Best_Time_to_Buy_and_Sell_Stock

```
/*
* Stock Buy And Sell
  Problem Statement: You are given an array of prices where prices[i] is the price of a given
  stock on an ith day.

  You want to maximize your profit by choosing a single day to buy one stock and choosing a
   different day in the
   future to sell that stock. Return the maximum profit you can achieve from this transaction.
    If you cannot achieve any profit, return 0.

  Examples:

  Example 1:
  Input: prices = [7,1,5,3,6,4]
  Output: 5
  Explanation: Buy on day 2 (price = 1) and
  sell on day 5 (price = 6), profit = 6-1 = 5.

  Note: That buying on day 2 and selling on day 1
  is not allowed because you must buy before
  you sell.

  Example 2:
  Input: prices = [7,6,4,3,1]
  Output: 0
  Explanation: In this case, no transactions are
  done and the max profit = 0.
*/
```

```java
Codeium: Refactor | Explain
public class Best_Time_to_Buy_and_Sell_Stock {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int maxProfit(int[] prices) {
        /*
         * BruteForce Approach: Find every possible profit and compare with the maxprofit and
         then set the maximum profit among them into maxprofit.
         * Time complexity: O(N^2) & Space complexity: O(1).
         * int maxProfit = 0;
           for(int i = 0; i < prices.length; i++){
               for(int j = i + 1; j < prices.length; j++){
                   if(prices[i] < prices[j]){
                       maxProfit = Math.max(maxProfit, prices[j] - prices[i]);
                   }
               }
           }
           return maxProfit;
        */

        // Optimized Solution: Time Complexity: O(N) & Space complexity: O(1)
        int minPrice = Integer.MAX_VALUE;
        int profit = 0;
        for(int i = 0; i < prices.length; i++){
            minPrice = Math.min(minPrice, prices[i]);
            profit = Math.max(profit, prices[i] - minPrice);
        }
        return profit;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int[] prices = {0, 1};
        int maxProfit = maxProfit(prices);
        System.out.println(maxProfit);
    }
}

```

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

Leaders_in_an_Array_problem

```java
/*
* Leaders in an Array
  Problem Statement: Given an array, print all the elements which are leaders. A Leader is
   an element that is greater than all of the elements on its right side in the array.

  Examples:

  Example 1:
  Input:
   arr = [4, 7, 1, 0]
  Output:
   7 1 0
  Explanation:
   Rightmost element is always a leader. 7 and 1 are greater than the elements in their right side.

  Example 2:
  Input:
   arr = [10, 22, 12, 3, 0, 6]
  Output:
   22 12 6
  Explanation:
   6 is a leader. In addition to that, 12 is greater than all the elements in its right side
   (3, 0, 6), also 22 is greater than 12, 3, 0, 6.
*/

```

```java
Codeium: Refactor | Explain
public class Leaders_in_an_Array_problem {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    static ArrayList<Integer> leaders(int arr[]){
        /*
         * BruteForce Approach: Time complexity: O(N^2) & Space complexity: O(1)
         * ArrayList<Integer> ans = new ArrayList<>();
           boolean leader = false;
           for(int i = 0; i < arr.length; i++){
               leader = false;
               for(int j = i + 1; j < arr.length; j++){
                   if(arr[i] < arr[j]){
                       leader = true;
                   }
               }
               if(!leader){
                   ans.add(arr[i]);
               }
           }
           return ans;
         */
        ArrayList<Integer> ans = new ArrayList<>();
        int max = arr[arr.length - 1];
        ans.add(max);
        for(int i = arr.length - 2; i >= 0; i--){
            if(max <= arr[i]){
                max = arr[i];
                ans.add(max);
            }
        }
        return ans;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int arr [] = {16,17,4,3,5,2};
        ArrayList<Integer> ans = leaders(arr);
        System.out.println(ans);
    }
}

```

```
> Task :Leaders_in_an_Array_problem.main()
[2, 5, 17]
```

Longest Consecutive Sequence in an Array

```java

/*
* Longest Consecutive Sequence in an Array
  Problem Statement: You are given an array of ‘N’ integers. You need to find the length of
  the longest sequence which contains the consecutive elements.

  Examples:

  Example 1:
  Input: [100, 200, 1, 3, 2, 4]
  Output: 4
  Explanation: The longest consecutive subsequence is 1, 2, 3, and 4.

  Example 2:
  Input: [3, 8, 5, 7, 6]
  Output: 4

  Explanation: The longest consecutive subsequence is 5, 6, 7, and 8.
*/

