Day-1

1)Finding Majority Element in an Array

$\geq \frac{n}{2}+1$ element freq

$n=9$

3 2

4 5

2 2

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}

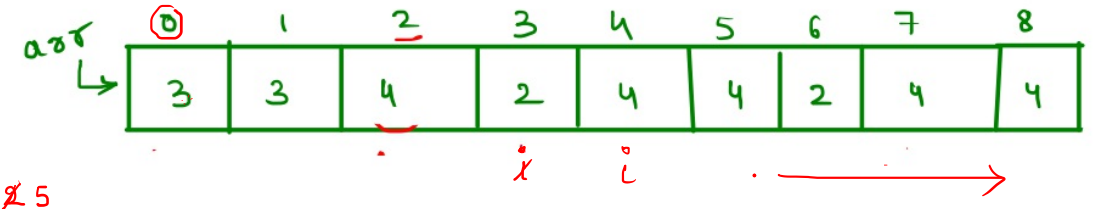
Output : No Majority Element

Explanation: There is no element whose frequency is greater than the half of the size of the array size.

Brute-Force

$n = 9 \rightarrow \frac{n}{2} = 4$

↳ 5



max_count = 5

index = 2

count = 0

(max_count) > $\frac{n}{2}$ → arr[index]

not > → maj not found

```
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");
}
```

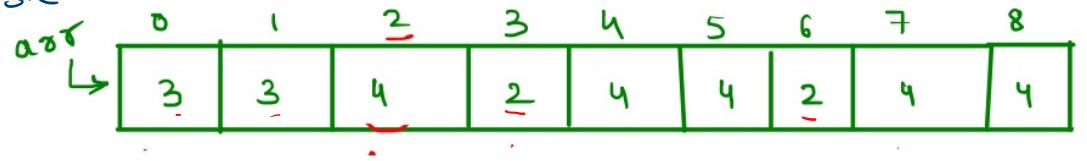
⇒ T.C : $O(n^2)$

S.C : $O(1)$

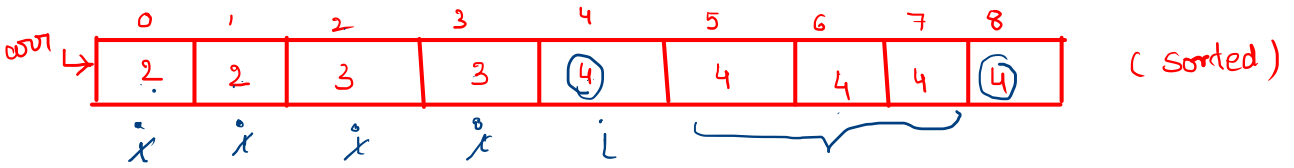
O(nlogn) T.C

O(n)/O(1) S.C

Sorting Approach



More than 1 maj ele → ?



$n = 9 \rightarrow \frac{n}{2} = 4$

let

Merge_sort ⇒ nlogn + n

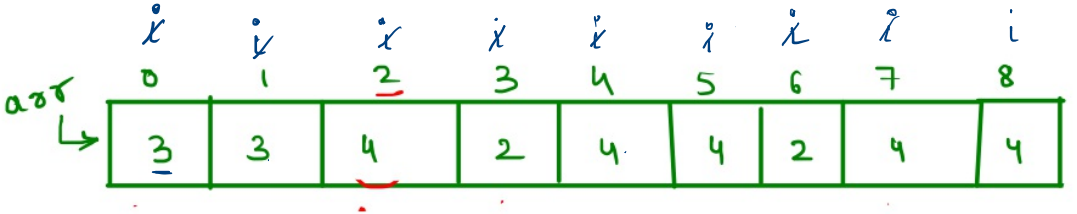
```
for(i=0; i<=n/2; i++)
{
    if( arr[i] == arr[i+n/2])
        return arr[i]
}
return -1
```

