Recursion - 1.2

Print in increasing and decreasing order

Objective: To print numbers 1 to n, in increasing as well as decreasing order.
We have to do two things, 1. Recurse for the remaining number
2. Print the numbers

Base Case : If n == 1:
print(1)
return;

Time Complexity: O(N)
Space Complexity: O(N), for the call stack

```
void dec(int n) {
    if (n == 1) {
        cout << "1" << endl;
        return;
    }
    cout << n << endl;
    dec(n - 1);
}
void inc(int n) {
    if (n == 1) {
        cout << "1" << endl;
        return;
    }
    inc(n - 1);
    cout << n << endl;
}
```

Find the first and last occurrence of an element using recursion

Objective: To find the first occurrence, we return as soon as we find the element. To find the last occurrence, we return the result we get from further function calls.

Base Case would be when you reach the end of the array.

Time Complexity: O(N)

Space Complexity: O(N) for function call stack

```
int firstocc(int arr[], int n, int i, int key) {
    if (i == n) {
        return -1;
    }
    if (arr[i] == key) {
        return i;
    }
    return firstocc(arr, n, i + 1, key);
}
int lastocc(int arr[], int n, int i, int key) {
    if (i == n) {
        return -1;
    }
    int restArray = lastocc(arr, n, i + 1, key);
    if (restArray != -1) {
        return restArray;
    }
    if (arr[i] == key) {
        return i;
    }
    return -1;
}
```

Check if Array is Sorted [Strictly increasing]

Objective: Check if the array is in strictly increasing order or not.

Time Complexity: O(N)

Space Complexity: O(N) for function call stack

```
bool sorted(int arr[], int n) {
    if (n == 1) {
        return true;
    }
    bool restArray = sorted(arr + 1, n - 1);
    return (arr[0] < arr[1] && restArray);

}
```