```

```java
Codeium: Refactor | Explain
public class Longest_Consecutive_Sequence_in_an_Array {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int longestConsecutive(int[] arr){
        if(arr.length == 0 || arr == null){
            return 0;
        }
        /*
         * Solution 1: Time complexity: O(NlogN) & Space Complexity: O(1).
         * Arrays.sort(arr);
           int ans = 1;
           int prev = arr[0];
           int curr = 1;
           for(int i = 1; i < arr.length; i++){
               if(arr[i] == prev + 1){
                   curr++;
               }
               else if(arr[i] != prev){
                   curr = 1;
               }
               prev = arr[i];
               ans = Math.max(ans, curr);
           }
           return ans;
        */
        HashSet<Integer> set = new HashSet<>();
        for(int i = 0; i < arr.length; i++){
            set.add(arr[i]);
        }
        int longestStreak = 0;
        for(int i = 0; i < arr.length; i++){
            if(!set.contains(arr[i] - 1)){
                int currNum = arr[i];
                int currentStreak = 1;
                while(set.contains(currNum + 1)){
                    currNum += 1;
                    currentStreak += 1;
                }
                longestStreak = Math.max(longestStreak, currentStreak);
            }
        }
        return longestStreak;
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int[] arr = {100, 102, 101, 1, 3, 2, 4};
        System.out.println(longestConsecutive(arr));
    }
}
```

```
> Task :Longest_Consecutive_Sequence_in_an_Array.main()
4
```

Majority Element that occurs more than N/2 times

```java
/*
* Find the Majority Element that occurs more than N/2 times
  Problem Statement: Given an array of N integers, write a program to return an element that occurs
   more than N/2 times in the given array. You may consider that such an element always exists in the array.

  Example 1:

  Input Format: N = 3, nums[] = {3,2,3}

  Result: 3

  Explanation: When we just count the occurrences of each number and compare with half of the size of the
  array, you will get 3 for the above solution.

*/
public class MajorityElement {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int majorityElement(int[] nums) {
        /*
        * BruteForce Approach: Time Complexity: O(N^2) & Space Complexity: O(1)
        * # In this solution we will find the frequency of each element and whichever frequency
        * is more than rest all and also more than n/2 that will be our answer
        * .(where n is the length of the array.)

        int maxFreq = 0;
        int ans = 0;
        for(int i = 0; i < nums.length; i++){
            int count = 0;
            for(int j = 0; j < nums.length; j++){
                if(nums[i] == nums[j]){
                    count++;
                }
            }
            if(maxFreq < count){
                maxFreq = count;
                ans = nums[i];
            }
        }
        if(maxFreq > nums.length/2){
            return ans;
        }
        else{
            return -1;
        }

        */

        /*Solution 2: if assume that the majority element always exists in the array,
         Then this could be our solution with
        Time Complexity: O(NlogN) & Space Complexity: O(1).

        Arrays.sort(nums);
        return nums[nums.length/2];

        */
```

```java
        /*
         * Solution 3: Using HashMap Time Complexity: O(NlogN) & Space Complexity: O(1),
          Time Complexity NlogN because it is unordered map.

        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);
            }
        }
        int maxFreq = 0, ans = 0;
        for(int value: map.keySet()){
            if(maxFreq < map.get(value)){
                maxFreq = map.get(value);
                ans = value;
            }
        }
        if(maxFreq > nums.length/2){
            return ans;
        }
        else{
            return -1;
        }

        // Solution 4: By Moore Voting Algorithm // this will only work if majority element exists.
        // Time Complexity: O(N) & Space Complexity: O(1)

        int count = 0, element = 0;
        for(int i = 0; i < nums.length; i++){
            if(count == 0){
                element = nums[i];
                count++;
            }
            else if(nums[i] != element){
                count--;
            }
            else{
                count++;
            }
        }
        return element;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int[] nums = {1, 2, 2, 1, 3, 1, 1};
        System.out.println(majorityElement(nums));
    }
}
```

