```
int longestOnes( str, k ){
  int s = 0;   // start
  int e = 0;   // end
  int zero_count = 0;
  int max_length = 0;
  for ( e ; e<str.length() ; e++){
    if (str [e] == '0' ) {
      zero_count ++;
    }
    while ( zero_count > k){
      if (str[start] == '0')
      start ++;
    }
    // zero_count <= k
    max_length = max(max_length, e-s+1);

  return max_length;
}
}
```

WINDOW BADA

WINDOW CONTRACT

MATRIX

1. Set Matrix zeros

1	0	1	1
0	0	0	1
1	1	0	1
1	0	1	1

BRUTE: ~~The~~ wherever we get 0, mark it's whole row and column as -1; After that make one more iteration and whenever arr[i] == -1 mark it as zero.

code:

```
for ( i=0 ; i<n )
    for ( j=0 ; j<m )
        if (arr [i] [j] == 0)
            MarkRow (i)
            MarkCol (j)

mark Row (i)
    for ( j=0 ; j<m ; j++)
        if ( arr [i] [j] != 0)
            arr [i] [j] = -1

mark col (j)
    for ( j=0 ; j<n ; i++)
        if ( arr [i] [j] != 0)
            arr [i] [j] = -1

zero_count - - ;
```

TC: $(n \times m) \times (n+m) \neq (n+m) \approx n^3$

SC:

Better

1	1	1	1
1	0	1	1
1	1	0	1
1	0	0	0

0
0
1
1
0

col → m size

row → n size

CODE:

```
col [m] = {0}, row [n] = {0}
for ( i = 0 → n ) {
    for ( j = 0 → n ) {
        if (arr[i][j] == 0) {
            row [i] = 1;
            col [j] = 1;
        }
    }
}

for ( i=0 → n)
    for ( j = 0 → m)
        if (row[i] || col [j])
            a[i] [j] = 0
```

TC: $O(n \times m) \times 2$

SC: $O(n) + O(m)$

OPTIMAL : col[0] = row[n]

row[0] = col[m]

col0 = 1	1	2	3
1	~~1~~ 0	~~1~~ 0	1
~~1~~ 0	0	1	1
~~1~~ 0	1	0	1
0	1	1	1

↳ start making zeros from here

☆

```
int col0 = 1
for (i = 0 → n)
    for (j = 0 → n)
        if (mat[i][j] == 0) {
row  →  mat[i][0] = 0;
~~Bcoz~~ → if (j == 0)
column → mat[0][j] = 0;
                else
Bcz column ka first → col0 = 0;
}
}
```

TC: O(N*m)

SC: O(1)

```
for (i = 1 → n)
    for (j = 1 → m)
        if (matrix[i][j] != 0) {
            if (mat[0][j] == 0 ||
        mat[i][0] == 0) {
            max[i][j] = 0;
        }
}
}
if (matrix[0][0] == 0) {
    for (j = 0 → m) max[0][j] = 0;
}
if (col0 == 0) {
    for (i = 0 → n)
        max[i][0] = 0;
```

(1 1 | 2 2 | 3 3)

(1 0 | 2 0)

(0 | 0)

we start from 1 bcoz 0th [i][j] already filled with 0's

Q) Rotate Matrix by 90°

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

→

7	4	1
8	5	2
9	6	3

i	j		j	i
[0]	[0]	→	[0]	[2]
[0]	[1]	→	[1]	[1]
[0]	[2]	→	[2]	[0]

0 → 2 1 → 1 2 → 0

i → (n-1) i → (n-1) - i j → (n-1) - i

[i] [j] → [j] [n-1-i]

BRUTE

```
ans[n][n]
for (i = 0 → n) {
    for (j = 0 → n) {
        ans[j][n-1-i] = mat[i][j]
    }
}
```

TC: O(N²)

SC: O(N²)

OPTIMAL SOLN

→ Transpose Matrix

→ Reverse elements row wise

[0] [1] → [1] [0]

[0] [2] → [2] [0]

