DSA - Problems And Solution.

Array

- Easy:

Problem: 1

Count Negative Sum of SubArray

Code:

```
    public class Solution {
        public static void main(String[] args) {
          Scanner sc = new Scanner(System.in);
          int n = sc.nextInt();
          int a[] = new int[n];
          for(int i = 0; i < n;i++)
          {
              a[i] = sc.nextInt();
          }
          int negcnt = 0, sum = 0;

          for(int i = 0; i < n;i++)
          {
              sum = 0;
              for(int j = i; j < n;j++)
              {
                    sum += a[j];
                    if(sum < 0)
                       negcnt++;
              }
          }

          System.out.println(negcnt);
        }
    }
```

Time complexity:
Space Complexity:

Problem: 2

Find Highest Sum of SubArray

Code:

```
public class Algorithm
{
	static int findMaxSumSubArray(int a[], int size)
	{
		int arraySum = 0;
		int maxSum = Integer.MIN_VALUE;

		for(int i = 0; i < size; i++)
		{
			arraySum += a[i];

			if(arraySum > maxSum)
				maxSum = arraySum;

			if(arraySum < 0)
				arraySum = 0;
		}
		return maxSum;
	}
	public static void main(String args[])
	{
		int totalElements;
		Scanner sc = new Scanner(System.in);
		totalElements = sc.nextInt();
		int array[] = new int[totalElements];

		for(int i = 0; i < totalElements; i++)
		{
			array[i] = sc.nextInt();
		}
		int sum = findMaxSumSubArray(array, array.length);

		System.out.println("Highest Sum of Subarray is: " + sum);

	}
}
```

Time complexity:
Space Complexity:

Problem: 3

SubArraySum eq to user-entered sum.

Code:

```
 int isSubArrayFound(int a[], int size, int sum)
 {
         int subArraySum = 0,start = 0;

 for(int i = 0; i <= size; i++)
 {
         while(subArraySum > sum && start < i)
         {
                 subArraySum -= a[start]; start++;
         }

 if(subArraySum == sum)
 {
                 System.out.println("SubArray Found Index: " + start
                         + " to " + (i - 1)); return 1;
 }

 subArraySum += a[i];

 }
 return subArraySum;
}
```

Time complexity:
Space Complexity:

Problem: 4

Build Array from Permutation

Code:

```
class Solution{
	public int[] buildArray(int[] nums)
	{
		int ans[] = new int[nums.length];

		for(int i = 0; i < nums.length; i++)
		{
			ans[i] = nums[nums[i]];
		}

		return ans;
	}
}
```

Time complexity: c1 + n = O(N)
Space Complexity: O(N)

Problem: 5

Concatenation of Array.

Code:

```
class Solution {
  public int[] getConcatenation(int[] nums) {
        int ans[] = new int[nums.length * 2];
        int n = nums.length;

        for(int i = 0; i < n; i++)
        {
                ans[i] = nums[i];
                ans[i+n] = nums[i];
        }
        return ans;
  }
}
```

Time complexity: c1 + n * c2 * c3 = O(N)
Space Complexity: N^2

Problem: 6

Running Sum of 1d Array

Code:

```
class Solution {
        public int[] runningSum(int[] nums)
        {
                for(int i = 1 ; i < nums.length; i++)
                        nums[i] = nums[i] + nums[i - 1];

                return nums;
        }
      }
```

Time complexity: O(N)
Space Complexity: O(1)

Problem: 7

Shuffle the Array.

Code:

```
class Solution
{
  public int[] shuffle(int[] nums, int n)
  {
        for(int i = 1; i < nums.length; i+=2)
        {
              int j = n;
              while(j > i)
              {
                    int temp = nums[j];
                    nums[j] = nums[j - 1];
                    nums[j - 1] = temp;
                    j–;
              }
              n++;
        }

        return nums;
  }
}
```

Time complexity: (n/ 2) * (m = n /2 -1) = O(N * M) (here N/2 = N)
Space Complexity: O(1)

Problem: 8

Kids With the Greatest Number of Candies

Code:

```
class Solution {
  public List<Boolean> kidsWithCandies(int[] candies, int extraCandies)
  {
        List<Boolean> result = new ArrayList<Boolean>(candies.length);
        int highCandy = 0;
        for(int i = 0 ; i < candies.length; i++)
        {
                highCandy = Math.max(highCandy,candies[i]);
        }

        for(int i = 0 ; i < candies.length; i++)
        {
                if(candies[i] + extraCandies >= highCandy)
                        result.add(true);
                else
                        result.add(false);
        }

        return result;
  }
}
```

Time complexity: n * c1 + n * c2 * c3 = O(N)
Space Complexity: O(N)

Problem: 9

Number of Good Pairs

Code:

```
class Solution {
  public int numIdenticalPairs(int[] nums)
  {
          HashMap<Integer,Integer> result = new HashMap<Integer,Integer>();
          int goodPair = 0;
          for(int i = 0 ; i < nums.length; i++)
          {
                  int totalFrd = result.getOrDefault(nums[i],0);
                  goodPair += totalFrd;
                  result.put(nums[i], totalFrd+1);
          }
          return goodPair;
  }
}
```

Time complexity: n * c1 * c2 * c3 = O(N)
Space Complexity: O(N)

Problem: 10

How Many Numbers Are Smaller Than the Current Number

Code:

```
class Solution {
        public int[] smallerNumbersThanCurrent(int[] nums)
        {
                int result[] = new int[nums.length];
                for(int i = 0; i < nums.length; i++)
                {
                        int cnt = 0;
                        for(int j = 0; j < nums.length;j++)
                        {
                                if(nums[j] < nums[i])
                                        cnt++;
                        }
                        result[i] = cnt;
                }
                return result;
        }
}
```

Time complexity: n * c1 * n * c2 * c3 = O(N^2)
Space Complexity: O(N)

Code:

```
class Solution {
    public int[] smallerNumbersThanCurrent(int[] nums) {
        int[] count = new int[101];
        int[] res = new int[nums.length];

        for (int i =0; i < nums.length; i++) {
            count[nums[i]]++;
        }

        for (int i = 1 ; i <= 100; i++) {
            count[i] += count[i-1];
        }

        for (int i = 0; i < nums.length; i++) {
            if (nums[i] == 0)
                res[i] = 0;
            else
                res[i] = count[nums[i] - 1];
        }

        return res;
    }
}
```

Time complexity: n * c1 + 100 * c2 + n * c3 = O(N)
Space Complexity: O(N)

Problem: 11

Create Target Array in the Given Order

Code:

```
class Solution {
        public int[] createTargetArray(int[] nums, int[] index)
        {
                int target[] = new int[nums.length];
                HashMap<Integer,Integer> map = new HashMap<Integer,Integer>();


                for(int i = 0; i < nums.length; i++)
                {
                        if(map.containsKey(index[i]) ||  target[index[i]] != 0)
                        {
                                int temp = i;
                                while(temp > index[i])
                                {
                                        target[temp] = target[temp - 1];
                                        temp--;
                                }
                                target[index[i]] = nums[i];
                        }
                        else
                        {
                                target[index[i]] = nums[i];
                                map.put(index[i],index[i]);

                        }
                }
                return target;
        }
}
```

Time complexity: n * m(while loop) = O(n * m)
Space Complexity: O(N)

Problem: 12

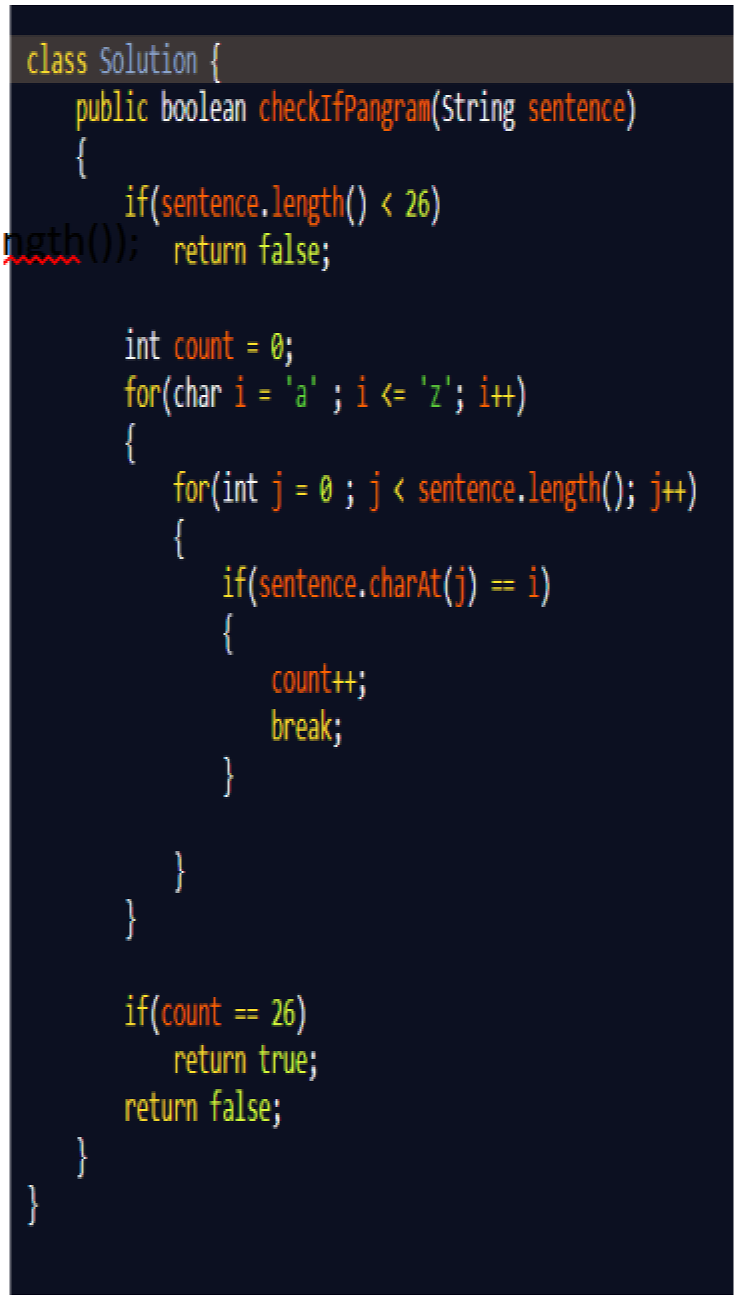
Check if the Sentence Is Pangram

Code:

```
      class Solution {
        public boolean checkIfPangram(String sentence)
        {
                List<Character> alphabets = new
              ArrayList<Character>(sentence.length());

                if(sentence.length() < 26)
                        return false;

                for(int i = 0; i < sentence.length(); i++)
                {
                      if(!alphabets.contains(sentence.charAt(i)))
                                alphabets.add(sentence.charAt(i));
                 }

                if(alphabets.size() == 26)
                        return true;

                return false;
        }
      }
```

