**Question 36:-**  **WAP to find Kth smallest element in unsorte array input arr[] = {7,10,4,20,15} k=4**

**Answer :-** KthSmallestUsingBinarySearch

**WAP to find Kth smallest element in unsorte array input arr[] = {7,10,4,20,15} k=3**

1. **Answer:-**  **Approach 1: Using Sorting**
   * In this approach, we sort the entire array using a sorting algorithm (such as quicksort, mergesort, or heapsort).
   * The time complexity of sorting an array of N elements is typically **O(N log N)**.
   * After sorting, we directly access the Kth element, which takes constant time (O(1)).
   * Therefore, the overall time complexity is **O(N log N)**.

**Question 37:- sort the array and calculate the cumulative frequency of each element of array input arr[] = {1,3,2,1,2,4} output 1->2 2->4 3->5 4->6**

# Answer:-  output form element -> cumulative frequency

*int*[] arr = {1, 3, 2, 1, 2, 4};

        countFrequency(arr);

        System.out.println();

*int*[] arr1 = {1, 2, 1, 2, 1, 2};

        countFrequency(arr1);

1 -> 2

1 -> 2

2 -> 4

2 -> 4

3 -> 5

4 -> 6

1 -> 3

1 -> 3

1 -> 3

2 -> 6

2 -> 6

2 -> 6

# The cumulative frequency of 1 is 2 (appears twice).

# The cumulative frequency of 2 is 4 (appears three times).

# The cumulative frequency of 3 is 5 (appears once).

# The cumulative frequency of 4 is 6 (appears once).

# The time complexity of this solution is O(n log n)

**Question 38:- get the majority element in array of integer containing duplicates majority element . A majority element is an element thet appear more than n/2 where n is the size of array input arr[] = {1,1,2,3,1,5,3,1,1,1} output = 1**

**Answer:-**  **arr[] = {1, 1, 2, 3, 1, 5, 3, 1, 1, 1} using Java. A majority element is an element that appears more than n/2 times (where n is the size of the array).**

### Approach: Counting Frequency

1. We’ll traverse through the array and keep track of the count of each element.
2. If the count of any element becomes greater than n/2, we’ll consider it as the majority element.
3. If no element satisfies this condition, there is no majority element.

In the Java implementation for this approach:

* **The element 1 appears 6 times, which is greater than n/2 (where n = 10).**

**The time complexity of this solution is O(n), where n is the size of the array, and the auxiliary space required is also O(n).**

**Question 39:- find of the length of longest consecutive elements sequence from a given unsorted array of integer Input arr[] = {49,1,3,200,2,4,70,5} The longest consecutive elements sequence is {1,2,3,4,5} therefore program will return its length in java**

**Answer:-**  **unsorted array arr[] = {49, 1, 3, 200, 2, 4, 70, 5}**

# use a HashSet to store all distinct elements from the array. Then, for each element, we check if its previous element (i.e., element - 1) exists in the set. If it does not, we start counting the consecutive elements from that element.

**OUTPUT - Length of the longest consecutive sequence is: 5**

# The longest consecutive sequence is {1, 2, 3, 4, 5}.

# The time complexity of this solution is O(n), where n is the size of the array, and the auxiliary space required is also O(n).

**Question 40:- find the max product formated by multiplication three numbers. (you can't sort the array watch out when there are negative numbers. array[] = {2,5,-2,6,-3,8,0,-7,-9,4} in java**

# Answer:-  arr[] = {2, 5, -2, 6, -3, 8, 0, -7, -9, 4}

1. Initialize three variables: max1, max2, and max3 to store the maximum positive numbers.
2. Initialize two variables: min1 and min2 to store the minimum negative numbers.
3. Traverse through the array and update these variables based on the current element.
4. The maximum product can be one of the following:
   * Product of the three maximum positive numbers: max1 \* max2 \* max3
   * Product of the two minimum negative numbers and the maximum positive number: min1 \* min2 \* max1
5. Return the maximum of the above two products.

