

## Best\_Time\_to\_Buy\_and\_Sell\_Stock

```
1  ✓ /*
2  ✓  * Stock Buy And Sell
3  Problem Statement: You are given an array of prices where prices[i] is the price of a given
4  stock on an ith day.
5
6  ✓  You want to maximize your profit by choosing a single day to buy one stock and choosing a
7  different day in the
8  ✓  future to sell that stock. Return the maximum profit you can achieve from this transaction.
9  | If you cannot achieve any profit, return 0.
10
11  Examples:
12
13  Example 1:
14  Input: prices = [7,1,5,3,6,4]
15  Output: 5
16  Explanation: Buy on day 2 (price = 1) and
17  sell on day 5 (price = 6), profit = 6-1 = 5.
18
19  Note: That buying on day 2 and selling on day 1
20  is not allowed because you must buy before
21  you sell.
22
23  Example 2:
24  Input: prices = [7,6,4,3,1]
25  Output: 0
26  Explanation: In this case, no transactions are
27  done and the max profit = 0.
28  */
29
30
```

```

31 Codeium: Refactor | Explain
   public class Best_Time_to_Buy_and_Sell_Stock {
32       Codeium: Refactor | Explain | Generate Javadoc | X
       public static int maxProfit(int[] prices) {
33           /*
34               * BruteForce Approach: Find every possible profit and compare with the maxprofit and
35               * then set the maximum profit among them into maxprofit.
36               * Time complexity: O(N^2) & Space complexity: O(1).
37               * int maxProfit = 0;
38               for(int i = 0; i < prices.length; i++){
39                   for(int j = i + 1; j < prices.length; j++){
40                       if(prices[i] < prices[j]){
41                           maxProfit = Math.max(maxProfit, prices[j] - prices[i]);
42                       }
43                   }
44               }
45               return maxProfit;
46           */
47
48           // Optimized Solution: Time Complexity: O(N) & Space complexity: O(1)
49           int minPrice = Integer.MAX_VALUE;
50           int profit = 0;
51           for(int i = 0; i < prices.length; i++){
52               minPrice = Math.min(minPrice, prices[i]);
53               profit = Math.max(profit, prices[i] - minPrice);
54           }
55           return profit;
56       }
57       Codeium: Refactor | Explain | Generate Javadoc | X
       public static void main(String[] args) {
58           int[] prices = {0, 1};
59           int maxProfit = maxProfit(prices);
60           System.out.println(maxProfit);
61       }
62     }
63

```

> Task :Best\_Time\_to\_Buy\_and\_Sell\_Stock.main()

1

## Leaders\_in\_an\_Array\_problem

```

1  /*
2  * Leaders in an Array
3  * Problem Statement: Given an array, print all the elements which are leaders. A Leader is
4  * an element that is greater than all of the elements on its right side in the array.
5
6  * Examples:
7
8  * Example 1:
9  * Input:
10     arr = [4, 7, 1, 0]
11  * Output:
12     7 1 0
13  * Explanation:
14     Rightmost element is always a leader. 7 and 1 are greater than the elements in their right side.
15
16  * Example 2:
17  * Input:
18     arr = [10, 22, 12, 3, 0, 6]
19  * Output:
20     22 12 6
21  * Explanation:
22     6 is a leader. In addition to that, 12 is greater than all the elements in its right side
23     (3, 0, 6), also 22 is greater than 12, 3, 0, 6.
24  */
25

```

```

29 Codeium: Refactor | Explain
public class Leaders_in_an_Array_problem {
30 Codeium: Refactor | Explain | Generate Javadoc | X
    static ArrayList<Integer> leaders(int arr[]){
31         /*
32          * BruteForce Approach: Time complexity: O(N^2) & Space complexity: O(1)
33          * ArrayList<Integer> ans = new ArrayList<>();
34          boolean leader = false;
35          for(int i = 0; i < arr.length; i++){
36              leader = false;
37              for(int j = i + 1; j < arr.length; j++){
38                  if(arr[i] < arr[j]){
39                      leader = true;
40                  }
41              }
42              if(!leader){
43                  ans.add(arr[i]);
44              }
45          }
46          return ans;
47          */
48          ArrayList<Integer> ans = new ArrayList<>();
49          int max = arr[arr.length - 1];
50          ans.add(max);
51          for(int i = arr.length - 2; i >= 0; i--){
52              if(max <= arr[i]){
53                  max = arr[i];
54                  ans.add(max);
55              }
56          }
57          return ans;
58      }
59 Codeium: Refactor | Explain | Generate Javadoc | X
    public static void main(String[] args) {
60         int arr [] = {16,17,4,3,5,2};
61         ArrayList<Integer> ans = leaders(arr);
62         System.out.println(ans);
63     }
64 }
65