Time complexity:O(N)
Space Complexity:O(N)

Code:



ngth());

```java
class Solution {
    public boolean checkIfPangram(String sentence)
    {
        if(sentence.length() < 26)
            return false;

        int count = 0;
        for(char i = 'a' ; i <= 'z'; i++)
        {
            for(int j = 0 ; j < sentence.length(); j++)
            {
                if(sentence.charAt(j) == i)
                {
                    count++;
                    break;
                }

            }
        }

        if(count == 26)
            return true;
        return false;
    }
}
```

Time complexity:26 * N = O(N)
Space Complexity:O(1)

Problem: 13

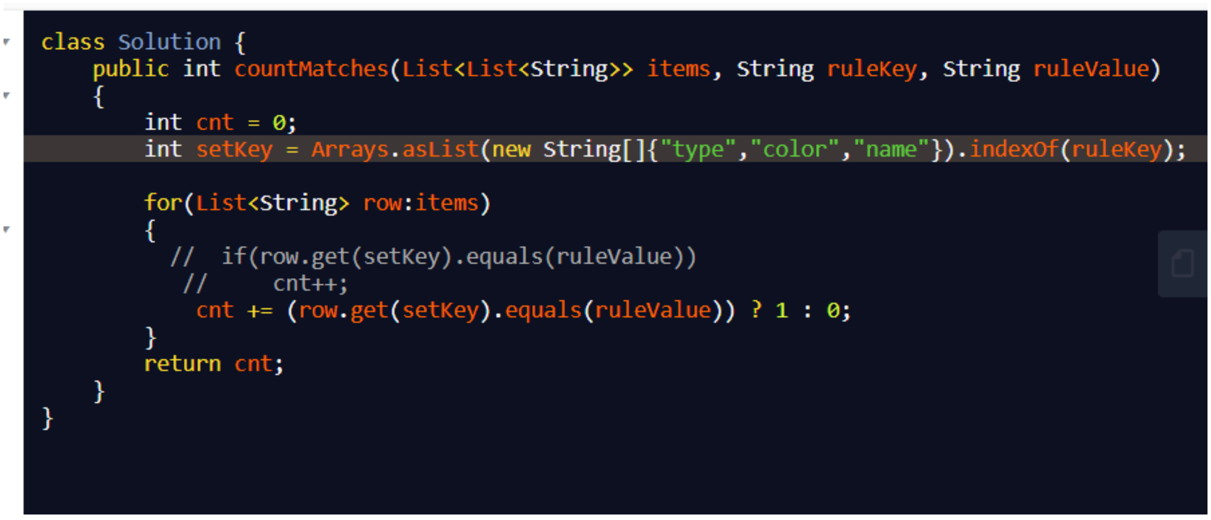
Count Items Matching a Rule

Code:

```
class Solution {
        public int countMatches(List<List<String>> items, String ruleKey, String
       ruleValue)
        {
                int cnt = 0, setKey = 0;

                if(ruleKey.equals("color"))
                        setKey = 1;
                if(ruleKey.equals("name"))
                        setKey = 2;
                for(List<String> row:items)
                {
                         if(row.get(setKey).equals(ruleValue))
                                 cnt++;
                }
                return cnt;
        }
}
```

Time complexity: O(N)
Space Complexity:O(1)

Code:



```
class Solution {
    public int countMatches(List<List<String>> items, String ruleKey, String ruleValue)
    {
        int cnt = 0;
        int setKey = Arrays.asList(new String[]{"type","color","name"}).indexOf(ruleKey);

        for(List<String> row:items)
        {
          //  if(row.get(setKey).equals(ruleValue))
           //      cnt++;
            cnt += (row.get(setKey).equals(ruleValue)) ? 1 : 0;
        }
        return cnt;
    }
}
```

Time complexity: O(N)
Space Complexity:O(N)

Problem: 14

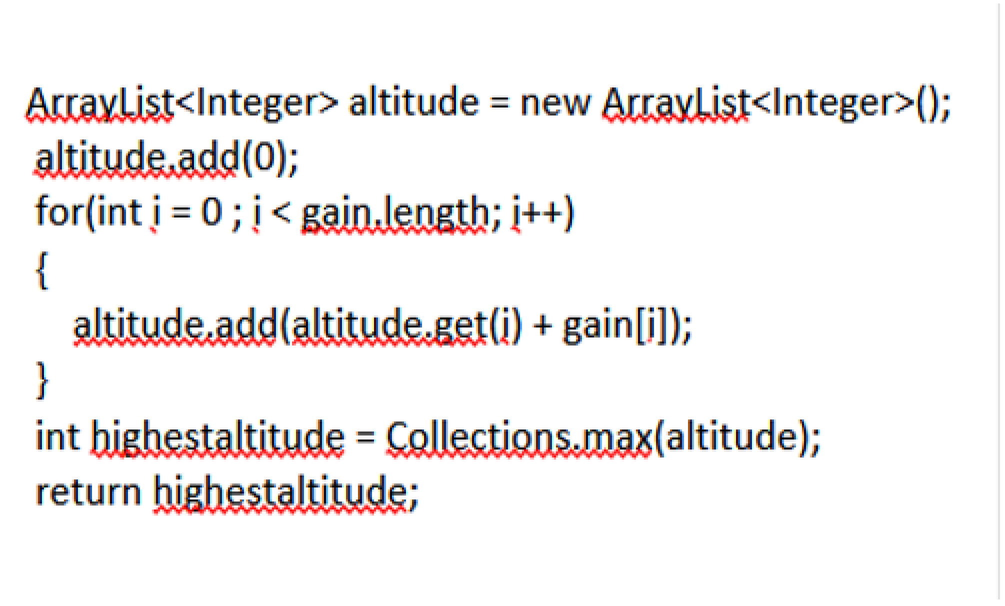
Find the Highest Altitude

Code:

```
class Solution {
        public int largestAltitude(int[] gain)
        {
                int highestaltitude = 0,sum = 0;

                for(int i = 0 ; i < gain.length; i++)
                {
                        sum += gain[i];
                        highestaltitude = Math.max(sum,highestaltitude);
                }

                return highestaltitude;
        }
}
```

Time complexity: O(N)
Space Complexity:O(1)

Code:



```
ArrayList<Integer> altitude = new ArrayList<Integer>();
 altitude.add(0);
 for(int i = 0 ; i < gain.length; i++)
 {
    altitude.add(altitude.get(i) + gain[i]);
 }
 int highestaltitude = Collections.max(altitude);
 return highestaltitude;
```

Time complexity:O(N)
Space Complexity:O(N)

Problem: 15

Cells with Odd Values in a Matrix

Code:

```
class Solution {
        static public int oddCells(int m, int n, int[][] indices)
        {
                int row[] = new int[m];
                int col[] = new int[n];

                for(int[] indice:indices)
                {
                        row[indice[0]]++;
                        Col[indice[1]]++;
                }

                int oddCount = 0;
                for(int i = 0 ; i < m; i++)
                {
                        for(int j = 0 ; j < n; j++)
                        {
                                if((row[i] + col[j]) % 2 != 0)
                                        oddCount++;
                        }
                }
        return oddCount;
        }
}
```

Time complexity: O(M * N)
Space Complexity:O(M * N)

Problem: 16

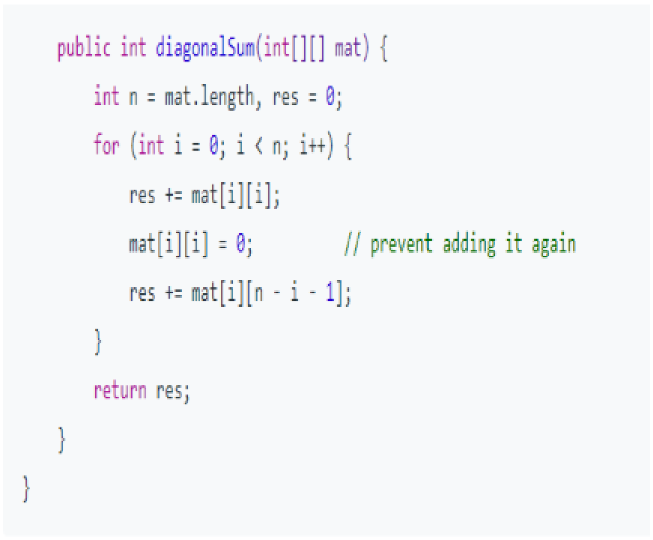
Matrix Diagonal Sum

Code:

```
class Solution {
        public int diagonalSum(int[][] mat)
        {
                int diagonalSum = 0;
                int n = mat.length;
                for(int i = 0; i < mat.length; i++)
                {
                        for(int j = 0 ; j < mat[i].length; j++)
                        {
                                diagonalSum += mat[i][j];
                                diagonalSum += mat[i][n - i - 1];
                        }
                }

                diagonalSum += (mat.length % 2 != 0) ? -mat[n/2][n/2] : 0;
                return diagonalSum;
        }
}
```

Time complexity:O(N)
Space Complexity:O(1)

