Sum_III

Hard > 3-Sum Problem.java

```
/*
 * 3 Sum : Find triplets that add up to a zero
   Problem Statement: Given an array of N integers, your task is to find unique triplets that add up to give a sum of zero.
    In short, you need to return an array of all the unique triplets [arr[a], arr[b], arr[c]] such that i!=j, j!=k,
    k!=i, and their sum is equal to zero.

   Examples:

   Example 1:
   Input: nums = [-1,0,1,2,-1,-4]
   Output: [[-1,-1,2],[-1,0,1]]
   Explanation: Out of all possible unique triplets possible, [-1,-1,2] and [-1,0,1] satisfy the condition of summing up to zero with i!=j!=k

   Example 2:
   Input: nums=[-1,0,1,0]
   Output: Output: [[-1,0,1],[-1,1,0]]
   Explanation: Out of all possible unique triplets possible, [-1,0,1] and [-1,1,0] satisfy the condition of summing up to zero with i!=j!=k
 */

```

```java
Codeium: Refactor | Explain
public class Sum_III {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static List<List<Integer>> tripletSum(int[] nums) {
        /*
         * BruteForce Approach: Time complexity: O(N^3*logk) & Space complexity: O(3*k)
         * List<List<Integer>> ans = new ArrayList<>();
           for(int i = 0; i < nums.length - 2; i++){
               for(int j = i + 1; j < nums.length - 1; j++){
                   for(int k = j + 1; k < nums.length; k++){
                       ArrayList<Integer> temp = new ArrayList<>();
                       if(i != j && i != k && j != k && nums[i] + nums[j] + nums[k] == 0){
                           temp.add(nums[i]);
                           temp.add(nums[j]);
                           temp.add(nums[k]);
                       }
                       if(temp.size() != 0){
                           ans.add(temp);
                       }
                   }
               }
           }
           return ans;
        */

        // Optimized Approach: Time complexity: O(N^3) & Space complexity: O(1).

        Arrays.sort(nums);
        List<List<Integer>> ans = new ArrayList<>();
        for(int i = 0; i < nums.length - 2; i++){
            if(i == 0 || (i > 0 && nums[i] != nums[i - 1])){
                int low = i + 1, high = nums.length - 1, sum = 0 - nums[i];
                while(low < high){
                    if(nums[low] + nums[high] == sum){
                        ans.add(Arrays.asList(nums[i], nums[low], nums[high]));
                        while(low < high && nums[low] == nums[low + 1]){
                            low++;
                        }
                        while(low < high && nums[high] == nums[high - 1]){
                            high--;
                        }
                        low++;
                        high--;
                    }
                    else if(nums[low] + nums[high] < sum){
                        low++;
                    }
                    else{
                        high--;
                    }
                }
            }
        }
        return ans;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int[] nums = {-1,0,1,2,-1,-4};
        System.out.println(tripletSum(nums));
    }
}
```

```
> Task :Sum_III.main()
[[-1, -1, 2], [-1, 0, 1]]
```