```

> Task :Leaders\_in\_an\_Array\_problem.main()  
[2, 5, 17]

## Longest Consecutive Sequence in an Array

```

1
2  ✓ /*
3  ✓ * Longest Consecutive Sequence in an Array
4      Problem Statement: You are given an array of 'N' integers. You need to find the length of
5      the longest sequence which contains the consecutive elements.
6
7      Examples:
8
9      Example 1:
10     Input: [100, 200, 1, 3, 2, 4]
11     Output: 4
12     Explanation: The longest consecutive subsequence is 1, 2, 3, and 4.
13
14     Example 2:
15     Input: [3, 8, 5, 7, 6]
16     Output: 4
17
18     Explanation: The longest consecutive subsequence is 5, 6, 7, and 8.
19     */
20
21

```

Codeium: Refactor | Explain

```
25 public class Longest_Consecutive_Sequence_in_an_Array {
26     Codeium: Refactor | Explain | Generate Javadoc | ✕
27     public static int longestConsecutive(int[] arr){
28         if(arr.length == 0 || arr == null){
29             return 0;
30         }
31         /*
32          * Solution 1: Time complexity: O(NlogN) & Space Complexity: O(1).
33          * Arrays.sort(arr);
34          int ans = 1;
35          int prev = arr[0];
36          int curr = 1;
37          for(int i = 1; i < arr.length; i++){
38              if(arr[i] == prev + 1){
39                  curr++;
40              }
41              else if(arr[i] != prev){
42                  curr = 1;
43              }
44              prev = arr[i];
45              ans = Math.max(ans, curr);
46          }
47          return ans;
48          */
49          HashSet<Integer> set = new HashSet<>();
50          for(int i = 0; i < arr.length; i++){
51              set.add(arr[i]);
52          }
53          int longestStreak = 0;
54          for(int i = 0; i < arr.length; i++){
55              if(!set.contains(arr[i] - 1)){
56                  int currNum = arr[i];
57                  int currentStreak = 1;
58                  while(set.contains(currNum + 1)){
59                      currNum += 1;
60                      currentStreak += 1;
61                  }
62                  longestStreak = Math.max(longestStreak, currentStreak);
63              }
64          }
65          return longestStreak;
66      }
67  }
```

Codeium: Refactor | Explain | Generate Javadoc | ✕

```
56 public static void main(String[] args) {
57     int[] arr = {100, 102, 101, 1, 3, 2, 4};
58     System.out.println(longestConsecutive(arr));
59 }
70 }
```

> Task :Longest\_Consecutive\_Sequence\_in\_an\_Array.main()