Code:



```
    public int diagonalSum(int[][] mat) {
        int n = mat.length, res = 0;
        for (int i = 0; i < n; i++) {
            res += mat[i][i];
            mat[i][i] = 0;          // prevent adding it again
            res += mat[i][n - i - 1];
        }
        return res;
    }
}
```

Time complexity:O(N)
Space Complexity:O(1)

Problem: 17

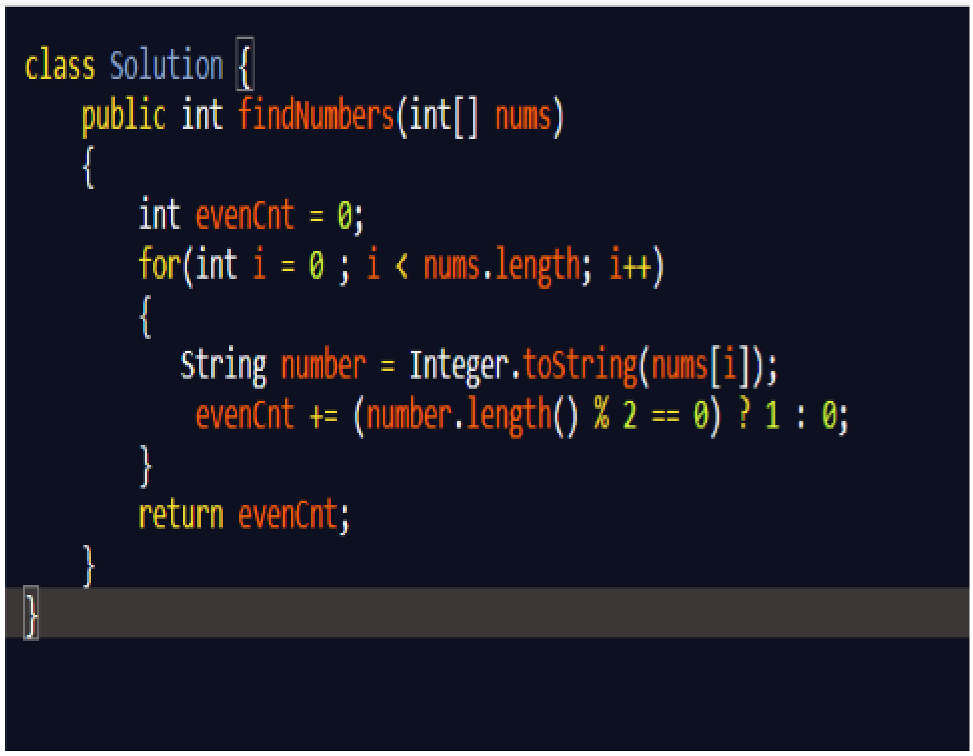
Find Numbers with Even Number of Digits

Code:

```
class Solution {
        public int findNumbers(int[] nums)
        {
                int evenCnt = 0;
                for(int i = 0 ; i < nums.length; i++)
                {
                        int cnt = 0;
                        while(nums[i] > 0)
                        {
                                cnt++;
                                nums[i] /= 10;
                        }
                        evenCnt += (cnt % 2 == 0) ? 1 : 0;
                }
                return evenCnt;
        }
}
```

Time complexity: O(N * M)
Space Complexity:O(1)

Code:



```
class Solution {
    public int findNumbers(int[] nums)
    {
        int evenCnt = 0;
        for(int i = 0 ; i < nums.length; i++)
        {
           String number = Integer.toString(nums[i]);
            evenCnt += (number.length() % 2 == 0) ? 1 : 0;
        }
        return evenCnt;
    }
}
```

Time complexity:
Space Complexity:

Problem: 18

Transpose Matrix

Code:

```
class Solution {
  public int[][] transpose(int[][] matrix)
  {
        int transpose[][]  = new int[matrix[0].length][matrix.length];

        for(int i = 0; i < transpose.length; i++)
        {
              for(int j = 0; j < transpose[i].length; j++)
              {
                    transpose[i][j] = matrix[j][i];
              }
        }
        return transpose;
  }
}
```

Time complexity: O(N * M)
Space Complexity:O(N * M)

Problem: 19

Flipping Image

Code:

```
class Solution {
  public int[][] flipAndInvertImage(int[][] image)
  {
        for(int i= 0; i < image.length;i++)
        {
              for(int j = image[i].length - 1,k = 0; j >= 0; j--)
              {
                    if(j > k)
                    {
                          int temp = image[i][j];
                          image[i][j] = image[i][k];
                          image[i][k] = temp;
                          k++;
                    }

                    if(image[i][j]==1)
                    {
                          image[i][j] = 0;
                    }
                    else
                    {
                          image[i][j] = 1;
                    }
              }
        }
        return image;
  }
}
```

Time complexity:O(M * N)
Space Complexity:O(M * N)

Problem: 20

Add to Array-Form of Integer

Code:

```
class Solution {
        public List<Integer> addToArrayForm(int[] num, int k)
        {
                String number = "";
                int n = num.length;
                ArrayList<Integer> numbers = new ArrayList<>(n);
                for(int i = num.length - 1 ; i >= 0; i--)
                {
                        numbers.add((num[i] + k) % 10);
                        k = (num[i] + k) / 10;
                }
                while (k>0)
                {
                        numbers.add(k % 10);
                        k = k /10;
                }
                Collections.reverse(numbers);
                return numbers;
        }
}
```

Time complexity:O(N)
Space Complexity:O(N)

Problem: 21

Maximum Population Year

Code:

```
class Solution {
        public int maximumPopulation(int[][] logs) {
        int[] year=new int[101];
        for(int[] log:logs){
                year[log[0]-1950]++;
                Year[log[1]-1950]--;
        }
        int maxNum=year[0],maxYear=1950;
        for(int i=1;i<year.length;i++){
                year[i] += year[i-1];
                if(year[i]>maxNum){
                        maxNum=year[i];
                        maxYear=i+1950;
                }
        }
        return maxYear;
        }
}
```

Time complexity:O(N)
Space Complexity:O(1)

Problem: 22

Lucky Numbers in a Matrix

Code:

```
class Solution {

static public List<Integer> luckyNumbers (int[][] matrix)
        {
                List<Integer> arrlist = new ArrayList<Integer>();
                int minElementArray[] = new int[matrix.length];
                int maxElementArray[] = new int[matrix[0].length];
                int min = 0,max = 0;
                for(int i = 0 ; i < matrix.length; i++)
                {
                          min = matrix[i][0];
                        for(int j = 0 ; j < matrix[i].length; j++)
                        {
                                min = Math.min(min, matrix[i][j]);
                        }

                        minElementArray[i] = min;
                 }


                for(int i = 0 ; i < matrix[0].length; i++)
                {
                          max = matrix[0][i];
                        for(int j = 0 ; j < matrix.length; j++)
                        {
                                max = Math.max(max, matrix[j][i]);
                        }

                        maxElementArray[i] = max;
                }


                for(int i = 0 ; i < minElementArray.length; i++)
                {
                        for(int j = 0 ; j < maxElementArray.length; j++)
                        {
                                if(maxElementArray[j] == minElementArray[i])
                                        arrlist.add(minElementArray[i]);
                        }

                }
                return arrlist;
        }
}
```

Time complexity: M * N + M * N + P = O(M * N)
Space Complexity: O(M * N)

Problem: 23

Maximum Subarray

Code:

```
class Solution {
        public int maxSubArray(int[] nums)
        {
        int arraySum = 0, maxSum = Integer.MIN_VALUE;
        for(int i = 0; i < nums.length; i++)
        {
        arraySum += nums[i];

        maxSum = Math.max(arraySum, maxSum);

                  if(arraySum < 0)
                        arraySum = 0;
        }
        return maxSum;
        }
}
```

Time complexity:O(N)
Space Complexity:O(1)

Problem: 24

Minimum Cost to Move Chips to The Same Position

Code:
```
class Solution {
   public int minCostToMoveChips(int[] position) {
      int evenTower = 0;
      int oddTower = 0;
      int length = position.length;
      for(int i = 0 ; i < length; i++)
      {
         if(position[i] % 2 == 0)
            evenTower++;
         else
            oddTower++;
      }

      return Math.min(evenTower,oddTower);
   }
}
```

Time complexity:O(N)

Space Complexity:O(1)

- Medium:

Problem: 1

Spiral Matrix

Code:

```
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> list = new ArrayList<Integer>();

        int matLength =  matrix.length;
        int matColLength = matrix[0].length;

        int left = 0, right = matColLength - 1,up = 0, down = matLength - 1;
        int direction = 1;

        while(true)
        {
            for(int i = left; i <= right; i++)
                list.add(matrix[up][i]);

            up++;
            if(left > right || up > down) break;


            for(int i = up; i <= down; i++)
                list.add(matrix[i][right]);

            right--;
           if(left > right || up > down) break;



            for(int i = right; i >= left; i--)
                list.add(matrix[down][i]);


            down--;
            if(left > right || up > down) break;



            for(int i = down; i >= up; i--)
                list.add(matrix[i][left]);
            left++;
             if(left > right || up > down) break;
        }

        return list;

    }
}
```

Time complexity:O(M*N)

Space Complexity:O(M*N)

Problem: 2

Spiral Matrix1

Code:

```
class Solution {
    public int[][] generateMatrix(int n) {
        int result[][] = new int[n][n];
        int left = 0, right = n - 1,up = 0, down = n - 1;
        int cnt = 1;

        while(true)
        {
            for(int i = left; i <= right; i++,cnt++)
                result[left][i] = cnt;

            up++;
            if(left > right || up > down) break;


            for(int i = up; i <= down; i++,cnt++)
                 result[i][right] = cnt;

            right--;
           if(left > right || up > down) break;



            for(int i = right; i >= left; i--,cnt++)
                 result[down][i] = cnt;


            down--;
            if(left > right || up > down) break;



            for(int i = down; i >= up; i--,cnt++)
                 result[i][left] = cnt;
            left++;
             if(left > right || up > down) break;
        }

        return result;
    }
}
```

Time complexity:O(M*N)