Sum_IV

```java
/*
* 4 Sum | Find Quads that add up to a target value Problem Statement:
* Given an array of N integers, your task is to find unique quads that add up to give a target value.
*  In short, you need to return an array of all the unique quadruplets [arr[a], arr[b], arr[c],
* arr[d]] such that their sum is equal to a given target.
*
  Note:
  0 <= a, b, c, d < n
  a, b, c, and d are distinct.
  arr[a] + arr[b] + arr[c] + arr[d] == target
  Example 1:

  Input Format: arr[] = [1,0,-1,0,-2,2], target = 0

  Result: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]

  Explanation: We have to find unique quadruplets from
  the array such that the sum of those elements is
  equal to the target sum given that is 0.

  The result obtained is such that the sum of the
  quadruplets yields 0.
*/

Codeium: Refactor | Explain
public class Sum_IV {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static List<List<Integer>> fourSum(int[] nums, int target) {
        /*
        * // BruteForce Appraoch: Time complexity: O(N^4 * logK) & Space complexity: O(1).
        * // It will only insert uniques quad.
        * List<List<Integer>> ans = new ArrayList<>();
          for(int a = 0; a < nums.length - 3; a++){
              for(int b = a + 1; b < nums.length - 2; b++){
                  for(int c = b + 1; c < nums.length - 1; c++){
                      for(int d = c + 1; d < nums.length; d++){
                          ArrayList<Integer> temp = new ArrayList<>();
                          if(nums[a] + nums[b] + nums[c] + nums[d] == target){
                              temp.add(nums[a]);
                              temp.add(nums[b]);
                              temp.add(nums[c]);
                              temp.add(nums[d]);
                          }
                          if(ans.contains(temp)){
                              // System.out.println("Hello World");
                          }
                          else if(temp.size() != 0){
                              ans.add(temp);
                          }
                      }
                  }
              }
          }
          return ans;
        */
```

```
/*
 * // Solution 2: Using three pointer & binary search.
 * // Time complexity: O(N log N + N³ logN) & Space complexity: O(1).
 * // It also contains duplicate quad.
 *
 * List<List<Integer>> ans = new ArrayList<>();
   Arrays.sort(nums);
   for(int i = 0; i < nums.length - 3; i++){
       for(int j = i + 1; j < nums.length - 2; j++){
           for(int k = j + 1; k < nums.length - 1; k++){
               int sum = nums[i] + nums[j] + nums[k];
               int item = target - sum;
               int start = k + 1;
               int end = nums.length;
               int mid = 0;
               while(start <= end){
                   mid = start + (end - start)/2;
                   if(nums[mid] == item){
                       ans.add(Arrays.asList(nums[i], nums[j], nums[k], nums[mid]));
                       break;
                   }
                   else if(nums[mid] > target){
                       end = mid - 1;
                   }
                   else{
                       start = mid + 1;
                   }
               }
           }
       }
   }
   return ans;
*/
```

```java

        // Optimized Solution: Time complexity: O(N^3) & Space complexity: O(1).
        List<List<Integer>> ans = new ArrayList<>();
        Arrays.sort(nums);
        for(int i = 0; i < nums.length - 1; i++){
            for(int j = i + 1; j < nums.length; j++){
                int target2 = target - nums[j] - nums[i];
                int front = j + 1;
                int back = nums.length - 1;
                while(front < back){
                    int twoSum = nums[front] + nums[back];
                    if(twoSum < target2){
                        front++;
                    }
                    else if(twoSum > target2){
                        back--;
                    }
                    else{
                        List<Integer> quad = new ArrayList<>();
                        quad.add(nums[i]);
                        quad.add(nums[j]);
                        quad.add(nums[front]);
                        quad.add(nums[back]);
                        ans.add(quad);
                        while(front < back && nums[front] == quad.get(2)){
                            front++;
                        }
                        while(front < back && nums[back] == quad.get(3)){
                            back--;
                        }
                    }
                }
                while(j + 1 < nums.length && nums[j + 1] == nums[j]) ++j;
            }
            while (i + 1 < nums.length && nums[i + 1] == nums[i]) ++i;
        }
        return ans;
    }
```

Codeium: Refactor | Explain | Generate Javadoc | ×

```java
    public static void main(String[] args) {
        int[] nums = {1,0,-1,0,-2,2};
        int target = 0;
        System.out.println(fourSum(nums, target));
    }
}

```

```
> Task :Sum_IV.main()
[[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]
```

Count_inversions_in_an_array

```
/*
 * Count inversions in an array
   Problem Statement: Given an array of N integers, count the inversion of the array (using merge-sort).

   What is an inversion of an array? Definition: for all i & j < size of array, if i < j
   then you have to find pair (A[i],A[j]) such that A[j] < A[i].

   Example 1:

   Input Format: N = 5, array[] = {1,2,3,4,5}

   Result: 0

   Explanation: we have a sorted array and the sorted array
   has 0 inversions as for i < j you will never find a pair
   such that A[j] < A[i]. More clear example: 2 has index 1
   and 5 has index 4 now 1 < 5 but 2 < 5 so this is not an
   inversion.
*/
```

```java
public class Count_inversions_in_an_array {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    static int merge(int arr[],int temp[],int left,int mid,int right)
    {
        int inv_count=0;
        int i = left;
        int j = mid;
        int k = left;
        while((i <= mid-1) && (j <= right)){
            if(arr[i] <= arr[j]){
                temp[k++] = arr[i++];
            }
            else
            {
                temp[k++] = arr[j++];
                inv_count = inv_count + (mid - i);
            }
        }

        while(i <= mid - 1)
            temp[k++] = arr[i++];

        while(j <= right)
            temp[k++] = arr[j++];

        for(i = left ; i <= right ; i++)
            arr[i] = temp[i];

        return inv_count;
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    static int merge_Sort(int arr[],int temp[],int left,int right)
    {
        int mid,inv_count = 0;
        if(right >= left){
            return 0;
        }
        else{
            mid = (left + right)/2;

            inv_count += merge_Sort(arr,temp,left,mid);
            inv_count += merge_Sort(arr,temp,mid+1,right);

            inv_count += merge(arr,temp,left,mid+1,right);
            return inv_count;
        }
    }