~~[0] [3] → [3] [0]~~

[1] [2] → [2] [1]

i.e, (i, j) → (j, i)

```
int n = mat.size();
for (i = 0 ; i < n-1)
    for (j = i+1 ; j < n)
        swap (mat[i][j], mat[j][i])
}
for (i = 0 ; i < n)
    reverse (mat[i].begin, mat[i].end)
}
```

Q) Spiral Matrix

	0	1	2	3	4	5
0	1	2	3	4	5	6
1	20	21	22	23	24	7
2	19	32	33	34	25	8
3	18	31	36	35	26	9
4	17	30	29	28	27	10
5	16	15	14	13	12	11

top

bottom

top = 0
left = 0
bottom = 5
right = 5

TC: O(N×M)
SC: O(N×M)

```
int n = mat.size();  // rows
int m = mat[0].size();  // columns
int left = 0, top = 0;
int right = m-1; left = n-1;
vector<int> ans;
while (top <= bottom && left <= right)
for (i = left → right)
    ans.push_back(mat[top][i])
}
top ++;
for (i = top → bottom) {
    ans.push_back(mat[i][right]);
}
right --
if (top <= bottom) {
for (i = right → left) {
    ans.push_back(mat[bottom][i])
}
bottom --;
for (i = bottom ; i >= top ) {
    ans.push_back (mat[i][left])
}
left ++;
}
return ans;
}
```

size_t is an unsigned integer type used to represent the size of objects in byte.

Search Pattern (KMP Algo)

txt = | a | a | a | a | a | d | a | a | a | d |

pat = | a | a | a | d |

LPS = | 0 | 1 | 2 | 0 |

```
int k = pat.size();
vector<int> lps (k, 0);
for (i = 1, len = 0 ; i < k) {
    lps[i++] = ++len;
}
else if (len)
    len = lps [len - 1];
else
    lps[i++] = 0;
}
for (i = 0, j = 0 ; i < txt.size() ) {
    if (txt[i] == pat[j]) {
        i++, j++;
        if (j == k) {
            res.push_back (i, j);
            j = lps [j-1];
        }
    }
    else if (j)
        j = lps [j-1];
    else
        i++;
}
return res;
```

Q) Find no. of subarrays with sum K. contiguous part of array

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

K = 3

BRUTE: Generate all subarrays

```
count = 0
for (i = 0 ; i<n ; i++) {
    for (j = i ; j<n ; j++) {
        sum = 0                 TC ≈ O(N^3)
        for (k = i → j)         SC : O(1)
            sum = sum + arr[k];
        if (sum == K)
            count++;
    }
}
```

BETTER: Within two loops TC ≈ $O(N^2)$

```
cnt = 0
for (i = 0 → n) {
    sum = 0
    for (j = i → n) {
        sum += arr[j].
    }
}
if (sum == K)
    cnt++;
```

OPTIMAL: (PREFIX SUM)

presum − 3 = 3 → 3

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

→ presum = 6 = K

Steps:
→ prefix sum = K
→ K = 3
→ look for subarray with sum (K − K).
→ Reverse engineering

S − K ?? → K

NO. of S − K = NO. of K

prefsum = S

Dry run: K = 3 HASH MAP

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

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

S − K = 6 − 3 = 3 → K = 3

S − K = 3 6

cnt = 0 1 2 3 4 5 8

6 − 3 = 3

6

(3, 1)
(6, 1)
(6, 2)
(3, 2)
(1, 1)
(0, 1)

(presum, cnt)
Key Value

TC: O(N log N)

```
unordered_map<int, int> mpp;
mpp[0] = 1;
int presum = 0, cnt = 0;
for (i = 0 ; i < arr.size(); i++)
{
    presum += arr[i];
    int remove = presum - K;
    cnt += mpp[remove];
    mpp[presum] += 1;
}
return cnt;
}
```

