Two-Sum

arr = {2, 6, 5, 8, 11} target = 14

Are there any 2 ele from the array which sums to 14.

form this 2 type of problems can be made

Q → are there any 2 ele which sums to target

Q → Tell me the location of those 2 items.

① **Brute** pick 1st ele and check it with other rem. elements
then pick 2nd ele " " " elements.

```
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 = $O(n^2)$

② **Better** (Hashing is used)

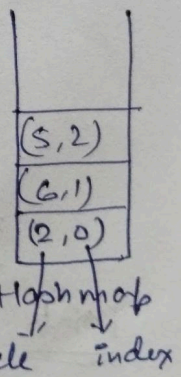
[2, 6, 5, 8, 11]

eg → x + y = target

8 + y = 14

we can find whether y exists in array or not using Hash table.

for 2 + 12 = 14
but in left of 2 there is no 12
for 6 + 8 = 14
for 5 + 9 = 14
for 8 + 6 = 14
in left of 8 there is 6
so we can easily return whatever is asked



Two-Sum

arr = {2, 6, 5, 8, 11} target = 14

Are there any 2 ele from the array which sums to 14.

from this 2 type of problems can be made

Q → are there any 2 ele which sums to target

Q → Tell me the location of those 2 items.

① **Brute** pick 1st ele and check it with other rem. elements
then pick 2nd ele " " " elements.

```
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 = $O(n^2)$

② **Better** (Hashing is used)

eg → x + y = target

8 + y = 14

we can find whether y exists in array or not using Hash table.

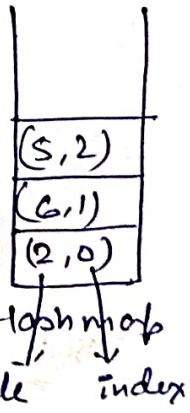
[2, 6, 5, 8, 11]

for 2 + 12 = 14
but in left of 2 there is no 12

for 6 + 8 = 14

for 5 + 9 = 14

for 8 + 6 = 14
in left of 8 there is 6
so we can easily return whatever is asked



Code

map (Hashmap / unordered map)

TC → O(N * logN) | O(N) → best, O(N²) → worst

SC → O(N) because we are using map.

```
{ map<int,int> mpp;
  for (int i=0; i<n; ++i)
  { int a = book[i];
    int more = target - a;
    if (mpp.find(more) != mpp.end())
        { return "yes"; } → or return {mpp[more], i};
    mpp[a] = i;
  }
  return "NO";  → or return {-1, -1}
}
```

(3) Optimal (does not use map instead use 2 pointer).

Step 1 → Sort the array and take 2 pointer left & right.

Step 2 → Sum both values
if sum < target ⇒ increment left
if sum > target ⇒ decrement right

Code

TC = O(N) + O(N logN)
Sorting

SC = O(1)

```
{ int left=0, right=n-1;
  sort(book.begin(), book.end());
  while (left < right) {
    int sum = book[left] + book[right];
    if(sum == target) { return "yes"; }
    else if (sum < target) left++;
    else right--;
  }
  return "NO";
}
```

points ★ for solving Q-2 the optimal is (2)nd Better we can't solve Q-2 with the 3rd approach bcoz SC will be TC if we solve that it is equal to 2nd approach.

3-Sum

Ques → Find out the triplets which sum to target.
triplets should be unique

① Brute arr = [-1, 0, 1, 2, -1, -4]

1st 2nd 3rd

(or)

1st 2nd 3rd

```
set<vector<int>> st;
for (i = 0 to n)
  {for (j = i+1 to n)
    { for (k = j+1 to n
      { if (arr[i]+arr[j]+arr[k]==0)
        { vector<int> temp = {num[i], num[j], num[k]}
          sort(temp.begin(), temp.end());
          st.insert(temp);
        }
      }
    }
  }
vector<vector<int>> ans (st.begin(), st.end());
return ans;
```

Triplets should be unique so for checking uniqueness.

TC = $O(n^3 * \log(\text{no. of unique}))$ $\cong O(n^3)$

SC = $2 * O(\text{no. of triplets})$

extra space → set.

② Better

Here we will be using set and hashing.

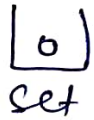
eg → arr = [-1, 0, 1, 2, -1, -4]
target = 0, we have to find triplets (unique) which sums to 0.