```

```java

    // public static int countInversions(int[] arr){
    //     /*
    //      * //BruteForce Approach: Time complexity: O(N^2) & Space complexity: O(1).
    //      * int count = 0;
    //        for(int i = 0; i < arr.length; i++){
    //            for(int j = i + 1; j < arr.length; j++){
    //                if(arr[i] > arr[j]){
    //                    count++;
    //                }
    //            }
    //        }
    //        return count;
    //     */

    // }
    Codeium: Refactor | Explain | ×
    public static void main(String[] args) {
        int arr[]={5,3,2,1,4};
        int n=arr.length;
        int[] temp = new int[arr.length];
        int ans = merge_Sort(arr,temp,0,n-1);
        System.out.println(ans);

        // System.out.println(countInversions(arr));
    }
}

```

```
> Task :Count_inversions_in_an_array.main()
0
```

Count_number_of_subarrays_with_given_xorK

```java

/*
 * Count the number of subarrays with given xor K
   Problem Statement: Given an array of integers A and an integer B. Find the total number of
   subarrays having bitwise XOR of all elements equal to B.

   Examples:

   Input Format:  A = [4, 2, 2, 6, 4] , B = 6
   Result: 4
   Explanation: The subarrays having XOR of their elements as 6 are  [4, 2], [4, 2, 2, 6, 4], [2, 2, 6], [6]

   Input Format: A = [5, 6, 7, 8, 9], B = 5
   Result: 2
   Explanation:The subarrays having XOR of their elements as 2 are [5] and [5, 6, 7, 8, 9]
*/

```

```java
public class Count_number_of_subarrays_with_given_xorK {
    public static int subsetXOR(int arr[], int k) {
        /*
            BruteForce Appraoch: Time complexity: O(N^N) & Space complexity: O(1)
            int count = 0;
            for(int i = 0; i < arr.length; i++){
                int currXor = 0;
                for(int j = i; j < arr.length; j++){
                    currXor ^= arr[j];
                    if(currXor == k){
                        count++;
                    }
                }
            }
            return count;
        */
        // Optimized Solution: Time complexity: O()
        HashMap<Integer, Integer> map = new HashMap<>();
        int count = 0;
        int xorr = 0;
        for(int i = 0; i < arr.length; i++){
            xorr = xorr ^ arr[i];
            if(map.get(xorr ^ k) != null){
                count += map.get(xorr ^ k);
            }
            if(xorr == k){
                count++;
            }
            if(map.get(xorr) != null){
                map.put(xorr, map.get(xorr) + 1);
            }
            else{
                map.put(xorr, 1);
            }
        }
        return count;
    }
    public static void main(String[] args) {
        int[] nums = {4, 94, 39, 36, 88, 87, 39, 67, 11, 6};
        int k = 15;
        System.out.println(subsetXOR(nums, k));
    }
}