# Maximum product of three numbers: 216

# The maximum product is obtained by multiplying the numbers 8, 6, and 5.

# The time complexity of this solution is O(n), where n is the size of the array.

**Question 41:- find the number which is not repeated in array of integers others are present for two times input 23,34,56,21,21,58,78,23,34**

**Answer:-**  The input array is: 23, 34, 56, 21, 21, 58, 78, 23, 34.

### Approach: Using XOR

# We can use the XOR operation to find the non-repeating number. The idea is as follows:

# Initialize a variable result to 0.

# Iterate through each element in the array and perform the XOR operation with result.

# The final value of result will be the non-repeating number.

* **OUTPUT --- The number 56 appears only once in the array.**

**The time complexity of this solution is O(n)**

**Question 42**:- **Given two array sort the value of using the second array. input String array1[] = {"a","b","c","d","e","f","g","h"} int array2[] = {0,1,1,0,1,2,2,0,1} Output {"a","d","h","b","c","e',"i","f","g"}**

# Answer:-  values of array1 using the second array array2, we can create a custom comparator that compares the elements based on their positions in array2. Then, we can use Arrays.sort() to sort array1

OUTPUT - Sorted array1: [a, d, h, b, c, e, f, g]

# The values in array1 are sorted based on their positions in array2.

# The time complexity of this solution is O(n log n)

**Question : 43 how to find the missing number in a given Array from number 1 to 100 ?**

1. **Answer:-**  Calculate the sum of all numbers from 1 to 100 using the formula: N \* (N + 1) / 2, where N is 100.
2. Iterate through the array and compute the sum of all the numbers in the array.
3. Subtract the sum of the array from the total sum calculated in step 1. The result will be the missing number.

# OUTPUT- /\* The missing number is: 5050\*/ with your actual array of integers. The output will be the missing number between 1 and 100.

**Question 44:-**

**what are jagged arrays in java? Arrays containing different length is known as jagged arrays Multimensional arrays are also known ar jagged arrays.**

**Answer:-**

**Question 45:- how to find all pairs on integer array whose sum is equals to given number in java**

### Answer:-  Using Nested Loops (Naive Approach)

# Use two nested loops to iterate through all possible pairs of elements in the array.

# Check if the sum of the pair equals the given number.

# Print the pairs that satisfy this condition.

OUTPU- D:\8Assignment\2ArraysAssignment\Assignment\36To61>java PairSumFinder45

(2, 4)

(5, 1)

**Question 46:- write a program to remove duplicate from array in java**

### Answer:-  Using Extra Space (Temporary Array)

# This approach creates a temporary array to store unique elements. It traverses the input array and copies all the unique elements to the temporary array. Finally, it updates the original array with the unique elements from the temporary array.

**Question47:-** **there are an array with every element repeated twice except one. Find that element? in java**

# Answer :- To find the element that appears only once in an array where every other element appears twice, we can use the XOR operation. The idea is based on the following two facts:

# XOR of a number with itself is 0.

# XOR of a number with 0 is the number itself.

# Ouestion 48:- How to find comman element in three sorted arrays in java?

### Answer:- Using Three Pointers

1. Initialize three pointers i, j, and k to 0. These pointers will point to the indices of the three arrays.
2. While all three pointers are within the bounds of their respective arrays:
   * If arr1[i], arr2[j], and arr3[k] are equal, print the common element.
   * Otherwise, increment the pointer that points to the smallest element among the three.
3. Repeat step 2 until any one of the pointers reaches the end of its array.

**OUTPUT-**

Common elements: 20 80

* The common elements in the three sorted arrays are 20 and 80.