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

Subarray_Sum_Equal_to_K

```
/*
 * Subarray with Given Sum
   Problem Statement: Subarray with Given Sum

   Given an array and a sum k, generate the subarray whose elements sum to k.

   Examples:

   Example 1:
   Input:
    arr = {1, 7, 3, 9}, k = 10

   Output: 7 3
   Explanation:
    Of all the subarrays, 7 and 3 sums to 10.

   Example 2:
   Input: arr = {2,1,3,4,5,6}, k = 10
   Output: 2 1 3 4
   Explanation: Of all the subarrays, 2, 1, 3 and 4 sums to 10

*/
```

```java
Codeium: Refactor | Explain
public class Subarray_Sum_Equal_to_K {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void subArrWithSumK(int nums[], int k)
    {
        /*Bruteforce Approach: Time complexity: O(N^2) & Space Complexity: O(1).
        for(int i = 0; i < nums.length; i++){
            int sum = 0;
            for(int j = i; j < nums.length; j++){
                sum += nums[j];
                if(sum == k){
                    for(int m = i; m <= j; m++){
                        System.out.print(nums[m] + " ");
                    }
                    System.out.println();
                }
            }
        }
        */

        // Optimized Solution: Time Complexity: O(N) to find subarray and
        O(N) to print subarray & Space complexity: O(1)
        int start = 0, end = -1, sum = 0;
        while (start < nums.length) {
          while ((end + 1 < nums.length) && (sum + nums[end + 1] <= k)){
            sum += nums[++end];
          }
          if (sum == k) {
            for (int p = start; p <= end; p++)
              System.out.print(nums[p] + " ");
            System.out.println();
          }

          sum -= nums[start];
          start++;
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int[] nums = { 1, 9, 3, 7 };
        int k = 10;
        subArrWithSumK(nums, k);
    }
}
```

```
> Task :Subarray_Sum_Equal_to_K.main()
1 9
3 7
```

Maximum_Subarray_Sum

```java
/*
* Kadane's Algorithm : Maximum Subarray Sum in an Array
  Problem Statement: Given an integer array arr, find the contiguous subarray (containing at least one number) which
  has the largest sum and return its sum and print the subarray.

  Examples:

  Example 1:
  Input: arr = [-2,1,-3,4,-1,2,1,-5,4]
  Output: 6
  Explanation: [4,-1,2,1] has the largest sum = 6.

  Examples 2:
  Input: arr = [1]
  Output: 1
  Explanation: Array has only one element and which is giving positive sum of 1.

*/
public class Maximum_Subarray_Sum {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int maxSubArray(int[] nums) {
        /*
         * BruteForce Solution: --> Time Complexity: O(N^2) & Space Complexity: O(1):
         * In this we find all possible subarrays sum and then find the maximum among them.
         * int maxSum = Integer.MIN_VALUE;
           for(int i = 0; i < nums.length; i++){
               int sum = 0;
               for(int j = i; j < nums.length; j++){
                   sum += nums[j];
                   maxSum = Math.max(maxSum, sum);
               }
           }
           return maxSum;

        */

        // Optimized Solution: Using Kadane's algorithm
        // Time complexity: O(N) & Space Complexity: O(1).

        int maxSum = Integer.MIN_VALUE;
        int currSum = 0;
        for(int i = 0; i < nums.length; i++){
            currSum += nums[i];
            if(maxSum < currSum){
                maxSum = currSum;
            }
            if(currSum < 0){
                currSum = 0;
            }
        }
        return maxSum;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int[] nums = {-2,1,-3,4,-1,2,1,-5,4};
        System.out.println(maxSubArray(nums));
    }
}
```

```
> Task :Maximum_Subarray_Sum.main()
6
```