4

## Majority Element that occurs more than $N/2$ times

```
7  /*
8  * Find the Majority Element that occurs more than  $N/2$  times
9  Problem Statement: Given an array of  $N$  integers, write a program to return an element that occurs
10 more than  $N/2$  times in the given array. You may consider that such an element always exists in the array.
11
12 Example 1:
13
14 Input Format:  $N = 3$ ,  $\text{nums}[] = \{3,2,3\}$ 
15
16 Result: 3
17
18 Explanation: When we just count the occurrences of each number and compare with half of the size of the
19 array, you will get 3 for the above solution.
20
21 */
```

```
26 public class MajorityElement {
27     Codeium: Refactor | Explain | Generate Javadoc | X
28     public static int majorityElement(int[] nums) {
29         /*
30          * BruteForce Approach: Time Complexity:  $O(N^2)$  & Space Complexity:  $O(1)$ 
31          * # In this solution we will find the frequency of each element and whichever frequency
32          * is more than rest all and also more than  $n/2$  that will be our answer
33          * |(where  $n$  is the length of the array.)
34
35         int maxFreq = 0;
36         int ans = 0;
37         for(int i = 0; i < nums.length; i++){
38             int count = 0;
39             for(int j = 0; j < nums.length; j++){
40                 if(nums[i] == nums[j]){
41                     count++;
42                 }
43             }
44             if(maxFreq < count){
45                 maxFreq = count;
46                 ans = nums[i];
47             }
48         }
49         if(maxFreq > nums.length/2){
50             return ans;
51         }
52         else{
53             return -1;
54         }
55
56         /*
57
58         /*Solution 2: if assume that the majority element always exists in the array,
59         Then this could be our solution with
60         Time Complexity:  $O(N\log N)$  & Space Complexity:  $O(1)$ .
61
62         Arrays.sort(nums);
63         return nums[nums.length/2];
64
65         */
66     }
67 }
```

```

67      * Solution 3: Using HashMap Time Complexity: O(NlogN) & Space Complexity: O(1),
68      Time Complexity NlogN because it is unordered map.
69
70      HashMap<Integer, Integer> map = new HashMap<>();
71      for(int i = 0; i < nums.length; i++){
72          if(map.containsKey(nums[i])){
73              map.put(nums[i], map.get(nums[i]) + 1);
74          }
75          else{
76              map.put(nums[i], 1);
77          }
78      }
79      int maxFreq = 0, ans = 0;
80      for(int value: map.keySet()){
81          if(maxFreq < map.get(value)){
82              maxFreq = map.get(value);
83              ans = value;
84          }
85      }
86      if(maxFreq > nums.length/2){
87          return ans;
88      }
89      else{
90          return -1;
91      }
92
93
94      // Solution 4: By Moore Voting Algorithm // this will only work if majority element exists.
95      // Time Complexity: O(N) & Space Complexity: O(1)
96
97      int count = 0, element = 0;
98      for(int i = 0; i < nums.length; i++){
99          if(count == 0){
100              element = nums[i];
101              count++;
102          }
103          else if(nums[i] != element){
104              count--;
105          }
106          else{
107              count++;
108          }
109      }
110      return element;
111
112      Codeium: Refactor | Explain | Generate Javadoc | ✕
113      public static void main(String[] args) {
114          int[] nums = {1, 2, 2, 1, 3, 1, 1};
115          System.out.println(majorityElement(nums));
116      }
117  }

```

> Task :MajorityElement.main()

1

## Subarray\_Sum\_Equal\_to\_K

```
7  /*
8  * Subarray with Given Sum
9  | Problem Statement: Subarray with Given Sum
10 |
11 | Given an array and a sum k, generate the subarray whose elements sum to k.
12 |
13 | Examples:
14 |
15 | Example 1:
16 | Input:
17 |   arr = {1, 7, 3, 9}, k = 10
18 |
19 | Output: 7 3
20 | Explanation:
21 |   Of all the subarrays, 7 and 3 sums to 10.
22 |
23 | Example 2:
24 | Input: arr = {2,1,3,4,5,6}, k = 10
25 | Output: 2 1 3 4
26 | Explanation: Of all the subarrays, 2, 1, 3 and 4 sums to 10
27 |
28 | */
29
```

```

30 public class Subarray_Sum_Equal_to_K {
31     public static void subArrWithSumK(int nums[], int k)
32     {
33         /*Bruteforce Approach: Time complexity:  $O(N^2)$  & Space Complexity:  $O(1)$ .
34         for(int i = 0; i < nums.length; i++){
35             int sum = 0;
36             for(int j = i; j < nums.length; j++){
37                 sum += nums[j];
38                 if(sum == k){
39                     for(int m = i; m <= j; m++){
40                         System.out.print(nums[m] + " ");
41                     }
42                     System.out.println();
43                 }
44             }
45         }
46         */
47
48         // Optimized Solution: Time Complexity:  $O(N)$  to find subarray and
49          $O(N)$  to print subarray & Space complexity:  $O(1)$ 
50         int start = 0, end = -1, sum = 0;
51         while (start < nums.length) {
52             while ((end + 1 < nums.length) && (sum + nums[end + 1] <= k)){
53                 sum += nums[++end];
54             }
55             if (sum == k) {
56                 for (int p = start; p <= end; p++)
57                     System.out.print(nums[p] + " ");
58                 System.out.println();
59             }
60
61             sum -= nums[start];
62             start++;
63         }
64     }
65
66     public static void main(String[] args) {
67         int[] nums = { 1, 9, 3, 7 };
68         int k = 10;
69         subArrWithSumK(nums, k);
70     }
71 }
72

```

> Task :Subarray\_Sum\_Equal\_to\_K.main()

