* Dutch Nat Flag Algorithm.
Purpose : Sort an array containing three distinct categories (0s, 1s, 2s) in one pass
When to use: When you need to partition an array into three sections. Ideal for problems that require segregating items, 3 distinct

* Moore's Voting Algorithm.
Purpose : Find the majority element (ele > n/2 times) in array.
When to use : When the array has guaranteed majority element
: Useful in prblm where to determine the most dominant ele

* Kadane's Algorithm.
Purpose : Find max sum subarray with 1D Array.
When to use : Max subarray problems, specially in DP.
: Suitable for both +ve, -ve;

Prefix sum.

→ When you query a range of an array to find out the product, sum, difference or any thing for a range of array use prefix sum.

eg. CS. 1310 XOR queries of subarray.

DATE

# ARRAYS

1. Basics →

Array is data structure which store similar elements.

Eg. Array → 

The above array can be declared as.

int arr[] = new int[6]

int arr[6] → global array. Here max len → $10^7$

main ()
{
int arr[6] → max length of array inside main → $10^6$
}

Q. largest element in array

arr[] = [3, 2, 1, 5, 2].

Way to solve.
Brute
↓
Better
↓
Optimal.

Brute

→ Sort the array last element is largest.

code.

psvm {
int[] arr = {3,2,1,5,2};
Arrays.sort(arr);
sout (arr[arr.length-1]);
}

TC → O(NlogN)

SC → O(1)

Better / Optimal

→ Now here consider the first element as largest element. And iterate over the array.

largest = arr[0] = 3.

arr = {3,2,1,5,2}.

i=0.

i=i+1

largest = 3

arr = {3,2,1,5,2} arr[1] = 2.

i=1

Compare if arr[i] > largest.

If yes then substitute it if no go ahead.
largest = 3 arr = [3,2,1,5,2] i = 2 arr[i] = 1
Now compare is arr[i] > 3 No so go ahead.

largest = 3 i = 3 arr[i] = 5
Now compare is arr[i] > 3 Yes 5 > 3 ∴ largest = 5 Now.

largest = 5 i = 4 arr[i] = 2.
Compare arr[i] > 5. No Go ahead.
No elements remaining stop.

Code.

```
main () {
    int [] arr = { 3,2,1,5,2};
    int large = arr[0] // 3 is first element i.e arr[0] = 3.

    for (int i=0; i < arr.length; i++) {
        if (arr[i] > large) {
            large = arr[i];
        }
    }
    Sout (large);
}
```

TC → O(N)
SC → O(1)

ATHARVA COLLEGE OF ENGINEERING

ATHARVA EDUCATIONAL TRUST'S
ATHARVA COLLEGE OF ENGINEERING, MUMBAI



Q. Second largest element in array without sorting.

Brute.
Sort the array
Since the last element is largest ; if no duplicates exist the second ~~largest~~ last element will be ~~sec~~ largest

But if duplicates exist .Its a problem.
For eg :- consider arr [5, 2, 9, 3, 1, 5, 9, 7]

→ Sort the array.
arr → 1, 2, 3, 5, 5, 7, 9, 9.
largest → 9.
for second largest run a for loop from second last element to 0 and put a condition that when arr[i] != largest
second largest = arr[i].

Code →

```
main (){
    int[] arr = {5,2,9,3,1,5,9,7}
    Arrays.sort(arr);
    int n = arr.length;
    int largest = arr[arr.length-1];
    int 2nd largest = 0.

    for(int i = n-2; i >= 0; i--){
        if(arr[i] != largest){
            2nd largest = arr[i]
        }
    }
}.
```

TC → O(N log N) + O(N) → ≈ O(N log N)
↑ sorting ↑ for loop.

SC → O(1).

Better.

① Here we will do first pass find largest element
keep largest as arr[0].

```
int largest = arr[0];
for (int i=0; i<n; i++){
    if (arr[i] > largest){
        largest = arr[i]
    }
}
```

② Here do second pass and find second largest.

```
int sec largest = -1
for (int i=0; i<n; i++){
    if (arr[i] > sec largest && arr[i] != largest)
    { sec largest = arr[i];
    }
}
```

TC → O(N+N)
SC → O(1)

Best-/Optimal-

Here only 1 pass will take place./The logic stays consider arr[0] as largest and slargest → -1 so traverse array if a num > largest. slargest = largest & largest → num. & if num is less than largest but greater than slargest, slargest → num.

```
int largest = arr[0]
int slargest = -1

for(int i=1; i<n; i++){
    if (arr[i] > largest){
        slargest = largest
        largest = arr[i];
    }
    if (arr[i] > slargest && arr[i] < largest){
        slargest = arr[i];
    }
}

print (s largest).
```

3. Check if the array is sorted or not

arr = [1, 2, 2, 3, 3, 6]

```
for(int i=1; i<n; i++){
    if(arr[i] < arr[i-1]){
        return false
    }
}
return true;
```

2. Remove duplicates in-place from sorted array

arr[] = [1,1,2,2,2,3,3]

Brute → Define HashSet

arr[] = [1, 1, 2, 2, 2, 3, 3]
i → i → i → i → i → i → i
Add to set, No, Add to ret, No, No, add to ret, No

| 3 | HashSet |
|---|---|
| 2 | |
| 1 | |

Code.

```
HashSet<Integer> hs = new HashSet<>();
for(int i=0; i<n; i++){
    hs.put(arr[i]);
}
for(int i=0; i<hs.size(); i++)
    arr[i] = hs.
int i = 0;
for(Integer num: hs)
{ arr[i] = num;
    i++;
}
return hs.size()
```

TC → N log N + N ← Next pair

Optional.

Here apply two pointer approach.

Here we will say. keep a pointer i=0, Now take another pointer j=1 and itrate. if arr[j] != arr[i] that means new element is encountered i++ → arr[i] = arr[j].

arr[] = [1, 1, 2, 2, 2, 3, 3]

i=0 ; j=1.

Now arr[j] != arr[i] → i++

count = 0

i=0

```
for(j=1; j<n; j++){
    if(arr[j] != arr[i]){
        i++;
        arr[i] = arr[j];
        count++;
    }
}
return count.
```

[1, 1, 2, 2, 2, 3, 3] arr[j] != arr[i] → i++, arr[i] = arr[j]

[1, 2, 2, 2, 2, 3, 3] arr[j] != arr[i] → i++, arr[i] = arr[j]

[1, 2, 3, 2, 2, 3, 3].

ATHARVA COLLEGE OF ENGINEERING

ATHARVA EDUCATIONAL TRUST'S
ATHARVA COLLEGE OF ENGINEERING, MUMBAI



Q. Left rotate the array by one place.
arr[] = [1,2,3,4,5] → [2,3,4,5,1]

Here store the first element in a variable and then shift all elements to left by one place.