Spiral Traversal of Matrix

```
/*
 * Spiral Traversal of Matrix
 Problem Statement: Given a Matrix, print the given matrix in spiral order.

 Examples:

 Example 1:
 Input: Matrix[][] = { { 1, 2, 3, 4 },
                      { 5, 6, 7, 8 },
                      { 9, 10, 11, 12 },
                      { 13, 14, 15, 16 } }

 Outhput: 1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10.
 Explanation: The output of matrix in spiral form.

 Example 2:
 Input: Matrix[][] = { { 1, 2, 3 },
                      { 4, 5, 6 },
                      { 7, 8, 9 } }

 Output: 1, 2, 3, 6, 9, 8, 7, 4, 5.
 Explanation: The output of matrix in spiral form.
*/
```

```java
Codeium: Refactor | Explain
public class Print_the_matrix_in_spiral_manner {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static List<Integer> spiralOrder(int[][] matrix) {
        int top = 0;
        int bottom = matrix.length - 1;
        int left = 0;
        int right = matrix[0].length - 1;
        int dir = 0;
        List<Integer> ans = new ArrayList<>();
        while(top <= bottom && left <= right){
            if(dir == 0){
                for(int i = left; i <= right; i++){
                    ans.add(matrix[top][i]);
                }
                top++;
            }
            else if(dir == 1){
                for(int i = top; i <= bottom; i++){
                    ans.add(matrix[i][right]);
                }
                right--;
            }
            else if(dir == 2){
                for(int i = right; i >= left; i--){
                    ans.add(matrix[bottom][i]);
                }
                bottom--;
            }
            else if(dir == 3){
                for(int i = bottom; i >= top; i--){
                    ans.add(matrix[i][left]);
                }
                left++;
            }
            dir = (dir + 1) % 4;
        }
        return ans;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        List<Integer> ans = spiralOrder(matrix);
        System.out.println(ans);
    }
}

```

```
> Task :Print_the_matrix_in_spiral_manner.main()
[1, 2, 3, 6, 9, 8, 7, 4, 5]
```

Rearrange_Array_Elements_by_Sign_Positive_Negative

```java
public class Rearrange_Array_Elements_by_Sign_Positive_Negative {
    public static int[] rearrangeArray(int[] nums) {
        /*
         * Brute Force Solution: Time Complexity: O(N) & Space complexity: O(N)
         * int[] ans = new int[nums.length];
           int k = 0;
           for(int i = 0; i < nums.length; i++){
               if(nums[i] > 0){
                   ans[k] = nums[i];
                   k+=2;
               }
           }
           int m = 1;
           for(int i = 0; i < nums.length; i++){
               if(nums[i] < 0){
                   ans[m] = nums[i];
                   m+=2;
               }
           }
           return ans;
        */

        // Solution 2: Using one loop: Time Complexity: O(N) & Space complexity: O(N)
        int[] ans = new int[nums.length];
        int k = 0;
        int j = 1;
        for(int i = 0; i < nums.length; i++){
            if(nums[i] > 0){
                ans[k] = nums[i];
                k+=2;
            }
            else{
                ans[j] = nums[i];
                j+=2;
            }
        }
        return ans;
    }
    public static void main(String[] args) {
        int[] nums = {3,1,-2,-5,2,-4};
        int[] ans = rearrangeArray(nums);
        System.out.println(Arrays.toString(ans));
    }
}
```

```
> Task :Rearrange_Array_Elements_by_Sign_Positive_Negative.main()
[3, -2, 1, -5, 2, -4]
```

Rotate_Image_by_90_degree

```
/*
 * Rotate Image by 90 degree
   Problem Statement: Given a matrix, your task is to rotate the matrix 90 degrees clockwise.

   Note: Rotate matrix 90 degrees anticlockwise

   Examples:

   Example 1:
   Input: [[1,2,3],[4,5,6],[7,8,9]]
   Output: [[7,4,1],[8,5,2],[9,6,3]]
   Explanation: Rotate the matrix simply by 90 degree clockwise and return the matrix.

   Example 2:
   Input: [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]
   Output:[[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]
   Explanation: Rotate the matrix simply by 90 degree clockwise and return the matrix
 */


```

```java
public class Rotate_Image_by_90_degree {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void rotate(int[][] matrix) {
        /*
        * BruteForce Approach: we will insert our each row of original array in the temp array from last column
        to first and then finally we will copy the temp array into the original array.
        * Time complexity: O(N^2) & Space complexity: O(N^2)
        *
        * int m = matrix.length;
          int n = matrix[0].length;
          int[][] temp = new int[m][n];
          int row = 0;
          int k = n - 1;
          while(row < m){
              for(int i = 0; i < m; i++){
                  temp[i][k] = matrix[row][i];
              }
              row++;
              k--;
          }
          for(int i = 0; i < m; i++){
              for(int j = 0; j < n; j++){
                  matrix[i][j] = temp[i][j];
              }
          }
        */
        // Solution 2: Time complexity: O(N^2) & Space complexity: O(1)
        for(int i = 0; i < matrix.length; i++){
            for(int j = i; j < matrix[0].length; j++){
                int temp = matrix[i][j];
                matrix[i][j] = matrix[j][i];
                matrix[j][i] = temp;
            }
        }
        for(int i = 0; i < matrix.length; i++){
            for(int j = 0; j < matrix[0].length/2; j++){
                int temp = matrix[i][j];
                matrix[i][j] = matrix[i][matrix.length - 1 - j];
                matrix[i][matrix.length - 1 - j] = temp;
            }
        }
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        rotate(matrix);
        for(int row = 0; row < matrix.length; row++){
            for(int col = 0; col < matrix[row].length; col++){
                System.out.print(matrix[row][col] + " ");
            }
            System.out.println();
        }
    }
}
```

```
> Task :Rotate_Image_by_90_degree.main()
7 4 1
8 5 2
9 6 3
```