Space Complexity:O(M*N)

Problem: 3

Set Matrix Zeroes

Code:

```
class Solution {
   public void setZeroes(int[][] matrix) {
      int m = matrix.length - 1;
      int n = matrix[0].length - 1;
      int k = 0;

      while(k<=n &&matrix[0][k] != 0) k++;

      for(int i = 1; i <=m; i++)
      {
         for(int j = 0; j <= n; j++)
         {
           if(matrix[i][j] == 0)
           {
              matrix[i][0] = 0;
              matrix[0][j] = 0;
           }
         }
      }

      for(int i = 1; i <= m; i++)
      {
         for(int j = n; j >= 0; j--)
         {
           if( matrix[i][0] == 0 || matrix[0][j] == 0)
             matrix[i][j] = 0;
         }
      }

      if(k <= n)
         Arrays.fill(matrix[0],0);
   }
}
```

Time complexity:O(M*N)

Space Complexity:O(1)

Problem: 4

Product of Array Except Self

Code:

```
class Solution {
   public int[] productExceptSelf(int[] nums) {
      int length = nums.length - 1;
      int answer[] = new int[length + 1];
      int mul = 1, index = 0;

      for(int i = 0 ; i <= length; i++)
      {
         mul *= nums[i];
         if(mul == 0)
         {
            index = i;
            break;
         }
      }

      if(mul != 0)
      {
         for(int i = 0 ; i <= length; i++)
            answer[i] = mul / nums[i];

         return answer;
      }
      mul = 1;
      for(int k = 0; k <= length; k++)
      {
         if(k == index) continue;
                  mul *= nums[k];
      }
      answer[index] = mul;
      return answer;
   }
}
```

Time complexity:O(N)

Space Complexity:O(N)

Code:

```
public int[] productExceptSelf(int[] nums) {

    int len = nums.length;
    int [] output = new int[len];

    int leftMult = 1, rightMult = 1;

    for(int i = len-1; i >= 0; i--){
        output[i] = rightMult;
        rightMult *= nums[i];
    }
    for(int j = 0; j < len; j++){
        output[j] *= leftMult;
        leftMult *= nums[j];

    }

    return output;

}
```

Time complexity:O(N)

Space Complexity:O(N)

Problem: 5

Sort Colors

Code:

```
class Solution {
    public void sortColors(int[] nums) {
        int length = nums.length;
        int start = 0;
        int end = length - 1;
        int index = 0;
       while(index <= end)
       {
           if(nums[index] == 0)
           {
               int temp = nums[start];
               nums[start] = nums[index];
               nums[index] = temp;

               start++;
           }
           if(nums[index] == 2)
           {
               int temp = nums[end];
               nums[end] = nums[index];
               nums[index] = temp;

               index--;
               end--;
           }
          index++;
       }
    }
}
```

Time complexity:O(N)

Space Complexity:O(N)

Code:(DUTCH NATIONAL FALG ALGORITHM)

```
class Solution {
    public void sortColors(int[] nums)
    {
        int low=0;
        int high=nums.length-1;
        int mid=0;
        while(mid<=high)
        {
            switch(nums[mid])
            {
                case 0:
                    {
                        int temp=nums[low];
                        nums[low]=nums[mid];
                        nums[mid]=temp;
                        low++;
                        mid++;
                        break;
                    }
                case 1:
                    {
                        mid++;
                        break;
                    }
                case 2:
                    {
                        int temp=nums[mid];
                        nums[mid]=nums[high];
                        nums[high]=temp;
                        high--;
                        break;
                    }
            }
        }

    }
}
```

Time complexity:O(N)

Space Complexity:O(N)

Problem: 6

Rotate Array

Code:

```
class Solution {
   public void rotate(int[] nums, int k) {
         int length = nums.length - 1;
         k %= nums.length;
         reverse(nums, 0 , length);
         reverse(nums, 0, k - 1);
         reverse(nums, k, length);
   }

   public static void reverse(int[] nums, int start,int end)
   {
      while(start < end)
      {
         int temp = nums[start];
         nums[start] = nums[end];
         nums[end] = temp;

         start++;
         end--;
      }
   }
}
```

Time complexity:O(N)

Space Complexity:O(1)

Problem: 7

Jump Game

Code:

```
class Solution {
    public boolean canJump(int[] nums) {
        int length = nums.length;
        int reachable = 0;
        for(int i = 0; i < length; i++)
        {
          if(i > reachable)
            return false;
         reachable = Math.max(reachable, i + nums[i]);
        }

        return true;
    }
}
```

Time complexity:O(N)

Space Complexity:O(1)

Problem: 8

Find First and Last Position of Element in Sorted Array

Code:

```
class Solution {
   public int[] searchRange(int[] nums, int target) {
      int array[] = new int[] {-1,-1};
      int length = nums.length;
      int start = 0, end = length - 1;

      while(start <= end)
      {
         int mid = (start + end) / 2;
         int num = nums[mid];

         if(num == target)
         {
                  int temp = mid;
                  while(temp >= 0 && nums[temp] ==  num)
                           temp--;
            array[0] = temp + 1;

            while(mid < length && nums[mid] == num)
                  mid++;
            array[1] = mid - 1;
         }

         if(target < num)
            end = mid - 1;
         else
            start = mid + 1;
      }

      return array;
   }
}
```

Time complexity:O(N)

Space Complexity:O(1)

Code:
Time complexity:
Space Complexity:

String

- Easy

Problem: 1

Defanging an IP Address.

Code:

```
class Solution {

  boolean validIp(String ip)
  {
          String ipParts[] = ip.split(".");

          for(int i = 0; i < ipParts.length ;i++)
          {
          if(Integer.valueOf(ipParts[i]) > 255 || Integer.valueOf(ipParts[i]) < 0)
          return false;
          }
          return true;
  }
 }
```

Time complexity:
Space Complexity:

Code:

```
 public String defangIPaddr(String address)
        {
        // if(validIp(address))
        address = address.replace(".","[.]");
        return address;
        // return null;
        }
```

Time complexity:
Space Complexity:

Problem: 2

Shuffle String.

Code:

```
class Solution {
  public String restoreString(String s, int[] indices)
  {
        int n = indices.length;
          char result[] = new char[n];

          for(int i =0 ;i < indices.length; i++)
          {
          result[indices[i]] = s.charAt(i);
          }
          return String.valueOf(result);
  }
}
```

Time complexity:
Space Complexity:

Problem: 3

Count Items Matching a Rule

Code:

```
class Solution {
        public int countMatches(List<List<String>> items, String ruleKey, String
ruleValue)
        {
        int cnt = 0;
        int setKey = Arrays.asList(new
        String[]{"type","color","name"}).indexOf(ruleKey);

        for(List<String> row:items)
        {
        //  if(row.get(setKey).equals(ruleValue))
        //          cnt++;
        cnt += (row.get(setKey).equals(ruleValue)) ? 1 : 0;
        }
        return cnt;
        }
}
```

Time complexity:
Space Complexity:

Problem: 4

Sorting the Sentence

Code:

```
class Solution {
  public String sortSentence(String s)
  {
          String[] splitString = s.split(" ");
          String result[] = new String[splitString.length];
          StringBuilder temp = new StringBuilder();
          for(int i = 0 ; i < splitString.length; i++)
          {
          int index = (splitString[i].charAt(splitString[i].length() - 1) - '0');
          result[index - 1] = splitString[i].substring(0,splitString[i].length() - 1);
          }
          for(int i = 0 ; i < result.length; i++)
          {
           temp.append(result[i] + " ");
          }
          return temp.substring(0,temp.length() - 1);
  }
}
```

Time complexity:
Space Complexity:

Problem: 5

Check If Two String Arrays are Equivalent

Code:

```
class Solution {
        public boolean arrayStringsAreEqual(String[] word1, String[] word2)
        {
        String w1 = new String();
        String w2 = new String();

        int length = word1.length > word2.length ? word1.length : word2.length;
        for(int i = 0; i < length; i++)
        {
        if(i < word1.length)
                w1 += word1[i];
        if(i < word2.length)
                w2 += word2[i];

        }
        return w1.equals(w2);
        }
}
```

Time complexity:

Space Complexity:

Problem: 6

To Lower Case

Code:

```
class Solution
{
  public String toLowerCase(String s)
  {
          char array[] = s.toCharArray();

          int i = 0;
          while(i < array.length)
          {
          if(Character.isUpperCase(array[i]))
          array[i] = Character.toLowerCase(array[i]);
          i++;
          }
          return String.valueOf(array);
  }
}
```

Time complexity:
Space Complexity:

Code:

```
class Solution
{
        public String toLowerCase(String s)
        {
        char array[] = s.toCharArray();
        int length = array.length;
        for(int i = 0 ; i < length; i++)
        {
        array[i] = (array[i] >= 65 && array[i] <= 90)?((char)(array[i] + 32)):array[i];
        }
        return new String(array);
        }
}
```

Time complexity:
Space Complexity:

Problem: 7

Determine if String Halves Are Alike

Code:

```
class Solution {

        boolean isVowel(char c)
        {
        if(c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' ||
        c == 'A' || c == 'E' || c == 'I' || c == 'O' ||c == 'U' )
        return true;
        return false;
        }
        public boolean halvesAreAlike(String s)
        {
        String half1 = s.substring(0,s.length()/2);
        String half2 = s.substring(s.length()/2,s.length());

        int half1Cnt = 0;
        int half2Cnt = 0;
        for(int i = 0 ; i < half1.length(); i++)
        {
        if(isVowel(half1.charAt(i)))
                half1Cnt++;

        if(isVowel(half2.charAt(i)))
                half2Cnt++;
        }

        return half1Cnt == half2Cnt;
        }
}
```

Time complexity:
Space Complexity:

Code:

```
class Solution {
        public boolean halvesAreAlike(String s)
        {
        String vowels = "AEIOUaeiou";
        int cnt = 0;
        int mid = s.length() / 2;
        for(int i = 0, j = mid; i < mid; i++,j++)
        {
                if(vowels.indexOf(s.charAt(i)) >= 0)
                cnt++;
                if(vowels.indexOf(s.charAt(j)) >= 0)
                cnt--;
        }
        return cnt == 0;
        }
}
```