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

TC → O(N)
SC → O(1)

Q. Left rotate the array by d places.
arr[] = [1,2,3,4,5] → [3,4,5,1,2] d = 2.
Now if d = 5 then it will bring array back to its original place.
Now d = 9 then d = 5 will bring it back to original now remaining 9-5 = 4 so basically we need to do 4 rotations.

So first step = d = d % N (N is len. of arr).
Now do similarly like above store the d elements.

① 
```
temp[] = new int[d];
for(int i=0; i<d; i++){
    temp[i] = arr[i];
}
```

② Now shift the elements.

```
for(int i = d; i < n; i++){
    arr[i-d] = arr[d];
}
```

② Now allocate the rotation.

```
for(i = n-d; i < n; i++){
    a[i] = temp[i-(n-d)];
}
```

TC = O(N)
SC = O(1)

Optimal Solution.

arr[] = [1, 2, 3, 4, 5, 6, 7] d=3, n=7.

First step reverse (a, a+d) → [3, 2, 1, 4, 5, 6, 7]. → O(d)
2nd reverse (a+d, a+n) → [3, 2, 1, 7, 6, 5, 4]. O(n-d)
3rd reverse (a, a+n) → [4, 5, 6, 7, 1, 2, 3]. O(n).

TC → O(2n)
SC → O(1)

Q Move all non-zero to front | zero to the end.

arr[] = [1, 0, 2, 3, 2, 0, 0, 4, 5, 1) → [1, 2, 3, 2, 4, 5, 1, 0, 0, 0).

Brute.
Step 1 → Store all non-zero elements in arr | List | Set.
Step 2 → Replace the original array.

① ArrayList<Int> al = new AL<>()

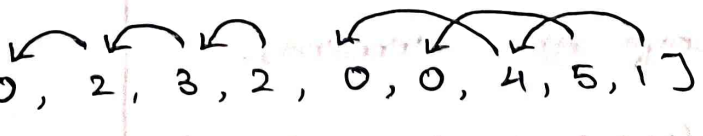
```
for (int i=0; i<n; i++){
    if (arr[i] != 0){
        al.add(arr[i]);
    }
}
```

② for (i=0; i < al.size(); i++){
    arr[i] = al.get(0);
}

③ for (i=al.size(); i<n; i++){
    arr[i] = 0
}

TC → O(N) + O(x) + O(N-x)
→ O(2N)
SC → O(x)

Optimal:
Two pointer approach.
→ keep pointer i at where the next non-zero element should be placed.
→ Iterate



```
int i = 0, count = 0;
for (int j=0; j<n; j++){
    if (arr[j] != 0) {
        arr[i] = arr[j]; i++;
        arr[j] = 0 count++
        }
    }.

for (int j= count; j<n; j++ {
    arr [j] = 0;
    }.
```

TC → O(N)
SC → O(1)

Linear Search.

Q. Search the given element is present in array or not.

arr[] = [6, 7, 8, 4, 1] num=4.

step :- apply for loop and see if the element is present.

```
for (int i= 0; i<n; i++){
    if (arr[i] == num){
        return true;
        }
    }
return false;
```

TC → O(N)
SC → O(1)

Q. Union of two sorted arrays

arr1[]: [1, 1, 2, 3, 4, 6] arr2[] [2, 3, 4, 4, 5, 6]

result = [1, 2, 3, 4, 5, 6].

1. Bruteforce.

Step 1 - Take HashSet.

Step 2 - pull all the elements of both arr1 & arr2 in set.

Step 3 - Sorth the elements inside set.

| | | Hash set | i = |
|---|---|---|---|
| arr1 | [1, 1, 2, 3, 4, 5] → | 1 | 0 |
| | → | 1 | 1 |
| | → | 1 2 | 2 |
| | → | 1 2 3 | 3 |
| | → | 1 2 3 4 | 4 |
| | → | 1 2 3 4 5 | 5. |

| | | HS | i = | |
|---|---|---|---|---|
| arr2 | [2, 3, 4, 4, 5, 6] | 1 2 3 4 5 | 0 | No change in set |
| | | 1 2 3 4 5 | 1 | " Already present |
| | | 1 2 3 4 5 | 2 | " |
| | | 1 2 3 4 5 | 3 | " |
| | | 1 2 3 4 5 | 4 | " |
| | | 1 2 3 4 5 6 | 5 | Add 6. |

Code.

```
TreeSet<Integer> ts = new Treeset<>();

for (int i=0; i<n1; i++){
    ts.add(arr1[i]);
}

for (int i=0; i<n2; i++){
    ts.add(arr2[i]);
}

int[] arr2 = new int[ts.size];
for (Integer num: ts){
    arr[i] = num; i++;
}
```

Treeset gives you ordered (sorted) arr

TC = O(n1 log n) + O(n2 log n) + O(n1+n2)

SC = O(n1+n2) + O(n1+n2)

↑ ans.

Optimal →
Two pointer approach keep pointer at i for arr1
" " j for arr2.

if (arr1[i] already present in ans i++
if (arr2[j] " " " j++

if (arr1[i] < arr2[j]) add arr1[i] to ans i++
if (arr2[j] < arr1[i] add arr2[j] to ans j++.

arr1[] = [1, 1, 2, 3, 4, 5]
arr2[] = [2, 3, 4, 4, 5, 6]

| | i \| arr[i] | j \| arr[j] | ans. |
|---|---|---|---|
| ans.add (arr[i]) | 0 \| 1 | 0 \| 2 | 1 |
| No change already in ans | 1 \| 1 | 0 \| 2 | 1 |
| Add 2, i++, j++. | 2 \| 2 | 0 \| 2 | 1, 2. |
| Add 3, i++, j++ | 3 \| 3 | 1 \| 3 | 1, 2, 3 |
| Add 4, i++, j++ | 4 \| 4 | 2 \| 4 | 1, 2, 3, 4 |
| No change 4 already in ans, i++ | 5 \| 5 | 3 \| 4. | 1, 2, 3, 4. |
| Add 5, i++, j++ | 5 \| 5 | 4 \| 5 | 1, 2, 3, 4, 5 |
| Add 6, j++ | x | 5 \| 6 | 1, 2, 3, 4, 5, 6. |
| | x | x | |

TC → O(n1) + O(n2)
SC → O(n1 + n2) → ans.

Q. Intersection of two sorted Arrays.
A[] = [1, 2, 2, 3, 3, 4, 5, 6] B[] = [2, 3, 3, 5, 6, 6, 7].

Brute force.
Take each element from A and see if it is present in B.
If present add to ans.

Take a visited array to handle duplicates efficiently.

Take i=0 j=0.
a[i] = 1 iterate over B[].

