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

Difficulty Level : Hard • Last Updated : 05 Jul, 2022

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
```

Recommended PracticeMaximum IndexTry It!

inner loop when you see an element greater than the picked element and keep updating the maximum j-i so far.

C++

```
// CPP program for the above approach
#include <bits/stdc++.h>
using namespace std;

/* 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;
        }
    }
```

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

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

C

```
// C program for the above approach
#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();
```

Java

```java
// Java program for the above approach
class FindMaximum {
    /* 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;
    }

    /* Driver program to test above functions */
    public static void main(String[] args)
    {
        FindMaximum max = new FindMaximum();
        int arr[] = { 9, 2, 3, 4, 5, 6, 7, 8, 18, 0 };
        int n = arr.length;
        int maxDiff = max.maxIndexDiff(arr, n);
        System.out.println(maxDiff);
    }
}
```

Python3

```python
# Python3 program to find the maximum
# j – i such that arr[j] > arr[i]
```

```
def maxIndexDiff(arr, n):
    maxDiff = -1
    for i in range(0, n):
        j = n - 1
        while(j > i):
            if arr[j] > arr[i] and maxDiff < (j - i):
                maxDiff = j - i
            j -= 1

    return maxDiff


# driver code
arr = [9, 2, 3, 4, 5, 6, 7, 8, 18, 0]
n = len(arr)
maxDiff = maxIndexDiff(arr, n)
print(maxDiff)

# This article is contributed by Smitha Dinesh Semwal
```

C#

```
// C# program to find the maximum
// j – i such that arr[j] > arr[i]
using System;
class GFG {
    // For a given array arr[], returns
    // the maximum j-i such that arr[j] > arr[i]
    static 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;
            }
```

```
    // Driver program
    public static void Main()
    {

        int[] arr = { 9, 2, 3, 4, 5, 6, 7, 8, 18, 0 };
        int n = arr.Length;
        int maxDiff = maxIndexDiff(arr, n);
        Console.Write(maxDiff);
    }
}
// This Code is Contributed by Sam007
```

PHP

```
<?php
// PHP program to find the maximum
// j – i such that arr[j] > arr[i]

// For a given array arr[], returns
// the maximum j – i such that
// arr[j] > arr[i]
function maxIndexDiff($arr, $n)
{
    $maxDiff = -1;

    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;
}

// Driver Code
```

```
echo $maxDiff ;

// This code is contributed by Sam007
?>
```

Javascript

```
<script>
// JavaScript program for the above approach

/* For a given array arr[],
returns the maximum j – i such
that arr[j] > arr[i] */
function maxIndexDiff(arr, n)
{
    let maxDiff = -1;
    let 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;
}

    // Driver code
    let arr = [ 9, 2, 3, 4, 5, 6, 7, 8, 18, 0 ];
    let n = arr.length;
    let maxDiff = maxIndexDiff(arr, n);
    document.write(maxDiff);

// This code is contributed by Manoj.
</script>
```

Time Complexity: $O(n^2)$

Auxiliary Space: $O(1)$

Method 2: Improvising the Brute Force Algorithm and looking for BUD, i.e Bottlenecks, unnecessary and duplicated works. A quick observation actually shows that we have been looking to find the first greatest element traversing from the end of the array to the current index. We can see that we are trying to find the first greatest element again and again for each element in the array. Let's say we have an array with us for example [1, 5, 12, 4, 9] now we know that 9 is the element that is greater than 1, 5, and 4 but why do we need to find that again and again. We can actually keep a track of the maximum number moving from the end to the start of the array. The approach will help us understand better and also this improvisation is great to come up with in an interview.

Approach :

1. Traverse the array from the end and keep a track of the maximum number to the right of the current index including self
2. Now we have a monotonous decreasing array, and we know we can use binary search to find the index of the rightmost greater element
3. Now we will just use binary search for each of the elements in the array and store the maximum difference of the indices and that's it we are done.

C++

