14) Sort an Array 0's, 1's & 2's (DNF Alg

→ Brute :- (Direct sorting)

logic : use sort()

TC : O(n log n)

SC : O(1)

code: sort(arr.begin(), arr.end());

→ Better :- (counting sort)

logic: count no. of 0, 1, 2 & overwrite

TC: O(n)

SC: O(1)

code:

```
int cnt0=0, cnt1=0, cnt2=0;
for(int num : arr) {
    if(num==0) cnt0++;
    else if(num==1) cnt1++;
    else cnt2++;
}
for(int i=0 → cnt0) arr[i]=0;
for(int i=cnt0 → cnt0+cnt1) arr[i]=1;
for(int i=cnt0+cnt1 → cnt0+cnt1+cnt2) arr[i]=2;
```

→ optimal :- (Dutch National flag Algo)

logic: use three pointers low, mid, high

TC: O(n)

SC: O(1)

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

mid ↑ (index 0)

High ↑ (index 9)

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

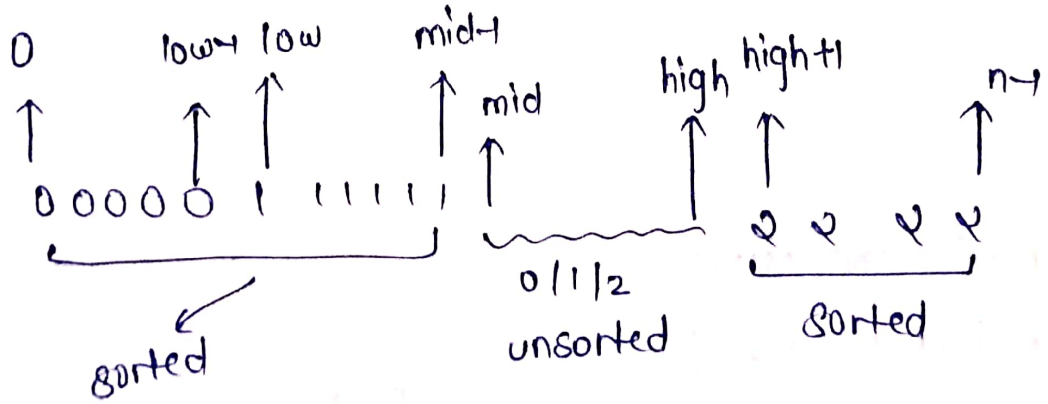
a[mid] == 1 → mid++

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

[0 ..... low-1] → 0 extreme left

[low ..... mid-1] → 1

[high+1 ..... n-1] → 2 extreme right



15. Leaders in an Array :- everything should be greater

arr[] = [10, 22, 12, 3, 0, 6]

→ [22, 12, 6]

→ Brute :-

```
for(int i=0 → n) {
    lead = true;
    for(j=i+1 → n) {
        if(a[j] > a[i]) {
            lead = false
            break
        }
    }
    new.add(arr[i]) if lead = true
}
```

TC :- $\approx O(n^2)$

SC :- O(n)

↓

for storing ans

→ optimal :-

```
for int maxi = INT_MIN;
for(int i=n-1 → 0) {
    if(a[i] > maxi) {
        ans.add(a[i]);
    }
    maxi = max(maxi, a[i]);
}
return ans;
}
```

TC :- $\approx O(n)$

SC :- O(n)

↓

for storing ans

Longest Consecutive Sequence :- (Google)

arr[ ] = [104, 4, 100, 1, 101, 3, 2, 1, 1]

→ Brute :-

arr[ ] = [104, 4, 100, 1, 101, 3, 2, 1, 1]

(Compare each & every digit)

cnt = 1 2 3

x = 100 ?
101 ?
102 ?
103

longest = 1 2 4
↑ Initially

Code :-

```
long = 1
for (i=0 → n) {
    x = arr[i];
    cnt = 1;
    While (ls(arr, x+1) == true) {
        x = x+1;
        cnt++;
    }
}
```

TC :- O(n²)

SC :- O(1)