```

```
> Task :Count_number_of_subarrays_with_given_xorK.main()
1
```

Find_the_repeating_and_missing_numbers

```
/*
 * Find the repeating and missing numbers
   Problem Statement: You are given a read-only array of N integers with values
   also in the range [1, N] both inclusive.
    Each integer appears exactly once except A which appears twice and B which is missing. The task is to
     find the repeating and missing numbers A and B where A repeats twice and B is missing.

   Example 1:
   Input Format:  array[] = {3,1,2,5,3}
   Result: {3,4)
   Explanation: A = 3 , B = 4
   Since 3 is appearing twice and 4 is missing

   Example 2:
   Input Format: array[] = {3,1,2,5,4,6,7,5}
   Result: {5,8)
   Explanation: A = 5 , B = 8
   Since 5 is appearing twice and 8 is missing

*/
```

```
Codeium: Refactor | Explain
public class Find_the_repeating_and_missing_numbers {
    static int x, y;
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void findMissingRepeating(int[] arr, int n){
        /*
        * // Solution 1: Time complexity: O(N) & Space complexity: O(N).
        * int[] substitute = new int[arr.length + 1];
          for(int i = 0; i < arr.length; i++){
              substitute[arr[i]]++;
          }
          for(int i = 1; i < substitute.length; i++){
              if(substitute[i] == 0){
                  System.out.println("The missing number is: " + i);
              }
              if(substitute[i] > 1){
                  System.out.println("The repeating number is: " + i);
              }
          }
        */

        /*
          // Solution 2: https://www.youtube.com/watch?v=5nMGY4VUoRY&list=PLgUwDviBIf0rPG3Ictpu74YWBQ1CaBkm2&index=4
          // Time complexity: O(N) & Space complexity: O(1).
          long len = arr.length;

          long S = (len * (len+1) ) /2;
          long P = (len * (len +1) *(2*len +1) )/6;
          long missingNumber=0, repeating=0;

          for(int i=0;i<arr.length; i++){
            S -= arr[i];
            P -= arr[i]*arr[i];
          }

          missingNumber = (S + P/S)/2;

          repeating = missingNumber - S;

          System.out.println("Missing Number is: " + missingNumber);
          System.out.println("repeating Number is: " + repeating);
        */
```

```
    // Solution 3: Time complexity: O(N) & Space complexity: O(1).

    /* Will hold xor of all elements
   and numbers from 1 to n  */
   int xor1;

   /* Will have only single set bit of xor1 */
   int set_bit_no;

   int i;
   x = 0;
   y = 0;

   xor1 = arr[0];

   /* Get the xor of all array elements  */
   for (i = 1; i < n; i++)
      xor1 = xor1 ^ arr[i];

   /* XOR the previous result with numbers from
  1 to n*/
   for (i = 1; i <= n; i++)
      xor1 = xor1 ^ i;

   /* Get the rightmost set bit in set_bit_no */
   set_bit_no = xor1 & ~(xor1 - 1);

   /* Now divide elements into two sets by comparing
rightmost set bit of xor1 with the bit at the same
position in each element. Also, get XORs of two
sets. The two XORs are the output elements. The
following two for loops serve the purpose */
   for (i = 0; i < n; i++) {
      if ((arr[i] & set_bit_no) != 0)
         /* arr[i] belongs to first set */
         x = x ^ arr[i];

      else
         /* arr[i] belongs to second set*/
         y = y ^ arr[i];
   }
   for (i = 1; i <= n; i++) {
      if ((i & set_bit_no) != 0)
         /* i belongs to first set */
         x = x ^ i;

      else
         /* i belongs to second set*/
         y = y ^ i;
   }
```

```java
        // at last do a linear check which amont x and y is missing or repeating

        /* *x and *y hold the desired output elements */
        boolean flag1 = false;
        boolean flag2 = false;
        for(int k = 0; k < arr.length; k++){
            if(x == arr[k]){
             flag1 = true;
            }
            if(y == arr[k]){
             flag2 = true;
            }
        }
        if(!flag1){
            System.out.println("Missing Number is: " + x);
            System.out.println("repeating Number is: " + y);
        }
        if(!flag2){
            System.out.println("Missing Number is: " + y);
            System.out.println("repeating Number is: " + x);
        }
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int[] arr = {3,1,2,5,3};
        int n = 5;
        findMissingRepeating(arr, n);
    }
}

```

```
> Task :Find_the_repeating_and_missing_numbers.main()
Missing Number is: 4
repeating Number is: 3
```