1 9

3 7

Maximum\_Subarray\_Sum



```

/*
 * 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.
 */

```

```

27 public class Maximum_Subarray_Sum {
    Codeium: Refactor | Explain | Generate Javadoc | X
28     public static int maxSubArray(int[] nums) {
29         /*
30          * BruteForce Solution: --> Time Complexity: O(N^2) & Space Complexity: O(1):
31          * In this we find all possible subarrays sum and then find the maximum among them.
32          * int maxSum = Integer.MIN_VALUE;
33          * for(int i = 0; i < nums.length; i++){
34              int sum = 0;
35              for(int j = i; j < nums.length; j++){
36                  sum += nums[j];
37                  maxSum = Math.max(maxSum, sum);
38              }
39          }
40          return maxSum;
41
42          */
43
44          // Optimized Solution: Using Kadane's algorithm
45          // Time complexity: O(N) & Space Complexity: O(1).
46
47          int maxSum = Integer.MIN_VALUE;
48          int currSum = 0;
49          for(int i = 0; i < nums.length; i++){
50              currSum += nums[i];
51              if(maxSum < currSum){
52                  maxSum = currSum;
53              }
54              if(currSum < 0){
55                  currSum = 0;
56              }
57          }
58          return maxSum;
59      }
    Codeium: Refactor | Explain | Generate Javadoc | X
60     public static void main(String[] args) {
61         int[] nums = {-2,1,-3,4,-1,2,1,-5,4};
62         System.out.println(maxSubArray(nums));
63     }
64 }

```

> 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 } }

Output: 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.

\*/

Codeium: Refactor | Explain

```
28 public class Print_the_matrix_in_spiral_manner {
    Codeium: Refactor | Explain | Generate Javadoc | X
29     public static List<Integer> spiralOrder(int[][] matrix) {
30         int top = 0;
31         int bottom = matrix.length - 1;
32         int left = 0;
33         int right = matrix[0].length - 1;
34         int dir = 0;
35         List<Integer> ans = new ArrayList<>();
36         while(top <= bottom && left <= right){
37             if(dir == 0){
38                 for(int i = left; i <= right; i++){
39                     ans.add(matrix[top][i]);
40                 }
41                 top++;
42             }
43             else if(dir == 1){
44                 for(int i = top; i <= bottom; i++){
45                     ans.add(matrix[i][right]);
46                 }
47                 right--;
48             }
49             else if(dir == 2){
50                 for(int i = right; i >= left; i--){
51                     ans.add(matrix[bottom][i]);
52                 }
53                 bottom--;
54             }
55             else if(dir == 3){
56                 for(int i = bottom; i >= top; i--){
57                     ans.add(matrix[i][left]);
58                 }
59                 left++;
60             }
61             dir = (dir + 1) % 4;
62         }
63         return ans;
64     }
    Codeium: Refactor | Explain | Generate Javadoc | X
65     public static void main(String[] args) {
66         int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
67         List<Integer> ans = spiralOrder(matrix);
68         System.out.println(ans);
69     }
70 }
71
```

> Task :Print\_the\_matrix\_in\_spiral\_manner.main()  
[1, 2, 3, 6, 9, 8, 7, 4, 5]

## Rearrange\_Array\_Elements\_by\_Sign\_Positive\_Negative

```
Codeium: Refactor | Explain | Generate Javadoc | X
3 public class Rearrange_Array_Elements_by_Sign_Positive_Negative {
4     public static int[] rearrangeArray(int[] nums) {
5         /*
6          * Brute Force Solution: Time Complexity: O(N) & Space complexity: O(N)
7          * int[] ans = new int[nums.length];
8          * int k = 0;
9          * for(int i = 0; i < nums.length; i++){
10              if(nums[i] > 0){
11                  ans[k] = nums[i];
12                  k+=2;
13              }
14          }
15          * int m = 1;
16          * for(int i = 0; i < nums.length; i++){
17              if(nums[i] < 0){
18                  ans[m] = nums[i];
19                  m+=2;
20              }
21          }
22          * return ans;
23          */
24
25          // Solution 2: Using one loop: Time Complexity: O(N) & Space complexity: O(N)
26          int[] ans = new int[nums.length];
27          int k = 0;
28          int j = 1;
29          for(int i = 0; i < nums.length; i++){
30              if(nums[i] > 0){
31                  ans[k] = nums[i];
32                  k+=2;
33              }
34              else{
35                  ans[j] = nums[i];
36                  j+=2;
37              }
38          }
39          return ans;
40      }
41      public static void main(String[] args) {
42          int[] nums = {3,1,-2,-5,2,-4};
43          int[] ans = rearrangeArray(nums);
44          System.out.println(Arrays.toString(ans));
45      }
46  }
47
```