→ Better :-

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

↓ sort

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

cnt = 0 1 2 3 4
1 2 3

last smaller = INTMIN
1 2 3 4
100 101

longest = 1 2 4

→ Code :-

```
longest = 1;
for(i=0 → n) {
    if(arr[i]-1 == last_smaller) {
        cnt++;
        last_smaller = arr[i];
    }
    else if(arr[i] != last_smaller){
        cnt = 1;
        last_smaller = arr[i];
    }
    longest = max(longest, cnt);
}
```

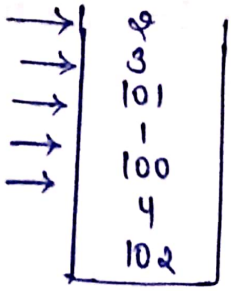
TC :- O(n log n)
SC :- O(1)

→ Optimal :-

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

⇒ a cnt = 1 2 3 4
1 2 3

longest = 1 4

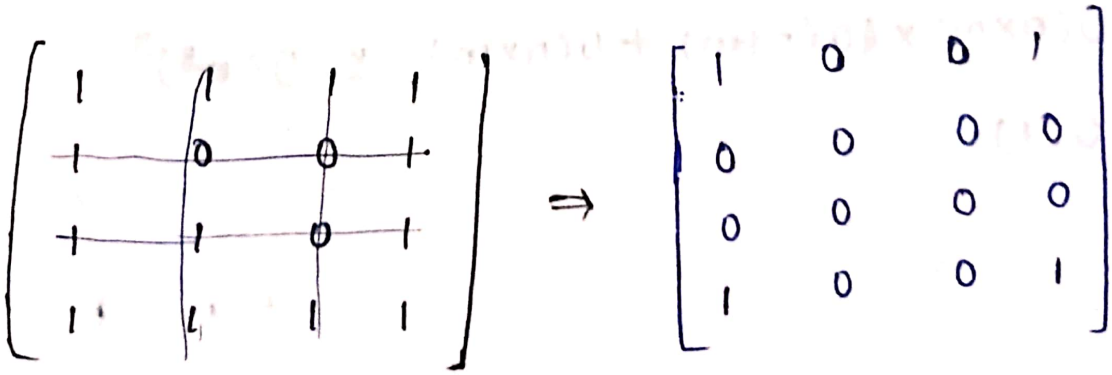


unordered set

Code :-

```
int longest = 1;
if(arr.size() == 0) return 0;
unordered_set<int> st;
for(int i=0 → n){
    st.insert(arr[i]);
}
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++;
        }
        longest = max(longest, cnt);
    }
return longest;
```

Set Matrix Zeroes :-



→ Brute :-

Code :-

```
for (i=0→n) {
  for (j=0→m) {
    if (arr[i][j]==0) {
      markRow(i);
      markCol(j);
    }
  }
}

for (i=0→m) {
  for (j=0→m) {
    if (arr[i][j]==-1) {
      arr[i][j]=0;
    }
  }
}
```

```
markRow(i) {
  for (j=0; j<m; j++) {
    if (arr[i][j] != 0)
      arr[i][j] = -1;
  }
}

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

⇒ Dry Run :-

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & -1 & -1 & 1 \\ -1 & 0 & 0 & -1 \\ -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

TC :- $O(n \times m) \times O(n+m) + O(n \times m) \simeq O(n^3)$

SC :- $O(1)$

Better :-

	0	$\emptyset_1$	$\emptyset_1$	0
0	1	1	1	1
$\emptyset_1$	1	0	1	1
$\emptyset_1$	1	1	0	1
$\emptyset_1$	1	0	0	1

col → m size

→ n size

⇒

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

Code :-

