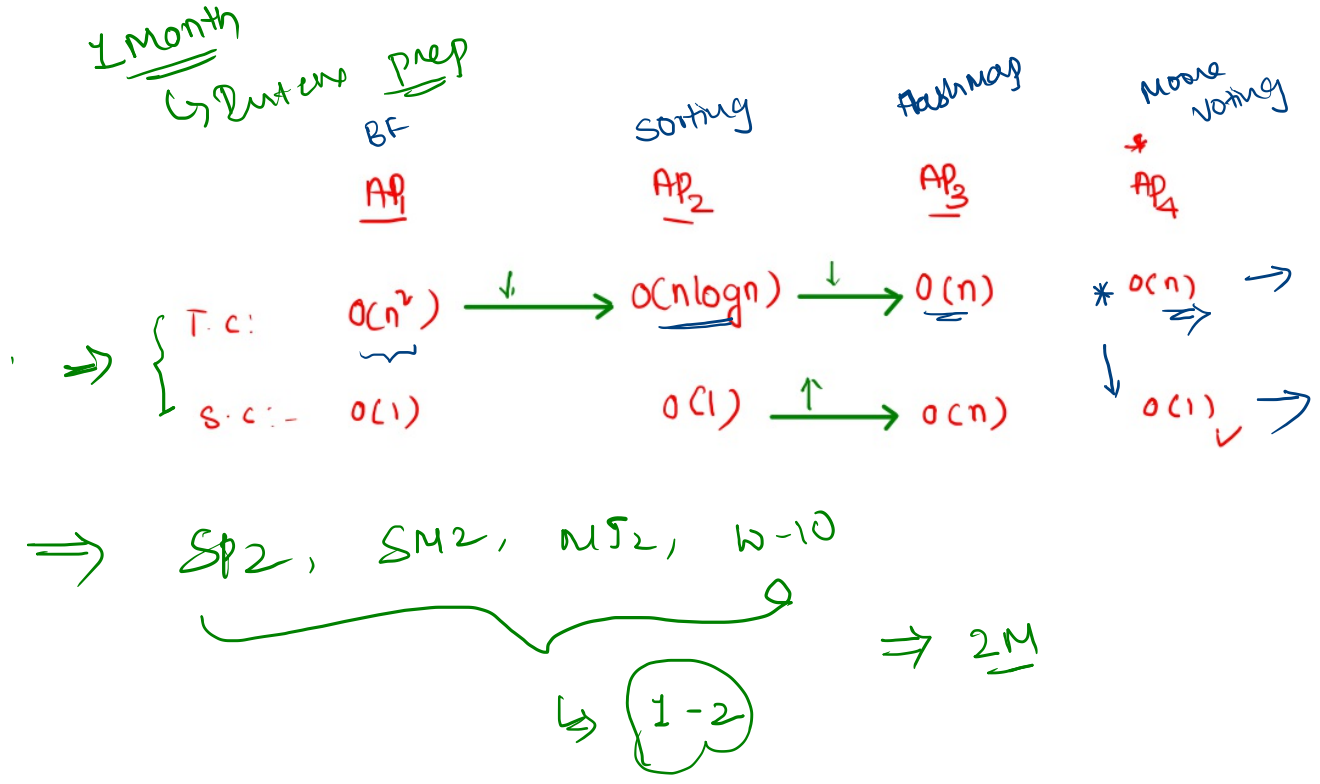
(m) maj_index = 0 3 4

Count = 1 2 1 0

= 1 0

= 1 2 1 1 3

arr[i] v/s arr[m]


1 Month
Interview prep
BF
Sorting
Hashmap
Moore voting
AP1
AP2
AP3
AP4
T.C:
S.C:-
O(n²)
O(1)
O(nlogn)
O(1)
O(n)
O(n)
O(n)
O(1)
SP2, SM2, MS2, W-10
2M
1-2