(a) [-1, 0, 1, 2, -1, -4]
i ↑ j=1

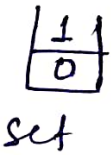
arr[k] = target − (arr[i] + arr[j])

(i) -1+0 = -1 → we need +1 but right now set is empty
j=1
so before incrementing j, add arr[j] to the set

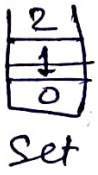

Set


Set

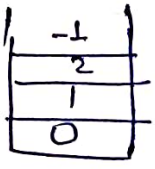
(ii) -1+1 = 0, which is present in the set.
j=2 so 1 triplet is {-1, 1, 0}
before moving j forward add arr[j] into set


Set

(iii) j=3 -1+2 = 1, we need -1, which is not in set
add arr[j] into set


Set

(iv) j=4 -1+(-1) = -2, so we need +2 which is present in set.
so other triplet is {-1, -1, 2}
now add arr[j] into set


set

(v) -1-4 = -5, we need +5 which is not present in set
so add arr[j] into set.

∴ iteration of j is over now empty the set and make new set for new iteration of i = 2.

There is one more set in which we are adding our triplets but before adding we sort those triplets and then add into a outer set.

Code

```
vector<vector<int>> tripletat (int n, vector<int> &num)
{
  set<vector<int>> st;
  for (i=0 to n)
   { set<int> hashset;
     for (j=i+1 to n)
      {  int third = -(num[i]+num[j]);
         if (hashset.find(third) != hashset.end())
           {  vector<int> temp = {num[i], num[j], third};
              sort(temp.begin(), temp.end());
              st.insert(temp);
           }
         hashset.insert(num[j])
      }
   }
  vector<vector<int>> ans(st.begin(), st.end());
  return ans;
}
```

TC ⇒ $N^2 \times \log(\text{size of inner set})$

SC ⇒ $O(N) + O(\underbrace{\text{No. of unique triplets}}_{\text{outer set}})$

(3) Optimal — 2 pointer approach.

Step 1 ⇒ Firstly sort the given array

arr[] = [-2 -2 -2 -1 -1 -1 0 0 0 2 2 2 2]

i j K

i → constant, move j or K

(i) arr[i] arr[j] arr[k]
-2 -2 +2 = -2

∵ -2 < target so we will move the left pointer i

i j
0 1

so j = 2

(ii) -2 -2 +2 = -2
same as above
so ↑es j

(iii) j = 3
-2 -1 +2 = -1
-1 < target so j++

(iv) j = 4
-2 -1 +2 = -1
same as above

(v) j = 5
same as above

(vi) j = 6
-2 +0 +2 = 0 → target.
so we get our 1st triplet in already sorted manner
[-2, 0, 2].

```
if sum < target
    j++;
if sum > target
    k--;
```

So now increase i but the i=1 is -2 which is same as previous so we will move i upto a new different value from previous so now i = 3.

Code

```
vector<vector<int>> triplet (int n , vector<int> &num)
{
  vector<vector<int>>ans ;
  Sort(num.begin() , num.end());
  for (i=0 to n)
  { if (i>0 && num[i]==num[i-1]) continue;
    int j = i+1;
    int k = n-1;
    while ( j<k)
    {   int sum = num[i] +num[j] + num[k];
        if(sum <0)  j++;
        else if (sum > 0)  k--;
        else
        { vector<int> temp = { num[i], num[j] , num[k] }
          ans.push_back (temp);
          j++; k--;
          while (j<k && num[j]==num[j-1])  j++;
          while (j<k && num[k]==num[k+1])  k--;
        }
    }
  }
```

4-Sum

Problem →

arr[] = [1, 0, -1, 0, -2, 2]

target = 0

[i != j != k != l]

find the indexs of those 4 no.s whose sum is equal to target.

① Brute force

{ i j k l } 4 loops

TC = O(n^4)

SC ⇒ O(no. of quads) * 2

② Better

num[l] = target − (num[i] + num[j] + num[k])

Now in this we will use 3 loops & a hashmap.