Time complexity:
Space Complexity:

Problem: 8

Decrypt String from Alphabet to Integer Mapping

Code:

```
class Solution {
  public String freqAlphabets(String s)
  {
          int length = s.length();

          StringBuilder answer = new StringBuilder();

          for(int i = length - 1; i >= 0; i--)
          {
           char ch = s.charAt(i);

           if(ch == '#')
           {
                String temp = String.valueOf(s.charAt(i - 2)) +
                  String.valueOf(s.charAt(i - 1));
                answer.append((char)(Integer.parseInt(temp) + 'a' - 1));

                i-=2;
           }
           else {
                answer.append((char)(s.charAt(i) - '0' + 96));
                  }
          }
          answer.reverse();
          return answer.toString();
  }
}
```

Time complexity:
Space Complexity:

Problem: 9

Number of Strings That Appear as Substrings in Word

Code:

```
class Solution {
  public int numOfStrings(String[] patterns, String word) {
          int subWord = 0;
          for(int i = 0; i < patterns.length; i++)
          {
          subWord += (word.contains(patterns[i]))?1:0;
          }
          return subWord;
  }
}
```

Time complexity:
Space Complexity:

Problem: 10

Robot Return to Origin

Code:

```
    class Solution {
      public boolean judgeCircle(String moves) {
                int verticalMove = 0,horizontalMove = 0;
                for(int i = 0 ; i < moves.length(); i++)
                {
                char ch = moves.charAt(i);

                if(ch == 'D' || ch == 'U')
                verticalMove+=(ch == 'U')?1:-1;
                else
                horizontalMove+=(ch == 'R')?1:-1;
                }
                }
                return (verticalMove == 0) && (horizontalMove==0);
      }
    }
```

Time complexity:
Space Complexity:

Code:

```
class Solution {
        public boolean judgeCircle(String moves) {
        int verticalMove = 0,horizontalMove = 0;
        for(int i = 0 ; i < moves.length(); i++)
        {
        char ch = moves.charAt(i);
                switch (ch) {
                case 'L':
                horizontalMove--;
                break;
                case 'R':
                horizontalMove++;
                break;
                case 'U':
                verticalMove++;
                break;
                case 'D':
                verticalMove--;
                break;
```

```
            }
            }
            return (verticalMove == 0) && (horizontalMove==0);
            }
}
```

Time complexity:
Space Complexity:

Problem: 11

Excel Sheet Column Title

Code:

```
class Solution {
  public String convertToTitle(int columnNumber) {
          StringBuilder result = new StringBuilder();

          while(columnNumber>0){
          columnNumber--;
          result.insert(0, (char)('A' +columnNumber % 26));
          columnNumber /= 26;
          }

          return result.toString();
  }
}
```

Time complexity:
Space Complexity:

Problem: 12

Implement strStr()

Code:

```
class Solution {
        public int strStr(String haystack, String needle) {
        int length = haystack.length();
        int needleLength = needle.length();

        if(needleLength > length)
        return -1;

        if(needleLength == 0 || haystack.compareTo(needle) == 0)
        return 0;

        char haystackArray[] = haystack.toCharArray();
        char needleArray[] = needle.toCharArray();
        int j = 0;

        char nch = needleArray[0];
        for(int i = 0; i <= length - needleLength; i++)
        {

        char ch = haystackArray[i];

        if(ch == nch)
        {
        j = 0;
                while((i + j) < length && j < needleLength && haystackArray[i+j] ==
needleArray[j])
                        j++;

                if(j == needleLength)
                        return i;
        }
        }
        return -1;
        }
}
```

Time complexity:
Space Complexity:

Code:

```
class Solution {
        public int strStr(String haystack, String needle) {
        int length = haystack.length();
        int needleLength = needle.length();

        if(needleLength > length)
        return -1;

        if(needleLength == 0 || haystack.compareTo(needle) == 0)
        return 0;

        char haystackArray[] = haystack.toCharArray();
        char needleArray[] = needle.toCharArray();
        int j = 0;

        char nch = needleArray[0];
        for(int i = 0; i < length; i++)
        {

        char ch = haystackArray[i];

        if(ch == nch)
        {
        j = 0;
                while((i + j) < length && j < needleLength && haystackArray[i+j] ==
needleArray[j])
                        j++;

                if(j == needleLength)
                        return i;
        }
        }
        return -1;
        }
}
```

Time complexity:
Space Complexity:

Code:

```
class Solution {
        public int strStr(String s, String t) {
        int m = s.length(), n = t.length();
        if (n == 0) return 0;
        for (int i = 0, j; i <= m - n; i++) {
        if (s.charAt(i + n - 1) != t.charAt(n - 1)) continue;
        for (j = 0; j < n && s.charAt(i + j) == t.charAt(j); j++);
        if (j == n) return i;
        }
        return -1;
        }
}
```

Time complexity:
Space Complexity:

Problem: 13

Long Pressed Name.

Code:

```
    class Solution {
      public boolean isLongPressedName(String name, String typed) {
            int m = name.length(), n = typed.length();
            int i = 0, j = 0;

            while(i < m && j < n){
            char c1 = name.charAt(i), c2 = typed.charAt(j);
            if(c1 != c2) return false; // we are handling different chars, no!

                    // count of consecutive c1/c2
            int count1 = 0;
            while(i < m && name.charAt(i) == c1){
            count1++;
            i++;
            }

                    // count of consecutive c1/c2
            int count2 = 0;
            while(j < n && typed.charAt(j) == c2){
            count2++;
            j++;
            }

            if(count2 < count1) return false;
            }

              // they both reach the end
            return i == m && j == n;
      }
    }
```

Time complexity:
Space Complexity:

Problem: 14

Valid Palindrome

Code:

```
class Solution {
  public boolean isPalindrome(String s) {
        int start = 0;
        int end = s.length() - 1;
        s = s.toLowerCase();
        while(start < end)
        {
         while(start <= end && !Character.isLetterOrDigit(s.charAt(start)))
              start++;

         while(end > -1 && !Character.isLetterOrDigit(s.charAt(end)))
              end--;

        if(start == s.length() && end == -1)
        return true;

         char ch1 = s.charAt(start);
         char ch2 = s.charAt(end);

        if(ch1 != ch2)
              return false;
        start++;
        end--;
        }
        return true;
  }
}
```

Time complexity:
Space Complexity:

Problem: 15

Longest Common Prefix

Code:

```
class Solution {
        public static  String longestCommonPrefix(String[] strs) {
        String preFix = strs[0];

        for(int i = 0 ; i < strs.length; i++)
        {
        while(strs[i].indexOf(preFix) != 0)
                preFix = preFix.substring(0, preFix.length() - 1);

        if(preFix.length() == 0)
                break;
        }
        return preFix;
        }
}
```

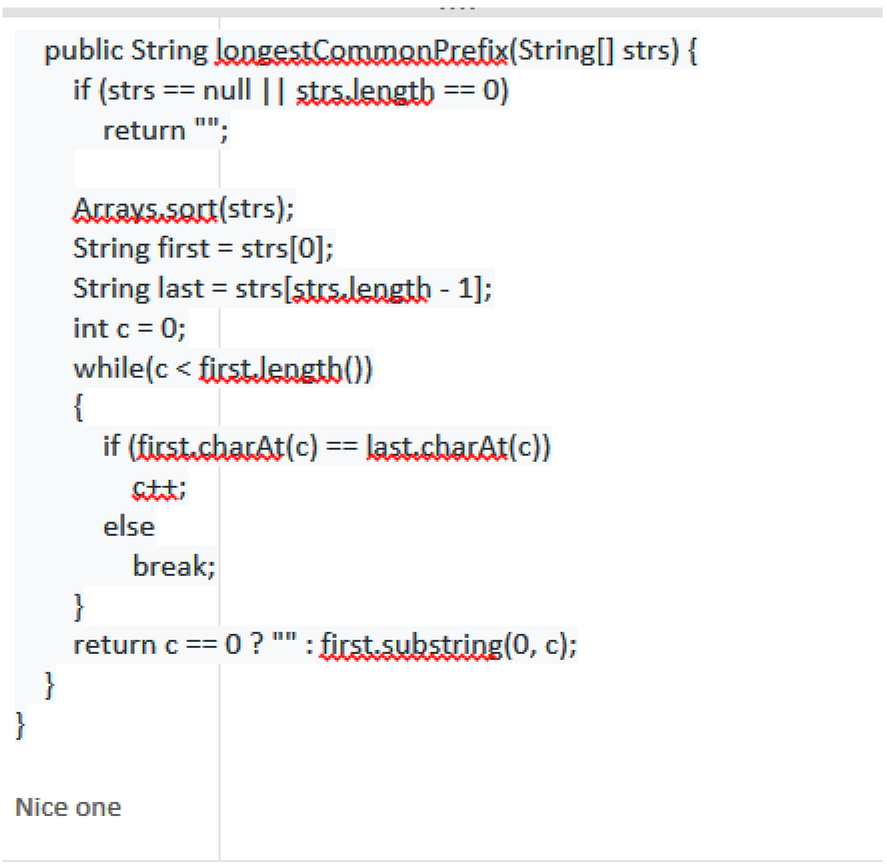
Time complexity:
Space Complexity:

Code:

```
class Solution {
      public static int findMinLength(String s[])
      {
                int minLength = Integer.MAX_VALUE;
                for ( int i=0; i<s.length; i++){
                 int length = s[i].length();
                if(length <= minLength){
                minLength = length;
                }
                }
                return minLength;
      }

        public static  String longestCommonPrefix(String[] strs) {
        int minLength = findMinLength(strs);
        int length = strs.length;
        String preFix = "";
        for(int i = 0 ; i < minLength; i++)
        {
            int j = 0;
            char ch = strs[j].charAt(i);
            j++;
            while(j < length && strs[j].charAt(i) == ch)
            {
                  j++;
            }

            if(j == strs.length)
                  preFix += ch;
        }

        return preFix;
        }
```