```
using namespace std;

int main()
{
    vector<long long int> v{
        34, 8, 10, 3, 2, 80, 30, 33, 1
    };
    int n = v.size();
    vector<long long int> maxFromEnd(n + 1, INT_MIN);

    // create an array maxfromEnd
    for (int i = v.size() - 1; i >= 0; i--) {
        maxFromEnd[i] = max(maxFromEnd[i + 1], v[i]);
    }

    int result = 0;

    for (int i = 0; i < v.size(); i++) {
        int low = i + 1, high = v.size() - 1, ans = i;

        while (low <= high) {
            int mid = (low + high) / 2;

            if (v[i] <= maxFromEnd[mid]) {

                // We store this as current answer and look
                // for further larger number to the right
                // side
                ans = max(ans, mid);
                low = mid + 1;
            }
            else {
                high = mid - 1;
            }
        }
        // keeping a track of the
        // maximum difference in indices
        result = max(result, ans - i);
    }
    cout << result << endl;
}
```

```c
/* C program to implement
the above approach */

/* For a given array arr[],
calculates the maximum j - i
such that arr[j] > arr[i] */
#include <limits.h>
#include <stdio.h>

/* Function for maximum of
two numbers in C */
int max(int num1, int num2)
{
    return (num1 > num2 ) ? num1 : num2;
}

int main()
{
    int v[] = { 34, 8, 10, 3, 2, 80, 30, 33, 1 };
    int n = sizeof(v) / sizeof(v[0]);
    int maxFromEnd[n+1];
    for (int i = 0; i < n+1; i++) {
        maxFromEnd[i] = INT_MIN;
    }
    // create an array maxfromEnd
    for (int i = n - 1; i >= 0; i--) {
        maxFromEnd[i] = max(maxFromEnd[i + 1], v[i]);
    }

    int result = 0;

    for (int i = 0; i < n; i++) {
        int low = i + 1, high = n - 1, ans = i;

        while (low <= high) {
            int mid = (low + high) / 2;

            if (v[i] <= maxFromEnd[mid]) {

                // We store this as current answer and look
                // for further larger number to the right
```

```
            else {
                high = mid - 1;
            }
        }
        // keeping a track of the
        // maximum difference in indices
        result = max(result, ans - i);
    }
    printf("\n %d", result);
}

/* This code is contributed by Pushpesh Raj */
```

Java

```
// Java program to implement
// the above approach

// For a given array arr[],
// calculates the maximum j – i
// such that arr[j] > arr[i]
import java.util.*;
class GFG{

public static void main(String[] args)
{
  int []v = {34, 8, 10, 3, 2,
             80, 30, 33, 1};
  int n = v.length;
  int []maxFromEnd = new int[n + 1];
  Arrays.fill(maxFromEnd, Integer.MIN_VALUE);

  // Create an array maxfromEnd
  for (int i = v.length - 1; i >= 0; i--)
  {
    maxFromEnd[i] = Math.max(maxFromEnd[i + 1],
                             v[i]);
  }

  int result = 0;
```

```
            ans = i;

        while (low <= high)
        {
          int mid = (low + high) / 2;

          if (v[i] <= maxFromEnd[mid])
          {
            // We store this as current
            // answer and look for further
            // larger number to the right side
            ans = Math.max(ans, mid);
            low = mid + 1;
          }
          else
          {
            high = mid - 1;
          }
        }

        // Keeping a track of the
        // maximum difference in indices
        result = Math.max(result, ans - i);
      }
      System.out.print(result + "\n");
    }
    }

    // This code is contributed by shikhasingrajput
```

Python3

```
# Python3 program to implement
# the above approach

# For a given array arr,
# calculates the maximum j – i
# such that arr[j] > arr[i]
```

```
    n = len(v);
    maxFromEnd = [-38749432] * (n + 1);

    # Create an array maxfromEnd
    for i in range(n - 1, 0, -1):
        maxFromEnd[i] = max(maxFromEnd[i + 1],
                            v[i]);

    result = 0;

    for i in range(0, n):
        low = i + 1; high = n - 1; ans = i;

        while (low <= high):
            mid = int((low + high) / 2);

            if (v[i] <= maxFromEnd[mid]):