[1 2 -1 -2 2 0 -1]

i j k

3 values b/w j & k.

hashmap

TC = n^3 × log(n)

SC = O(N) + O(quads) * 2

↓

set

Code

```
set<vector<int>> st;
for (i=0 to n)
  { for (j = i+1 to n)
      set <long long> hashset;
      { for( k = j+1 to n)
          { long long sum = nums[i] + nums[j];
            sum = sum + nums[k];
            long long fourth = target - (sum);
            if (hashset.find(fourth) != hashset.end())
              { vector<int> temp = {nums[i], nums[j], nums[k],
                                    nums[l]};
                sort (temp.begin(), temp.end());
                st.insert(temp);
              }
            hashset.insert(nums[k]);
          }
      }
  }
vector<vector<int>> ans (st.begin(), st.end());
return ans;
```

(3) **optimal** does not use hashset instead uses 2 pointer method.

Stp1 → Sort the array.

arr = [1 1 1 2 2 2 3 3 3 4 4 4 5 5]

i j k l

TC → $O(n^3 \times n) \simeq O(n^3)$

SC → O(no. of quads)

Code

```
int n= nums.size();
vector<vector<int>>ans;
sort(nums.begin(), nums.end());
for(i=0 to n)
{ if(i>0 && nums[i]==nums[i-1]) continue;
   for(j = i+1 to n)
   { if(j!=(i+1) && nums[j]==nums[j-1] continue;
      int k = j+1;
      int l = n-1;
      while(k<l)
      { long long sum = nums[i];
         sum += nums[j];
         sum += nums[k];
         sum += nums[l];
         if(sum==target)
         { vector<int> temp ={nums[i], nums[j], nums[k], nums[l]};
            ans.push_back(temp);
            k++; l--;
            while(k<l && nums[k]==nums[k-1]) k++;
            while(k<l && nums[l]==nums[l+1]) l--;
         }
         else if(sum<target) k++;
         else l--;
      }
   }
}
return ans;
```

Longest Common Sequence

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

eg [101, 102, ⇒ 2 ?
[1 2 3 4 ⇒ len = 4]
[100, 101, 102 → len = 3]
} longest length = 4.

① Brute

```
longst = 1                                   TC = O(n²)
for (i = 0 to n)
 { x = arr[i]
   count = 1;
   while (linearsearch(arr, x+1) == true)
    {   x = x+1;
        count = count+1;
    }
 }
```

② Better → In this 1st ly sort the array and take lastsmaller for comparing current ele.

```
count=0; longst = 1 , lastsmaller = INT_MIN;
for (i=0 to n)                               TC = O(n)
{ if (arr[i]-1 == lastsmaller)
   { count++;
     lastsmaller = arr[i];
   }
  else if (arr[i] != lastsmaller)
   { counter = 1;
     lastsmaller = arr[i];
   }
longst = max (longst, counter);
```

③ Optimal → using set for storing all ele uniquely.
(unordered)

We will do ele by ele approach.

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

Sol → set = {102, 4, 100, 1, 101, 3, 2}.

(i) take 102 (in optimal we will work back of that no.)
search for 101 ⟶ It is present so we will not start from this

(ii) take 4
search for 3 → " " " "

(iii) take 100
search for 99 ⟶ not present so probably this is starting point (100)
so now look for 101 → yes
102 → yes
103 → NO
} counter = 3 (as of now).

(iv) take 1
search for 0 → not present " " "
so now look for 2 → yes
3 → yes
4 → yes
} counter = 4 (as of now)

(v) take 101
search for 100 → yes, so move forward.

(vi) take 3
search for 2 → yes " "

(vii) take 2
search for 1 → yes " "

So longest = 4.

Issue kam aage ke ele search or previous search, agar previous wala set me nahi hai iska matlab ye starting point ho skta hai ek consecutive sequence ka isliye ab iske aage ke ele ko dekhenge.

Code

```
int n= a.size();
if (n==0) return 0;
longt = 1;
unordered_set<int > st;
for (i= 0 to n)
 { st.insert(a[i]);}
```

} O(n) → worst case

```
for (auto it : st) {
    if (st.find(it-1)== st.end())
     {   int cnt = 1;
         int x = it;
         while (st.find(x+1) != st.end())
         { x=x+1;
           cnt = cnt+1;
         }
         longest = max(longest, cnt);
     }
}
return longest;
```

TC → If we use unordered_set then O(1) → avg
& O(n) → worst case when collision happen (rare case) → worst

$O(n) + O(2n) = O(3n)$

SC → O(N)

Longest Subarray with Sum K

Subarray means → contigous part of the array.

arr = [1 2 3 1 1 1 1 4 2 3]

K = 3

ans → subarrays of sum = 3 → [1 2] → len = 2
[1 1 1] → len = 3
[3] → len = 1
} longest = 3.