Set Matrix Zero

```java
/*
 * Set Matrix Zero
   Problem Statement: Given a matrix if an element in the matrix is 0 then you will have to set its entire column and row to 0 and then return the matrix.

   Examples:

   Examples 1:
   Input: matrix=[[1,1,1],[1,0,1],[1,1,1]]
   Output: [[1,0,1],[0,0,0],[1,0,1]]
   Explanation: Since matrix[2][2]=0.Therfore the 2nd column and 2nd row wil be set to 0.

   Examples 2:
   Input: matrix=[[0,1,2,0],[3,4,5,2],[1,3,1,5]]
   Output:[[0,0,0,0],[0,4,5,0],[0,3,1,0]]
   Explanation:Since matrix[0][0]=0 and matrix[0][3]=0. Therefore 1st row, 1st column and 4th column will be set to 0
*/

public class Set_Matrix_Zeroes {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void setZeroes(int[][] matrix) {
        /*
         * BruteForce Approach: Using Extra matrix which is the copy of original matrix.
         * Time Complexity: O((N*M)*(N + M)). O(N*M)) & Space complexity: O(m * n)
         * int m = matrix.length;
           int n = matrix[0].length;
           int[][] temp = new int[m][n];
           for(int row = 0; row < matrix.length; row++){
               for(int col = 0; col < matrix[row].length; col++){
                   temp[row][col] = matrix[row][col];
               }
           }
           for(int row = 0; row < matrix.length; row++){
               for(int col = 0; col < matrix[row].length; col++){
                   if(matrix[row][col] == 0){
                       int ind = row - 1;
                       while(ind >= 0){
                           if(matrix[ind][col] != 0){
                               temp[ind][col] = 0;
                           }
                           ind--;
                       }
                       ind = row + 1;
                       while(ind < matrix.length){
                           if(matrix[ind][col] != 0){
                               temp[ind][col] = 0;
                           }
                           ind++;
                       }
                       ind = col - 1;
                       while(ind >= 0){
                           if(matrix[row][ind] != 0){
                               temp[row][ind] = 0;
                           }
                           ind--;
                       }
                       ind = col + 1;
                       while(ind < matrix[row].length){
                           if(matrix[row][ind] != 0){
                               temp[row][ind] = 0;
                           }
                           ind++;
                       }
                   }
               }
           }

           for(int row = 0; row < matrix.length; row++){
               for(int col = 0; col < matrix[row].length; col++){
                   if(matrix[row][col] == -1){
                       matrix[row][col] = 0;
                   }
               }
           }
           for(int row = 0; row < matrix.length; row++){
               for(int col = 0; col < matrix[row].length; col++){
                   matrix[row][col] = temp[row][col];
               }
           }
        */
```

```java
        // Optimized Approach: Time complexity: O(2*(N*M)) & Space complexity: O(1)
        int m = matrix.length;
        int n = matrix[0].length;
        int x = 1;
        int y = 1;
        for(int j = 0; j < n; j++){
            if(matrix[0][j] == 0){
                x = 0;
            }
        }
        for(int i = 0; i < m; i++){
            if(matrix[i][0] == 0){
                y = 0;
            }
        }
        for(int i = 1; i < m; i++){
            for(int j = 1; j < n; j++){
                if(matrix[i][j] == 0){
                    matrix[i][0] = 0;
                    matrix[0][j] = 0;
                }
            }
        }
        for(int j = 1; j < n; j++){
            if(matrix[0][j] == 0){
                for(int i = 0; i < m; i++){
                    matrix[i][j] = 0;
                }
            }
        }
        for(int i = 1; i < m; i++){
            if(matrix[i][0] == 0){
                for(int j = 0; j < n; j++){
                    matrix[i][j] = 0;
                }
            }
        }
        if(y == 0){
            for(int i = 0; i < m; i++){
                matrix[i][0] = 0;
            }
        }
        if(x == 0){
            for(int j = 0; j < n; j++){
                matrix[0][j] = 0;
            }
        }
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int[][] matrix = {{0, 1, 2, 0}, {3, 4, 5, 2}, {1, 3, 1, 5}};
        setZeroes(matrix);
        for(int row = 0; row < matrix.length; row++){
            for(int col = 0; col < matrix[row].length; col++){
                System.out.print(matrix[row][col] + " ");
            }
            System.out.println();
        }
    }
}