Code.

```
for(i=0 → n1)
    for(j=0 → n2)
    {
        if (a[i]==b[j] && vis[j]==0)
            ans.add(a[i]);
            vis[j] = 1;
            break;
        }
        if (b[j] > a[i]) break;
    }
```

TC → O(n1 × n2)
SC → O(n2)

Optimal.

Two pointer

Here take two pointer i for a[], j for b[] now if
if a[i] = b[j] add to ans.
if a[i] < b[j] i++
if b[j] < a[i] j++.

eg A = [1, 2, 2, 3, 3, 4, 5, 6]
B = [2, 3, 3, 5, 6, 6, 7].

Steps done.

| i | arr[i] |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 2 |
| 3 | 3 |
| 4 | 3 |
| 5 | 4 |
| 6 | 5 |
| 7 | 6 |

| j | b[j] |
|---|---|
| 0 | 2 |
| 0 | 2 |
| 1 | 3 |
| 1 | 3 |
| 2 | 3 |
| 3 | 5 |
| 3 | 5 |
| 4 | 6 |

b[j] > a[i] i++.
b[j] = a[i] add ans, i++, j++
b[j] > a[i] i++
b[j] = a[i] " " "
" " " " "
b[j] > a[i] i++
b[j] = a[i] " "
b[j] > a[i]
break.

Q. Find Missing no

You are given an array of size n-1 all nos appear once except one number find that number.

arr[] = [1, 2, 4, 5] N=5

→

Brute force.

consider every number and then search if present in array.

code

```
for (i = 1 ; i <= N ; i++){
    for( j=0 ; i,< n-1 ; j++)        → boolean p = false
        if ( i == arr[j]){
            p = true; break;
        }
    }
    if (!p) { return i };
```

TC → O(N×N).

SC → O(1)

Better.

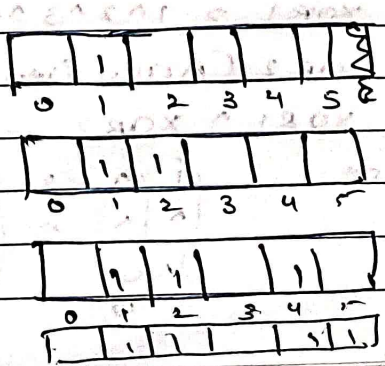
Here consider a Hash array.

for every element encountered in arr[] increment hash array

Iterate Hash array to find the remaining element.

| i | arr[i] |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 1 | 2 |
| 2 | 4 |
| 3 | 5 |

Hash array.



Code

```
int[] hash = new int[N+1];
for(Integer num: arr){
    hash[num] = 1;
    }
for(i=0 → hash.len){
    if(hash[i] == 0){
        return i;
        }
```

TC → O(N+N)
SC → O(N)

Optimal.

1) Sum of N natural nos.

We know sum of n natural no = $\frac{N \times (N+1)}{2}$.

Now calculate the sum of all array elements.

sum - sumArray = Ans.

Code.

```
int sumA = 0
int sum = N*(N+1)/2;
for(i=0; i<n; i++){
    sumA += arr[i];
    }.
return sum - sumA;
```

TC → O(N) SC → O(1)

2) XOR.

We know $x \wedge x = 0$ & $x \wedge 0 = x$.

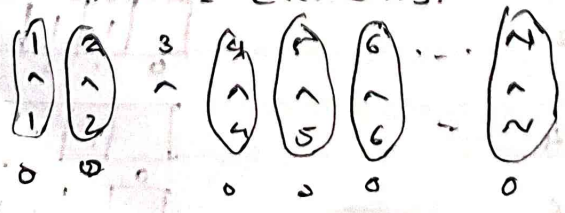
Using the same concept.

take XOR1 = $1 \wedge 2 \wedge 3 \wedge 4 \ldots \wedge N$.

take XOR2 (Array Elements) = $1 \wedge 2 \wedge 4 \wedge \ldots \wedge N$.

Now XOR1 ^ XOR2 = Elements.

since




= $0 \wedge 3 = 3$.

TC → O(N+N)
SC → O(1).

Q. Find Max consecutive 1's.
arr[] : [1, 1, 0, 0, 1, 1, 1, 0]

→ Here take two var count and max
reset count to zero every time 0 is encountered
increment count by 1 everytime 1 is encountered.

```
int max = 0
    count = 0
for(int no : arr){
    if(no == 0){
        count = 0;
    }
    else {
        count++;
        max = Math.max(max, count);
    }
return count;
```

Q. Find the number that appear once, and other twice.
arr[] = [1, 1, 2, 3, 3, 4, 4]

→ Bruteforce
Pick a number
Do linear search if duplicate exist.
find the num with one occurence.

→
```
for(i=0 → n){
    num = arr[i]
    count = 0
    for(j = 0 → n){
        if(arr[i] == arr[j] && i != j){
            count++;
        }
    if (count == 1) return num;
}.
```

TC → O(N×N)
SC → O(1)

Better;
Use Hashmap and store key, value pair.
Now using loop store elements along with their frequency.

Code.

```
Hashmap<Integer, Integer> hp = new Hashmap<>();

for(Integer no : nums){
    hp.put(no, hp.getOrDefault(no, 0)+1)
}
for(Map.Entry<Integer, Integer> ent : hp.entrySet()){
    if(ent.getValue() == 1){
        return ent.getKey();
    }
}
return -1;
```

TC → O(N)
SC → O(N)

Optimal:

Apply XOR operations.
When two occurence XOR happens it will result in 0.

x ^ x ^ y ^ y ^ z ^ z ^ a
0 ^ 0 ^ 0 ^ a.
0 ^ 0 ^ a
0 ^ a = a. ∴ 0 ^ a = a

Code.

```
int XOR = 0
for(Int no : nums){
    XOR ^= no;
}
return XOR;
```

TC → O(N)
SC → O(1)

ATHARVA EDUCATIONAL TRUST'S
ATHARVA COLLEGE OF ENGINEERING, MUMBAI



Q. Longest subarray with sum K

arr[] = [1, 2, 3, 1, 1, 1, 1, 4, 2, 3] K=3.

PREFIX SUM

Brute force

Generate all subarrays

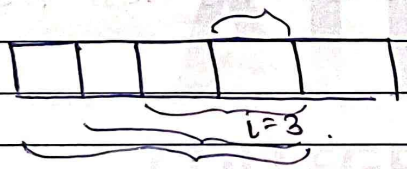
When encountered sum of of $K, take len

```
int max = 0
for (int i = 0; i < n; i++) {
    sum = 0
    for (int j = i ; j < n ; j++ {
        sum += arr[j];
        if (sum == k){
            max = Math.max (max, j-i+1 );
            }
```

TC → O(N×N)

SC → O(1)

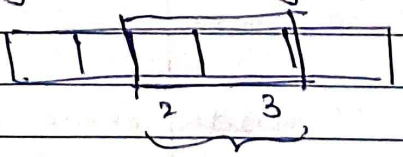
Better (Prefix Sum).