Codeium: Refactor | Explain | Generate Javadoc | X

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

## Rotate\_Image\_by\_90\_degree

```
1  ✓ /*
2  ✓  * Rotate Image by 90 degree
3      Problem Statement: Given a matrix, your task is to rotate the matrix 90 degrees clockwise.
4
5      Note: Rotate matrix 90 degrees anticlockwise
6
7      Examples:
8
9      Example 1:
10     Input: [[1,2,3],[4,5,6],[7,8,9]]
11     Output: [[7,4,1],[8,5,2],[9,6,3]]
12     Explanation: Rotate the matrix simply by 90 degree clockwise and return the matrix.
13
14     Example 2:
15     Input: [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]
16     Output: [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]
17     Explanation: Rotate the matrix simply by 90 degree clockwise and return the matrix
18  */
19
20
```

```

21 public class Rotate_Image_by_90_degree {
    Codeium: Refactor | Explain | Generate Javadoc | X
22     public static void rotate(int[][] matrix) {
23         /*
24          * BruteForce Approach: we will insert our each row of original array in the temp array from last column
25          * to first and then finally we will copy the temp array into the original array.
26          * Time complexity:  $O(N^2)$  & Space complexity:  $O(N^2)$ 
27          */
28         int m = matrix.length;
29         int n = matrix[0].length;
30         int[][] temp = new int[m][n];
31         int row = 0;
32         int k = n - 1;
33         while(row < m){
34             for(int i = 0; i < m; i++){
35                 temp[i][k] = matrix[row][i];
36             }
37             row++;
38             k--;
39         }
40         for(int i = 0; i < m; i++){
41             for(int j = 0; j < n; j++){
42                 matrix[i][j] = temp[i][j];
43             }
44         }
45     }
46     // Solution 2: Time complexity:  $O(N^2)$  & Space complexity:  $O(1)$ 
47     for(int i = 0; i < matrix.length; i++){
48         for(int j = i; j < matrix[0].length; j++){
49             int temp = matrix[i][j];
50             matrix[i][j] = matrix[j][i];
51             matrix[j][i] = temp;
52         }
53     }
54     for(int i = 0; i < matrix.length; i++){
55         for(int j = 0; j < matrix[0].length/2; j++){
56             int temp = matrix[i][j];
57             matrix[i][j] = matrix[i][matrix.length - 1 - j];
58             matrix[i][matrix.length - 1 - j] = temp;
59         }
60     }
61 }
    Codeium: Refactor | Explain | Generate Javadoc | X
62 public static void main(String[] args) {
63     int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
64     rotate(matrix);
65     for(int row = 0; row < matrix.length; row++){
66         for(int col = 0; col < matrix[row].length; col++){
67             System.out.print(matrix[row][col] + " ");
68         }
69         System.out.println();
70     }
71 }
72 }

```

> Task :Rotate\_Image\_by\_90\_degree.main()