                # We store this as current
                # answer and look for further
                # larger number to the right side
                ans = max(ans, mid);
                low = mid + 1;
            else:
                high = mid - 1;

        # Keeping a track of the
        # maximum difference in indices
        result = max(result, ans - i);

    print(result, end = "");

# This code is contributed by Rajput-Ji
```

C#

```
// C# program to implement
// the above approach
```

```
class GFG{

public static void Main(String[] args)
{
  int []v = {34, 8, 10, 3, 2,
             80, 30, 33, 1};
  int n = v.Length;
  int []maxFromEnd = new int[n + 1];

  for (int i = 0;
           i < maxFromEnd.Length; i++)
    maxFromEnd[i] = int.MinValue;

  // Create an array maxfromEnd
  for (int i = v.Length - 1;
           i >= 0; i--)
  {
    maxFromEnd[i] = Math.Max(maxFromEnd[i + 1],
                             v[i]);
  }

  int result = 0;

  for (int i = 0; i < v.Length; i++)
  {
    int low = i + 1,
        high = v.Length - 1,
    ans = i;

    while (low <= high)
    {
      int mid = (low + high) / 2;

      if (v[i] <= maxFromEnd[mid])
      {
        // We store this as current
        // answer and look for further
        // larger number to the right side
        ans = Math.Max(ans, mid);
        low = mid + 1;
      }
```

```
    }

    // Keeping a track of the
    // maximum difference in indices
    result = Math.Max(result, ans - i);
  }
  Console.Write(result + "\n");
}
}

// This code is contributed by shikhasingrajput
```

Javascript

```
<script>

    // Javascript program to implement
    // the above approach

    // For a given array []arr,
    // calculates the maximum j – i
    // such that arr[j] > arr[i]

    let v = [34, 8, 10, 3, 2, 80, 30, 33, 1];
    let n = v.length;
    let maxFromEnd = new Array(n + 1);

    for (let i = 0; i < maxFromEnd.length; i++)
      maxFromEnd[i] = Number.MIN_VALUE;

    // Create an array maxfromEnd
    for (let i = v.length - 1; i >= 0; i--)
    {
      maxFromEnd[i] = Math.max(maxFromEnd[i + 1], v[i]);
    }

    let result = 0;

    for (let i = 0; i < v.length; i++)
    {
```

```
            let mid = parseInt((low + high) / 2, 10);

            if (v[i] <= maxFromEnd[mid])
            {
              // We store this as current
              // answer and look for further
              // larger number to the right side
              ans = Math.max(ans, mid);
              low = mid + 1;
            }
            else
            {
              high = mid - 1;
            }
          }

          // Keeping a track of the
          // maximum difference in indices
          result = Math.max(result, ans - i);
        }
        document.write(result);

    </script>
```

Output

```
6
```

Time complexity : O(N*log(N))

Space complexity: O(N)

Method 3 O(nLgn): Use hashing and sorting to solve this problem in less than quadratic complexity after taking special care of the duplicates.

Approach :

1. Traverse the array and store the index of each element in a list (to

and j.

4. For j consider the last index from the list of possible indexes of the element and for i consider the first index from the list. (As the index was appended in ascending order).
5. Keep updating the max difference till the end of the array.

Below is the implementation of the above approach:

C++

```
// C++ implementation of
// the hashmap approach
#include <bits/stdc++.h>
using namespace std;

// Function to find maximum
// index difference
int maxIndexDiff(vector<int>& arr, int n)
{

    // Initialise unordered_map
    unordered_map<int, vector<int> > hashmap;

    // Iterate from 0 to n - 1
    for (int i = 0; i < n; i++) {
        hashmap[arr[i]].push_back(i);
    }

    // Sort arr
    sort(arr.begin(), arr.end());
    int maxDiff = INT_MIN;
    int temp = n;