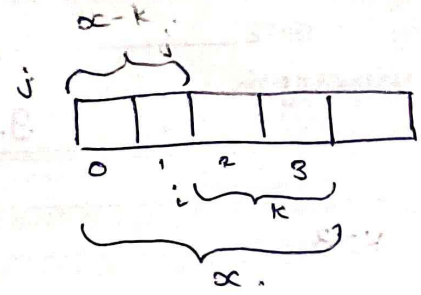

Here consider subarrays ending at i=3 now there exist 4 subarrays ending at i=3 i.e 0-3, 1-3, 2-3, 3-3.

Here the approach is reversed approach we find the total sum till i from 0 = x

→ Let's say hypothetically from i = 2 - 3 the subarray sum is k.




Do reverse engineering.




If the subarray i=2-3 is $\Sigma k$ and it $\sum_{i=0}^{3} = x$ then.
if there exist a subarray starting from j=0 to j having a sum k-x then i=2-3 has $\Sigma = k$.

Code Explanation.
Take a Hashmap.

arr [] = [1, 2, 3, 1, 1, 1, 1, 4, 2, 3] k=3 max = 3

| i | arr[i] | Sum(i) | Hashmap | Explanation. |
|---|---|---|---|---|
| 0 | 1 | 1 | (0,-1)(1,0) | Input sum(i), check if x-k exist. -2 doesnt exist. |
| 1 | 2 | 1+2=3 | (1,0), (3,1) | Check if x-k exist, 0 exist Update max = max(max, i-j) = 2 |
| 2 | 3 | 3+3=6 | (1,0) (3,1) (6,2) | Check if x-k exist, 3 does exist. Update max = max(max, i-j) doesnt change. |
| 3. | 1 | 6+1=7 | (0,-1) (1,0) (3,1) (6,2) (7,3) | Check 7-3 exist no. |
| 4. | 1 | 7+1=8 | (0,-1) (1,0) (3,1) (6,2) (7,3) (8,4) | Check 8-3 =5 exist no. |
| 5. | 1 | 8+1=9 | (0,-1) (1,0) (3,1) (6,2) (7,3) (8,4) (9,5) | Check if 9-3=6 exist, yes Max = max(max, i-j) = 3. |
| 6. | 1 | 9+1=10 | " " " (10,6) | Check 10-3<7 exist, yes update max & i = max 3. |
| 7. | 4 | 10+4=14 | " " (14,7) | Check 14-3<11 exist no. |
| 8. | 2 | 14+2=16 | " " (16,8) | Check 16-3=13 exist no. |
| 9 | 3. | 16+3=19 | " " (19,9) | check 19-3=16 exist yes if yes update max. |

Here keep in mind one case
This whole algorithm is valid for positive value array. but if it contains negatives as well then there is slight change

arr[] = {2, 0, 0, 3}. K = 3.

i = 0 sum = 2 len = 0 hp = (2, 0)
i = 1 sum = 2 len = 0 hp = (2, 1)
i = 2 sum = 2 len = 0 hp = (2, 2)
i = 3 sum = 5 len = hp = (2, 2) (5, 3) ; 5-3 available yes.

but index for 5 here is 2 so now subarray for sum = 3 is 3 - 2 = 1 lenght.
but the longest length for sum = 3 is 1 - 3 length 3.

code max.
2, 0, 0, 3
longest

So here do a slight change if the hashmap contains a sum, key-value pair do not update it since we want the longest subarray.

TC → O(N).
SC → O(N)

Optimal.

Two pointer approach.

arr[] = [1, 2, 3, 1, 1, 1, 1, 3, 3] K = 6.

Simple approach count sum till l to r if sum exceeds k move l++ and if sum ~~exceeds~~ is less than k move r++ when sum is found update max l.

[1, 2, 3, 1, 1, 1, 1, 3, 3]  k = 6.  maxL = 0 3 5

0 1 2 3 4 5 6 7 8

| sum | l | r. | Condition. |
|---|---|---|---|
| 0+1=1 | 0 | 0 | Check sum <, >, = k ✗ sum < k, r++. |
| 1+2=3 | 0 | 1 | sum < k, r++. |
| 3+3=6 | 0 | 2 | sum == k, update maxL = max(maxL, r-l+1) |
| 6+1=7 | 0 | 3 | sum > k, l++, sum = sum - arr[l]. |
| 7-1=6. | 1 | 3 | sum == k, maxL = max(maxL, r-l+1), No change |
| 6+1=7 | 1 | 4 | sum > k, l++, sum = sum - arr[l] |
| 7-2=5 | 2 | 4 | sum < k, r++ |
| 5+1=6 | 2 | 5 | sum == k, maxL = max(maxL, r-l+1). |
| 6+1=7 | 2 | 6 | sum > k, l++, sum = sum - arr[l]. |
| 7-3=4 | 3 | 6 | sum < k, r++ |
| 4+3=7 | 3 | 7 | sum > k, l++, sum = sum - arr[l]. |
| 7-1=6 | 4 | 7 | sum == k, maxL = max(maxL, r-l+1), No change. |
| 6+3=9 | 4 | 8 | sum > k, l++, sum = sum - arr[l]. |
| 9-1=8 | 5 | 8 | sum > k, l++. |
| 8-1=7 | 6 | 8 | sum > k, l++ |
| 7-1=6 | 7 | 8 | sum == k, Update maxL. |
| | | | Now stops as r++ will be beyond range. |

TC → O(N)

SC → O(1)

✗ Sliding Window approach only works for positive values & not negative

ATHARVA EDUCATIONAL TRUST'S
ATHARVA COLLEGE OF ENGINEERING, MUMBAI



# 2-Sum

arr[] = [2, 6, 5, 8, 11] target = 14.

Q.1. Does there exist two element a and b s.t a+b = target

Q.2. Given an array there exist two element s.t a+b = target find two elements

Brute force.

1) Pick one element (a)
2) Check with all other elements (b) if a+b = target.

```
for (i=0; i<n; i++){
    for(j=0; j<n; j++){
        if (i==j) continue;
        if (arr[i] + arr[j] == target)
        {
            return i,j;
        }
    }
}
```

TC → O(N²)
SC → O(1)

Better Solution
Hashing
if a = any element, then b should be target - a
∴ By using this concept we will try to find if there exist any corresponding value of a (target - a) i.e b. to satisfy the given condition.

| i | arr[i] | HM | Explanation |
|---|---|---|---|
| 0 | 2 | | Does 14-2 = 12 exist NO. |
| 1 | 6 | (2,0) | Does 14-6=8 exist, NO. |

| i | arr[i] | HM | Explanation |
|---|---|---|---|
| 2 | 5 | (2,0),(6,1) | Does 14-5=9 exist NO. |
| 3 | 8 | (2,0),(6,1),(5,2) | Does 14-8=6 exist yes. Update i,j in map |
| 4 | 11 | (2,0),(6,1),(5,2),(8,3) | Does 14-11=3 exist NO. |

TC → O(N)
SC → O(N).