Time complexity:
Space Complexity:

Code:



```
    public String longestCommonPrefix(String[] strs) {
        if (strs == null || strs.length == 0)
            return "";

        Arrays.sort(strs);
        String first = strs[0];
        String last = strs[strs.length - 1];
        int c = 0;
        while(c < first.length())
        {
            if (first.charAt(c) == last.charAt(c))
                c++;
            else
                break;
        }
        return c == 0 ? "" : first.substring(0, c);
    }
}
```

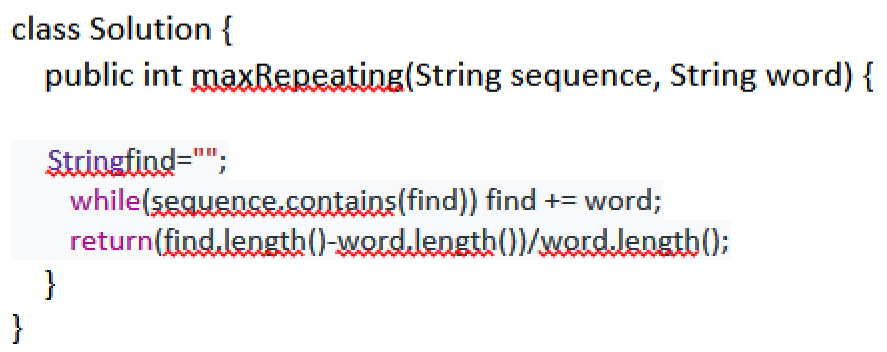
Nice one

Time complexity:
Space Complexity:

Problem: 16

Maximum Repeating Substring.

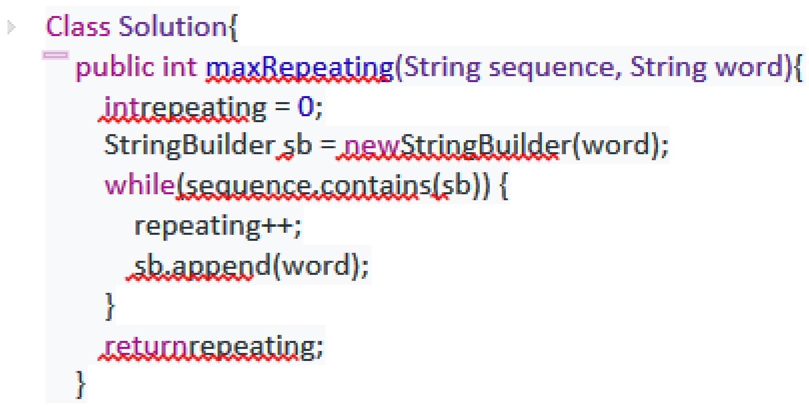
Code:



```
class Solution {
   public int maxRepeating(String sequence, String word) {

    Stringfind="";
      while(sequence.contains(find)) find += word;
      return(find.length()-word.length())/word.length();
   }
}
```

Time complexity:
Space Complexity:

Code:



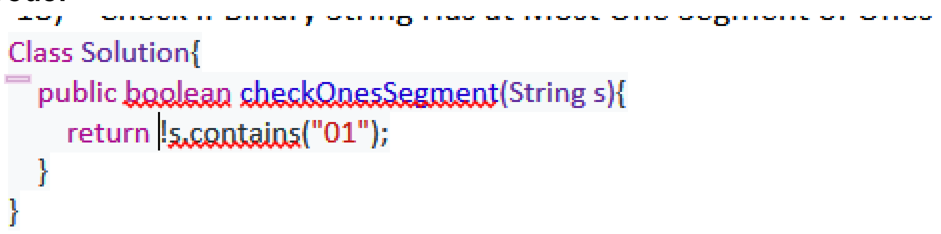
```
Class Solution{
  public int maxRepeating(String sequence, String word){
     intrepeating = 0;
     StringBuilder sb = newStringBuilder(word);
     while(sequence.contains(sb)) {
        repeating++;
        sb.append(word);
     }
     returnrepeating;
  }
```

Time complexity:
Space Complexity:

Problem: 17

Check if Binary String Has at Most One Segment of Ones

Code:



```
Class Solution{
    public boolean checkOnesSegment(String s){
        return !s.contains("01");
    }
}
```

Time complexity:

Space Complexity:

Problem: 18

Longest Common Prefix

Code:
```
class Solution {
	public String mergeAlternately(String word1, String word2) {
	int word1Length = word1.length();
	int word2Length = word2.length();
	StringBuilder string = new StringBuilder();
	int i;
	for(i = 0; i < word1Length; i++)
	{
	string.append(word1.charAt(i));
	if(i < word2Length) string.append(word2.charAt(i));
	}

	for(int j = i; j < word2Length; j++) string.append(word2.charAt(j));

	return string.toString();
	}
}
```

Time complexity:

Space Complexity:

Code:
```
class Solution {
	public String mergeAlternately(String word1, String word2) {
	StringBuilder sb = new StringBuilder();
	int lenmax = Math.max(word1.length(),word2.length());
	for(int i=0;i<=lenmax-1;i++)
	{
	if(i<word1.length()) sb.append(word1.charAt(i));
	if(i<word2.length()) sb.append(word2.charAt(i));
	}
	return sb.toString();
	}
}
```

Time complexity:

Space Complexity:

Problem: 19

Roman to Integer

Code:

```
class Solution {
  public int romanToInt(String s) {
        HashMap<Character, Integer> hashMap = new HashMap<>();
        hashMap.put('I', 1);
        hashMap.put('V', 5);
        hashMap.put('X', 10);
        hashMap.put('L', 50);
        hashMap.put('C', 100);
        hashMap.put('D', 500);
        hashMap.put('M', 1000);

        int value = 0;
        int length =  s.length();
        for(int i = 0; i < length; i++)
        {
                char ch = s.charAt(i);
                if(i + 1 < length && ch == 'I' && (s.charAt(i + 1) == 'V' || s.charAt(i +
                1) == 'X' ))
                {
                        value += hashMap.get(s.charAt(i + 1)) - 1;
                        i++;
                }
                else   if(i + 1 < length &&ch == 'X' && (s.charAt(i + 1) == 'L' ||
                s.charAt(i + 1) == 'C' ))
                {
                        value += hashMap.get(s.charAt(i + 1)) - 10;
                        I++;
                }
                else   if(i + 1 < length && ch == 'C' && (s.charAt(i + 1) == 'D' ||
                s.charAt(i + 1) == 'M' ))
                {
                        value += hashMap.get(s.charAt(i + 1)) - 100;
                        I++;
                }
                else
                        value += hashMap.get(ch);
        }

        return value;
  }
}
```

Time complexity:
Space Complexity:

Code:

```
class Solution {
        public int romanToInt(String S) {
        int ans = 0, num = 0;
        for (int i = S.length()-1; i >= 0; i--) {
        switch(S.charAt(i)) {
                case 'I': num = 1; break;
                case 'V': num = 5; break;
                case 'X': num = 10; break;
                case 'L': num = 50; break;
                case 'C': num = 100; break;
                case 'D': num = 500; break;
                case 'M': num = 1000; break;
        }
        if (4 * num < ans) ans -= num;
        else ans += num;
        }
        return ans;
        }
}
```

Time complexity:
Space Complexity:

Problem: 20

Valid Parentheses

Code:

```
class Solution {
  public boolean isValid(String s) {
        Stack<Character> stack = new Stack<Character>();
        for (char c : s.toCharArray()) {
        if (c == '(')
        stack.push(')');
        else if (c == '{')
        stack.push('}');
        else if (c == '[')
        stack.push(']');
        else if (stack.isEmpty() || stack.pop() != c)
        return false;
        }
        return stack.isEmpty();
  }
}
```

Time complexity:

Space Complexity:

Problem: 21

Length of Last Word

Code:

```
class Solution {
  public int lengthOfLastWord(String s) {
          s = s.trim();
          int index = s.length() - 1;
          int length = 0;
          while(index >=0 && s.charAt(index--) != ' ')
          length++;

          return length;
  }
}
```

Time complexity:

Space Complexity:

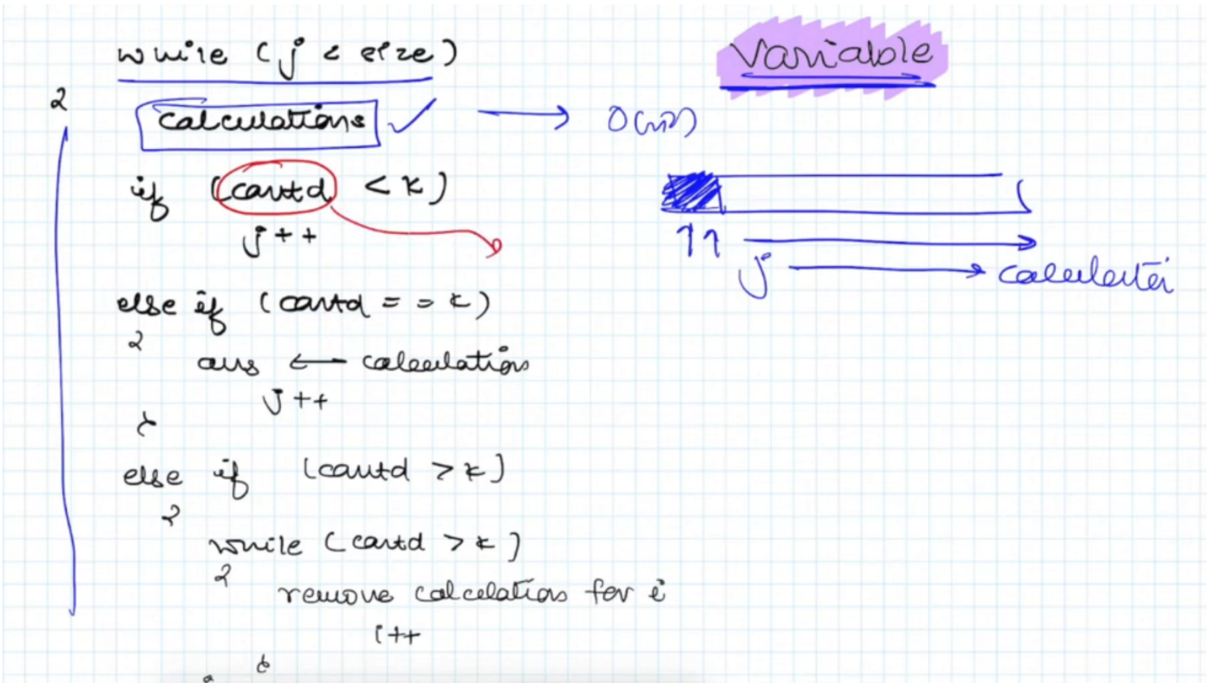
Code:

```
class Solution {
        public int lengthOfLastWord(String s) {
        return s.trim().length()-s.trim().lastIndexOf(" ")-1;
        }
}
```