    // Iterate from 0 to n - 1
    for (int i = 0; i < n; i++) {
        if (temp > hashmap[arr[i]][0]) {
            temp = hashmap[arr[i]][0];
        `
```

```
    }
    return maxDiff;
}

// Driver Code
int main()
{

    int n = 9;
    vector<int> arr{ 34, 8, 10, 3, 2, 80, 30, 33, 1 };

    // Function Call
    int ans = maxIndexDiff(arr, n);
    cout << "The maxIndexDiff is : " << ans << endl;

    return 1;
}
```

Java

```
// Java implementation of
// the hashmap approach
import java.io.*;
import java.util.*;

class GFG{

// Function to find maximum
// index difference
static int maxIndexDiff(ArrayList<Integer> arr, int n)
{

    // Initialise unordered_map
    Map<Integer,
    ArrayList<Integer>> hashmap = new HashMap<Integer,
                                      ArrayList<Integer>>();

    // Iterate from 0 to n - 1
    for(int i = 0; i < n; i++)
```

```
            }
            else
            {
                hashmap.put(arr.get(i), new ArrayList<Integer>());
                hashmap.get(arr.get(i)).add(i);
            }
        }

        // Sort arr
        Collections.sort(arr);
        int maxDiff = Integer.MIN_VALUE;
        int temp = n;

        // Iterate from 0 to n - 1
        for(int i = 0; i < n; i++)
        {
            if (temp > hashmap.get(arr.get(i)).get(0))
            {
                temp = hashmap.get(arr.get(i)).get(0);
            }
            maxDiff = Math.max(maxDiff,
            hashmap.get(arr.get(i)).get(
                hashmap.get(arr.get(i)).size() - 1) - temp);
        }
        return maxDiff;
    }

    // Driver Code
    public static void main(String[] args)
    {
        int n = 9;
        ArrayList<Integer> arr = new ArrayList<Integer>(
            Arrays.asList(34, 8, 10, 3, 2, 80, 30, 33, 1));

        // Function Call
        int ans = maxIndexDiff(arr, n);

        System.out.println("The maxIndexDiff is : " + ans);
    }
    }
```

```
# Python3 implementation of the above approach
n = 9
a = [34, 8, 10, 3, 2, 80, 30, 33, 1]

# To store the index of an element.
index = dict()
for i in range(n):
    if a[i] in index:

        # append to list (for duplicates)
        index[a[i]].append(i)
    else:

        # if first occurrence
        index[a[i]] = [i]

# sort the input array
a.sort()
maxDiff = 0

# Temporary variable to keep track of minimum i
temp = n
for i in range(n):
    if temp > index[a[i]][0]:
        temp = index[a[i]][0]
    maxDiff = max(maxDiff, index[a[i]][-1]-temp)

print(maxDiff)
```

C#

```
// C# implementation of
// the hashmap approach

using System;
using System.Collections.Generic;

public class GFG
```

```
static int maxIndexDiff(List<int> arr, int n)
{
  Dictionary<int,List<int>> hashmap = new Dictionary<int,List<int>>();

  // Iterate from 0 to n - 1
  for(int i = 0; i < n; i++)
  {
    if(hashmap.ContainsKey(arr[i]))
    {
      hashmap[arr[i]].Add(i);
    }
    else
    {
      hashmap.Add(arr[i], new List<int>());
      hashmap[arr[i]].Add(i);
    }
  }

  // Sort arr
  arr.Sort();

  int maxDiff = -1;
  int temp = n;

  // Iterate from 0 to n - 1
  for(int i = 0; i < n; i++)
  {
    if(temp > hashmap[arr[i]][0] )
    {
      temp = hashmap[arr[i]][0];
    }
    maxDiff = Math.Max(maxDiff,hashmap[arr[i]][hashmap[arr[i]].Count -
  }
  return maxDiff;
}

// Driver Code
static public void Main (){
  int n = 9;
  List<int> arr = new List<int>();
  arr.Add(34);
```

```
    arr.Add(80);
    arr.Add(30);
    arr.Add(33);
    arr.Add(1);

