**Q1 : Set Matrix Zeroes**

**Approach** : Store the state of each row and column in the first row and column respectively and store the state of first row and column in two separate variables and then set the matrix to zeroes accordingly.

**Time Complexity** : O ( rc ).

**Q2 : Sort 0 1 2**

**Approach** : One approach will be to just sort (O(nlogn)). Another approach will be to do count sort. The complexity of count sort will be O(n) but it is a 2 pass algorithm. Most optimal approach will be to use three pointers (low, mid and high). Traverse and swap the array accordingly such that from [0 to low-1] there should only be 1 present and from [high-1 to n-1]only 2 should be present.Iterate until mid <= high.

**Time Complexity** : O ( n ).

**Q3 : Max Sum Subarray (Kadane’s Algo)**

**Approach** : Iterate the array and keep track of the running sum, as soon as the sum becomes negative make it to 0 and update the max sum after every iteration.

**Time Complexity** : O ( n )

**Q4 : Pascal’s Triangle**

**Approach** : The first row is always {1}. To fill the next row using the current row just push one extra 1 to the next row and then push the sum of the current element and the next element of the current row(Iterate in the current row). At the end push one more extra 1 to the vector.

**Time Complexity** : O ( n^2 )

**Q5 : Stock Buy and Sell**

**Approach** : Simply traverse the array simultaneously calculating min until the current index. Subtract then min from the current element and take the maximum of all the result.

**Time Complexity** : O ( n )

**Q6 : Rotate Matrix (90 deg)**

**Approach** : The matrix will be square unless extra space is allowed. Simply take the transpose of the matrix and reverse each row in-place.

*Taking transpose of a matrix.*

If it is a rectangular matrix then we will have to take extra space.

For Square matrix =>

For(int i=0;i<r;i++)

{

for(int j=0;j<I;j++)

{

Swap(m[I][j],m[j][i]);

}

}

**Time Complexity** : O(r\*c)

**Q7 Rotate Matrix( Clockwise by 1 element )**

**Approach** : Simply traverse from the outer boundary using 4 for loops and swap accordingly.

**Time Complexity** : O(r\*c) because each element is visited only once.

**Q8 Merge Intervals**

**Approach** : First sort the intervals a/c to the start time. Now take the first element and move until the end time is >= start time of the next element simultaneously updating the end time also and then similarly for the next elements.

**Time Complexity** : O(nlogn) for sorting + O(n) for traversal.