① Brute

Generate all subarray.

like → ① → [1 2
[1 2]
[1 2 3]
⋮
[1 2 3 — — 3]

② [2]
[2 3]
[2 3 1]
⋮
[2 3 1 — -3]

etc.

Code

```
len=0
for (i=0 to n)
{ for (j=i to n)
  { sum=0;
    for (k=i to j)
      { sum+=arr[k]; }
    if (sum==K) len= max(len, j-i+1)
  }
}
print(len);
```

TC = O(n³)

SC = O(1)

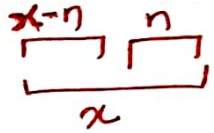
(1) Better Brute (2nd)

In brute force we move on summing elem in subarray sum fes. Here we can do one thing that is stop iterating when sum is greater than target (K).

$T(=O(n^2)$

```
for (i=0 to n)
{   sum=0
    for (j=i to n)
    {  sum+ =arr[j];
       if(s==k) len = max(len , j-i+1)
    }
}
```

(2) Better use hashing. [Prefix Sum]



arr = [1 2 3 1 1 1 1 4 2 3]
(0 1 2 3 4 5 6 7 8 9)

k = 3

Hashmap

initially prefix sum = 0 (1) len = 0 2

ele → 1

ele → 1+2 = 3

ele → 1+2+3 = 6

3 3 ... 3 (6) we need sum = 3

does there someone with sum = 3 → ~~so move~~ Yes

so one more subarray set of length = 1

ele → 1+2+3+1 = 7

4 3 ... (7) does there someone with sum = 4 → no so move

ele → 1+2+3+1+1 = 8 " "

ele → 1+2+3+1+1+1 = 9 " " sum = 6 → Yes

6 3→sum ... (9) so new subarray of length = 3

[1 1 1]

etc

Code

```
int longestSubarrayWithSumK(vector<int> a, long long k)
{
    map<long long, int> preSumMap;
    long long sum = 0;
    int maxLen = 0;
    for(i = 0 to a.size())
    {   sum += a[i];
        if (sum == k)
        {   maxLen = max(maxLen, i+1); }
        long long rem = sum - k;
        if(preSumMap.find(rem) != preSumMap.end())
        { int len = i - preSumMap[rem];
          maxLen = max(maxLen, len);
        }
        if (preSumMap.find(sum) == preSumMap.end())
        { preSumMap[sum] = i; }

    } or
    return maxLen;
```

TC → using ordered map → $O(N \times \log N)$] → donot have any collision
unordered map → $O(N \times 1)$

SC → $O(N)$

Optimal (2 pointers approach).

arr = [1 2 3 1 1 1 1 3 3]
↑ ↑

Code

```
int longestSubarraySumK(vector<int> a, long long k)
{
    int left = 0, right = 0;
    long long sum = a[0];
    int maxlen = 0;
    int n = a.size();
    while (right < n)
    {
        while (left <= right && sum > k)
        {
            sum -= a[left];
            left++;
        }
        if (sum == k)
        {
            maxlen = max(maxlen, right - left + 1);
        }
        right++;
        if (right < n) sum += a[right];
    }
    return maxlen;
}
```

Sort an array of 0's 1's & 2's.

[Optimal → Dutch National Flag Algorithm]

arr[] = [0, 1, 2, 0, 1, 2, 1, 2, 0, 0, 0, 1]

① Brute → Sort an array

TC → O(N log N) } Merge Sort.
SC → O(N)

② Better

In one loop count the no. of zero, 1 and 2.
and then from starting we fill 0, 1, 2 based on the count.