K = −10

Q) arr = [10 2 −2 −20 10]

i = 0 sum = 10 max = 10

i = 1 sum = 12 max = 22

i = 2 sum = 10

i = 3 sum = −10 max = 0

(0, 3)
(0, 2)
(10, 2)
(12, 1)
(10, 1) K
(0, 1) −

Count = 1 2 3

Q) PASCAL'S △

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

① Given row, & column
→ tell me element at that place

Eg: R=5 C=3 ans = 6

② Print any nth row of Pascal △

Eg: N=5 ans: 1 4 6 4 1

③ Print entire Pascal △

$^{10}C_3 = \frac{10 \times 9 \times 8}{3 \times 2 \times 1}$

↳ shortcut that we are going to use

$^{R-1}C_{C-1}$

$= {}^4C_2 = 6$

④

```
fn NCR (n, r)   \\ (r-1, c-1)
{
    res = 1
    for (i=0; i<r; i++) {
        res = res * (n-i);
        res = res / (i+1);  (pehle 1 se divide phir 2 se and so on)
    }
    return res
}
```

⑤ Nth row → n element (n=5)

Column	1	2	3	4	5
Row = 5	1	4	6	4	5

$^4C_1$ ~~[illegible]~~

$^4C_2 = 6$

$= \frac{4 \times 3}{1 \times 2}$

1 4 $\frac{3 \times 4}{1}$

ans = 1 at column 1

= ans * $\frac{4}{1}$ at " 2

ans × 3 at " 3

ans × $\frac{4}{3}$ at col 4

ans × $\frac{5}{4}$ at col 5

```
ans = 1
print(ans)
for (i = 1 → n)
    ans = ans * (n-i)
    ans = ans / (i)
    print(ans)
}
```

TC: O(n)
SC: O(1)

③

```
ans = []
for (row = 1 → n)
{   temp = []
    for (col = 1 → row)
    {
        temp.add(nCr(row-1, col))
    }
    ans.add(temp);
}
print(ans)
```

SC(n²)

OPTIMAL

```
vector<int> generateRow(int row){
    long long ans = 1;
    vector<int> ansRow;
    ansRow.push_back(1);
    for (int col = 1; col < row; col++)
    {
        ans = ans * (row - col);
        ans = ans / (col);
        ansRow.push(ans);
    }
    return ansRow;
}
vector<vector<int>> pascalΔ(int n) {
    vector<vector<int>> ans;
    for (i = 1 → <= n)
        vector <int> temp = generateRow(i);
        ans.push_back(temp)
    }
    return ans;
}
```

Q) 3-SUM

arr[] = [-1, 0, 1, 2, -1, -4]

arr[i] + arr[j] + arr[k] = 0

(i != j != k)

ans = [-1, 2, -1] ✓

[0, 1, -1] ✓

[2, 2, -4] ✗

BRUTE: set <vector <int>> st;

vector <vector <int>> ans;

for (i = 0 → n) { TC: $O(n^3)$ + log

for (j = i+1 → n) {

for (k = j+1 → n) {

if (arr[i] + j + k == 0)

{

//unique vector <int> temp = {num[i], j, k};

sort (temp)

st.insert (temp)

}

}

}

vector <vector <int>> ans (st.begin(), st.end());

return ans;

BETTER: $O(N^2)$ // HASHING

arr[i] + arr[j] + arr[k] = 0

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

→ Get rid of one loop

arr[] = [-1, 0, 1, 2, -1, -4]

i j j j

2 ~~(-1+1)~~ 0

→ arr[k] = -(-1+0) = 1

Not present in hash so

Move j

→ again arr[k] = -(-1+1) = 0

present in hash map (-1, 1, 0)

↓ sort (-1, 0, 1)

→ -(-1+2) = -1 ✗

→ -(-1-1) = 2 ✓

↳ (-1, -1, 2) sorted

-1
2
1
0

→ -(-1-4) = 5 ✗

Move i = 1 j = 2

empty hash set

arr[] = [-1, 0, 1, 2, -1, -4]

i j j

2
1

→ -(0+1) = -1 ✗

→ -(1+0+2) = -2 ✗

→ -(0-1) = 1 ✓ [0, -1, 1]