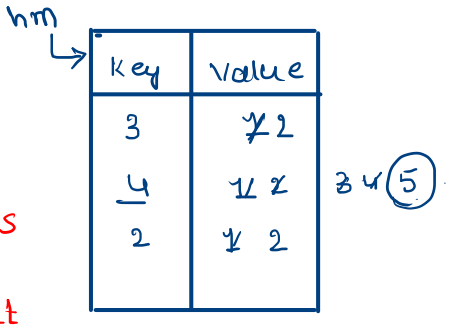
→ O(n) T.C

→ O(n) S.C

HashMap

↳ Key + value pair



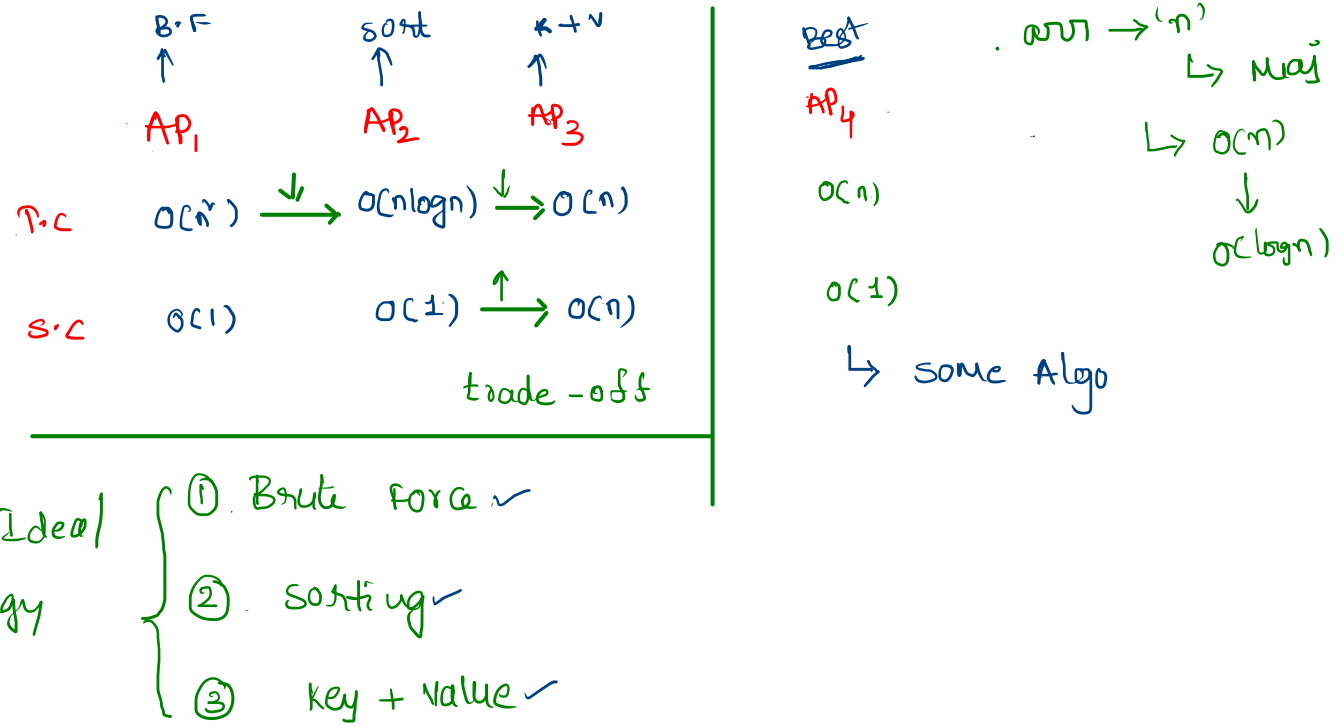


S.C → worst case

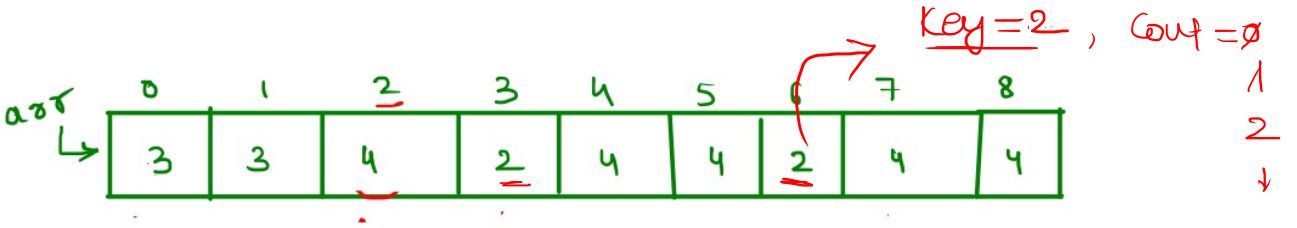
↳ All elements can be distinct

temp = 5 → $> \frac{n}{2}$

key = 4


B.F
sort
K+V
AP1
AP2
AP3
T.C
O(n²)
O(nlogn)
O(n)
S.C
O(1)
O(1)
O(n)
trade-off
Best
AP4
O(n)
O(1)
Array → 'n'
↳ Max
↳ O(n)
O(logn)
↳ some Algo
Technique/Idea/
strategy
① Brute force ✓
② sorting ✓
③ key + value ✓

Moore-Voting Algo



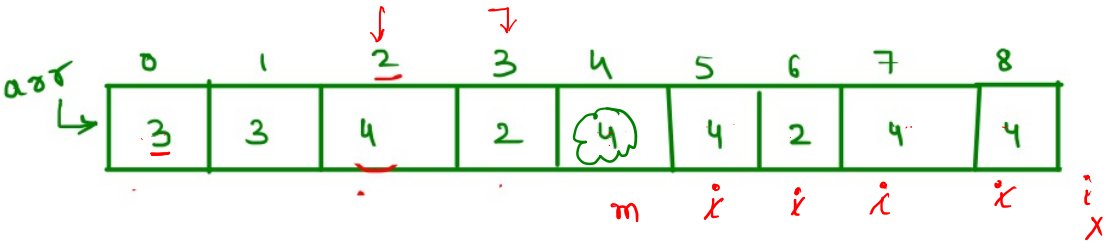
Contains 2 steps

* (1) : return some element as maj element.

✓ (2) : It is our respo to check, that element as maj or not.

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 = 8 4