Optimal.
2 pointer approach.

Sort the array.
Apply Greedy
Keep left ptr at 0, right ptr at last.

[2,5,6,8,11]
 ↑ l  ↑ r
arr[l] + arr[r] = 2+11 = 13
which is less than k = 14.
Move l ahead
(As we move l it will increase the sum).

[2,5,6,8,11]
 l  r
arr[l]+arr[r] = 11+5 = 16. > k.
Move r Behind (As we move r behind it will decrease sum)

[2,5,6,8,11]
 l  r
arr[l]+arr[r] = 8+5 = 13 < k.
Move l ahead (will increase sum)

[2,5,6,8,11]
   l r
arr[l] + arr[r] = 8+6 = 14 == k.
∴ we found ans.

ATHARVA EDUCATIONAL TRUST'S
ATHARVA COLLEGE OF ENGINEERING, MUMBAI

12

Q. Sort an array of 0's 1's & 2's.
arr[] = {0, 1, 2, 0, 1, 2, 1, 2, 0, 0, 0, 1}

Brute force
// Use any sorting algorithm, Use Merge Sort instead.

TC → N Log N
SC → N

Better Solution.
Loop entire array and increase the count for each element encountered.
Take three var, count 0, count 1, coun 2.

Code

```
count 0, c0, count 1 = 0, count 2 = 0

for (i=0; i<n; i++) {
    if (a[i] == 0) count0++;
    if (a[i] == 1) count1++;
    else count2++;
}

for (i=0; i<count0; i++)
    a[i] = 0;
for (i=count0; i<count0+count1; i++)
    a[i] = 1;
for (i=count0+count1; i<n; i++) {
    a[i] = 2;
}
```

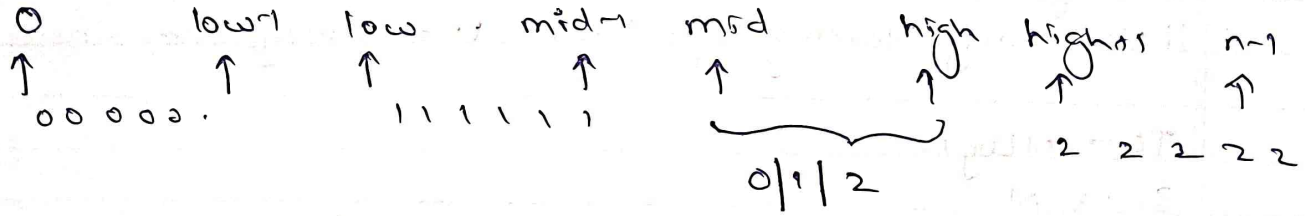
TC → O(N)
SC → O(1)

# Dutch National Flag Algorithm

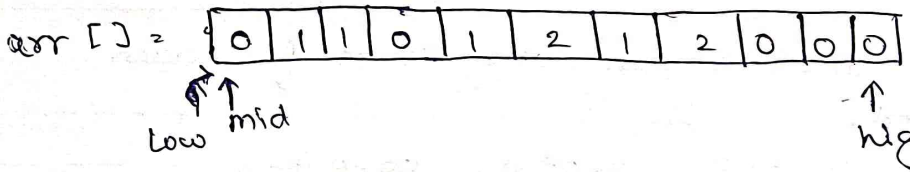
→ Based on 3 var low, mid, high.

→ Based on 3 rules [0 ... low-1] → 0 | [low ... mid-1] → 1 |
[high+1, n-1] → 2

So the structure of array looks like,



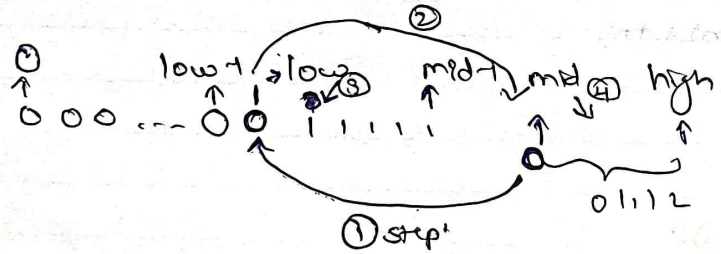

So while starting out here mid - high is your whole array.




Here low, mid points at 0, high point at end., The above pointers also follow the structure here 0... low-1 → 0 , low - mid-1 → 1 , mid → high → 0|1|2.

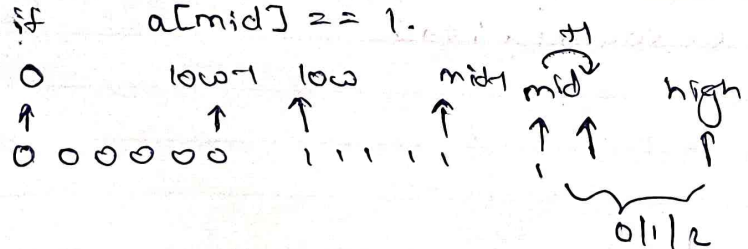
Now there are three cases for a[mid].

if a[mid] == 0.



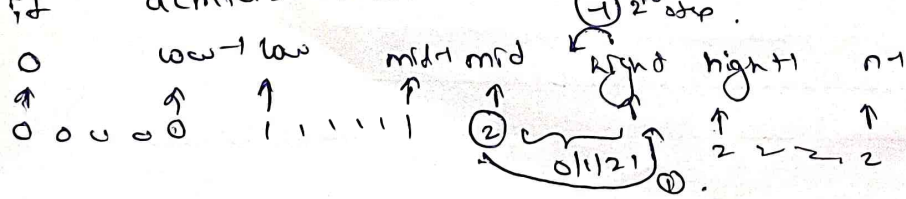

Step.
swap(a[low], a[mid])
low++, mid++;

if a[mid] == 1.




Step.
Just move mid++;

if a[mid] == 2.