→ -(0-4) = 4 ✗

sort ↳ [-1, 1, 0]

↓

already taken

set <vector <int>> st;

for (i = 0 → n) {

set <int> hashset;

for (int j = i+1 → n) {

int third = -(arr[i] + arr[j])

if (hashset.find(third) != hashset.end()) {

TC → $N^2 \times \log(M)$

vector <int> temp = {num[i], [j], [k]};

sort (temp.begin(), temp.end());

st.insert (temp);

}

SC → $O(N) + O(no.\ of\ ans) \times 2$

hashset.insert (num[j]);

}

vector <vector <int>> ans (st.begin(), st.end());

return ans;

}

OPTIMAL: (i) sort array

(ii) 2-pointer approach

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

(const) i ↓ j ↓ j ↓ j ↓

k ↑

→ -2 + -2 + 2 = -2 ✗

→ -2 + -1 + 2 = -1 ✗

→ -2 + 0 + 2 = 0 ✓ (-2, 0, 2) already sorted

→ Move k upto one place

also j

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

i j j k k

→ (-1, -1, 2)

→ -1 + 0 + 2 ≠ 0 ✗

→ [..., 0, 0, 0, 2, 2, 2, 2]

i j k k

(0 + 0 + 2) > 0 ✗

Reduce k ...

(0 + 0 + 0) = 0 ✓

```
vector<vector<int> ans;
sort (num.begin(), num.end())
for ( i = 0 → n) {
    if (i > 0 && num[i] == num[i-1]
        continue; // it means go back
    to for loop ; don't go down

    int j = i+1;
    int k = n-1;
    while (j < k) {
        int sum = num[i] + j + k
        if (sum < 0) {
            j++;
        else if (sum > 0) {
            k--;
        else {
            vector<int> temp = {num[i], j, k}
            ans.push_back (temp);
            j++;
            k--;
        }
    while (num[j] == num[j-1]) j++;
    while (num[k] == num[k+1]) k--;
    }
}
return ans;
}
```

TC: O(n log n) + O(n²)

SC: O(no. of unique triplets)

temp arr. is for returning not in our algorithms

Q) 4 sum

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

target = K

a[i] + a[j] + a[k] + a[l] = target = 0

ans: [-2, -1, 1, 2]

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

BRUTE: → 4 loops

→ calculate sum

→ sum = target O(N⁴)

temp = { } } O(no. of quadrupl)

→ sort temp

→ st. insert temp

BETTER: O(N³ × log M)

→ 3 loops

→ nums[l] = target − (a[i] + j + k)

→ hash map

OPTIMAL: ① Sort given array

i j k

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

k

l l

target = 8

→ (1, 1, 2, 5) move l

→ (1 + 1 + 2 + 4)

```
int n = a.size();
vector<vector<int>> ans;
SORT
for (i = 0 -> n)
  if (i > 0 && a[i] == a[i-1])
    continue;

  for (j = i+1 ; j < n)
    if (j != i+1 && a[j] ==
      = a[j-1]) continue;

    int k = j+1;
    int l = n-1;
    while (k < l) {
      long long sum = a[i]
      sum += a[j];
          += a[k];
          += a[l];
      if (sum == target) {
vector<int> temp = {a[i], j, k, l}
ans.push (temp);
k++, l--;
while (a[k] == a[k-1] && k<l) k++;
while (a[l] == a[l+1] && k<l) l--;
      }
      else if (sum < target) k++;
      else l--;
    }
  }
}
return ans
```

---

3) MAJORITY ELEMENT (>N/3 Time)

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

n = 8 TC : O(n²) / SC : O(1)

CODE

```
ls = []
for (i = 0 -> n-1) {
  if (ls.size() == 0 ||
    ls[0] != nums[i])
  {
    count = 0;
    for (j = 0 -> n-1)
      if (nums[j] == nums[i])
        count ++;
    if (count > n/3)
      ls.add (nums[i])
```