count = 1 0 1 2 1 2 3

O(n) T.C

O(1) S.C

temp = 4, cnt = 0

for(i=0; i<n; i++)

{ if(arr[i] == temp)

count++

}

→ Count > n/2 → temp

→ -1

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

↳

2) Two elements whose sum is close to zero

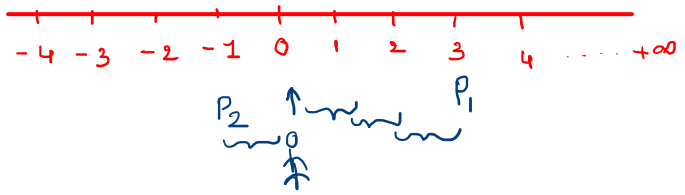
arr[]= { 1,2,6,9,-5,-2 }

$O(n^2)$ T.C

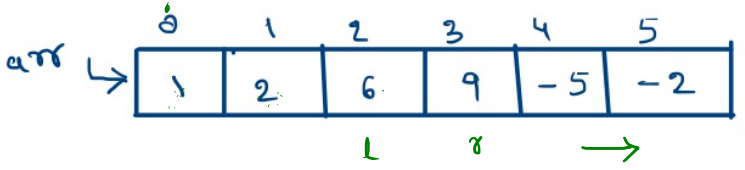
$O(1)$ S.C

Brute-Force

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



$(a+b) \simeq 0$



P_1 : (1, 2) = 3

P_2 : (-2, 1) = -1

Abs → sum

min_l = 0 ⊘ 1

min_r = 1 ⊘ 5

min_sum = 3 - 1 0

sum = arr[l] + arr[r] = 8 - 3 0

```
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] to arr[min_r]
`
```