```

```
> Task :Set_Matrix_Zeroes.main()
0 0 0 0
0 4 5 0
0 3 1 0
```

Sort an array of 0s, 1s and 2s

```java
/*
* Sort an array of 0s, 1s and 2s
Problem Statement: Given an array consisting of only 0s, 1s and 2s. Write a program to in-place sort the array
without using inbuilt sort functions. ( Expected: Single pass-O(N) and constant space)

Example 1:

Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]

Input: nums = [2,0,1]
Output: [0,1,2]

Input: nums = [0]
Input: nums = [0]

*/

public class SortColors {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void swap(int[] nums, int low, int high){
        int temp = nums[low];
        nums[low] = nums[high];
        nums[high] = temp;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void sortColors(int[] nums) {
        /*
        * Brute Force Approach: Using sorting--> Time Complexity: O(NlogN) & Space Complexity: O(1)
        * Arrays.sort(nums);
        *
        */

        /*
        Solution 2: Using Count Sort --> Time Complexity: O(N) & Space Complexity: O(1)
        int countZero = 0, countOne = 0, countTwo = 0;
        for(int i = 0; i < nums.length; i++){
            if(nums[i] == 0){
                countZero++;
            }
            else if(nums[i] == 1){
                countOne++;
            }
            else{
                countTwo++;
            }
        }
        int k = 0;
        while(k < countZero){
            nums[k++] = 0;
        }
        while(k < (countOne + countZero)){
            nums[k++] = 1;
        }
        while(k < nums.length){
            nums[k++] = 2;
        }
        */
```

```
            // Optimized Approach: "Dutch National Flag problem" Time Complexity: O(N) & Space Complexity: O(1)
            int low = 0;
            int mid = 0;
            int high = nums.length - 1;
            while(mid <= high){
                switch(nums[mid]){
                    case 0: swap(nums, low++, mid++);
                    break;
                    case 1: mid++;
                    break;
                    case 2: swap(nums, high--, mid);
                    break;
                }
            }
        }
Codeium: Refactor | Explain | Generate Javadoc | ×
        public static void main(String[] args) {
            int[] nums = {2,0,2,1,1,0};
            sortColors(nums);
            System.out.println(Arrays.toString(nums));
        }
    }

```

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

Two Sum : Check if a pair with given sum exists in Array

```
/*
* Two Sum : Check if a pair with given sum exists in Array
  Problem Statement: Given an array of integers nums[] and an integer target, return indices of the
   two numbers such that their sum is equal to the target.

  Note: Assume that there is exactly one solution, and you are not allowed to use the same element twice.
   Example: If target is equal to 6 and num[1] = 3, then nums[1] + nums[1] = target is not a solution.

  Example 1:

  Input: nums = [2,7,11,15], target = 9

  Output: [0,1]

  Explanation: Because nums[0] + nums[1] == 9,
  which is the required target, we return
  indexes [0,1]. (0-based indexing)
  Example 2:

  Input Format: nums = [3,2,4,6], target = 6

  Output: [1,2]

  Explanation: Because nums[1] + nums[2] == 6,
  which is the required target, we return
  indexes [1,2].

*/
```

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

        */
        int[] ans = new int[2];
        HashMap<Integer, Integer> map = new HashMap<>();
        for(int i = 0; i < nums.length; i++){
            if(map.containsKey(target - nums[i])){
                ans[1] = i;
                ans[0] = map.get(target - nums[i]);
                return ans;
            }
            map.put(nums[i], i);
        }
        return ans;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        int[] nums = {2,7,11,15};
        int target = 9;
        int[] ans = findTwoSum(nums, target);
        System.out.println(Arrays.toString(ans));
    }
}

```

```
> Task :TwoSum.main()
[0, 1]
```