Better

```
ls = [], map, min = (n/3)+1
for (i = 0 -> n-1) {
  mp{arr[i]}++;
  if (mp[arr[i]] == min)
    ls.add (arr[i])
```

OPTIMAL

```
int cnt1 = 0, cnt2 = 0;
int el1 = INT_MIN;
int el2 = INT_MIN;
for (i = 0 -> n) {
  if (cnt1 == 0 && el2 != v[i]) {
    cnt1 = 1;
    el1 = v[i];
  }
  else if (cnt2 == 0 && el1 !=
    v[i])
  {
    cnt2 = 1;
    el2 = v[i];
  }
  else if (v[i] == el1) cnt1 ++;
  else if (v[i] == el2) cnt2++;
  else {
    cnt1--, cnt2--;
  }
}
vector<int> ls;
cnt1 = 0, cnt2 = 0
```

Q) Merge Overlapping

BRUTE

```
int n = arr.size();
sort (arr.begin(), arr.end());
vector<vector<int>> ans;
for (int i = 0 -> n) {
    int start = arr[i][0];
    int end = arr[i][1];
    if (!ans.empty() && end <= ans.
        back()[1]) {
        continue;
    }
    for (int j = i+1; j < n; j++) {
        if (arr[j][0] <= end) {
            end = max(end, arr[j][1]);
        }
        else {
            break;
        }
    }
    ans.push_back({start, end});
}
return ans;
}
```

OPTIMAL

```
int n = arr.size();
sort (arr.begin(), arr.end());
vector<vector<int>> ans;
for (i = 0 -> n) {
    if (ans.empty() || arr[i][0] >
        ans.back()[1]) {
        ans.push_back(arr[i]);
    }
    else {
        ans.back()[i] = max(ans.
            back()[i], arr[i][i]);
    }
}
return ans;
}
```

Q) Merge two Sorted arrays without extra space

arr1 = [1 3 5 7] //n

arr2 = [0 2 6 8 7] //m

Sorted arr = [0 1 2 3 5 6 7 7 8]

arr1 = [0 1 2 3]

arr2 = [5 6 7 7 8]

BRUTE:

```
vector<int> arr3;
int left = 0, right = 0,
index = 0;
while (left < n && right < m)
    if (arr1[left] <= arr2[right])
        arr3[index] = arr1[left]
        left ++, index ++;
    else
        arr3[index] = arr2[right];
        right ++, index ++;
while (left < n) {
    arr3[index ++] = arr1[left++]
}
while (right < m) {
    arr3[index ++] = arr2[right ++]
}
int index = 0;
for (i = 0 -> n+m; i++)
    if (i < n) arr1[i] = arr3[i]
    else
        arr2[i-n] = arr3[i]
```

TC: O(m+n) + O(m+n)

SC: O(m+n)

OPTIMAL : SC : $O(1)$
TC : $O(m+n)$

arr 1 = [1 3 ~~7~~ 2 ~~7~~ 0]
arr 2 = [~~0~~ 7 ~~7~~ 5 6 8 9]

→ 7 > 0 , swap
→ 5 > 2 , swap
→ 5 > 6 ✗
→ 3 > 8 ✗
→ 3 > 9 ✗

Now sort arr 1 , arr 2

```
int left = n-1;
int right = 0;
while (left >= 0 && right < m)
{
    if(arr1[left] > arr2[right]){
        swap (arr1[left], arr2[right]);
        left--, right++;
    }
    else {
        break;
    }
}
sort (arr1, arr1+n);
sort (arr2, arr2+m);
}
```

TC: $O(\min(n,m)) + O(n\log n) + O(m\log m)$
SC: $O(1)$

2nd Optimal soln : (Gap Method) → Shell Sort

```
int len = (n+m)
int gap = (len/2) + (len%2);
while (gap > 0){
    int left = 0;
    int right = left + gap;
    while (right < len) {
        if (left < n && right >= n)
```