7 4 1

8 5 2

9 6 3

## Set Matrix Zero

```
1  /*
2  * Set Matrix Zero
3  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.
4
5  Examples:
6
7  Examples 1:
8  Input: matrix=[[1,1,1],[1,0,1],[1,1,1]]
9  Output: [[1,0,1],[0,0,0],[1,0,1]]
10 Explanation: Since matrix[2][2]=0. Therefore the 2nd column and 2nd row will be set to 0.
11
12 Examples 2:
13 Input: matrix=[[0,1,2,0],[3,4,5,2],[1,3,1,5]]
14 Output: [[0,0,0,0],[0,4,5,0],[0,3,1,0]]
15 Explanation: Since matrix[0][0]=0 and matrix[0][3]=0. Therefore 1st row, 1st column and 4th column will be set to 0
16 */
17
18
19 public class Set_Matrix_Zeroes {
20     Codeium: Refactor | Explain | Generate Javadoc | X
21     public static void setZeroes(int[][] matrix) {
22         /*
23          * BruteForce Approach: Using Extra matrix which is the copy of original matrix.
24          * Time Complexity:  $O((N*M)*(N + M))$ .  $O(N*M)$  & Space complexity:  $O(m * n)$ 
25          */
26         int m = matrix.length;
27         int n = matrix[0].length;
28         int[][] temp = new int[m][n];
29         for(int row = 0; row < matrix.length; row++){
30             for(int col = 0; col < matrix[row].length; col++){
31                 temp[row][col] = matrix[row][col];
32             }
33         }
34         for(int row = 0; row < matrix.length; row++){
35             for(int col = 0; col < matrix[row].length; col++){
36                 if(matrix[row][col] == 0){
37                     int ind = row - 1;
38                     while(ind >= 0){
39                         if(matrix[ind][col] != 0){
40                             temp[ind][col] = 0;
41                         }
42                         ind--;
43                     }
44                     ind = row + 1;
45                     while(ind < matrix.length){
46                         if(matrix[ind][col] != 0){
47                             temp[ind][col] = 0;
48                         }
49                         ind++;
50                     }
51                     ind = col - 1;
52                     while(ind >= 0){
53                         if(matrix[row][ind] != 0){
54                             temp[row][ind] = 0;
55                         }
56                         ind--;
57                     }
58                     ind = col + 1;
59                     while(ind < matrix[row].length){
60                         if(matrix[row][ind] != 0){
61                             temp[row][ind] = 0;
62                         }
63                         ind++;
64                     }
65                 }
66             }
67         }
68         for(int row = 0; row < matrix.length; row++){
69             for(int col = 0; col < matrix[row].length; col++){
70                 if(matrix[row][col] == -1){
71                     matrix[row][col] = 0;
72                 }
73             }
74         }
75         for(int row = 0; row < matrix.length; row++){
76             for(int col = 0; col < matrix[row].length; col++){
77                 matrix[row][col] = temp[row][col];
78             }
79         }
80     }
81 }
```

```

80 // Optimized Approach: Time complexity: O(2*(N*M)) & Space complexity: O(1)
81 int m = matrix.length;
82 int n = matrix[0].length;
83 int x = 1;
84 int y = 1;
85 for(int j = 0; j < n; j++){
86     if(matrix[0][j] == 0){
87         x = 0;
88     }
89 }
90 for(int i = 0; i < m; i++){
91     if(matrix[i][0] == 0){
92         y = 0;
93     }
94 }
95 for(int i = 1; i < m; i++){
96     for(int j = 1; j < n; j++){
97         if(matrix[i][j] == 0){
98             matrix[i][0] = 0;
99             matrix[0][j] = 0;
100         }
101     }
102 }
103 for(int j = 1; j < n; j++){
104     if(matrix[0][j] == 0){
105         for(int i = 0; i < m; i++){
106             matrix[i][j] = 0;
107         }
108     }
109 }
110 for(int i = 1; i < m; i++){
111     if(matrix[i][0] == 0){
112         for(int j = 0; j < n; j++){
113             matrix[i][j] = 0;
114         }
115     }
116 }
117 if(y == 0){
118     for(int i = 0; i < m; i++){
119         matrix[i][0] = 0;
120     }
121 }
122 if(x == 0){
123     for(int j = 0; j < n; j++){
124         matrix[0][j] = 0;
125     }
126 }
127 }

```

[Codeium](#) [Replit](#) [Explain](#) [Generate JavaDoc](#) [^](#)

```

128 public static void main(String[] args) {
129     int[][] matrix = {{0, 1, 2, 0}, {3, 4, 5, 2}, {1, 3, 1, 5}};
130     setZeroes(matrix);
131     for(int row = 0; row < matrix.length; row++){
132         for(int col = 0; col < matrix[row].length; col++){
133             System.out.print(matrix[row][col] + " ");
134         }
135         System.out.println();
136     }
137 }
138 }
139