Largest_Subarray_with_0_Sum

```
/*
 * Length of the longest subarray with zero Sum
   Problem Statement: Given an array containing both positive and negative integers, we have to find the
   length of the longest subarray with the sum of all elements equal to zero.

   Example 1:
   Input Format: N = 6, array[] = {9, -3, 3, -1, 6, -5}
   Result: 5
   Explanation: The following subarrays sum to zero:
   {-3, 3} , {-1, 6, -5}, {-3, 3, -1, 6, -5}
   Since we require the length of the longest subarray, our answer is 5!

   Example 2:
   Input Format: N = 8, array[] = {6, -2, 2, -8, 1, 7, 4, -10}
   Result: 8

   Subarrays with sum 0 : {-2, 2}, {-8, 1, 7}, {-2, 2, -8, 1, 7}, {6, -2, 2, -8, 1, 7, 4, -10}
   Length of longest subarray = 8

   Example 3:
   Input Format: N = 3, array[] = {1, 0, -5}
   Result: 1

   Subarray : {0}
   Length of longest subarray = 1
   Example 4:

   Input Format: N = 5, array[] = {1, 3, -5, 6, -2}
   Result: 0
   Subarray: There is no subarray that sums to zero
*/
```

```java
Codeium: Refactor | Explain
public class Largest_Subarray_with_0_Sum {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int LargestSubarray(int[] nums){
        /*
         * BruteForce Approach: Time complexity: O(N^2) & Space complexity: O(1).
         * int ans = 0, temp = 0;
           for(int i = 0; i < nums.length; i++){
               int sum = 0;
               for(int j = i; j < nums.length; j++){
                   sum += nums[j];
                   if(sum == 0){
                       temp = (j - i) + 1;
                   }
                   ans = Math.max(ans, temp);
               }
           }
           return ans;
        */

        // Optimized Approach: Time complexity: O(NlogN) & Space complexity: O(N).
        HashMap<Integer, Integer> map = new HashMap<>();
        int maxi = 0;
        int sum = 0;
        for(int i = 0; i < nums.length; i++){
            sum += nums[i];
            if(sum == 0){
                maxi = i + 1;
            }
            else{
                if(map.get(sum) != null){
                    maxi = Math.max(maxi, i - map.get(sum));
                }
                else{
                    map.put(sum, i);
                }
            }
        }
        return maxi;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
       int[] nums = {15,-2,2,-8,1,7,10,23};
       System.out.println(LargestSubarray(nums));
    }
}

```

```
> Task :Largest_Subarray_with_0_Sum.main()
5
```

Majority Elements(>N/3 times)

```java
/*
 * Majority Elements(>N/3 times) | Find the elements that appears more than N/3 times in the array
   Problem Statement: Given an array of N integers. Find the elements that appear more than N/3 times in the array.
    If no such element exists, return an empty vector.

   Example 1:
   Input: N = 5, array[] = {1,2,2,3,2}
   Ouput: 2
   Explanation: Here we can see that the Count(1) = 1, Count(2) = 3 and Count(3) = 1.
   Therefore, the count of 2 is greater than N/3 times. Hence, 2 is the answer.

   Example 2:
   Input:  N = 6, array[] = {11,33,33,11,33,11}
   Output: 11 33
   Explanation: Here we can see that the Count(11) = 3 and Count(33) = 3. Therefore,
   the count of both 11 and 33 is greater than N/3 times. Hence, 11 and 33 is the answer.
*/

public class Majority_Element_II {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static List<Integer> majorityElement(int[] nums) {
        /*
         * // BruteForce Approach --> Time complexity: O(N^2) & Space complexity: O(1).
           List<Integer> ans = new ArrayList<>();
           for(int i = 0; i < nums.length; i++){
               int count = 1;
               for(int j = i + 1; j < nums.length; j++){
                   if(nums[i] == nums[j]){
                       count++;
                   }
               }
               if(count > nums.length/3){
                   ans.add(nums[i]);
               }
           }
           return ans;
        */

        /*
         * // Solution 2: Using HashMap --> Time compelxity: O(N) & Space complexity: O(N)
           List<Integer> ans = new ArrayList<>();
           HashMap<Integer, Integer> map = new HashMap<>();
           for(int i = 0; i < nums.length; i++){
               if(map.containsKey(nums[i])){
                   map.put(nums[i], map.get(nums[i]) + 1);
               }
               else{
                   map.put(nums[i], 1);
               }
           }
           for (Map.Entry<Integer, Integer> entry : map.entrySet()){
               if(entry.getValue() > nums.length/3){
                   ans.add(entry.getKey());
               }
           }
           return ans;
        */