Step.
swap[a[mid], a[high]].
high--;

ATHARVA EDUCATIONAL TRUST'S
ATHARVA COLLEGE OF ENGINEERING, MUMBAI



arr[] = {0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 2}.

| low | mid | high | Arr | Steps |
|---|---|---|---|---|
| 0 | 0→0 | 10 | | ~~low~~ swap (low, mid), low++, |
| 1 | 1→1 | 10 | 0 1 1 0 1 2 1 2 0 0 0 | mid++ |
| 1 | 2→1 | 10 | 0 1 1 0 1 2 1 2 0 0 0 | mid++ |
| 1 | 3→0 | 10 | 0 1 1 0 1 2 1 2 0 0 0 | swap(low, mid), low++, mid+ |
| 2 | 4→1 | 10 | 0 0 1 1 1 2 1 2 0 0 0 | mid++ |
| 2 | 5→1 | 10 | 0 0 1 1 1 2 1 2 0 0 0 | swap(mid, high), high-- |
| 2 | 5→0 | 9 | 0 0 1 1 1 0 1 2 0 0 2 | swap(low, mid, low++, mi |
| 3 | 6→1 | 9 | 0 0 0 1 1 1 1 2 0 0 2 | mid++ |
| 3 | 7→2 | 9 | 0 0 0 1 1 1 1 2 0 0 2 | swap(mid, high), high-- |
| 3 | 7→0 | 8 | 0 0 0 1 1 1 1 0 0 2 2 | swap(low, mid) low++, mi |
| 4 | 8→0 | 8 | 0 0 0 1 1 1 1 0 2 2 | swap(low, mid) low++, mi |
| 5 | 9→2 | 8 | 0 0 0 0 1 1 1 1 2 2 | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

TC → O(N)
SC → O(1)

Q. Find the Majority element.
arr[] = [2 2 3 3 1 2 2] majority element > N/2

Brute force
Take individual element and then count their occurance in the entire array.

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

TC → $O(N^2)$

SC → $O(1)$

Better → Hashing.

Declare hashmap

Traverse the array, increase the count for each occurrence.

Go through all HashMap values and if value $\geq n/2$ return key.

arr[] [2 2 3 3 1 2 2]

| i | arr[i] | Hashmap. |
|---|---|---|
| 0 | 2 | (2,1) |
| 1 | 2 | (2,2) |
| 2 | 3 | (2,2) (3,1) |
| 3 | 3 | (2,2) (3,2) |
| 4 | 1 | (2,2) (3,2) (1,1) |
| 5 | 2 | (2,3) (3,2) (1,1) |
| 6 | 2 | (2,4) (3,2) (1,1) |

```
HashMap<Int, Int> hp = new HP<>();
for (Integer num : nums)
    hp.put(num, hp.getOrDefault(num,0)+1);
}.
for (Map.Entry<Int, Int> entry : hp.entrySet())
    if (entry.getValue() > n/2)
        return entry.getKey();
}.
```

TC → O(N)

SC → O(1).

Optimal → Moore's Voting algorithm.

ATHARVA COLLEGE OF ENGINEERING

ATHARVA EDUCATIONAL TRUST'S
ATHARVA COLLEGE OF ENGINEERING, MUMBAI



arr[] = [7 7 5 7 5 1 5 7 5 5 7 7 5 5 5 5]

→ The core idea behind Moore's voting algorithm is based on the cancellation principle. If the array contains a majority element its occurrence must be more than half the size of the array.

Key insight:-
When you pair up different elements, they cancel out each other. The majority element will survive the cancellation process because it appears more than the others.

Given array [2, 2, 1, 1, 1, 2, 2]

| Index | Element | Count | Majority | Explanation |
|---|---|---|---|---|
| 0 | 2 | 1 | 2 | Count becomes 1, set ele = 2 |
| 1 | 2 | 2 | 2 | Same elem, count++ |
| 2 | 1 | 1 | 2 | Diff from ele, count-- |
| 3 | 1 | 0 | 2 | Diff from ele, count-- |
| 4 | 1 | 1 | 1 | Count = 0, assign new element |
| 5 | 2 | 0 | 1 | Different from ele, count-- |
| 6 | 2 | 1 | 2 | Count = 0, assign new element. |

8. ~~Maximum Subarray Sum~~

Key idea
- Maintain a candidate ele & count
- Traverse the array
  - Inc count if curr ele matches candidate
  - Decrease it otherwise
  - Reset the candidate when counter drops to 0

Constraints: Assumes majority element exists.

# KADANE'S ALGO.

Q] . arr[] = [-2, -3, 4, -1, -2, 1, 5, -3], ans = 7. , Maxi subarray sum.

→ Bruteforce.

Make all subarrays

Then take max sum.

```
max = 0
for (i = 0 ; i → n) {
    sum = 0
    for(j = i ; j <= n) {
        sum += arr[j];
    }
    if(max < Math.max(max, sum)
    }
return max;
```

TC → O(N²)

SC → O(1).

---

## Better / Optimal (Kadane's algorithm).

The core idea behind Kadane's Algorithm is to discard any subarray that has a negative sum because including it would reduce the sum of sub arr. If the cumulative sum (from start to potential subarray) become -ve, it will never help in forming a larger sum. In other words, if adding the current element drops the sum below 0, its best to start fresh from next element.

Approach:

→ Iterate through array.

→ Maintain two variables: sum (current sum) This variable accumulates the sum of elements in the current subarray.
maxi (Max sum). This var holds max subarray sum encountered so far.

→ Update rules during iteration: • Add the current element sum = sum + arr[]
• Update maxi if sum exceeds maxi
• Reset when -ve, if sum becomes -ve, reset it to 0, because -ve sum will only hinder the potential sum of sub array

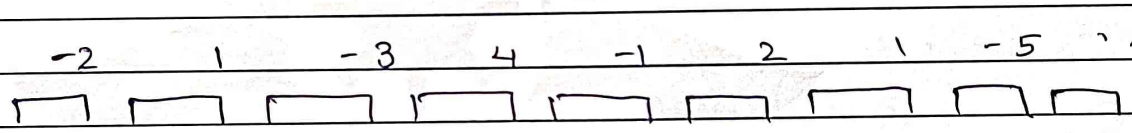
ATHARVA EDUCATIONAL TRUST'S
ATHARVA COLLEGE OF ENGINEERING, MUMBAI

15

Array = [-2, 1, -3, 4, -1, 2, 1, -5, 4].