```

> 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

```
8  /*
9  * Sort an array of 0s, 1s and 2s
10 Problem Statement: Given an array consisting of only 0s, 1s and 2s. Write a program to in-place sort the array
11 without using inbuilt sort functions. ( Expected: Single pass- $O(N)$  and constant space)
12
13 Example 1:
14
15 Input: nums = [2,0,2,1,1,0]
16 Output: [0,0,1,1,2,2]
17
18 Input: nums = [2,0,1]
19 Output: [0,1,2]
20
21 Input: nums = [0]
22 Input: nums = [0]
23
24 */
25
26
27 public class SortColors {
28     Codeium: Refactor | Explain | Generate Javadoc | X
29     public static void swap(int[] nums, int low, int high){
30         int temp = nums[low];
31         nums[low] = nums[high];
32         nums[high] = temp;
33     }
34     Codeium: Refactor | Explain | Generate Javadoc | X
35     public static void sortColors(int[] nums) {
36         /*
37          * Brute Force Approach: Using sorting--> Time Complexity:  $O(N\log N)$  & Space Complexity:  $O(1)$ 
38          * Arrays.sort(nums);
39          *
40          */
41
42         /*
43          * Solution 2: Using Count Sort --> Time Complexity:  $O(N)$  & Space Complexity:  $O(1)$ 
44          * int countZero = 0, countOne = 0, countTwo = 0;
45          * for(int i = 0; i < nums.length; i++){
46          *     if(nums[i] == 0){
47          *         countZero++;
48          *     }
49          *     else if(nums[i] == 1){
50          *         countOne++;
51          *     }
52          *     else{
53          *         countTwo++;
54          *     }
55          * }
56          * int k = 0;
57          * while(k < countZero){
58          *     nums[k++] = 0;
59          * }
60          * while(k < (countOne + countZero)){
61          *     nums[k++] = 1;
62          * }
63          * while(k < nums.length){
64          *     nums[k++] = 2;
65          * }
66          */
67     }
```

```

66 // Optimized Approach: "Dutch National Flag problem" Time Complexity: O(N) & Space Complexity: O(1)
67 int low = 0;
68 int mid = 0;
69 int high = nums.length - 1;
70 while(mid <= high){
71     switch(nums[mid]){
72         case 0: swap(nums, low++, mid++);
73         break;
74         case 1: mid++;
75         break;
76         case 2: swap(nums, high--, mid);
77         break;
78     }
79 }
80
81 Codeium: Refactor | Explain | Generate Javadoc | X
82 public static void main(String[] args) {
83     int[] nums = {2,0,2,1,1,0};
84     sortColors(nums);
85     System.out.println(Arrays.toString(nums));
86 }
87

```

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

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

```

6  /*
7  * Two Sum : Check if a pair with given sum exists in Array
8  Problem Statement: Given an array of integers nums[] and an integer target, return indices of the
9  two numbers such that their sum is equal to the target.
10
11 Note: Assume that there is exactly one solution, and you are not allowed to use the same element twice.
12 Example: If target is equal to 6 and num[1] = 3, then nums[1] + nums[1] = target is not a solution.
13
14 Example 1:
15
16 Input: nums = [2,7,11,15], target = 9
17
18 Output: [0,1]
19
20 Explanation: Because nums[0] + nums[1] == 9,
21 which is the required target, we return
22 indexes [0,1]. (0-based indexing)
23 Example 2:
24
25 Input Format: nums = [3,2,4,6], target = 6
26
27 Output: [1,2]
28
29 Explanation: Because nums[1] + nums[2] == 6,
30 which is the required target, we return
31 indexes [1,2].
32
33 */

```

```

39 public class TwoSum{
    Codeium: Refactor | Explain | Generate Javadoc | X
40     public static int[] findTwoSum(int[] nums, int target){
41         /*
42          * BruteForce Approach: Time Complexity: O(N^2) & Space complexity: O(1)
43          * int[] ans = new int[2];
44          for(int i = 0; i < nums.length; i++){
45              for(int j = i + 1; j < nums.length; j++){
46                  if(nums[i] + nums[j] == target){
47                      ans[0] = i;
48                      ans[1] = j;
49                  }
50              }
51          }
52          return ans;
53
54          */
55          int[] ans = new int[2];
56          HashMap<Integer, Integer> map = new HashMap<>();
57          for(int i = 0; i < nums.length; i++){
58              if(map.containsKey(target - nums[i])){
59                  ans[1] = i;
60                  ans[0] = map.get(target - nums[i]);
61                  return ans;
62              }
63              map.put(nums[i], i);
64          }
65          return ans;
66      }
    Codeium: Refactor | Explain | Generate Javadoc | X
67     public static void main(String[] args) {
68         int[] nums = {2,7,11,15};
69         int target = 9;
70         int[] ans = findTwoSum(nums, target);
71         System.out.println(Arrays.toString(ans));
72     }
73 }
74

```

> Task :TwoSum.main()

[0, 1]