***************************************MILESTONE 1**

1- Jewels and Stones

```
class Solution {
public:
   int numJewelsInStones(string jewels, string stones) {
      int count=0;         //Initializing a variable to store the no. of stone that are jewel
      for(int i=0;i<jewels.length();i++)
      {
         for(int j=0;j<stones.length();j++)
         {
            if(jewels[i]==stones[j])    //Comparing the jewels with stones one by one
            {
               count++;
            }
         }
      }
      return count;
       }
};
//Time Complexity-O(N^2)
//Space Complexity-O(1)
```

2-Merge Strings Alternately

```
class Solution {
public:
    string mergeAlternately(string word1, string word2) {
        string res="";            //Initializing a string named 'res'
        int n=word1.size();      //Storing the size of string word1 in a variable n
        int m=word2.size();     //Storing the size of string word2 in a variable m
        int ni=0,mi=0,x;          //Intializing two variable with 0 ,i.e, 'ni' & 'mi' will keep the count of word1 and word2 string
        if(n>m) x=m;             //The smaller of size of the strings will be considered for the loop
        else x =n;
        for(int i=0;i<2*x;i++)
        {
            if(i%2==0)             //At even positions of string 'res' characters of string word1 will occur
            {
                res+=word1[ni];
                ni++;
            }
            else{         //At odd positions of string 'res' characters of string word2 will occur
                res+=word2[mi];
                mi++;
            }
        }
        while(ni<n)             //String whose elements will be left out are to be added in the end
        {
            res+=word1[ni];
            ni++;
        }
        while(mi<m)
        {
            res+=word2[mi];
            mi++;
```

```
      }
      return res;
   }
};
//Time Complexity-O(N)
//Space Complexity-O(N)
```

3-Minimum Number of Steps to Make Two Strings Anagram

```
class Solution {
public:
   int minSteps(string s, string t) {
      int count = 0;  //Initialising a variable'count' to store the no. of steps taken to make string 't' an anagram.
      int frequency[26]={0};
      for(int i=0;i<s.length();i++)
      {
         frequency[s[i]-'a']++;
      }
      for(int j=0;j<t.length();j++)
      {
         if((frequency[t[j]-'a']--)<=0)
         {
            count++;
         }
      }
      return count;
   }
};
//Time Complexity-O(N+M) →2 for loops for length of 2 strings respectively
//Space Complexity-O(1)  → To store the frequency we have taken an array of size 26 ,i.e, constant
```

4-Spiral Matrix

```
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {
        if(matrix.size()==0) return {};   //If matrix is empty we need to return it empty
        int n=matrix.size();
        int m=matrix[0].size();
        vector<int> res;
        int dir=0;
        int top=0,down=n-1,left=0,right=m-1;
        while(top<=down && left<=right){
            if(dir==0){               //Condition that will handle the left to right traversal in the matrix
                for(int i=left;i<=right;i++){
                    res.push_back(matrix[top][i]);
                }
                top++;
            }
            else if(dir==1){                  //Condition that will handle the top to down traversal in the matrix
                for(int i=top;i<=down;i++){
                    res.push_back(matrix[i][right]);
                }
                right--;
            }
            else if(dir==2){                  //Condition that will handle the right to left traversal in the matrix
                for(int i=right;i>=left;i--){
                    res.push_back(matrix[down][i]);
                }
                down--;
            }
            else if(dir==3){                  //Condition that will handle the down to top traversal in the matrix
                for(int i=down;i>=top;i--){
                    res.push_back(matrix[i][left]);
                }
                left++;
            }
            dir = (dir+1)%4;    /*It will provide condition to be taken up next coz it moves like 0->1->2->3->0->1 .Similarly
the way matrix is need to be traversed*/
        }
        return res;
    }
};
```

```
//Time Complexity-O(N)
//Space Complexity-O(N)
```

5-Sort Array by Parity

```
class Solution {
public:
    vector<int> sortArrayByParity(vector<int>& nums) {
        int n=nums.size();
        int l=0,r=n-1,temp;
        while(l<r)
        {
            while(l<r && nums[l]%2==0)   //Regulating the left pointer for even elements
            {
                l++;
            }
            while(l<r && nums[r]%2!=0)   // Regulating the left pointer for odd elements
            {
                r--;
            }
            temp = nums[l];     //Swapping the right and left elements according to the even and odd element
            nums[l]=nums[r];
            nums[r]=temp;
        }
        return nums;
    }
};
//Time Complexity-O(N^2)
//Space Complexity-O(1)
```

---

6. Best Time to Buy and Sell Stock

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int n=prices.size();
        int min=prices[0],maxProfit=0;
        for(int i=0;i<n;i++)
        {
            if(min>prices[i])               //Calculating the min price from the left
            {
                min=prices[i];
            }
            if(maxProfit<prices[i]-min)       /*For each price calculating the maximum profit and then
comparing it with the previous max calculated*/
```

```
         {
            maxProfit=prices[i]-min;
         }
      }
      return maxProfit;
   }
};
//Time Complexity-O(N)
//Space Complexity-O(1)
```

7. Best Time to Buy and Sell Stock II

```
class Solution {
public:
   int maxProfit(vector<int>& prices) {
     int n=prices.size();
     int maxProfit=0;
      for(int i=0;i<n-1;i++)
      {
         if(prices[i]<prices[i+1]) /*Comparing the immediate next values so as to compare the pairs with
min-max  values*/
         {
            maxProfit += prices[i+1]-prices[i];
         }
      }
      return maxProfit;
   }
};
//Time Complexity-O(N)
//Space Complexity-O(1)
```