Q1. Is Sorted : Check if array is sorted using Recursion.

Logic : Assume isSorted(v, i) checks if the array v[0....i] is sorted. So, at a given step, we have to check v[i] >= v[i-1] && isSorted(v, i-1) must be true.

Base Case - i == 0, one element is always sorted.

```
bool isSorted(vector<int>& v, int i) {
    if(v.size() == 0 || v.size() == 1)
        return true;
    if(i <= 0)
        return true;

    return v[i] >= v[i-1] && isSorted(v, i-1);
}
```

```
bool isSorted(int arr[], int size) {

    //base case
    if(size == 0 || size == 1 ){
        return true;
    }

    if(arr[0] > arr[1])
        return false;
    else {
        bool remainingPart = isSorted(arr + 1, size - 1 );
        return remainingPart;
    }

}
```

Babbar Code

Homework: Find sum of the elements of an array using Recursion.

Logic : Assume getSum(arr, n) gives the sum of the array arr containing n elements. Base case - n == 0 or n == 1.

```
int getSum(int arr[], int n) {
    if(n == 0) return 0;
    int sum = arr[0];
    sum += getSum(arr+1, n-1);
    return sum;
}
```

Call from main()

getSum(arr, n-1)

```
int sumOfArray(vector<int> v, int ind, int sum) {
    if(ind == v.size()) return 0;
    sum = v[ind];
    sum += sumOfArray(v, ind+1, sum);
    return sum;
}
```

Call from main()

sumOfArray(v, 0, 0)

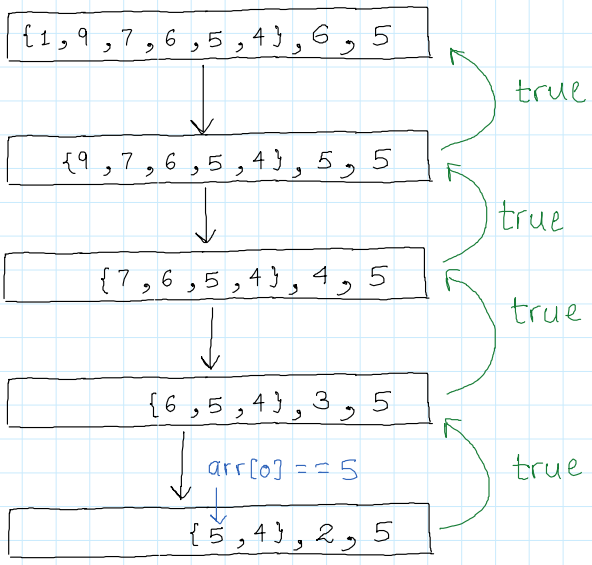
Q2. Search Element in array using Recursion. (Easy)

```
bool linearSearch(int arr[], int n, int val) {
    if(n < 0) return false;
    if(arr[0] == val)
        return true;
    // if the element is found even once, then we return true
    return linearSearch(arr+1, n-1, val);
}
```

Recursion Tree

Let arr[] = {1, 9, 7, 6, 5, 4}

val = 5 , n = 6



Binary Search Using Recursion :

```cpp
void printArr(int arr[], int l, int r) {
    cout << "Size of array is " << r-l+1 << endl;
    for(int i=l;i<=r;i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

bool binarySearch(int arr[], int val, int l, int r) {
    printArr(arr, l, r);
    if(l>r) return false;

    int mid = l + (r-l)/2;

    if(arr[mid] == val)
        return true;
    else if(arr[mid] > val)
        return binarySearch(arr, val, l, mid-1);
    else if(arr[mid] < val)
        return binarySearch(arr, val, mid+1, r);

    return false;
}
```

```
10
1 3 4 5 7 8 11 12 13 14

14
```

Output.txt

```
Size of array is 10
1 3 4 5 7 8 11 12 13 14
Size of array is 5
8 11 12 13 14
Size of array is 2
13 14
Size of array is 1
14
Found
```

If you are not able to understand this, watch the video numbered 12 to understand Binary Search.

Homework : first and last Position using Recursion.