Code

```
cut0 = 0 , cut1 = 1 , cut2 = 2
for ( i = 0 to n)
 {    if (a[i] == 0) cut0++;
      else if(a[i] == 1) cut1++;
       else if (a[i] == 2) cut2++;
 }
for (i = 0 ; i<cut0 ; ++i) a[i] = 0;
for (i = cut0; i<cut0+cut1 ; ++i) a[i] = 1;
for (i = cut0+cut1 ; i<n ; ++i) a[i] = 2;
```

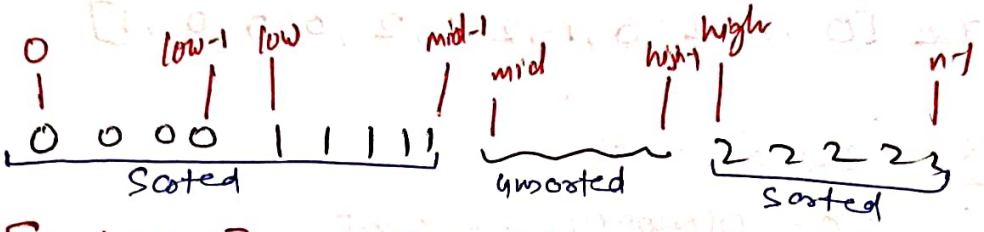
} O(N)

} O(N)

TC = O(2N)

SC = O(1)

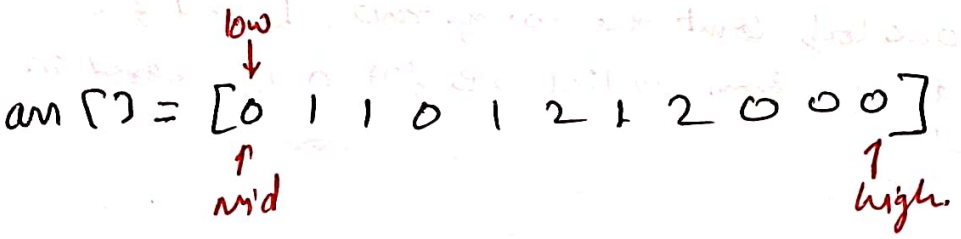
③ Optimal (Dutch National flag Algorithm)



[0 - low-1] → 0 extreme left

[low - mid-1] → 1

[high+1, n-1] → 2 extreme right



if a[mid] = 0 → swap(a[low], a[mid])
low++, mid++;

if a[mid] = 1 → ~~swap(a[1])~~ mid++;

if a[mid] = 2 → swap(a[high], a[mid])
high--;

Code

```
low = 0 , mid = 0, high = n-1;
while (mid ≤ high)
{   if (a[mid] == 0)
      { swap (a[low], a[mid]);
        low++; mid++; }
    else if (arr[mid] == 1){
            mid++; }
    else {
      swap (a[mid], a[high]);
      high--;
      }
}
```

TC = O(N)

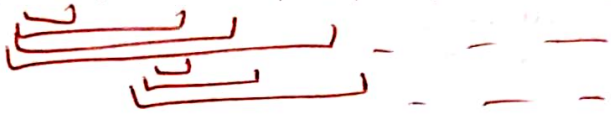
Maximum Subarray Sum

[Kadane's Algo]

Subarray means → contiguous part of the array.

① **Brute** . Try out all subarrays.

[-2, -3, 4, -1, -2, 1, 5, -3]



```
maxi = INT_MIN;
for (i=0 to n)
{   for (j = i to n)
    { sum = 0;
      for (k = i to j)
      { sum += arr[k]; }
      maxi = max(sum, maxi);
    }
}
```

TC = $O(N^3)$

SC = $O(1)$

② **Better**

```
maxi = INT_MIN;
for (i=0 to n)
{   sum = 0;
    for (j = i to n)
    { sum += arr[j];
      maxi = max(sum, maxi);
    }
}
```

TC = $O(N^2)$

SC = $O(1)$

③ **Optimal** (Kadane's Algo)

arr = [-2, -3, 4, -1, -2, 1, 5, -3]

maxi = 4 7

sum = 0 -2 4 / -2 0 3 x x 7 4

So max = 4

When sum < 0 then do sum = 0

code.

```
long long subarraysum (int arr[], int n)
{ long long sum=0 , maxi= LONG_INT;
  for(i=0 to n)
  { sum += arr[i];
    if (sum > maxi)
        maxi= sum;
    if (sum<0)
        sum=0;
  }
if(maxi<0) maxi=0;
return maxi;
```

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

few test case

(1) [-4 -2 -3 -1]

max subarray sum = -1

so if sum<0 then return empty subarray which means sum=0

Ques → Print any subarray of max. sum.

```
for(i=0 to n)
{ if (sum==0)  start=i;
  sum = sum + arr[i];
  if (sum > maxi)
  { maxi = sum;
    ansStart = start , ansEnd = i;
  }
  if (sum<0)
  { sum=0;
  }
}
```

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

Number of Subarrays with Sum K

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

Qus → Find the no. of subarrays whose sum is K.