Time complexity:

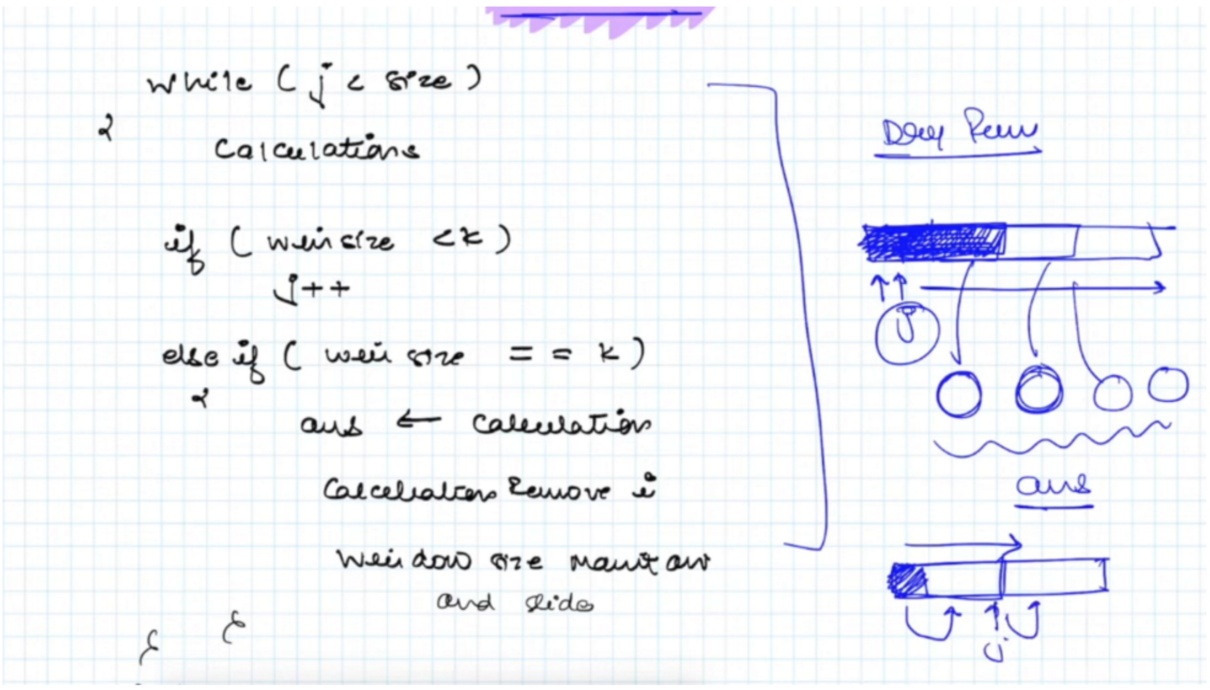
Space Complexity:

Sliding Window Technique



```
while (j < size)
{
    calculations ✓ → O(n)
    if (cantd < k)
        j++
    else if (cantd == k)
    {
        ans ← calculations
        j++
    }
    else if (cantd > k)
    {
        while (cantd > k)
        {
            remove calculations for i
            i++
        }
```

Variable
O(n)
j
calculation



```
while (j < size)
{
    calculations
    if (win size < k)
        j++
    else if (win size == k)
    {
        ans ← calculations
        calculations remove i
        window size maintain
        and slide
    }
}
```

Dry Run
j
ans
i

Problem: 1

Maximum Average Subarray

Code:
```
class Solution {
        public double findMaxAverage(int[] nums, int k) {
                double maxAverage = 0.0d;
                int sum = 0;
                int length = nums.length;
                for(int i = 0; i < k; i++)
                {
                        sum+=nums[i];
                }

                maxAverage = (double)sum / k;
                for(int i = k; i < length; i++)
                {
                        sum += nums[i] - nums[i - k];
                        double average = (double)sum / k;
                        maxAverage = Math.max(average,maxAverage);
                }
                return maxAverage;
        }
}
```

Time complexity: O(N)

Space Complexity:O(1)

Problem: 2

First Negative Number in every Window of Size K

Code:

```
static void firstnegative(int arr[], int k)
    {
        int length = arr.length;
        List<Integer> list = new ArrayList<>();

        for(int i = 0, j = 0; j < length; j++)
        {
            // add negative number to list
            if(arr[j] < 0)
                list.add(arr[j]);

            // check for hit window
            if((j - i + 1) == k)
            {
                // check is there are elements in list is empty
                // means no negative in window
                if(list.isEmpty())
                    System.out.print(0 +" ");
                // else means negative element exist in window
                else
                {
                    System.out.println(list.get(0));
                    // check if we pass from the window the added negative
                    number of list
                    // then remove
                    if(arr[i] == list.get(0))
                        list.remove(0);
                }
                i++;
            }
        }
    }
```

Time complexity:O(N)

Space Complexity:O(N)

Problem: 3

Maximum Sum Subarray of size K

Code:

```
	static int maxSubArraySum(int arr[], int k)
		{
			int max = Integer.MIN_VALUE;
			int sum = 0;
			for(int i = 0; i < k; i++)
				sum += arr[i];


			max = sum;

			int length = arr.length;
			for(int i = k; i < length; i++)
			{
				sum = sum - arr[i - k] + arr[i];
				max = Math.max(sum, max);
			}
			return max;
		}
```

Time complexity:O(N)

Space Complexity:O(1)

Problem: 4

Minimum Sum Subarray of size K

Code:
```
static int minSubArraySum(int arr[], int k)
        {
                int min = 0;
                int sum = 0;
                for(int i = 0; i < k; i++)
                        sum += arr[i];


                min = sum;

                int length = arr.length;
                for(int i = k; i < length; i++)
                {
                        sum = sum - arr[i - k] + arr[i];
                        min = Math.min(sum, min);
                }
                return min;
        }
```

Time complexity:O(N)

Space Complexity:O(1)

Problem: 5

Substrings of Size Three with Distinct Characters

Code:

```
class Solution {
      static boolean isRepeat(String s)
       {
             HashMap<Character, Integer> map = new HashMap<Character, Integer>();
             for(int i = 0 ; i < 3; i++)
             {
                   if(map.containsKey(s.charAt(i)))
                         return false;
                   map.put(s.charAt(i), i);
             }
             return true;
       }

         public static int countGoodSubstrings(String s) {
         int length = s.length();
         int count = 0;
         StringBuilder str = new StringBuilder();
         for(int i = 0,j=0; i < length; i++)
         {
                 str.append(s.charAt(i));
                 if(i - j + 1 == 3)
                 {
                         if(isRepeat(str.toString()))
                         count++;
                         str.deleteCharAt(0);
                         j++;
                 }
         }

         return count;
         }
}
```

Time complexity: O(N) * 3 = O(N)

Space Complexity:O(N)

Code:
```
class Solution {

        public static int countGoodSubstrings(String s) {
                int length = s.length();
                int count = 0;
                HashMap<Character, Integer> map = new HashMap<Character,
        Integer>();
                for(int i = 0,j=0; i < length; i++)
                {
                        char ch = s.charAt(j);
                        char sch = s.charAt(i);
                        map.put(sch, map.getOrDefault(s.charAt(i),0) + 1);
                        if(i - j + 1 == 3)
                        {
                                if(map.size() == 3)
                                        count++;

                                int val = map.get(ch);
                                if((val - 1) == 0)
                                        map.remove(ch);
                                else
                                        map.put(ch, val - 1);
                                j++;

                        }
                }

        return count;
        }
}
```

Time complexity:O(N)

Space Complexity:O(N)

Problem: 6

Minimum Difference Between Highest and Lowest of K Scores

Code:

```
class Solution {
        public int minimumDifference(int[] nums, int k) {
              Arrays.sort(nums);
              int min = Integer.MAX_VALUE, max = Integer.MIN_VALUE;
              int length = nums.length;
              int diff = 0;

                diff = nums[k - 1] - nums[0];
                for(int i = k, j = 1; i < length; i++,j++)
                        diff = Math.min(diff, nums[i] - nums[j]);

                return diff;
        }
}
```

Time complexity: N * (LOG N) + N = N * LON(N)

Space Complexity:O(1)

Problem: 7

Sliding Window Maximum

Code:

```
      class Solution {
        public int[] maxSlidingWindow(int[] array, int window) {
                  List<Integer> list = new LinkedList<>();
                  int length = array.length;
                 int result[] = new int[length - window + 1];
                 int listSize, index = 0;
                 for(int i = 0, j = 0; i < length; i++)
                 {
                       listSize = list.size();
                       while(listSize > 0 && list.get(listSize - 1) < array[i])
                       {
                             list.remove(listSize - 1);
                             listSize--;
                       }
                      list.add(array[i]);

                       if(i - j + 1 == window)
                       {
                            result[index++] = list.get(0);

                             if(list.get(0) == array[j])
                                   list.remove(0);
                            j++;
                       }

                 }
                 return result;

        }
```