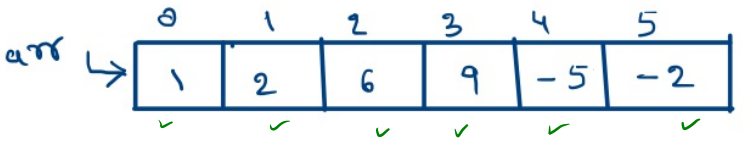
O(nlogn) T.C

O(1) S.C

sorting + 2 pointer

⇐ nlogn + n

→ arr.sort() → ~~O(1)~~

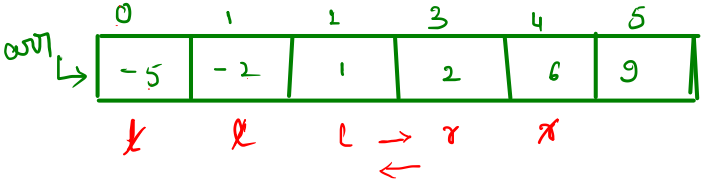


Sum = 0

min_sum = ~~999999999~~ +∞

min_l = 0 1

min_r = 5 4 3



sum = arr[l] + arr[r] = -5 + 9 = 4 > 0

if(sum > 0)

r--

else l++

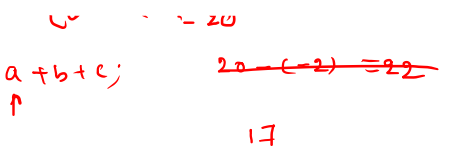
sorting + 2-pointer

```
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]
    }
```



(n-1) odd + 1 even → O(n) T.C

→ O(n) S.C

3)seperate odd and even

i/p arr[] = {12, 34, 45, ... 3}

→ even[] → 12, 34, 8, 90

→ odd[] → 45, 9, 3

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:

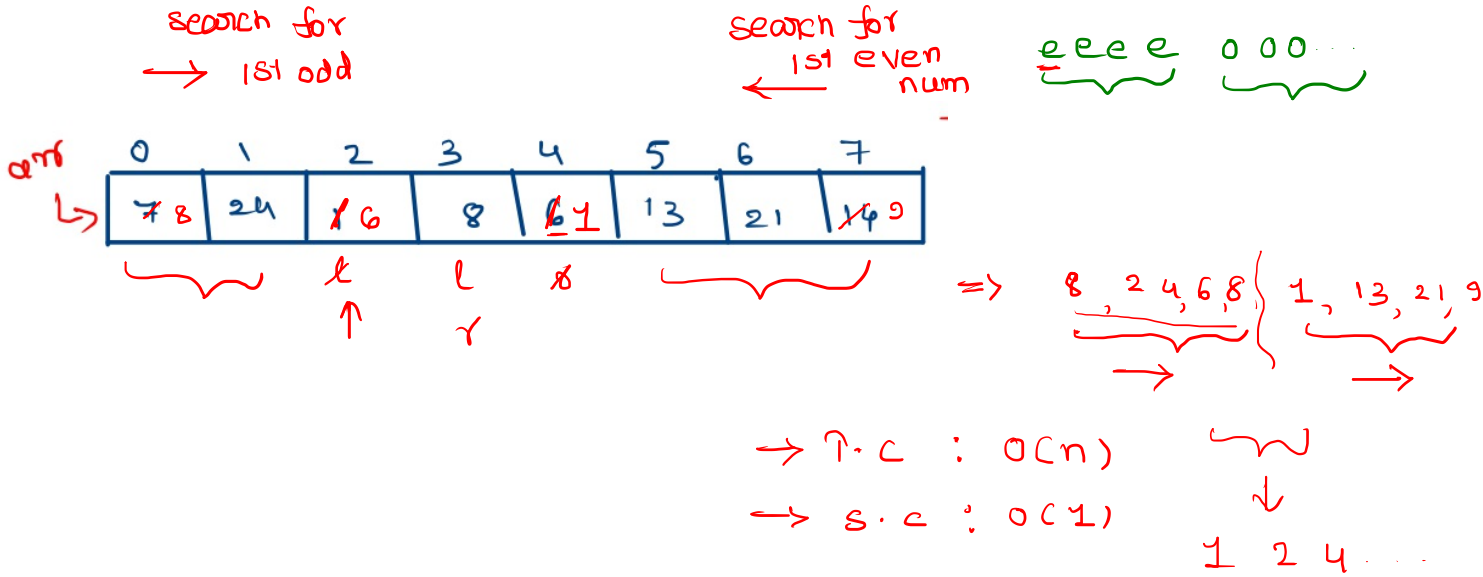
```
Input  = {12, 34, 45, 9, 8, 90, 3}
Output = {12, 34, 8, 90, 45, 9, 3}
```

