Given an array arr[], find the maximum j – i such tha arr[j] > arr[i]

Given an array arr[], find the maximum j – i such that arr[j] > arr[i].

Examples:

```
Input: {34, 8, 10, 3, 2, 80, 30, 33, 1}
Output: 6  (j = 7, i = 1)

Input: {9, 2, 3, 4, 5, 6, 7, 8, 18, 0}
Output: 8 ( j = 8, i = 0)

Input:  {1, 2, 3, 4, 5, 6}
Output: 5  (j = 5, i = 0)

Input:  {6, 5, 4, 3, 2, 1}
Output: -1
```

We strongly recommend that you click here and practice it, before moving on to the solution.

Method 1 (Simple but Inefficient)

Run two loops. In the outer loop, pick elements one by one from left. In the inner loop, compare the picke element with the elements starting from right side. Stop the inner loop when you see an element greater than th picked element and keep updating the maximum j-i so far.

C Java

```
#include <stdio.h>
/* For a given array arr[], returns the maximum j – i such that
    arr[j] > arr[i] */
int maxIndexDiff(int arr[], int n)
{
    int maxDiff = -1;
    int i, j;

    for (i = 0; i < n; ++i)
    {
        for (j = n-1; j > i; --j)
        {
            if(arr[j] > arr[i] && maxDiff < (j - i))
                maxDiff = j - i;
        }
    }

    return maxDiff;
}

int main()
{
    int arr[] = {9, 2, 3, 4, 5, 6, 7, 8, 18, 0};
    int n = sizeof(arr)/sizeof(arr[0]);
    int maxDiff = maxIndexDiff(arr, n);
    printf("\n %d", maxDiff);
    getchar();
    return 0;
}
```

Run on IDE

Time Complexity: O(n^2)

Method 2 (Efficient)

To solve this problem, we need to get two optimum indexes of arr[]: left index i and right index j. For an elemer arr[i], we do not need to consider arr[i] for left index if there is an element smaller than arr[i] on left side of arr[i Similarly, if there is a greater element on right side of arr[j] then we do not need to consider this j for right index So we construct two auxiliary arrays LMin[] and RMax[] such that LMin[i] holds the smallest element on left sid of arr[i] including arr[i], and RMax[j] holds the greatest element on right side of arr[j] including arr[j]. Afte constructing these two auxiliary arrays, we traverse both of these arrays from left to right. While traversing LMin and RMa[] if we see that LMin[i] is greater than RMax[j], then we must move ahead in LMin[] (or do i++) becaus all elements on left of LMin[i] are greater than or equal to LMin[i]. Otherwise we must move ahead in RMax[j] t look for a greater j – i value.

Thanks to celicom for suggesting the algorithm for this method.

C Java

```
#include <stdio.h>

/* Utility Functions to get max and minimum of two integers */
int max(int x, int y)
{
    return x > y? x : y;
}

int min(int x, int y)
{
    return x < y? x : y;
}

/* For a given array arr[], returns the maximum j – i such that
    arr[j] > arr[i] */
int maxIndexDiff(int arr[], int n)
{
    int maxDiff;
    int i, j;

    int *LMin = (int *)malloc(sizeof(int)*n);
    int *RMax = (int *)malloc(sizeof(int)*n);
```