```

```java
        // Optimized Solution: Boyer Moore Voting Algorithms.
        // Time complexity: O(N) & Space complexity: O(1).
        List<Integer> ans = new ArrayList<>();
        int num1 = -1; int num2 = 0; int count1 = 0; int count2 = 0;
        for(int i = 0; i < nums.length; i++){
            if(num1 == nums[i]){
                count1++;
            }
            else if(num2 == nums[i]){
                count2++;
            }
            else if (count1 == 0){
                num1 = nums[i];
                count1 = 1;
            }
            else if (count2 == 0){
                num2 = nums[i];
                count2 = 1;
            }
            else{
                count1--;
                count2--;
            }
        }
        int count01 = 0, count02 = 0;
        for(int i = 0; i < nums.length; i++){
            if(num1 == nums[i]){
                count01++;
            }
            if(num2 == nums[i]){
                count02++;
            }
        }
        if(count01 > nums.length/3){
            ans.add(num1);
        }
        if(count02 > nums.length/3){
            ans.add(num2);
        }
        return ans;
    }
```

```
Codeium: Refactor | Explain | Generate Javadoc | X
    public static void main(String[] args) {
        int[] nums = {-1, -1, -1};
        System.out.println(majorityElement(nums));
        /*
         * for BruteForce Solution:
         * List<Integer> majority = majorityElement(nums);
           HashSet<Integer> set = new HashSet<>(majority);
           for(int it: set){
               System.out.print(it + " ");
           }
        */
    }
}

```

```
> Task :Majority_Element_II.main()
[-1]
```

Maximum_Product_Subarrays

```
/*
 * Maximum Product Subarray in an Array
   Problem Statement: Given an array that contains both negative and positive integers, find the maximum product subarray.

   Examples:

   Example 1:
   Input:
    Nums = [1,2,3,4,5,0]
   Output:
    120
   Explanation:
    In the given array, we can see 1×2×3×4×5 gives maximum product value.


   Example 2:
   Input:
    Nums = [1,2,-3,0,-4,-5]
   Output:
    20
   Explanation:
   In the given array, we can see (-4)×(-5) gives maximum product value.
*/

```

```java
Codeium: Refactor | Explain
public class Maximum_Product_Subarrays {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int maxProductSubArray(int[] nums){
        /*
         * //BruteForce Approach: Time complexity: O(N^2) & Space complexity: O(1).
         * int ans = Integer.MIN_VALUE;
           for(int i = 0; i < nums.length; i++){
               int product = 1;
               for(int j = i; j < nums.length; j++){
                   product *= nums[j];
                   ans = Math.max(ans, product);
               }
           }
           return ans;
        */

        // Solution 2:  Time complexity: O(N) & Space complexity: O(1)
        int ans = nums[0];
        int max = ans;
        int min = ans;
        for(int i = 1; i < nums.length; i++){
            if(nums[i] < 0){
                int temp = max;
                max = min;
                min = temp;
            }
            max = Math.max(nums[i], max*nums[i]);
            min = Math.min(nums[i], min*nums[i]);
            ans = Math.max(ans, max);
        }
        return ans;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int nums[] = {2, 3, -2, 4};
        int answer = maxProductSubArray(nums);
        System.out.print("The maximum product subarray is: "+answer);
    }
}

```

```
> Task :Maximum_Product_Subarrays.main()
The maximum product subarray is: 6
```

Merge_Overlapping_Sub_intervals

```java
/*
 * Q. Merge Overlapping Sub-intervals.
   Problem Statement: Given an array of intervals, merge all the overlapping intervals and return an array of non-overlapping intervals.

   Examples

   Example 1:
   Input: intervals=[[1,3],[2,6],[8,10],[15,18]]
   Output: [[1,6],[8,10],[15,18]]
   Explanation: Since intervals [1,3] and [2,6] are overlapping we can merge them to form [1,6]
    intervals.

   Example 2:
   Input: [[1,4],[4,5]]
   Output: [[1,5]]
   Explanation: Since intervals [1,4] and [4,5] are overlapping we can merge them to form [1,5].
*/

```

```java
Codeium: Refactor | Explain
public class Merge_Overlapping_Sub_intervals {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static List<List<Integer>> merge(int[][] intervals) {
        Arrays.sort(intervals, (i1, i2) -> Integer.compare(i1[0], i2[0]));
        List<List<Integer>> ans = new ArrayList<>();
        for (int i = 0; i < intervals.length; i++) {
            if (ans.size() == 0 || ans.get(ans.size() - 1).get(1) < intervals[i][0]) {
                ArrayList<Integer> v = new ArrayList<>();
                v.add(intervals[i][0]);
                v.add(intervals[i][1]);
                ans.add(v);
            } else {
                ans.get(ans.size() - 1).set(1, Math.max(ans.get(ans.size() - 1).get(1), intervals[i][1]));
            }
        }

        return ans;
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int[][] arr = { { 1, 3 }, { 2, 6 }, { 8, 10 }, { 15, 18 } };
        List<List<Integer>> ans = merge(arr);
        System.out.println("Merged Overlapping Intervals are ");
        for (List<Integer> it : ans) {
            System.out.println(it.get(0) + " " + it.get(1));
        }
    }
}