Iterative Approach

```cpp
// return the index of the first occurance of val in vector v
pair<int, int> firstAndLastPosition(vector<int>& v, int val) {
    int n = v.size();
    int l = 0, r = n-1, mid;
    pair<int, int> ans(-1, -1);
    while(l <= r) {
        mid = (l+r)/2;
        if(v[mid] == val) {
            ans.first = mid;
            r = mid - 1;
        }
        else if(v[mid] > val)
            r = mid - 1;

        else
            l = mid + 1;
    }
```

we can also break this down into 2 functions.

```cpp
// return the index of the first occurance of val in vector v
pair<int, int> firstAndLastPosition(vector<int>& v, int val) {
    int n = v.size();
    int l = 0, r = n-1, mid;
    pair<int, int> ans(-1, -1);
    while(l <= r) {
        mid = (l+r)/2;
        if(v[mid] == val) {
            ans.first = mid;
            r = mid - 1;
        }
        else if(v[mid] > val)
            r = mid - 1;

        else
            l = mid + 1;
    }
    if(ans.first == -1)
        return ans;

    l = ans.first, r = n-1;
    while(l <= r) {
        mid = (l+r)/2;
        if(v[mid] == val) {
            l = mid + 1;
            ans.second = mid;
        }

        else if(v[mid] > val)
            r = mid - 1;
    }
    return ans;
}
```

we can also break this down into 2 functions.

Recursive Approach

```cpp
int firstOcc(vector<int>& v, int l, int r, int val) {
    if(l > r) return -1;
    int n = v.size(), ans = -1;
    int mid = l + (r-l)/2;
    if(v[mid] == val) {
        ans = mid;
        int t = firstOcc(v, l, mid-1, val);
        if(t != -1)
            ans = t;
    }
    else if(v[mid] > val) {
        ans = firstOcc(v, l, mid-1, val);
    }
    else {
        ans = firstOcc(v, mid+1, r, val);
    }
    return ans;
}

int lastOcc(vector<int>& v, int l, int r, int val) {
    if(l > r) return -1;
    int n = v.size(), ans = -1;
    int mid = l + (r-l)/2;
    if(v[mid] == val) {
        ans = mid;
        int t = lastOcc(v, mid+1, r, val);
        if(t != -1)
            ans = t;
    }
    else if(v[mid] > val) {
        ans = lastOcc(v, l, mid-1, val);
    }
    else {
        ans = lastOcc(v, mid+1, r, val);
    }
    return ans;
}
```

Homework: Find total no. of occurances of an element in a sorted array.

Solution : (last occurance - first occurance + 1)

Homework : Find the peak in the mountain array.

Solution :

```
int findPeak(int arr[], int l, int r) {
    if(l > r) return -1;
    int ans = -1;
    int mid = (l+r) / 2;
    if(arr[mid] > arr[mid-1] && arr[mid] > arr[mid+1])
        return mid;

    else if(arr[mid] < arr[mid+1])
        return findPeak(arr, mid+1, r);

    return findPeak(arr, l, mid-1);
}
```

Homework : Find Pivot in a rotated sorted array.

Solution :

```
int findPivotRec(vector<int>& nums, int l, int r) {
    if(l > r) return 0;
    int mid = l + (r - l)/2;
    // given that we don't rotate the array k times which is a
    multiple of n.
    if(mid>0 && nums[mid] < nums[mid-1])
        return mid;

    else if(nums[mid] >= nums[0])
        return findPivotRec(nums, mid+1, r);

    return findPivotRec(nums, l, mid);
}
```

Homework : Search in rotated sorted array. (easy)

Solution : Find pivot and then use binary search in the half according to the value to be found.

Homework : Find square root using Binary Search.

```
long long int squareRoot(int n, int l, int r) {
    if(l > r) return 0;
    long long int mid = (l+r)/2;

    if(mid*mid <= n && (mid+1)*(mid+1)>n)
        return mid;

    else if(mid*mid > n) {
        return squareRoot(n, l, mid-1);
    }

    return squareRoot(n, mid+1, r);
}
```