→ loop : n times

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.

n % 2 == 0 → even → push / add to even array

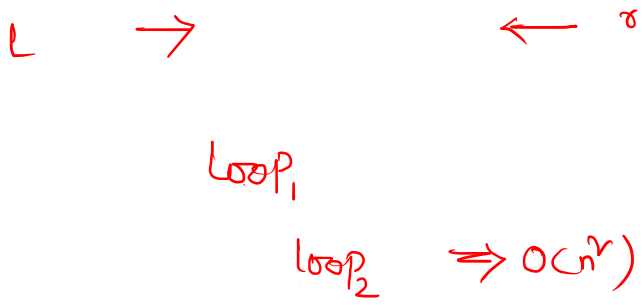
2-pointer


search for
→ 1st odd
search for
← 1st even
num
eeee
000
arr
0 1 2 3 4 5 6 7
78 24 16 8 61 13 21 19
l
r
⇒
8, 2, 4, 6, 8
1, 13, 21, 9
→ T.C : O(n)
→ S.C : O(1)
1 2 4

```
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

* 1) Triplet whose sum is x

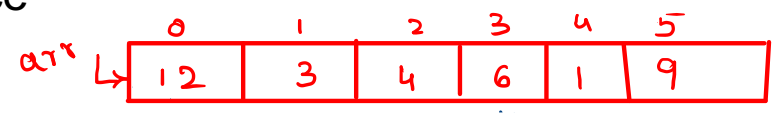
a + b + c = sum
↳ given

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.*

API - BF
→ T.C : O(n³)
→ S.C : O(1)
↳ 3 nested loops

```
n → for(i=0; i<n; i++)
    {
n →   for(j=i+1; j<n; j++)
      {
n →     for(k=j+1; k<n; k++)
        {
          if(arr[i]+arr[j]+
             arr[k]==sum)
          {
            return true
          }
        }
      }
}
```

Brute-Force

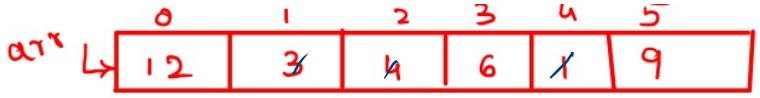


triplet → a+b+c

target = 24

Sorting + 2-pointer

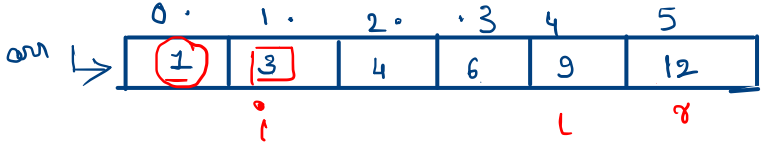
nlogn + ↳(n-2)*(n)



nlogn + n²-2n ⇒ T.C: O(n²)

S.C: O(1)

↓ Sorted



sum = arr[i] + arr[L] + arr[R]

Sum: 1 + 3 + 12 = 16

1 + 4 + 12 = 17

1 + 6 + 12 = 19

1 + 9 + 12 = 22

3 + 4 + 12 = 19

3 + 6 + 12 = 21

3 + 9 + 12 = 24

3, 9, 12

Sum = 16, target = 24

C_1 : sum == target

Print (triplet)

C_2 : sum < target

L++

C_3 : sum > target

R--

Eg:- 24

↑ given target

