

GeeksforGeeks

A computer science portal for geeks

Practice

IDE

Q&A

GeeksQuiz

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

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 picked element with the elements starting from right side. Stop the inner loop when you see an element greater than the picked element and keep updating the maximum j-i so far.

```
#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 element 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 side of arr[i] including arr[i], and RMax[j] holds the greatest element on right side of arr[j] including arr[j]. After 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++) because all elements on left of LMin[i] are greater than or equal to LMin[i]. Otherwise we must move ahead in RMax[j] to look for a greater j - i value.

Thanks to celicom for suggesting the algorithm for this method.

```

#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);

    /* Construct 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]);
}

```

```
/* Construct 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]);

/* Traverse both arrays from left to right to find optimum j - i
   This process is similar to merge() of MergeSort */
i = 0, j = 0, maxDiff = -1;
while (j < n && i < n)
{
    if (LMin[i] < RMax[j])
    {
        maxDiff = max(maxDiff, j-i);
        j = j + 1;
    }
    else
        i = i+1;
}

return maxDiff;
}

/* Driver program to test above functions */
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: $O(n)$

Auxiliary Space: $O(n)$

Please write comments if you find the above codes/algorithms incorrect, or find other ways to solve the same problem.



186 Comments Category: Arrays

Related Posts:

- [Longest Span with same Sum in two Binary arrays](#)
- [Count Inversions of size three in a give array](#)
- [Find the subarray with least average](#)
- [Count triplets with sum smaller than a given value](#)
- [Find zeroes to be flipped so that number of consecutive 1's is maximized](#)
- [Reorder an array according to given indexes](#)
- [Find maximum value of Sum\(i*arr\[i\]\) with only rotations on given array allowed](#)
- [Find maximum average subarray of k length](#)

(Login to Rate and Mark)

3.6

Average Difficulty : 3.6/5.0
Based on 8 vote(s)

☐

Add to TODO List

☐

Mark as DONE

Like Share 12 people like this. Be the first of your friends.

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

@geeksforgeeks, Some rights reserved

[Contact Us!](#)

[About Us!](#)

[Advertise with us!](#)