Q) Find the Missing and Repeating Number

arr = [4 3 6 2 1 1]
n = 6 ans = {1, 5}

BRUTE:

```
for (i = 1 → n)
    cnt = 0
    for (j = 0 → n-1)
        if (arr[j] == i)
            cnt++;
    if (cnt == 2) repeating = i;
    else if (cnt == 0) missing = i;
    if ( != -1 && != -1)
        break;
}
```

TC: $O(N^2)$
SC: $O(1)$

BETTER: HASH ~~MAP~~ ARR

1 to 6

hash arr	0	~~2~~ 1	~~0~~ 1	~~1~~ 1	~~1~~ 1	0	~~1~~ 1
	0	1	2	3	4	5	6

size = 7

repeating = 1 , missing = 5

```
int n = a.size();
int hash[n+1] = {0};
for (i=0 → n)
    hash[a[i]]++;
int repeating = -1, missing = -1;
for (int i = 1; i<=n; i++)
    if (hash[i] == 2) repeating = i
    else if (hash[i] == 0) missing = i
    if (repeating != -1 && missing != -1) {
        break
    }
return {repeating, missing};
```

TC: $O(2n)$
SC: $O(n)$

Q) OPTIMAL 1 : XOR (TOUGH)

Q) OPTIMAL 2 : By SUM

repeating = 1 = x, missing = 5 = y

1. SUM

$(1+2+3+4+5+1) - (1+2+3+4+5)$

$\Rightarrow 1 - 5 = -4$

i.e $x - y = -4$ — (1)

2. SUM of SQUARES

$(4^2+3^2+6^2+2^2+5^2+1^2) - (1^2+2^2+3^2+4^2+5^2)$

$1^2 - 5^2 = -24$

$\Rightarrow x^2 - y^2 = -24$

$(x-y)(x+y) = -24$

$x + y = 6$ — (2)

$x = 1, y = 5$

CODE :

```
int n = a.size();
int sN = (n * (n+1)) / 2;
int s2N = (n * (n+1) * (2n+1))/6;
int s = 0, s2 = 0;
for (i=0 -> n)
    s += a[i]
    s2 += (i+1) a[i] + (i+1) a[i];
int val1 = s - sN;
int val2 = s2 - s2N;
val2 = val2 / val1 ; // x+y
int x = (val1 + val2) / 2;
int y = x - val1;
return {(int)x, (int)y};
```

TC : O(N)
SC : O(1)

Q) COUNT INVERSIONS

arr[] = [5 3 2 4 1]

$i < j$ && $a[i] > a[j]$

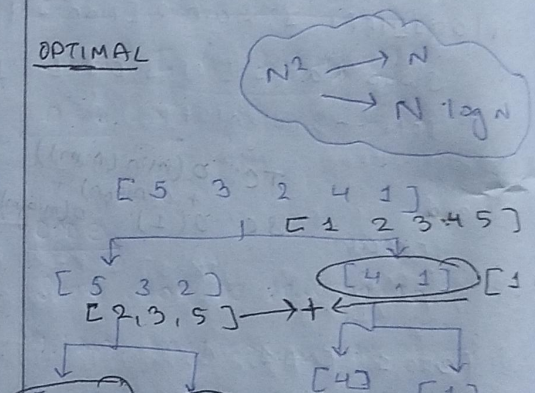
BRUTE :

```
cnt = 0
for (i=0 -> n-1)
    for (j = i+1 -> n)
        if (a[i] > a[j])
            cnt++;
```

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

OPTIMAL

$N^2 \rightarrow N$
$\rightarrow N \log N$



TC : N log N

Q) REVERSE PAIRS (HARD)

arr = [40 25 19 12 9 6 2]

i < j && a[i] > 2 × arr[j]

→ Run merge sort program

→ void fun ( )

int right = mid + 1;

for (i = low ; <= mid)

while (right <= high && a[i] > 2 * a[right]) right++;

cnt += (right - (mid + 1));

}

}