```
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;
}
```

C1

C2

C3

3) seperate zero's and ones

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

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

AP1:- zero[] ✓
one[] ✓
loop1 : n times
+
T.C : O(n)
S.C : O(n)

AP2:- count approach.

Step1 : → count # of zeros : zc

Step2 : → # of ones = n - zero's : oc

Step3 → zc times → print all zeros
oc times → n n ones

T.C : O(n)
S.C : O(1)

Count Approach

- arr[] = { 1,1,0,1,0,0,0,1}
- o/p :- { 0,0,0,0,1,1,1,1}

2-pointer Approach

→ arr[] = { 1,1,0,1,0,0,0,1}

→ o/p :- { 0,0,0,0,1,1,1,1}

2-ptr:

```
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--;
        }
    }
}
```

even: arr[i] % 2 == 0

* Seperate +ve and -ve numbers

1. even & odd
2. zero's & ones

T.C : O(n)
S.C : O(1)

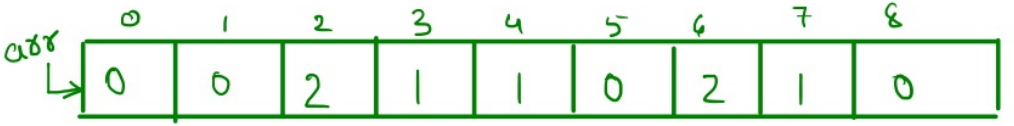
* * * separate 0's , 1's and 2's

AP1 :- zero[] ✓
oan[] ✓
tan[] ✓
} O(n) T.C
O(n) S.C

AP2 :- count app

✓ zc
✓ oc
✓ tc
} O(n) T.C
O(1) S.C
extra loops

- make a count




