18. Maximum Index

Medium Accuracy: 42.72% Submissions: 30463 Points: 4

Given an array **A[]** of **N** positive integers. The task is to find the maximum of **j - i** subjected to the constraint of **A[i] <= A[j]**.

Example 1:

```
Input:
N = 2
A[] = {1, 10}
Output:
1
Explanation:
A[0]<=A[1] so (j-i) is 1-0 = 1.
```

$O(N^2)$ ←

① We've to find the max index such that given condⁿ is satisfied for the values stored.

② For the farthest dis max approach would be to look for all the elements starting from array and for every element from the starting

(11) Efficient Approach :-

(1) ∵ We need to maximize $|i-j|$ for given condn $a[i] <= a[j]$.

(2) We only need the greatest element when starting from right.

(3) And the smallest when starting from right.

(smallest) → ← (greatest)

Loop will be iterated only in case same cond^{n} fails.

```
int maxIndexDiff(int arr[], int n)
{
    if(n==1){
        return 0;
    }
    int maxDiff;
    int i, j;

    int *LMin = new int[n];
    int *RMax = new int[n];

    //Constructing LMin[] such that LMin[i] stores the minimum value
    //from (arr[0], arr[1], ... arr[i]).
    LMin[0] = arr[0];
    for (i = 1; i < n; ++i)
        LMin[i] = min(arr[i], LMin[i-1]);

    //Constructing RMax[] such that RMax[j] stores the maximum value
    //from (arr[j], arr[j+1], ..arr[n-1]).
    RMax[n-1] = arr[n-1];
    for (j = n-2; j >= 0;  j)
        RMax[j] = max(arr[j], RMax[j+1]);
```

(1) These arrays will contain smallest element present on the left.

(2) Used for the second largest one present at the right of given array index.

Bottom up manner

```
i = 0, j = 0, maxDiff = -1;
//Traversing both arrays from left to right to find optimum j-i.
//This process is similar to merge() of MergeSort.
while (j < n && i < n)
{
    if (LMin[i] <= RMax[j])
    {
        //Updating the maximum difference.
        maxDiff = max(maxDiff, j-i);
        j = j + 1;
    }
    else
        i = i+1;
}
//returning the maximum difference.
return maxDiff;
```

maximizing distance by iterating forwards in array elements.

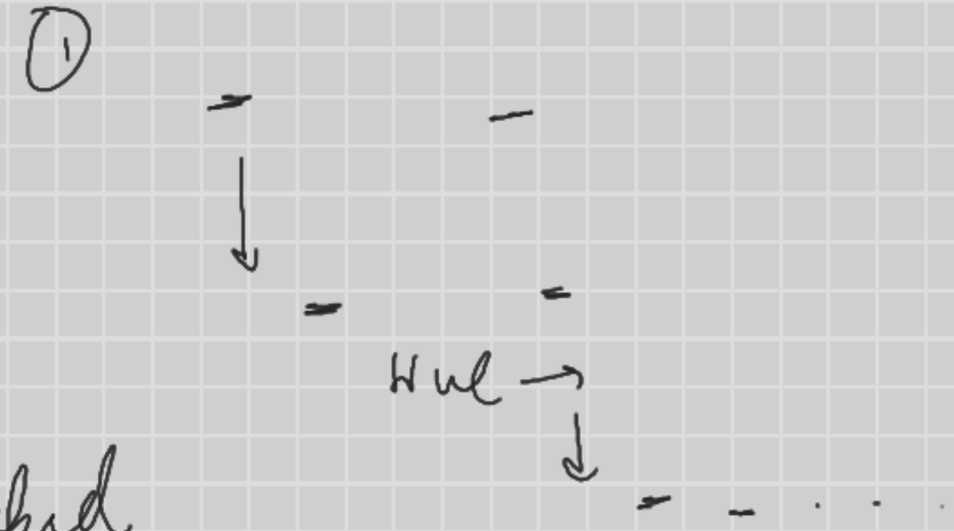
∴ it is working in bottom up manner hence the indices would keep on increasing.

LMin → Decreasing array.

RMax → Decreasing array.

```
i = 0, j = 0, maxDiff = -1;
//Traversing both arrays from left to right to find optimum j-i.
//This process is similar to merge() of MergeSort.
while (j < n && i < n)
{
    if (LMin[i] <= RMax[j])
    {
        //Updating the maximum difference.
        maxDiff = max(maxDiff, j-i);
        j = j + 1;
    }
    else
        i = i+1;
}
//returning the maximum difference.
return maxDiff;
```

failing condition

① 

② It ensures that all possible indices are looked for and then it proceeds in this way.