Array

How to count inversions in a sorted array?

Two elements arr[i] and arr[j] in an array arr[] form an inversion if a[i] > a[j] and i < j. How to count all inversions in an unsorted array.

In an integer array, there is 1 to 100 number, out of one is duplicate, how to find?

In this case, you can simply add all numbers stored in an array, and the total sum should be equal to n(n+1)/2. Now just subtract actual sum to expected sum, and that is your duplicate number. There is a brute force way of checking each number against all other numbers, but that will result in the performance of O(n^2) which is not good.

– One way of solving this problem is using a Hashtable or HashMap data structure. You can traverse through an array, and store each number as key and number of occurrence as value. At the end of traversal, you can find all duplicate numbers, for which occurrence is more than one. In Java if a number already exists in HashMap then calling get(index) will return number otherwise it returns null. this property can be used to insert or update numbers in HashMap.

– We can use one Set and one List to keep repeating and non-repeating character separately. Once we finish scanning through String, which is O(n), we can get the magic character by accessing List which is O(1) operator. Since List is an ordered collection get(0) returns the first element.

– First step: Scan String and store count of each character in HashMap.
– Second Step: Traverse String and get a count for each character from Map.
Since we go through String from first to the last character, when the count for any character is 1, we break, it's the first non repeated character.
Here the order is achieved by going through String again.

How do you find the missing number in a given integer array of 1 to 100?

1) Sum of the series: Formula: n (n+1)/2(but only work for one missing number)
2) Use BitSet, if an array has more than one missing elements.

integers as an index.

How do you find the duplicate number on a given integer array?

We can use Set interface, particularly LinkedHashSet, because that also keep the order on which elements are inserted into Set.
The problem with an array is not finding duplicates, it's about removing them. Since an array is a static, fixed length data structure, you can not change its length. This means, deleting an element from an array requires creating a new array and copying content into that array.

If your input array contains lots of duplicates then this may result in lots of temporary arrays. It also increases the cost of copying contents, which can be very bad. Given this restriction, you need to come out with a strategy to minimize both memory and CPU requirements. So you need to do is to convert your array into ArrayList first then subsequently create a LinkedHashSet from that ArrayList. Regarding removing duplicate permanently from result array, one approach could be to count a number of duplicates and then create an array of right size i.e. length – duplicates and then copying content from intermediate result array to final array, leaving out elements which are marked duplicate.

We use two variables largest and smallest to store the maximum and minimum values from the array. Initially largest is initialized with Integer.MIN_VALUE and smallest is initialized with Integer.MAX_VALUE. In each iteration of the loop, we compare current number with largest and smallest and update them accordingly. Since if a number is larger than largest, it can't be smaller than smallest, which means you don't need to check if the first condition is true, that's why we have used if-else code block, where else part will only execute if the first condition is not true.

How do you find all pairs of an integer array whose sum is equal to a given number?

Questions:
Does the array contain only positive or negative numbers?
What if the same pair repeats twice, should we print it every time?
Is reverse of pair is acceptable e.g. can we print both (4,1) and (1,4) if given sum is 5.
Do we need to print only distinct pair? does (3, 3) is a valid pair forgiven sum of 6?

– The first solution is a brute-force, naive but genuine. You take one number from the array and then loop through the array and output pairs which is equal to the given sum. (n^2)
– In order to find two numbers in an array whose sum equals a given value, we probably don't need to compare each number with other. What we can do here is to store all

operation in the hash table. So total complexity of solution would be O(N)

– A more efficient in-place solution would be to sort the array and use two pointers to scan through the array from both direction i.e. beginning and end. If the sum of both the values are equal to given number then we output the pair and advance them. If the sum of two numbers is less than k then we increase the left pointer, else if the sum is greater than k we decrement the right pointer, until both pointers meet at some part of the array. The complexity of this solution would be O(NlogN) due to sorting.