```
int col[m] = {0};
int row[m] = {0};

for(int i → n) {
    for(j → m) {
        if(mat[i][j] == 0) {
            row[i] = 1;
            col[j] = 1;
        }
    }
}

for (i → n) {
    for (j → m) {
        if(row[i] || col[j]) {
            mat[i][j] = 0;
        }
    }
}

return mat;
```

TC :- $O(m \times n) + O(m \times n)$

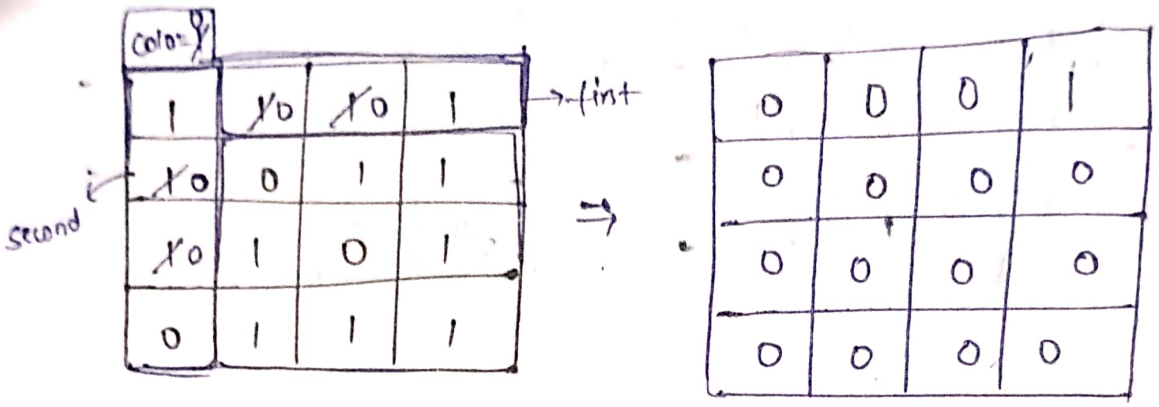
$\simeq 2O(m \times n)$

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

→ optimal :-

col[0] → row[n] → mat[...][0]

row[0] → col[m] → mat[0][...]



1	X0	X0	1
X0	0	1	1
X0	1	0	1
0	1	1	1

⇒

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

code :-

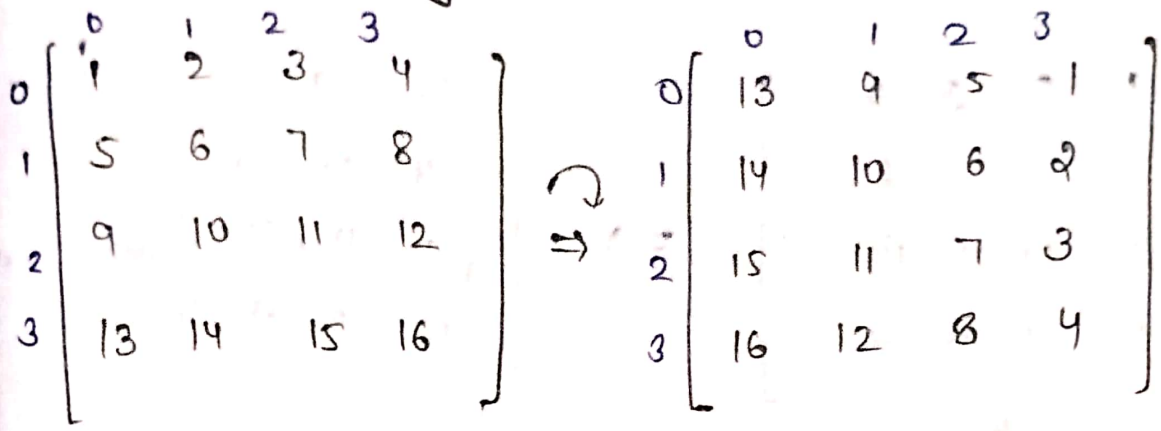
```
int col0 = 1;
for (i=0 → n) {
  for (j=0 → m) {
    if (mat[i][j] == 0) {
      // marks the i-th row
      mat[i][0] = 0;
      if (j != 0) {
        // mark the jth col
        mat[0][j] = 0;
      }
      else
        col0 = 0;
    }
  }
}
```

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

TC :- 2O(m×n)

SC :- O(1)

Rotate matrix by 90° :-



→ **Brute :-**