Sol

① Brute Generate all subarrays.

1
1 2
1 2 3 ——
. 1 2 3 ——-3.

2
2 3
2 3 -3
⋮
2 3 -3 — — -3

— — . etc.

```
counter = 0
for (i = 0 to n)
{ for (j = i to n)
  { sum = 0
    for (k = i to j)
        sum = sum + arr[k];
    if (sum == k)
        counter++;
  }
}
```

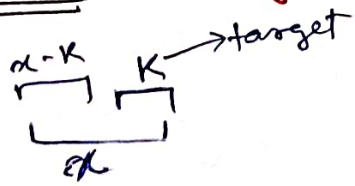
TC = O(n³)

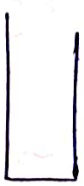
② Better

```
cnt = 0
for(i=0 to n)
{ sum = 0;
  for (j = i to n)
    sum += arr[j];
  if (sum == k)
    cnt++;
}
```

TC = O(n²)

③ Optimal (using prefix sum) + (hashmap)





Hashmap <Presum, counter>

arr = [1 2 3 -3 1 1 1 4 2 -3] K = 3

dry run

presum = 0 1 3 6 3 4 5 6 10 12 9

cut = 0 1 2 3 4 5 6 8

for ele → 1 : 1-3=-2, 3 → K; -2 is not in the hashmap.

for ele → 2 : 3-3=0, 3; 0 is present in hashmap; 2+1=3

for ele → 3 : 6-3=3, 3; 3 is present in hashmap; 3+2+1=6

for ele → 1 : 4-3=1, 3 → K; 1 is a presum so if we exclude '1' then there is a sum of K(3)

for ele → 1 : 3, 3; presum 3 has 2 occurrences so which on removal give sum = 3.

Hashmap (presum, cut)
(9,1)
(12,1)
(10,1)
(5,1)
(4,1)
(6,2)
(3,2)
(1,1)
(0,1)

etc.

Code

```
{ unordered_map <int, int> mpp;
  mpp[0] = 1;
  int preSum = 0, cut = 0;
  for (i = 0 to n)
  { preSum += arr[i];
    int remove = preSum - K;
    cut += mpp[remove];
    mpp[preSum] += 1;
  }
  return cut;
}
```

Number of Subarrays with XOR as K

XOR → compares 2 bits and if 2 bits are diff. → returns 1
2 bits are same → " 0

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

$4 \wedge 2 =$
$100 \wedge 010 = 110 = 6$

$\boxed{7 \wedge 7 = 0}$

(XOR is used in bit manipulation tricks)

arr = [4 2 2 6 4] K=6

Ans → So no. of subarrays with XOR=6 are.
[4 2] [2 2 6] [4 2 2 6 4]
count = 4

① $x \wedge x = 0$
② $x \wedge 0 = x$

① **Brute** generate all subarrays and count the subarrays with XOR as K.

```
for(i=0 to n)
{ for(j=i to n)
  { xor=0
    for(k=i to j)
      { xor = xor^arr[k]; }
    if(xor==k) cnt++;
  }
}
```

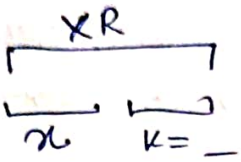
TC = $O(n^3)$
SC = $O(1)$

② **Better**

```
for(i=0 to n)
{ xor=0
  for(j=i to n)
  { xor = xor^arr[j];
    if(xor==k) cnt++;
  }
}
```

TC = $O(n^2)$
SC = $O(1)$

③ optimal using hashmap. and similar approach like prefixSum.