```

```
> Task :Merge_Overlapping_Sub_intervals.main()
Merged Overlapping Intervals are
1 6
8 10
15 18
```

Merge_two_sorted_arrays_without_extra_space

```
/*
 * Merge two Sorted Arrays Without Extra Space
   Problem statement: Given two sorted arrays arr1[] and arr2[] of sizes n and m in non-decreasing order.
    Merge them in sorted order. Modify arr1 so that it contains the first N elements and modify arr2
    so that it contains the last M elements.

   Examples:

   Example 1:
   Input:
   n = 4, arr1[] = [1 4 8 10]
   m = 5, arr2[] = [2 3 9]

   Output:
   arr1[] = [1 2 3 4]
   arr2[] = [8 9 10]

   Explanation:
   After merging the two non-decreasing arrays, we get, 1,2,3,4,8,9,10.

   Example2:

   Input:
   n = 4, arr1[] = [1 3 5 7]
   m = 5, arr2[] = [0 2 6 8 9]

   Output:
   arr1[] = [0 1 2 3]
   arr2[] = [5 6 7 8 9]

   Explanation:
   After merging the two non-decreasing arrays, we get, 0 1 2 3 5 6 7 8 9.
*/

```

```java
Codeium: Refactor | Explain
public class Merge_two_sorted_arrays_without_extra_space {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    static void swap(int a,int b)
    {
        int temp=a;
        a=b;
        b=temp;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void merge(int arr1[], int arr2[], int n, int m)
    {
        /*
         * // solution 1: using extra Space.
         * // Time complexity: O(n*log(n))+O(n)+O(n)) & Space complexity: O(N).
         * int[] temp = new int[m + n];
           int j = 0;
           for(int i = 0; i < n; i++){
               temp[j++] = arr1[i];
           }
           for(int i = 0; i < m; i++){
               temp[j++] = arr2[i];
           }
           Arrays.sort(temp);
           int u = 0;
           for(int i = 0; i < n; i++){
               arr1[i] = temp[u++];
           }
           for(int i = 0; i < m; i++){
               arr2[i] = temp[u++];
           }
        */

        /*
         * //Solution 2: using insertions sort:
         * // Time complexity: O(n*m) & Space complexity: O(1).
         * int k;
           for(int i = 0; i < n; i++){
               if(arr1[i] > arr2[0]){
                   int temp = arr1[i];
                   arr1[i] = arr2[0];
                   arr2[0] = temp;
               }
               int first = arr2[0];
               // insertion sort is used here
               for (k = 1; k < m && arr2[k] < first; k++) {
                arr2[k - 1] = arr2[k];
               }
               arr2[k - 1] = first;
           }
        */

    }
```

```
85
86          // Optimized Solution:
87          // Time complexity: O(n+m) & Space Complexity: O(1)
88          int gap =(int) Math.ceil((double)(n + m) / 2.0);
89          while (gap > 0) {
90            int i = 0;
91            int j = gap;
92            while (j < (n + m)) {
93              if (j < n && arr1[i] > arr1[j]) {
94                swap(arr1[i], arr1[j]);
95              } else if (j >= n && i < n && arr1[i] > arr2[j - n]) {
96                swap(arr1[i], arr2[j - n]);
97              } else if (j >= n && i >= n && arr2[i - n] > arr2[j - n]) {
98                swap(arr2[i - n], arr2[j - n]);
99              }
100             j++;
101             i++;
102           }
103           if (gap == 1) {
104             gap = 0;
105           } else {
106             gap =(int) Math.ceil((double) gap / 2.0);
107           }
108         }
109     }
        Codeium: Refactor | Explain | Generate Javadoc | ×
110     public static void main(String[] args) {
111         int arr1[] = {1,4,7,8,10};
112       int arr2[] = {2,3,9};
113         merge(arr1, arr2, arr1.length, arr2.length);
114         System.out.println(Arrays.toString(arr1));
115         System.out.println(Arrays.toString(arr2));
116     }
117 }
118
```

```
> Task :Merge_two_sorted_arrays_without_extra_space.main()
[1, 4, 7, 8, 10]
[2, 3, 9]
```