```
i  j      i  j  (n-1)-j
[0][0] → [0][3]
[0][1] → [1][3]
[0][2] → [2][3]
[0][3] → [3][3]

i  j      j  (n-i)
[1][0] → [0][2]
[1][1] → [1][2]
[1][2] → [2][2]
[1][3] → [3][2]
```

Code :-

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

TC :- $O(n^2)$

SC :- $O(n^2)$

→ Optimal :-

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \Rightarrow \begin{bmatrix} 13 & 9 & 5 & 1 \\ 14 & 10 & 6 & 2 \\ 15 & 11 & 7 & 3 \\ 16 & 12 & 8 & 4 \end{bmatrix}$$

⇓ Transpose

	0	1	2	3
0	1	5	9	13
1	2	6	10	14
2	3	7	11	15
3	4	8	12	16

(reverse each row)

⇒ Transpose :- i j → j i

[0][1] → [1][0]
[0][2] → [2][0]
[0][3] → [3][0]
[1][2] → [2][1]
[1][3] → [3][1]
[2][3] → [3][2]

(i=0 → n-1, j=i+1 → n)

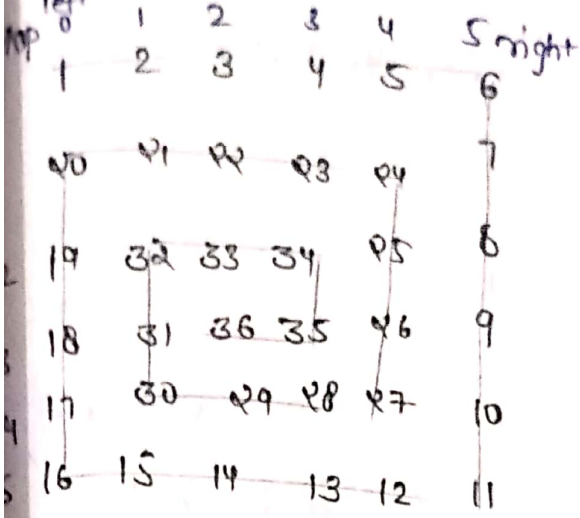
⇒ code :-

```
for(i=0 → n-1){
    for(j=i+1 → n){
        swap(mat[i][j], mat[j][i]);
    }
}

for(i=0 → n){ → n
    reverse(mat[i].begin(), mat[i].end()); → n/2
}
```

TC :- O(n/
+O(n×
SC :- O(1)

Spiral matrix :-



(right → bottom → left → top)

top = 0 bottom = 5

left = 0 right = 5

code :-

```
while (top<= bottom && left <= right) {
    for (i=left → right)
      push(a[top][i])
    top++;
    for (i=top → bottom)
       push(a[i][right])
    right--;
    if (top<=bottom) {
       for (i=right → left) {
            push (a[bottom][i]) } }
      bottom--;
    }
    if( left <=right) {
       for (i= bottom → top) {
          push(a[i][left])
       }
    }
    left++;
```

20] Count Subarrays Sum equals K :-

arr[ ] = [1, 2, 3, -3, 1, 1, 1, 4, 2, -3] ans = 8

→ Brute :-

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

TC :- $\leq O(n^3)$

SC :- O(1)

→ Better :-

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

1 → 1
1 2 → 3
1 2 3 → 6
1 2 3 -3 → 3
!
!
so on.

⇒ TC :- $\leq O(n^2)$

SC :- O(1)

optimal :- (prefix Sum + map)

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

(3, 3, 6)

code :-

```
mpp[0]=1;
for(i=0 → n){
    sum = sum+arr[i];
    prefixsum = sum-k;
    count = count + mpp[prefix sum];
    mpp[sum] ++;
}
return count;
```

sum (key)	cnt (value)
9	1
12	1
10	1
5	1
4	1
6	2
3	2
1	1
0	1

cnt = 0 1 2 3 4 5 6

TC :- ≤ O(n x log n)

SC :- ≤ O(n)