0 ∞

| i | sum | maxi | Explaination |
|---|---|---|---|
| 0 | -2 | -2 | sum = 0+(-2) = -2 .·. maxi = max(-∞, -2) = -2, sum<0 sum=0 |
| 1 | 0+1=1 | 1 | sum = 0+1 = 1, maxi = max(-2, 1) = 1 |
| 2 | 1+-3=-2 | 1 | sum = 1+(-3) = -2, maxi = max(1, -2) = 1, sum<0 ∴ sum=0, |
| 3 | 0+4=4 | 4 | sum = 0+4 = 4, maxi = max(1, 4) = 4. |
| 4 | 4+(-1)=3 | 4 | sum = 4+(-1) = 3, maxi = max(4, 3) = 4. |
| 5 | 3+2=5 | 5 | sum = 3+(2) = 5, maxi = max(4, 5) = 5 |
| 6 | 5+1=6 | 6 | sum = 5+1 = 6, maxi = max(5, 6) = 6. |
| 7 | 6-5=1 | 6 | sum = 6+(-5) = 1, maxi = max(6, 1) = 6, |
| 8 | 1+4=5 | 6 | sum = 1+4 = 5, maxi = max(6, 5) = 5 |




| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Sum: | -2→0 | 1 | 1-3=-2→0 | 0+4=4 | 4-1=3 | 3+2=5 | 5+1=6 | 6-5=1 | 1+4=5 |
| maxi | -2 | 1 | 1 | 4 | 4 | 5 | 6 | 6 | 6. |

Code.

```
long maxi = Long.MIN_VALUE;
long sum = 0;

for(int i = 0; i < n; i++)
    sum += arr[i];

    if(sum > maxi){
        maxi = sum;
    }
    if(sum < 0){
        sum = 0
    }
}
return maxi;
```

TC → O(N)
SC → O(1)

Q. Rearrange array elements by sign. n/2 -ve, n/2 +ve.
arr[] = [3, 1, -2, -5, 2, -4] → [3, -2, 1, -5, 2, -4]
+ - + - + -

→ Brute →
Take an array named pos of size n/2. pos[] = new int[n/2]
Take an array named neg of size n/2. neg[] = new int[n/2]

Iterate over array put +ve elements in pos and -ve in neg.
eg arr = [3, 1, -2, -5, 2, -4]
pos = [3, 1, 2]
neg = [-2, -5, -4]

Now we know elements from pos will be at index 0, 2, 4 ... in ans & elements from neg will be at index 1, 3, 5 ... in ans

Code.
```
int[] pos = new int[n/2]
int[] neg = new int[n/2]
int k = 0, l = 1
for(i=0 → n){
    if(arr[i] > 0){
        pos[k] = arr[i];
        k++;
    }
    if(arr[i] < 0){
        neg[l] = arr[i];
        l++;
    }
}

for(i = 0; i < n/2; i++){
    arr[2*i] = pos[i]
    arr[2*i+1] = neg[i]
}
```
Here TC → O(N/2 + N)
SC → O(N)

Optimal → Not huge improvement

Take two pointers posIndex & negIndex = 1. | Take ans array.
then if you encounter +ve element in arr, ans[posIndex] = arr[i]
else ans[negIndex] = arr[i].

```
int n = nums.length.
int[] ans = new int[n];
int posI = 0 , negI = 1 ;
for (int i=0 → n) {
    if (nums[i] < 0) {
      ans[negI] = nums[i];
       negI += 2 ;
    }
    else {
      ans[posI] = nums[i];
       posI += 2 ;
    }
  }
  return ans;
```

TC → O(N)
SC → O(N)

Q. Next Permutation

arr[] = [3 1 2] Now next pn will be 1 2 3

arr[] = 1 1 5 Now next pn will be 1 5 1

→ The pn of 3 1 2.

1 2 3 → 1 3 2 → 2 1 3 → 2 3 1 → 3 1 2 → 3 2 1

Brute force

→ Generate all pn in sorted order.

→ Linear search the given array.

→ Next index return.

Optimal.

Consider this case

for 2 nos 1 2 → 2 1 These are the only 2 options.

Now for 3 nos

1 2 3 There exists a pattern that if least two numbers

1 3 2 are in asc then swap them if not.

2 1 3 check for.

2 3 1

3 1 2

3 2 1



Consider this 2 1 5 4 3 2 0

By thinking logically it will make 2 3 0 1 2 4 5 as next pn.

So the structure of number is.

1 | 5 4 3 2 0

↑ Break point

5 4 3 2 0 → dec order.

Since it is dec order Everything after break point is in dec order

which means it is as high as possible. To get next permutation, we need to increase break point

12. Leader in an array

arr[] = [10, 22, 12, 3, 0, 6] Here an element is considered as leader if all the elements to its right are smaller than it.

Bruteforce.

→ Take an element check all the element to its right
→ If no element is greater than it then add to Ans.

```
ArrayList<Integer> ans = new ArrayList<>();

for(int i=0; i<n; i++){
    boolean found = true;
    for(int j=i+1; j<n; j++){
        if(arr[j] > arr[i]){
            found = false
        }
    }
    if(found){
    ans.add(arr[i])
    }.
```

TC → O(N*N)
SC → O(1).

Optimal:

→ Iterate from right to left.
→ Track the greatest element encountered so far.
→ If curr element > max element then it is greater than all the elements to right:

```
int max = -1.
for(int i = n-1; i >= 0; i++){
    if(arr[i] > max){
        ans.add(arr[i]);
        max = arr[i];
    }
}
```

ATHARVA EDUCATIONAL TRUST'S
ATHARVA COLLEGE OF ENGINEERING, MUMBAI



Q. Longest Consecutive sequence.

arr[] = [102, 4, 100, 1, 101, 3, 2, 1, 1]

longest con seq → 1, 2, 3, 4. len = 4.

Bruteforce.

→ Take an element.

Search for ele+1 in the arr if exist then +1.

```
longest = 1
for(i=0 → n){
    x = arr[i]
    count = 1.
    while(ls(arr, x+1))
    {
        x = x+1;
        cnt = cnt+1;
    }
}
ls(arr, num)
{
for(i=0; i<n; i++){
    if(arr[i] = num)
        return true;
    }
    return false;
}
```

TC → O(N) x O(N)

SC → O(1).

Better:-

Here, Sort the array and then see.

[1, 1, 1, 2, 2, 2, 2, 3, 3, 4, 100, 100, 101, 102, 102],

Here we can see atleast that the consecutive seq lies, as a whole part, with duplicates in it

Our main focus will be to eliminate duplicates.