Program to generate Pascal's Triangle

```java
/*
 * Program to generate Pascal's Triangle
 Problem Statement: Given an integer N, return the first N rows of Pascal's triangle.

 In Pascal's triangle, each number is the sum of the two numbers
 directly above it as shown in the figure below:


 Example 1:

 Input Format: N = 5

 Result:
     1
    1 1
   1 2 1
  1 3 3 1
 1 4 6 4 1

 Explanation: There are 5 rows in the output matrix. Each row
 corresponds to each one of the rows in the image shown above.
 Example 2:

 Input Format: N = 1
 Result: 1
*/

import java.util.List;
import java.util.ArrayList;

Codeium: Refactor | Explain
public class Pascals_Triangle {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static List<List<Integer>> generate(int numRows) {
        List<List<Integer>> ans = new ArrayList<>();
        List<Integer> row, pre = null;
        for (int i = 0; i < numRows; i++) {
            row = new ArrayList<Integer>();
            for (int j = 0; j <= i; j++) {
                if (j == 0 || j == i) {
                    row.add(1);
                } else {
                    row.add(pre.get(j - 1) + pre.get(j));
                }
            }
            pre = row;
            ans.add(row);
        }
        return ans;
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int n = 5;
        System.out.println(generate(n));
    }
}
```

```
> Task :Pascals_Triangle.main()
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]
```

Reverse_Pairs

```
/*
 * Count Reverse Pairs
   Problem Statement: Given an array of numbers, you need to return the count of reverse pairs. Reverse Pairs are those
   pairs where i<j and arr[i]>2*arr[j].

   Examples:

   Example 1:
   Input: N = 5, array[] = {1,3,2,3,1)
   Output: 2
   Explanation: The pairs are (3, 1) and (3, 1) as from both the pairs the condition arr[i] > 2*arr[j] is satisfied.

   Example 2:
   Input: N = 4, array[] = {3,2,1,4}
   Output: 1
   Explaination: There is only 1 pair  ( 3 , 1 ) that satisfy the condition arr[i] > 2*arr[j]
*/


```

```java
public class Reverse_Pairs {
    public static int merge(int[] nums, int low, int mid, int high) {
        int cnt = 0;
        int j = mid + 1;
        for(int i = low;i<=mid;i++) {
            while(j<=high && nums[i] > (2 * (long) nums[j])) {
                j++;
            }
            cnt += (j - (mid+1));
        }

        ArrayList<Integer> temp = new ArrayList<>();
        int left = low, right = mid+1;
        while(left <= mid && right<=high) {
            if(nums[left]<=nums[right]) {
                temp.add(nums[left++]);
            }
            else {
                temp.add(nums[right++]);
            }
        }

        while(left<=mid) {
            temp.add(nums[left++]);
        }
        while(right<=high) {
            temp.add(nums[right++]);
        }

        for(int i = low; i<=high;i++) {
            nums[i] = temp.get(i - low);
        }
        return cnt;
    }
    public static int mergeSort(int[] nums, int low, int high) {
        if(low>=high) return 0;
        int mid = (low + high) / 2;
        int inv = mergeSort(nums, low, mid);
        inv += mergeSort(nums, mid+1, high);
        inv += merge(nums, low, mid, high);
        return inv;
    }
```

```java
    static int reversePairs(int arr[]) {
        /*
         * // BruteForce Approach: Time complexity: O(N^2) & Space complexity: O(1).
         * int Pairs = 0;
           for (int i = 0; i < arr.length; i++) {
               for (int j = i + 1; j < arr.length; j++) {
                   if (arr[i] > 2 * arr[j]){
                       Pairs++;
                   }
               }
           }
           return Pairs;
        */

        // solution 2: Optimized -> Time compelxity: O( N log N ) + O (N) + O (N) & Space compelxity: O(N).
        return mergeSort(arr, 0, arr.length - 1);
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int arr[] = { 1, 3, 2, 3, 1 };
        System.out.println("The Total Reverse Pairs are " + reversePairs(arr));
    }
}

```

```
> Task :Reverse_Pairs.main()
The Total Reverse Pairs are 2
```