    // Function Call
    int ans = maxIndexDiff(arr, n);

    Console.WriteLine("The maxIndexDiff is : " + ans );
  }
}

// This code is contributed by rag2127.
```

Javascript

```
<script>
// JavaScript implementation of
// the hashmap approach

// Function to find maximum
// index difference
function maxIndexDiff(arr,n)
{

    // Initialise map in JavaScript
    let hashmap = new Map()

    // Iterate from 0 to n - 1
    for (let i = 0; i < n; i++) {

        hashmap[arr[i]] = hashmap[arr[i]] || []
        hashmap[arr[i]].push(i)

    }

    // Sort arr
    arr.sort((a,b)=> (a - b))

    let maxDiff = 0
```

```
            if (temp > hashmap[arr[i]][0]) {
                temp = hashmap[arr[i]][0]
            }
            maxDiff = Math.max( maxDiff,hashmap[arr[i]][hashmap[arr[i]].lengt

        }
        return maxDiff
    }

    // Driver Code

    let n = 9
    const arr = [ 34, 8, 10, 3, 2, 80, 30, 33, 1 ]

    // Function Call
    let ans = maxIndexDiff(arr, n)
    document.write(`The maxIndexDiff is : ${ans}`)

     // This code is contributed by shinjanpatra
     </script>
```

Output

```
 The maxIndexDiff is : 6
```

Time complexity : O(N*log(N))

Auxiliary Space: O(N)

Method 4 (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 the right index. So we construct two auxiliary arrays LMin[] and RMax[] such that LMin[i] holds the smallest element on left side of

traversing LMin[] and RMax[] 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.

Working Example:

Lets consider any example [7 3 1 8 9 10 4 5 6]

what is maxRight ?

Filling from right side 6 is first element now 6 > 5 so again we fill 6 till we reach 10 > 6 :

[10 10 10 10 10 10 6 6 6] this is maxR

[7 3 1 1 1 1 1 1 1] this is minL

now we see that how to reach answer from these to and its proof !!!

lets compare first elements of the arrays now we see 10 > 7,

now we increase maxR by 1 till it becomes lesser than 7 i.e at index 5

hence answer till now is. 5-0 = 5

now we will increase minL we get 3 which is lesser than 6 so we increase maxR till it reaches last index and the answer becomes 8-1= 7

so we see how we are getting correct answer

before index i.

in previous hint, make 2 arrays,

First, will store smallest occurring element before the element

Second, will store largest occurring element after the element

Traverse the Second array, till the element in second array is larger than or equal to First array, and store the index difference. And if it becomes smaller, traverse the first array till it again becomes larger.

And store the max difference of this index difference.

Below is the implementation of the above approach:

C++

```
#include <bits/stdc++.h>
using namespace std;

/* 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 = new int[(sizeof(int) * n)];
    int* RMax = new int[(sizeof(int) * n)];

    /* Construct LMin[] such that
    LMin[i] stores the minimum value
    from (arr[0], arr[1], ... arr[i]) */
```

```
    /* 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 Code
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);
    cout << maxDiff;
    return 0;
}

// This code is contributed by rathbhupendra
```

C

```
#include <stdio.h>
```

```
}

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;
    }
```

```
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;
}
```

Java

```
class FindMaximum {
    /* 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 RMax[] = new int[n];
        int LMin[] = new 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]);
```

```
            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 the above functions */
    public static void main(String[] args)
    {
        FindMaximum max = new FindMaximum();
        int arr[] = { 9, 2, 3, 4, 5, 6, 7, 8, 18, 0 };
        int n = arr.length;
        int maxDiff = max.maxIndexDiff(arr, n);
        System.out.println(maxDiff);
    }
}
```

Python3

```
# Utility Functions to get max
# and minimum of two integers
def max(a, b):
    if(a > b):
        return a
    else:
        return b
```

```
        return b

# For a given array arr[],
# returns the maximum j - i
# such that arr[j] > arr[i]
def maxIndexDiff(arr, n):
    maxDiff = 0;
    LMin = [0] * n
    RMax = [0] * n