$\Rightarrow x \wedge K = XR$

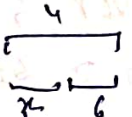
$\Rightarrow x \wedge K \wedge K = XR \wedge K$

$\Rightarrow \boxed{x = XR \wedge K}$

→ Here from the starting XOR is XR and we are looking for K and from starting we are looking for x in the hashmap.

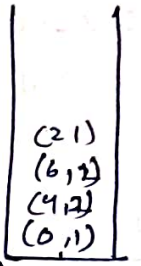
→ arr = [4 2 2 6 4] K = 6

for ele 4 → XR = 4



↳ $XR \wedge K \Rightarrow 4 \wedge 6 = 2$

So no 2 is present previously. cnt = 0



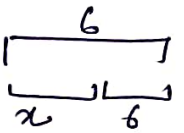
Ans.
cnt = 4

100
110
010

100
010
110

110
010
100

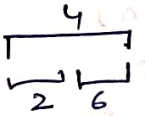
for ele 2 → XR = 4 ^ 2 = 6



$x = 6 \wedge 6 = 0$ (which is present

cnt = 1

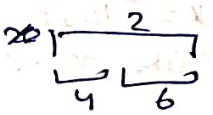
for ele 2 → XR = 6 ^ 2 = 4



$x = 2$ (not present)

cnt = 1

for ele 6 → XR = 4 ^ 6 = 2



code

```
int XR = 0;
map<int, int> mpp;          ]→ NlogN or L
mpp[XR]++;
int cnt = 0;
for(i = 0 to a.size())
{   XR = XR ^ a[i];
    int x = XR ^ K;
    cnt += mpp[x];
    mpp[XR]++;
}
return cnt;
```

TC = O(N)

SC = O(N)

Set Matrix zeroes

Ques → Given an m×n integer matrix, if an element is 0, set its entire row and column to 0. You must do it inplace.

① Brute

→ First we will use 2 loops (nested loops) to traverse all the cells of the matrix.
→ if any cell (i,j) contains the value 0, we will mark all cells in row i and column j with -1 except those which contain 0.
→ we will perform step 2 for every cell containing 0.
→ finally we will mark all the cells containing -1 with 0.
→ Thus the given matrix will be modified according to the question.

Code

```
for (i=0 to n)
{ for (j=0 to m)
   { if(a[i][j]==0)
      { MarkRow(i)
        MarkCol(j)
      }
   }
}
```

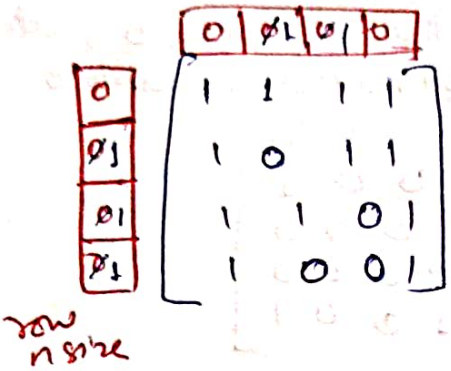
```
markRow(i)
{ for(j=0 to m)
   { if (a[i][j]!=0)
        a[i][j]=-1;
   }
}
```

```
markCol(j)
{ for(i=0;i<n;++i)
   { if(a[i][j]!=0)
        a[i][j]=-1;
   }
}
```

```
for (i=0 to n)
{ for(j=0 to n)
   { if(a[i][j]==-1)
        a[i][j]=0;
   }
}
```

TC = (nm)(n+m) + (nm)
≈ O(n³)

② Better (using 2 extra arrays)



→ Declare 2 arrays row[n] & col[m] all stored with 0.

→ Traverse each cells, if any cell (i,j) contains 0, we will mark ith index of row array i.e. row[i] and jth index of col array col[j] as 1. (It signifies that all the elements in the ith row and jth column will be 0 in the final matrix)

→ we again traverse the entire matrix and we will put 0 into all the cells (i,j) for which either row[i] or col[j] is marked as 1.

Code

```
col[m]={0}, row[n]={0}
for(i=0 to n)
{ for(j=0 to m)          } O(nm)
  { if(a[i][j]==0)
      { row[i]=1;
        col[j]=1;
      }
  }
}
```

```
for(i=0 to n)
{ for(j=0 to m)          } O(nm)
  {
    if(row[i] || col[j])
      a[i][j]=0.
```

TC = O(2×n×m)

SC = O(n) + O(m)