2) Two elements whose sum is close to zero

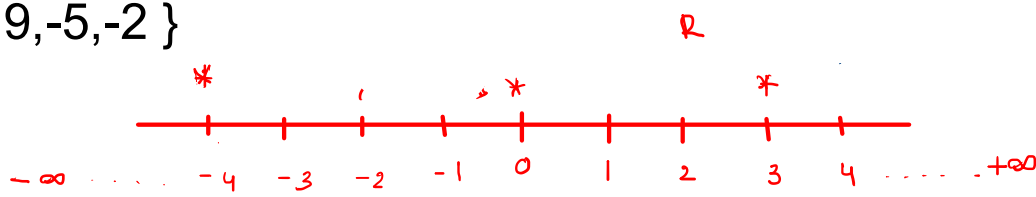
arr[]= { 1,2,6,9,-5,-2 }

O(n²)

O(1)

Brute-Force

arr[]= { 1,2,6,9,-5,-2 }



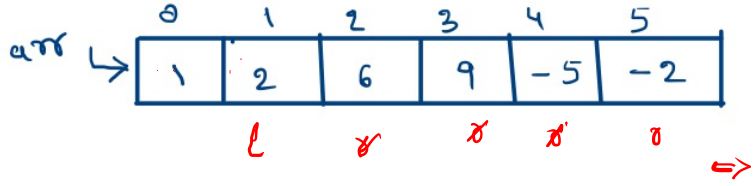
min_l = ∅ ∅ 1

min_r = ∅ ∅ 5

min_sum = -1

3

1



Sum = ∅ -5 0

0

① (-2, 0) (or) ② (1, -2)

-2 -1

Abs(-2) Abs(-1)

2 1 ✓

```
function  minAbsSumPair(int  arr[], int  arr_size)
{

  int  l, r, min_sum, sum, min_l, min_r;
  if(arr_size < 2)
  {
    System.out.println("Invalid Input");
    return;
  }
  min_l = 0;
  min_r = 1;
  min_sum = arr[0] + arr[1];
  for(l = 0; l < arr_size - 1; l++)
  {
    for(r = l+1; r < arr_size; r++)
    {
      sum = arr[l] + arr[r];
      if(Math.abs(min_sum) > Math.abs(sum))
      {
        min_sum = sum;
        min_l = l;
        min_r = r;
      }
    }
  }
  print(arr[min_l] , arr[min_r])
}
```

OJ – Contest

$(a+b) \simeq 0$

1, 0

2/3

1/0

⇒ ↳ X

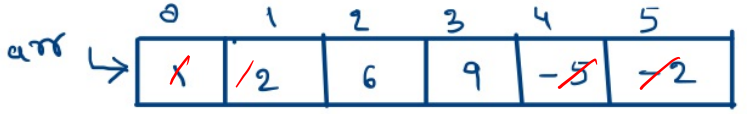
→ O(nlogn)

→ O(n)/O(1)

** sorting + 2 pointer

-ve → 0 ← +ve

↳ (l < r)



nlogn + n

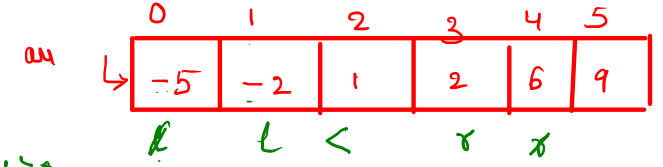
0 ~ 1

-3

min_l = 0 1

min_r = 4 3

min_sum = 4 1 0



sum > 0

(7) (l++)

(1) r--

sum = 4 1 -3 0

-3

sum < 0

↳ l++

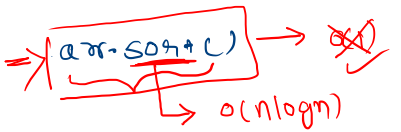
(*)