Three var.

lastSmaller : This var keeps track of last no added to consecutive seq

cnt : This var counts len of curr consecutive seq.

longest : This var stores the max len.

arr [1, 1, 1, 2, 2, 3, 3, 4, 100, 100, 101, 102, 102].

| i | arr[i] | lastS, -∞ | cnt = 0 | longest = 1 | |
|---|---|---|---|---|---|
| 0 | 1 | -∞ → 1 | 1 | max(1,0)=1 | (arr[i] - 1) != lastS, |
| 1 | 1 | 1 | 1 | max(1,1)=1 | (arr[i] - 1) != lastS. |
| 2 | 1 | 1 | 1 | max(1,1)=1 | (arr[i] - 1) != lastS. |
| 3 | 2 | 1 → 2 | 2 | max(1,2)=2 | (arr[i] - 1) == lastS, count ++, longest max |
| 4 | 2 | 2 | 2 | max(2,2)=2 | arr[i] - 1 != lastS. |
| 5 | 3 | 2 → 3 | 3 | max(2,3)=3 | (arr[i] - 1) == lastS, count ++, longest.max. |
| 6 | 3 | 3 | 3 | max(3,3)=3 | arr[i] - 1 != lastS |
| 7 | 4 | 3 → 4 | 4 | max(3,4)=4 | arr[i] - 1 == lastS, count ++, longest.max |
| 8. | 100 | 4 → 100 | 0 → 1 | max(0,4)=4 | arr[i] != lastS && arr[i] - 1 != lastS. count = 1, lastS = arr[i]. |
| 9. | 100 | 100 | 1 | max(1,4)=4 | |
| 10 | 101 | 100 → 101 | 2 | max(2,4)=4 | |
| 11 | 102 | 101 → 102 | 3 | max(3,4)=4 | |

Optimal Approach (See striver notes).

## Set Matrix 0.

Q. Set Entire row and col' to 0 if 0 is encountered.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Brute force Soln:
Take an array of size m×n as ans.
Copy entire array (given) to ans.
Traverse given array whenever 0 is encountered set row, col to 0

```
setZeroes B(int[][] matrix){
  int m = matrix.len
  int n = matrix[0].len
  int[][] ans = new int[m][n];
  copy (matrix, ans);

  for(int i=0; i<m; i++){
    for(int j=0; j<n; j++){
      if(ans[i][j] == 0){
        setRow(matrix, i)
        setCol(matrix, j)
      }
    }
  }
}
```

```
setRow(int[][] m, int row){
  for(int i=0; i<m[0].len; i++){
    m[row][i] = 0;
  }

setCol(int[][] m, int col){
  for(int i=0; i<m.len; i++){
    m[i][col] = 0;
  }

copy(int[][] m, int[][] n){
  for(i=0 → n)
    for(j=0 → n){
      m[i][j] = n[i][j];
    }
```

TC → O(m×n)
SC → O(m×n).

Better →
Here we utilize the Hashmaps.
→ Take two hashmap, row | col and
→ 0 encountered, store row, col in respective hashmap.
→ Iterate hashmap values and set row, col to 0.

```
HashSet<> row = new HashSet<>();
HashSet<> col = new HashSet<>()

int m = matrix.length;
int n = matrix[0].length;

for(int i=0; i<m; i++){
    for(int j=0; j<n; j++){
        if(matrix[i][j]==0){
            row.add(i);
            col.add(j);
        }
    }
}

for(Integer val : row){
    for(int i=0; i<n; i++){
        matrix[val][i]=0;
    }
}

for(Integer val : col){
    for(int j=0; j<m; j++){
        matrix[i][val]=0;
    }
}
```

TC → O(m×n)
SC → O(m+n)

Optimal Solution:

Initially the brute-force used an extra matrix, HashSet to mark the Rows & Cols that need to be 0'zed out. But:

- The matrix itself can act as flagging system.
- By marking the first row and first column as flags, we eliminate the need for external space

Steps:

- When iterating through the matrix, every time we encounter 0.
- Instead of storing it to row/col we mark the first row & col that should be zeroed out.
- The first row and columns are used as flags, but they themselves may need to be zeroed out (case where the first row & col has 0)
- Here we manage two boolean var, row/col here if the first row/col has 0 we mark them true.

for Code see

ATHARVA EDUCATIONAL TRUST'S
ATHARVA COLLEGE OF ENGINEERING, MUMBAI

20

## Pascal Triangle

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
1   5   10  10  5   1
```

1) Given R and C, Tell the element.
2) Print any Nth Row of Pascals Triangle
3) Given N, print the entire triangle.

→ Here remember $^{R-1}C_{c-1}$ represents the element at that particular or $^{R-1}C_{c-1}$ when R represents the row, c represents col, So element In 5th row and 3rd col will be.

$$^{5-1}C_{3-1} = {}^{4}C_{2} = {}^{n}C_{r} = \frac{n!}{(n-r)!\,r!} = \frac{4!}{2!\,2!}$$

So Brutforce,
find (R-1)! then divide it by (C-1)! (R-C)!

```
int r = R, c = 3.
int num = 1, den = 1
for(int i = r-1; i >= 1; i--){
    num *= i;
}
for(int i = c-1; i >= 1; i--){
    den *= i;
}
for(int i = r-c; i >= 1; i--){
    den *= i;
}
return num/den;
```

TC → O((R-1) + (c-1) + (R-c))
SC → O(1)

Optimal.
Now understand, $^{7}C_{2} = \frac{7!}{2!\,5!} = \frac{7\times6\times(5\times4\times3\times2\times1)}{(2\times1)\times(5\times4\times3\times2\times1)}$

Here they are getting cancelled so no need to calculate (n-r)!

fun ncr

```
int r = 5 , c = 3.
int num = 1
int den = 1

for(int i = r-1 ; i > r-c ; i--){
    num *= i;
}.

for(int i = c-1 ; i > 1 ; i--){
    den *= i;
}.

return num / den
```

TC → O((c-1) + (c-1))
→ O(2c-2)
SC → O(1)

2> Print entire Row.

Bruteforce.

Now consider the formula $^{R-1}C_{c-1}$ and then find each element in the row

```
for(int = 0; i<c; i++){
    print( NCR(R,i) );
}.
```

TC → O(N × c).

Optimal.

1 5 10 10 5 1 Consider this eg.

1 2 3 4 5 6

$^{6-1}C_{1-1} = {}^{5}C_0 = \frac{5\times4\times3\times2\times1}{0!\,(5\times4\times3\times2\times1)} = \frac{1}{1}$

$^{6-1}C_{2-1} = {}^{5}C_1 = \frac{5(4\times3\times2\times1)}{(1\times(4\times3\times2\times1)} = \frac{5}{1}$.

$^{6-1}C_{3-1} = {}^{5}C_2 = \frac{5\times4}{1\times2}$

$^{6-1}C_{4-1} = {}^{5}C_3 = \frac{5\times4\times3}{1\times2\times3}$.

$^{6-1}C_{5-1} = {}^{5}C_4 = \frac{5\times4\times3\times2}{1\times2\times3\times4}$.

$^{6-1}C_{6-1} = {}^{5}C_5 = \frac{5\times4\times3\times2\times1}{1\times2\times3\times4\times5}$.

Here you need to understand that.

```
int r = 6   ans = 1
for(int c=1 ; c <= r ; c++){
    if(c == 1) print ans ;
    else {
        ans = ans * (r-c+1);
        ans = ans / (c-1);
```