ANANYA SINGH- MILESTONE 1

Q1. https://leetcode.com/problems/jewels-and-stones/

```
class Solution {
    public int numJewelsInStones(String jewels, String stones) {
        int stoneslen= stones.length(), jewelslen= jewels.length(), count=0;
        for(int i=0; i<stoneslen; i++)
        {
            for(int j=0; j<jewelslen; j++)
            {
                if(stones.charAt(i)==jewels.charAt(j))
                    count++;
            }
        }
        return count;
    }
}
```

Q2. https://leetcode.com/problems/merge-strings-alternately/

```
class Solution {
    public String mergeAlternately(String word1, String word2) {
       String merged=""; String longstr="";
        int word1len= word1.length(), word2len= word2.length();
        int minlen= Math.min(word1len, word2len);

        if(word1len<word2len)
            longstr=word2;
        else if(word2len<word1len)
            longstr=word1;
         for(int i=0; i<minlen; i++)
            merged+=""+word1.charAt(i)+word2.charAt(i);
         if(word1len!=word2len)
        {
            for(int j=minlen; j<longstr.length(); j++)
                merged+=""+longstr.charAt(j);
        }
        return merged;
    }
}
```

Q3. https://leetcode.com/problems/minimum-number-of-steps-to-make-two-strings-anagram/

```
class Solution {
    public int minSteps(String s, String t) {
        int index=0, steps=0;
        List<Character> list=new ArrayList<Character>();

        for (int i=0; i<s.length(); i++) {
            list.add(s.charAt(i));
        }
        for(int i=0; i<t.length(); i++)
        {
            index = list.indexOf(t.charAt(i));
          if(index!=-1)
              list.set(index, ' ');
          else
              steps++;
        }
        return steps;
    }
}
```

Q4. https://leetcode.com/problems/spiral-matrix/

```
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> list=new ArrayList<Integer>();
        int firstRow=0, firstCol=0, lastRow=matrix.length-1,
lastCol=matrix[0].length-1;
        int count=(matrix.length)*(matrix[0].length);

        while(firstRow<=lastRow && firstCol<=lastCol)
        {
            if(list.size()==count)
                break;

            if(list.size()<count)
            {
             for(int i=firstCol; i<=lastCol; i++)
               {    list.add(matrix[firstRow][i]);  }
                firstRow++;

               for(int i=firstRow; i<=lastRow; i++)
              {    list.add(matrix[i][lastCol]);  }
```

```
            lastCol--;

            if(firstRow<=lastRow)
            {
               for(int i=lastCol; i>=firstCol; i--)
                {   list.add(matrix[lastRow][i]); }
               lastRow--;
            }

            if(firstCol<=lastCol)
            {
               for(int i=lastRow; i>=firstRow; i--)
                {   list.add(matrix[i][firstCol]);  }
               firstCol++;
            }
          }

       }
       return list;
    }
}
```

Q5. https://leetcode.com/problems/sort-array-by-parity/

```
class Solution {
    public int[] sortArrayByParity(int[] nums) {
        int[] arr= new int[nums.length];
        int j=0, len=((nums.length)-1);
        for(int i=0; i<nums.length; i++)
        {
            if(nums[i]%2==0)
            {
              arr[j]=nums[i];
              j++;
            }
            else
            {
               arr[len]=nums[i];
               len--;
            }
        }
        return arr;
    }
}
```

Q6. https://leetcode.com/problems/best-time-to-buy-and-sell-stock/

```
class Solution {
    public int maxProfit(int[] prices) {
        int minbuy=prices[0], maxprofit=0;

        for(int i=1; i<prices.length; i++)
        {
            if(prices[i]>minbuy)
            {
                if((prices[i]-minbuy)>maxprofit)
                    maxprofit=prices[i]-minbuy;
            }

            else if(prices[i]<minbuy)
                minbuy=prices[i];
        }

        return maxprofit;
    }
}
```

Q7. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii

```
class Solution {
    public int maxProfit(int[] prices)
    {
        int i=0, maxprofit=0;
        while(i<prices.length-1)
        {
            if(prices[i+1]>prices[i])
            {
                maxprofit+=(prices[i+1]-prices[i]);
            }
            i++;
        }
        return maxprofit;
    }
}
```