    # Construct LMin[] such that
    # LMin[i] stores the minimum
    # value from (arr[0], arr[1],
    # ... arr[i])
    LMin[0] = arr[0]
    for i in range(1, n):
        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 in range(n - 2, -1, -1):
        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, j = 0, 0
    maxDiff = -1
    while (j < n and i < n):
        if (LMin[i] <= RMax[j]):
            maxDiff = max(maxDiff, j - i)
            j = j + 1
        else:
            i = i + 1

    return maxDiff
```

```
    n = len(arr)
    maxDiff = maxIndexDiff(arr, n)
    print (maxDiff)

# This code is contributed
# by gautam karakoti
```

C#

```
// C# program to find the maximum
// j – i such that arr[j] > arr[i]
using System;

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

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

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

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

        // Construct LMin[] such that LMin[i]
        // stores the minimum value
        // from (arr[0], arr[1], ... arr[i])
        LMin[0] = arr[0];
```

```
        // 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
    public static void Main()
    {

        int[] arr = { 9, 2, 3, 4, 5, 6, 7, 8, 18, 0 };
        int n = arr.Length;
        int maxDiff = maxIndexDiff(arr, n);
        Console.Write(maxDiff);
    }
}
// This Code is Contributed by Sam007
```

PHP

```php
// For a given array arr[],
// returns the maximum j - i
// such that arr[j] > arr[i]
function maxIndexDiff($arr, $n)
{
    $maxDiff = 0;
    $LMin = array_fill(0, $n, NULL);
    $RMax = array_fill(0, $n, NULL);

    // 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
```

```php
// Driver Code
$arr = array(9, 2, 3, 4, 5,
             6, 7, 8, 18, 0);
$n = sizeof($arr);
$maxDiff = maxIndexDiff($arr, $n);
echo $maxDiff;

// This code is contributed
// by ChitraNayal
?>
```

Javascript

```javascript
<script>
    // Javascript program to find the maximum
    // j – i such that arr[j] > arr[i]

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

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

    // For a given array arr[], returns
    // the maximum j-i such thatarr[j] > arr[i]
    function maxIndexDiff(arr, n)
    {
        let maxDiff;
        let i, j;

        let RMax = new Array(n);
        let LMin = new Array(n);
```

```
        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;
    }

    let arr = [ 9, 2, 3, 4, 5, 6, 7, 8, 18, 0 ];
    let n = arr.length;
    let maxDiff = maxIndexDiff(arr, n);
    document.write(maxDiff);
</script>
```

Output

```
8
```

Time Complexity: O(n)

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

Another Approach: (only using one extra array): We consider an auxiliary array : rightMax[] , such that, rightMax[i] = max element of the subarray arr[i...(n-1)], the largest or equal element after arr[i] element Suppose (arr[i], arr[jLast]) is a pair, such that arr[jLast] is the last greater or equal element than arr[i]. For the pairs ending with arr[jLast] : (arr[k], arr[jLast]) for all k = (i+1) to jLast we don't need to consider (jLast – k) because (jLast – i) > (jLast – k) for all such k's. So we can skip those pairs. Traversing from left to right of both arrays : arr[] and rightMax[] , when we first encounter rightMax[j] < arr[i[, we know that jLast = j-1, and we can skip the pairs (arr[k], arr[jLast]) for all k = (i+1) to jLast. And also rightMax[] is non increasing sequence , so all elements at right side of rightMax[j] is smaller than or equal to rightMax[j]. But there may be arr[x] after arr[i] (x > i) such that arr[x] < rightMax[j] for x > i, so increment i when rightMax[j] < arr[i] is encountered.

Below is the implementation of the above approach:

C++

```
#include <bits/stdc++.h>
using namespace std;