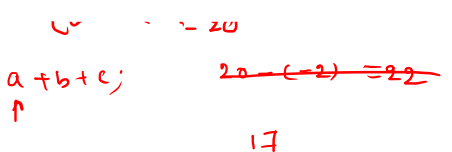
sum > 0

↳ r--

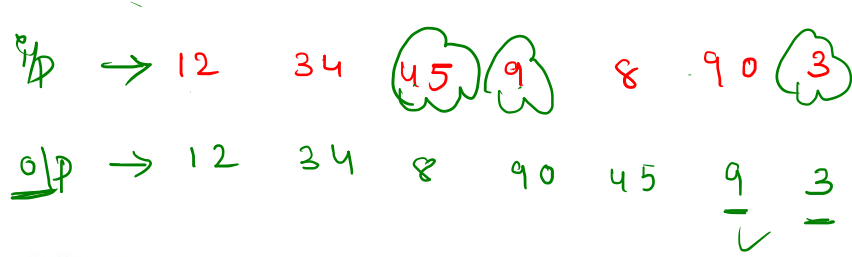
sorting + 2-pointer

```
static  void  minAbsSumPair(int  arr[], int  n)
    {

      int  sum, min_sum = 999999;

      int  l = 0, r = n-1;

      int  min_l = l, min_r = n-1;
      if(n < 2)
      {
        System.out.println("Invalid Input");
        return;
      }
      while(l < r)
      {
        sum = arr[l] + arr[r];
        if(Math.abs(sum) < Math.abs(min_sum))
        {
          min_sum = sum;
          min_l = l;
          min_r = r;
        }
        if(sum < 0)
          l++;
        else
          r--;
      }
       print(arr[min_l] to arr[min_r]
    }
```





⇒ 3)seperate odd and even



Given an array A[], write a function that segregates even and odd numbers. The functions should put all even numbers first, and then odd numbers.

Example:

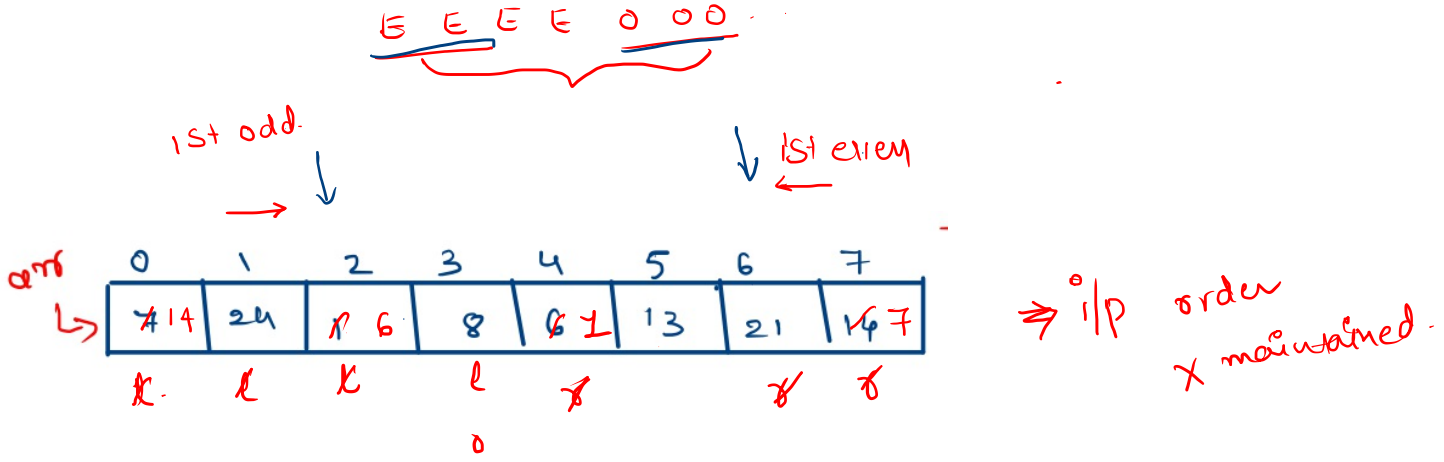
```
Input  = {12, 34, 45, 9, 8, 90, 3}
Output = {12, 34, 8, 90, 45, 9, 3}
```

i/p order maintained. ⇒ ✓ E[], O[] → combine.

i/p order not maintained. ⇒ x

In the output, the order of numbers can be changed, i.e., in the above example, 34 can come before 12 and 3 can come before 9.