The time complexity of this solution is **O(n1 + n2 + n3)**, where n1, n2, and n3 are the sizes of arr1, arr2, and arr3

# Ouestion 50:- how to find the smallest positive value integer value that cna be represented as sum of any subset of a given arrav in java input {1,2,3,-4,-1,4} output {-4,1,-1,2,3,4}

# Answer:- how to find the smallest positive value integer value that can be represented as sum of any subset of a given array in java input {1,1,3,6,10,11,15} output 2:

Explanation:

1. Sort thethe array in non-decreasing order.
2. Initialize res as 1 (the smallest possible answer).
3. Traverse the array. For each element arr[i]:
   * If arr[i] is greater than res, we found the gap, and res is the final result.
   * Otherwise, increment res by arr[i].

For the given input {1, 1, 3, 6, 10, 11, 15}, the output will be **2**.

# Ouestion 51:- How to arrange array in alternating positive and negative number input(1,2,3,-4,-1,4} output {-4,1,-1,2,3,4} in java

**Answer:-** The output for arr1 will be [-4, 1, -1, 2, 3, 4], and for arr2, it will be [-5, 5, -2, 2, -8, 4, 7, 1, 8, 0]. In java 52 Question.

Ouestion 52:- How to find if there is a sub arrays sum equals 30 zero input {4,2,-3,1,6} output true in java

# Answer:- To determine if there exists a subarray with a sum equal to **zero** in the given array {4, 2, -3, 1, 6},

In this code:

* We maintain a HashSet called prefixSums to store the cumulative sum of elements encountered so far.
* As we iterate through the array, we add each element to the current sum.
* If the current sum is zero or has been seen before, we return true.
* Otherwise, we add the current prefix sum to the set.
* If no such subarray is found, we return false.

For the input {4, 2, -3, 1, 6}, the output will be true, as there is a subarray with zero sum from index 1 to 3 (i.e., {2, -3, 1}).

# Ouestion 53:- How to remove duplicate from place such that each element appear only one andreturn thenew length input array A = [1,1,2] return length = 2 and A is now [1,2] return length

**Answer:-**  To remove duplicates from an array in-place while ensuring that each element appears only once, you can follow this approach in Java:

* We maintain a pointer uniqueCount to keep track of the unique elements.
* As we iterate through the array, if the current element is different from the last unique element, we move it to the next unique position.
* The final value of uniqueCount represents the new length of the array after removing duplicates.

For the input array A = [1, 1, 2], the output will be:

New array A: 1 2

New length: 2

# Ouestion 54:-

**Answer:-**

# Ouestion 55:- find sub array with maximum sum in an array positive and negative numbers? array [-2,1,-3,4,-1,2,1,-5,4] the contiguous sub array [4,-1,2,1] has the largest sum = 6

# Answer:-  To find the contiguous subarray with the maximum sum in an array containing both positive and negative numbers, we can use **Kadane’s Algorithm**. This algorithm efficiently computes the maximum sum subarray

Explanation:

1. We maintain two variables: maxSoFar (stores the maximum sum found so far) and maxEndingHere (stores the maximum sum contiguous subarray ending at the current index).
2. For each element in the array:
   * Update maxEndingHere by taking the maximum of the current element and the sum of the previous subarray ending at the current index.
   * Update maxSoFar by taking the maximum of the current maxSoFar and the updated maxEndingHere.
3. The final value of maxSoFar represents the maximum sum subarray.

OUTPUT

input array [-2, 1, -3, 4, -1, 2, 1, -5, 4], the output will be:

Maximum sum of contiguous subarray: 6

The subarray [4, -1, 2, 1] indeed has the largest sum of

# Ouestion 56 :- HOW TO FIND sub array with largest product in array of both positive and negative value Example array =2,3,-2,4] the contiguous subarray[2,3] has the largest product = 6

**Answer:-** To find the contiguous subarray with the largest product in an array containing both positive and negative values, we can use **Kadane’s Algorithm** with a slight modification. This algorithm efficiently computes the maximum product subarray.