```
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}
```

Dutch National Falg Algorithm

EA

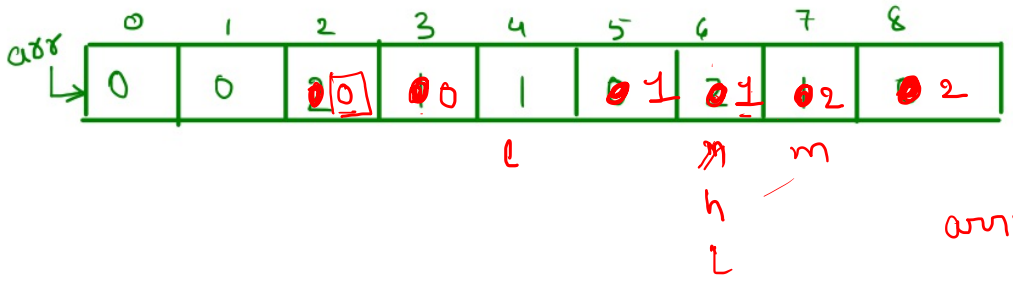
l < h ✗

m < h

0's 1's → 2 ptrs

0's 1's and 2's → 3 ptrs

< l, r, mid >



arr[mid]

0 0 0 0 1 1 1 2 2

0 → swap(arr[low], arr[mid])
l++, mid++

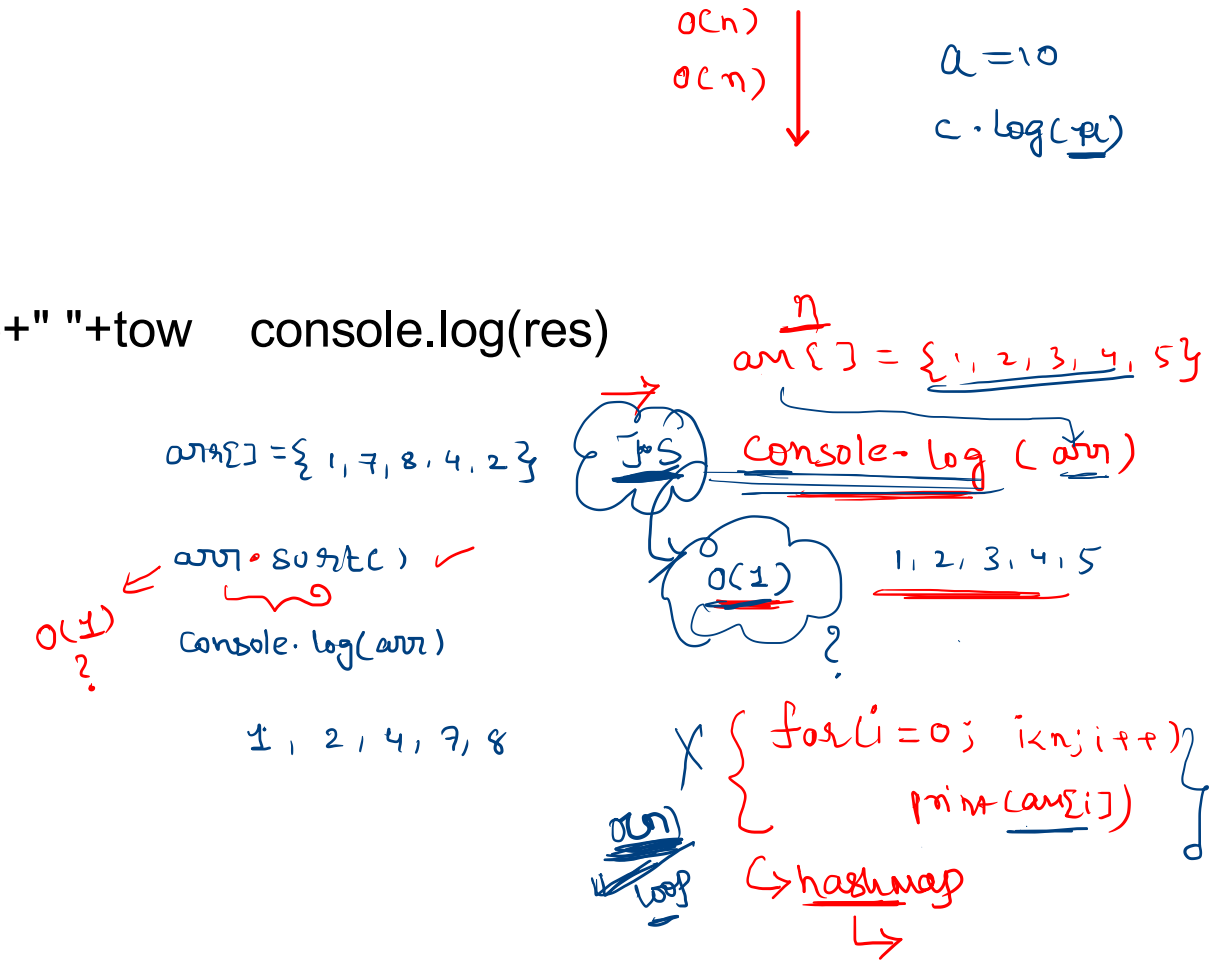
1 → mid++

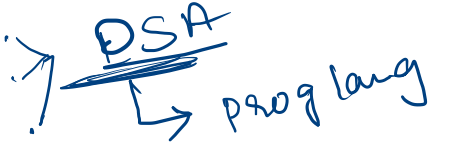
2 → swap(arr[mid], arr[high])
high--

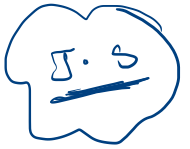
?

```
static  void  sort012(int  a[], int  arr_size)
    {
        int  lo = 0,  hi = arr_size - 1,mid = 0;
        while  (mid <= hi) {
            switch  (a[mid]) {
            case  0: {
                swap(arr[low],arr[mid])
                lo++;
                mid++;
                break;
            }
            case  1:
                mid++;
                break;
            case  2: {
                swap(arr[mid],arr[high]
                hi--;
                break;
            }
            }
        }
    }
```

```
if(arr[i]===0)
{zero.push(arr[i])}
else if(arr[i]===1)
{one.push(arr[i])}
else{two.push(arr[i])}
var res = zero +" "+one+" "+tow    console.log(res)
```




DSA
prog lang


3.5


arr
10
rem
10.5

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 / , %]