2-pointer


E E E E O O O
1st odd
1st even
arr
0 1 2 3 4 5 6 7
14 24 6 8 7 13 21 7
⇒ i/p order
X maintained

```
segregateEvenOdd(int  arr[])
{
    /* Initialize left and right indexes */
    int  left = 0, right = arr.length - 1;

    while  (left < right)
    {
        /* Increment left index while we see 0 at left */
        while  (arr[left]%2  == 0  && left < right)
            left++;

        /* Decrement right index while we see 1 at right */
        while  (arr[right]%2  == 1  && left < right)
            right--;

        if  (left < right)
        {
            /* Swap arr[left] and arr[right]*/
            int  temp = arr[left];
            arr[left] = arr[right];
            arr[right] = temp;
            left++;
            right--;
        }
    }
}
```

Day-2

(a+b) → O(n²)

(a,b,c)

1) Triplet whose sum is x

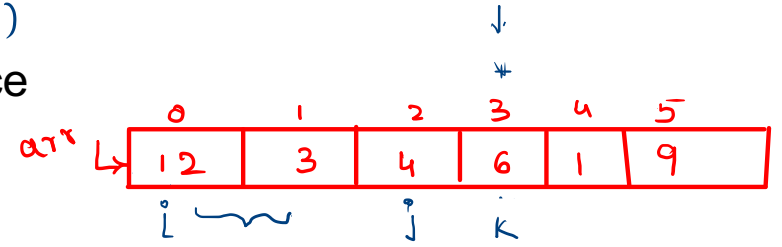
Input: *array = {12, 3, 4, 1, 6, 9}, sum = 24;*
Output: *12, 3, 9*
Explanation: *There is a triplet (12, 3 and 9) present in the array whose sum is 24.*
Input: *array = {1, 2, 3, 4, 5}, sum = 9*
Output: *5, 3, 1*
Explanation: *There is a triplet (5, 3 and 1) present in the array whose sum is 9.*

O(n³)

O(1)

Brute-Force

sum = 24 ;



12 + 3

15 + 4

6

4

9

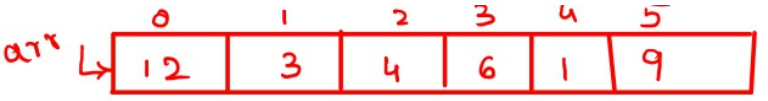
12 + 4 +

$O(n^2)$

$O(1)$ / $O(n)$

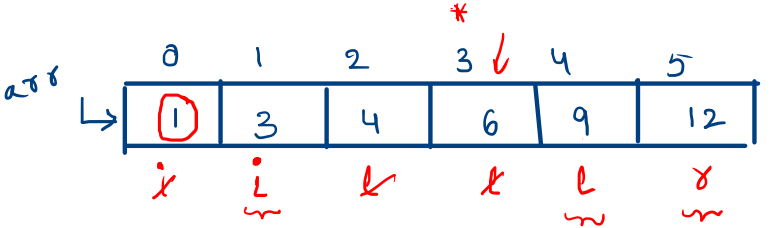
Sorting + 2-pointer

Sum = 24



↳ $n \log n + n^2$

M.S



→ Merge Sort

$O(n \log n)$ $O(n)$

$3 + (4 + 12) = 19$

$6 + 12 = 21$

$9 + 12 = 24$

$O(n)$ → while(l < r)

{

}

Sort +

```
boolean find3Numbers(int A[], int arr_size, int sum)
{
    int l, r;

    for(int i = 0; i < arr_size - 2; i++) {
        l = i + 1;
        r = arr_size - 1;
        while (l < r) {
            if (A[i] + A[l] + A[r] == sum) {
                print("Triplet is " + A[i] + ", " + A[l] + ", " + A[r]);
                return true;
            }
            else if (A[i] + A[l] + A[r] < sum)
                l++;
            else
                r--;
        }
    }

    return false;
}
```

↳ target sum

3) seperate zero's and ones ⇒ Seperate even & odd problem

arr[] = { 1,1,0,1,0,0,0,1}

o/p :- { 0,0,0,0,1,1,1,1}

API → sort() ⇒ O(nlogn) T.C
+
O(n) S.C

O(n) ✓

O(1) ✓

Count Approach

→ arr[] = { 1,1,0,1,0,0,0,1}

→ o/p :- { 0,0,0,0,1,1,1,1}

cout=0 1 2 3 4

n=8;

ZC = 4

OC = |n - zc| = 4

→ O(n) + O(zc)

loop1 → n times.