Explanation:

1. We maintain three variables: maxEndingHere, minEndingHere, and maxSoFar.
2. For each element in the array:
   * Update maxEndingHere by considering the maximum of the current element, the product of the previous maximum, and the product of the previous minimum.
   * Update minEndingHere by considering the minimum of the same three values.
   * Update maxSoFar with the maximum value for each index.
3. The final value of maxSoFar represents the maximum product subarray.

For the given input array [2, 3, -2, 4], the output will be:

Maximum product of contiguous subarray: 6

The subarray [2, 3] indeed has the largest product of 6.

# Ouestion 57:- find length of longest consecutive sequence in array of integers to find the contigous sub arrays example [100,4,200,1,3,2] consecutive sequence is [1,2,3,4] return length 4

# Answer:- find the length of the longest consecutive sequence in an array of integers, we can use an efficient algorithm that runs in O(n) time. Let’s solve this problem for the given example:

# Given array: [100, 4, 200, 1, 3, 2]

# The longest consecutive elements sequence is [1, 2, 3, 4], and its length is 4.

In this code:

* We use a HashSet to store all the numbers for efficient lookups.
* For each number, we check if it is the start of a sequence (i.e., no previous smaller number exists).
* If it is the start, we extend the sequence by checking consecutive larger numbers.
* We keep track of the maximum length encountered.

For the given input array, the output will be:

Length of longest consecutive sequence: 4

The sequence [1, 2, 3, 4] indeed has the longest consecutive elements

# Ouestion 58 - find the element that appear more than n/k times? input {3,1,2,2,1,2,3,3} and k = 4 then output [2,3]

**Answer:-** To find the elements in an array that appear more than **n/k** times, where **n** is the size of the array and **k** is a given integer, we can use the following approaches:

1. **Sorting and Counting:**
   * First, sort the array.
   * Then, count the frequency of distinct elements by checking adjacent elements.
   * If the frequency of an element is greater than **n/k**, print that element.
   * This approach has a time complexity of **O(n\*logn)** due to sorting.

Output: 2 3

# Ouestion 59:- how to reverse array place in java

# Answer:- To reverse an array in-place (i.e., without creating a new array), you can use the **swapping** approach.

In this code:

* We maintain two pointers (left and right) that start from the beginning and end of the array, respectively.
* We swap the elements at these pointers and move them towards each other until they meet in the middle.
* The array is reversed in-place.

For the given input array {10, 20, 30, 40, 50}, the output will be:

Reversed array:

50 40 30 20 10

# Ouestion 60:- it is not possible to reach (M, N) from (1 ,1) then return -1 as your output input1 integer value of N where 1<N input2 M where M<106.

**Answer:-** To solve this problem, we can use dynamic programming to find the minimum number of steps needed to reach cell (N, M) starting from the current position (1, 1). The rule is that if you are at cell (x, y), then from there you can either move to cell (x, x+y) or to cell (x+y, y) in one step

In this code:

* We use a 2D array dp to store the minimum number of steps needed to reach each cell.
* We fill in the array based on the given rule.
* If it is not possible to reach (N, M) from (1, 1), we return -1.

For the given input (N = 3, M = 2), the output will be:

Minimum number of steps: 2

The minimum steps to reach cell (3, 2) from (1, 1) are 2

# Ouestion 48:- write a program to print spiral of the given array int arr[][] = {{10,20,30},{40,50,60},{70,80,90}} ; Output 10 20 30 60 70 90 80 70 40 50

# Answer:- To print the given 2D array in a spiral form, we can follow the spiral traversal order: left to right (first row), top to bottom (last column), right to left (last row), and bottom to up (first column). Here’s the Java code to achieve this:

OUTPUT:- 10 20 30 60 70 90 80 70 40 50

# Ouestion 48:-

**Answer:-**

# Ouestion 48:-

**Answer:-**

# Ouestion 48:-

**Answer:-**