Time complexity: O(N * M)

Space Complexity:O(N)

Code:

```
public int[] maxSlidingWindow(int[] a, int k) {
        int n = a.length;
        int[] res = new int[n - k + 1];
        Deque<Integer> dq = new ArrayDeque<Integer>();
        int i=0,j=0,l=0;
        while(j < n) {
            while(dq.size() > 0 && a[j] > dq.peekLast()){
                dq.pollLast();
            }
            dq.add(a[j]);

            if(j-i+1 < k) j++;

            else if(j-i+1==k)
            {
                res[l++] = dq.peekFirst();
                if(dq.peekFirst()==a[i]) {
                    dq.remove(a[i]);
                }
                i++;
                j++;
            }
        }
        return res;
    }
}
```

Time complexity:O(N)

Space Complexity:O(N)

Problem: 8

Longest K unique characters substring

Code:

```
class Solution {
        public static int longestkSubstr(String s, int k) {
        HashMap<Character, Integer> map = new HashMap<>();
        int length = s.length();
        int longest = -1;
        for(int i = 0, j = 0; i < length; i++)
        {
           map.put(s.charAt(i), map.getOrDefault(s.charAt(i), 0) + 1);
           if(map.size() == k)
                longest = Math.max(longest, (i - j + 1));
           if(map.size() > k)
           {
                char c = s.charAt(j);
                int value = map.get(c);
                if(value - 1 == 0)
                      map.remove(c);
                else
                      map.put(c, value - 1);
                j++;
           }
        }
        return longest;
        }
}
```

Time complexity:O(N)

Space Complexity:O(N)

Problem: 9

Longest Substring Without Repeating Characters.

Code:

```
class Solution {
  public int lengthOfLongestSubstring(String s) {
  List<Character> list = new LinkedList<>();
          int longest = 0;
          int length = s.length();

          for(int i = 0;  i < length; i++)
          {
          char c = s.charAt(i);
           if(list.contains(c));
           {
                   int index = list.indexOf(c);
                   while(index >= 0)
                   {
                           list.remove(index);
                           index--;
                   }
           }
           list.add(c);
           longest = Math.max(longest,list.size());
          }
          return longest;
  }
}
```

Time complexity:O(N)

Space Complexity:O(N)

Code:

```
 HashMap<Character, Integer> map = new HashMap<>();
        int longest = 0;
        int length = s.length();

        for(int i = 0, j = 0; i < length; i++)
        {
        char c = s.charAt(i);
        if(map.containsKey(c))
        {
                j = Math.max(j, map.get(c) + 1);
        }
        map.put(c, i);
        longest = Math.max(longest, (i - j) + 1);
        }
```

Time complexity:O(N)

Space Complexity:O(N)

Bit Manipulation

Problem: 1

Add Binary

Code:

```
class Solution {
   public String addBinary(String a, String b) {
     StringBuilder string = new StringBuilder();

       int aLength = a.length();
       int bLength = b.length();

       int i = aLength - 1, j = bLength - 1;
       int sum = 0;
       boolean carry = false;
       while(i >= 0 || j >= 0)
       {
          carry = false;
          if(i >= 0) sum += a.charAt(i) - '0';
          if(j >= 0) sum += b.charAt(j) - '0';

          string.append(sum % 2);

          if(sum > 1)
          {
                    sum = 1;
                    carry = true;
          }
          else sum = 0;
          i--;
          j--;
       }
       if(carry) string.append(1);
       return string.reverse().toString();
   }
}
```

Time complexity: O(MAX(a,b))

Space Complexity:O(1)

Problem: 2

Single Number

Code:

```
class Solution {
   public int singleNumber(int[] nums) {
      int XOR = nums[0];
      int length = nums.length;
      for(int i = 1; i < length; i++)
         XOR ^= nums[i];

      return XOR;
   }
}
```

Time complexity: O(N)

Space Complexity: O(1)

Problem: 3

Reverse Bits

Code:

```
public class Solution {
   // you need treat n as an unsigned value
   public static int reverseBits(int n) {
      int res=0;
       for(int i=0;i<32;i++){
             res = ( res << 1 ) | ( n & 1 );
             n = n >> 1;
        }
   return res;
   }

}
```

Time complexity: O(1)

Space Complexity: O(1)

Code:

```
class Solution {
   public static int reverseBits(int n) {
       if (n == 0) {
          return 0;
       }

       int result = 0;
       int power = 31;

       while (n != 0) {
          result |= (n & 1) << power;
          n >>= 1;
          power--;
       }

       return result;
   }
}
```

Time complexity: O(1)

Space Complexity: O(1)

Problem: 4

Number of 1 Bits

Code:

```
public class Solution {
   // you need to treat n as an unsigned value
   public int hammingWeight(int n) {
      int count = 0;
      int bit = 0;
      while(bit <= 31)
      {
         count += (n & 1);
         n = n >> 1;
         bit++;
      }
      return count;
   }
}
```

Time complexity:O(1)

Space Complexity:O(1)

Code:

```
public class Solution {
   public int hammingWeight(int n) {
      int count = 0;
      int bit = 0;
      while(n !=0)
      {
         count += (n & 1);
         n = n >>> 1;
      }
      return count;
   }
}
```

Time complexity:O(1)

Space Complexity:O(1)

Problem: 5

Power of Two

Code:

```
class Solution {
    public boolean isPowerOfTwo(int n) {
        if(n <= 0) return false;
        return ((n) & (n - 1)) == 0;
    }
}
```

Time complexity:O(1)

Space Complexity:O(1)

Problem: 6

Missing Number

Code:

```
class Solution {
    public int missingNumber(int[] nums) {
         int length = nums.length;
         int sum = 0;
         int totalSum = (length * ( length + 1)) / 2;
        for(int i = 0 ; i  < length; i++)
            sum += nums[i];

         return totalSum - sum;
    }
}
```

Time complexity:O(N)

Space Complexity:O(1)

Code:

```
class Solution {
    public int missingNumber(int[] nums) {
        int length = nums.length;
        int XOR = length;

        for(int i = 0; i < length; i++)
        {
            XOR ^= i;
            XOR ^= nums[i];
        }

        return XOR;
    }
}
```

Time complexity:O(N)

Space Complexity:O(1)

Problem: 7

Counting Bits

Code:

```
class Solution {
   int countBit(int n)
   {
      int count = 0;
      while(n > 0)
      {
         count += (n & 1);
         n = n >> 1;
      }
      return count;
   }
   public int[] countBits(int n) {
      int result[] = new int[n + 1];

      for(int i = 0; i <= n; i++)
         result[i] = countBit(i);

      return result
   }
}
```

Time complexity:O(N)

Space Complexity:O(N)

Code:(JS)

```
class Solution {

    var countBits = function(n) {
    var result = [];
    result[0] = 0;
    for (var i = 1; i <= n; i++) {
        result[i] = result[i >> 1] + (i & 1);
    }
    return result;

   }
}
```

Time complexity:O(N)

Space Complexity:O(N)

Problem: 8

Power of Four

Code:

```
class Solution {
   public boolean isPowerOfFour(int n) {
      double sqrt = Math.sqrt(n);
      if(sqrt > 0 && (sqrt % 2 == 0 || sqrt == 1))
         return (((int)sqrt) & (int)(sqrt - 1)) ==0;

      return false;
   }
}
```

Time complexity: O(1)

Space Complexity:O(1)

Problem: 9

Find the Difference

Code:
```
class Solution {
   public char findTheDifference(String s, String t) {
      int sum1 = 0, sum2 = 0;
      for(int i = 0; i < t.length(); i++)
      {
         if(i < s.length()) sum1 += s.charAt(i) - 'a';
         sum2+= t.charAt(i) - 'a';
      }
      return (char)((sum2 - sum1) + 'a');
   }
}
```

Time complexity: O(t)

Space Complexity:O(1)

Code:
```
class Solution {
   public char findTheDifference(String s, String t) {
      char XOR = 0;
      for(int i = 0, j = 0; i < t.length(); i++,j++)
      {
         if(j < s.length()) XOR ^= s.charAt(j);
         XOR ^= t.charAt(i);
      }
      return XOR;
   }
}
```

Time complexity: O(t)

Space Complexity:O(1)

Problem: 10

Convert a Number to Hexadecimal

Code:

```
class Solution {

   public String toHex(int num) {
      String result = "";
       if (num == 0)
           return "0";
       char array[] = new char[]{'0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'};
      while(num != 0)
      {
          int n = num & 15;
          result = array[n] + result;
          num = num >>> 4;
      }
       return result;
   }
}
```

Time complexity:O(N)

Space Complexity:O(!)

Problem: 11

Hamming Distance

Code:
```
class Solution {
    public int hammingDistance(int x, int y) {
        int count = 0;
        while(x > 0 || y > 0)
        {
            if(((x & 1) == 1 && (y & 1) == 0) || (((x & 1) == 0) && ((y & 1) == 1))) count++;
            x = x >> 1;
            y = y >> 1;
        }
        return count;
    }
}
```

Time complexity:O(MAX(x,y))

Space Complexity:O(1)

Problem: 12

Number Complement

Code:
```
class Solution {
   public int findComplement(int num) {
      int result = num;
      int pow = 0;
      while(result > 0)
      {
         if(((1 << pow) & num) > 0)
                  num = ~(1 << (pow)) & num;
         else
                  num = (1 << pow) | num;
         result = result >> 1;
         pow++;
      }

      return num;
   }
}
```

Time complexity:O(N)

Space Complexity:O(1)

Problem: 15

Longest Common Prefix

Code:
Time complexity:
Space Complexity:

Problem: 15

Longest Common Prefix

Code:
Time complexity:
Space Complexity:

Problem: 15

Longest Common Prefix

Code:

Time complexity:

Space Complexity:

Problem: 15

Longest Common Prefix

Code:

Time complexity:

Space Complexity:

Problem: 15

Longest Common Prefix

Code:

Time complexity:

Space Complexity:

Problem: 15

Longest Common Prefix

Code:
Time complexity:
Space Complexity: