1. **If the array is sorted using recursion:**

Type1 : you have to check if a[0] < a[1] or not and everytime you have to call recursion make the array shorter. Suppose at first array was 1,2,3,4. In next recursive call array will be 2,3,4. Next recursive call array will be 3,4. So that everytime there will be new a[0] and a[1] which will be checked if sorted

Type 2: you have to pass the array and array size to the function that will be recursive. Check if arr[n-1] > arr[n-2] is true or not. If not then the array is not sorted.

And every time during recursive call send the full array but send the size by making it smaller by one - **isSorted(arr,n-1)** like this. Base case will be if n==0 or n==1 means array size is 0 or 1 then array is sorted , return true

1. **Sum of array elements using recursion:**

**Base case:** When n = 0 means array size is 0, return arr[0] , as we will sum the elements, so we will return the first index value.

**Recurive call:** We will reduce the size by 1 and call the recusive function like sumOfArray(arr,n-1); and store this value to a variable. Let’s say **smallerInput**;

**Calculation:** We will return the summation of **smallerInput + arr[n]**

1. **Check if an element is present in the array using recursion:**

**Base Case:**

If n = 0, we will check if the element is = arr[0] or not . if equal , return true else return false

Then after that we will also check if arr[0] == element or not outside the base case because we need a one time match with any index value of the array. If any index value matches, we will get our ans

**Recurive call**: We will point the next index of the array and also reduce the size by 1 and send it as a recursive call like **bool smallerInput = check(arr+1,n-1);**

So, that every time we can get a new arr[0] value. Using this recursive call we are checking the array elements from left to right onwards.

**Calculation:** We will return the boolean recursive call that we have made.