loop2 → zc times.

loop3 → oc times

} print

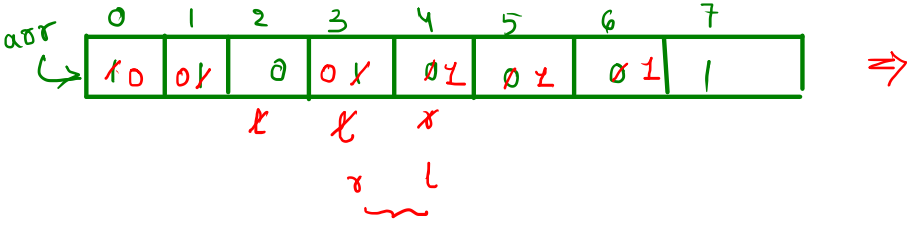
O(n) ✓

O(n)
O(1)

2-pointer Approach

→ arr[] = { 1,1,0,1,0,0,0,1}

→ o/p :- { 0,0,0,0,1,1,1,1}



arr[] = { 7, 1, 2, 3}

→ arr.sort() → O(nlogn)

↳ 1, 2, 3, 7 O(1)

T.C

O(1)

O(n)

→ console.log(arr)

↳ 1, 2, 3, 4

loop

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

n

```
void  segregate0and1(int  arr[], int  size)
{
    int  left = 0, right = size - 1;

     while  (left < right)
    {

        while  (arr[left] == 0  && left < right)
           left++;



        while  (arr[right] == 1  && left < right)
            right--;


        if  (left < right)
        {
            arr[left] = 0;
            arr[right] = 1;
            left++;
            right--;
        }
    }
}
```

EA | ORACLE

→ sort() : O(nlogn)

4) seperate 0s, 1s and 2s

Given an array **A[]** consisting 0s, 1s and 2s. The task is to write a function that sorts the given array. The functions should put all 0s first, then all 1s and all 2s in last.

Examples:

```
Input: {0, 1, 2, 0, 1, 2}
Output: {0, 0, 1, 1, 2, 2}

Input: {0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1}
Output: {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2}
```

O(n)
O(1)

make a count

i i i →

```
Input: {0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1}
Output: {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2}
```

zc = 0 1

oc = 0 1 2

tc = 0

} loop1 : O(n)

+

zc → loop2

+

oc → loop3

+

tc → loop4

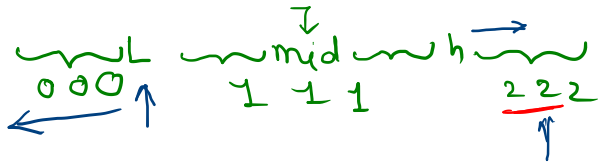
} O(n)

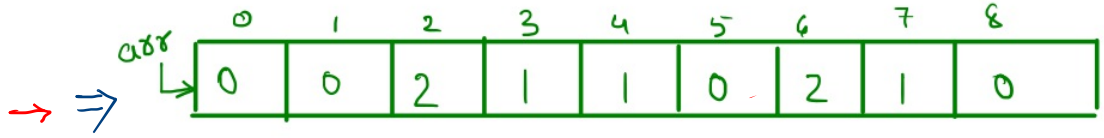
O(n)

O(1)

3 Colors

Dutch National Falg Algorithm :-





⇒ G G R B G R

R R G G B

0 0 1 1 2

0 → swap(arr[l], arr[mid])
l++, mid++

1 → mid++

2 → swap(arr[mid], arr[h])
h--
mid++ ?

```
static  void  sort012(int  a[], int  arr_size)
    {
        int  lo = 0,  hi = arr_size - 1,mid = 0;

         while  (mid <= hi) {

            switch  (a[mid]) {
                'R'
            case  0: {
                swap(arr[low],arr[mid])
                lo++;
                mid++;
                break;
            }
                'G'
            case  1:
                mid++;
                break;
                'B'
            case  2: {
                swap(arr[mid],arr[high]
                hi--;
                break;
            }
            }
        }
    }
```

3)All twice except one

```
Input:  ar[] = {7, 3, 5, 4, 5, 3, 4}
Output: 7
```

Check given integer is even or odd [without using / , %]