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

```
        //rightMax[i] = max{ arr[i...(n-1] }



        int maxDist = INT_MIN;
        int i = 0, j = 0;
        while(i<n && j<n)
        {
            if(rightMax[j] >= arr[i])
            {
                maxDist = max( maxDist, j-i );
                j++;
            }
            else // if(rightMax[j] < leftMin[i])
                i++;
        }

        return maxDist;


}

// Driver Code
int main()
{
    int arr[] = { 34,8,10,3,2,80,30,33,1};
    int n = sizeof(arr) / sizeof(arr[0]);
    int maxDiff = maxIndexDiff(arr, n);
    cout << maxDiff;
    return 0;
}

// This code is contributed by Sourashis Mondal
```

Java

```
import java.util.*;

class GFG{
```

```
  static int maxIndexDiff(int arr[], int n)
  {

    int []rightMax = new int[n];
    rightMax[n-1]= arr[n-1];
    for(int i = n-2; i>=0; i--)
      rightMax[i] = Math.max(rightMax[i+1] , arr[i]);

    // rightMax[i] = max{ arr[i...(n-1] }
    int maxDist = Integer.MIN_VALUE;
    int i = 0, j = 0;
    while(i < n && j < n)
    {
      if(rightMax[j] >= arr[i])
      {
        maxDist = Math.max( maxDist, j-i );
        j++;
      }
      else // if(rightMax[j] < leftMin[i])
        i++;
    }

    return maxDist;


  }

  // Driver Code
  public static void main(String[] args)
  {
    int arr[] = {34, 8, 10, 3, 2, 80, 30, 33, 1};
    int n = arr.length;
    int maxDiff = maxIndexDiff(arr, n);
    System.out.print(maxDiff);
  }
}

// This code is contributed by Rajput-Ji
```

Python3

```
    rightMax = [0] * n
    rightMax[n - 1] = arr[n - 1]
    for i in range(n - 2, -1, -1):
        rightMax[i] = max(rightMax[i + 1], arr[i])

    # rightMax[i] = max arr[i...(n-1]
    maxDist = -2**31
    i = 0
    j = 0

    while (i < n and j < n):
        if (rightMax[j] >= arr[i]):
            maxDist = max(maxDist, j - i)
            j += 1

        else:

            # if(rightMax[j] < leftMin[i])
            i += 1

    return maxDist

# Driver Code
arr = [ 34, 8, 10, 3, 2, 80, 30, 33, 1 ]
n = len(arr)
maxDiff = maxIndexDiff(arr, n)

print(maxDiff)

# This code is contributed by Shubham Singh
```

C#

```
/* For a given array arr[],
returns the maximum j – i such that
arr[j] > arr[i] */
using System;
```

```
  {

    int []rightMax = new int[n];
    rightMax[n - 1] = arr[n - 1];
    int i = 0, j = 0;
    for(i = n - 2; i >= 0; i--)
      rightMax[i] = Math.Max(rightMax[i+1] , arr[i]);

    // rightMax[i] = max{ arr[i...(n-1] }
    int maxDist = Int32.MinValue;
    i = 0;
    while(i < n && j < n)
    {
      if(rightMax[j] >= arr[i])
      {
        maxDist = Math.Max( maxDist, j - i);
        j++;
      }
      else // if(rightMax[j] < leftMin[i])
        i++;
    }

    return maxDist;
  }

  // Driver Code
  public static void Main()
  {
    int[] arr = {34, 8, 10, 3, 2, 80, 30, 33, 1};
    int n = arr.Length;
    int maxDiff = maxIndexDiff(arr, n);
    Console.Write(maxDiff);
  }
}

// This code is contributed by Shubham Singh
```

Javascript

```
<script>
```

```
function maxIndexDiff(arr, n)
{
    var rightMax = new Array(n).fill(0);;
    rightMax[n - 1] = arr[n - 1];
    for(var i = n - 2; i >= 0; i--){
        rightMax[i] = Math.max(rightMax[i+1] , arr[i]);
    }

    // rightMax[i] = max{ arr[i...(n-1] }
    var maxDist = Number.MIN_VALUE;
    var i = 0;
    var j = 0;
    while(i < n && j < n)
    {
        if(rightMax[j] >= arr[i])
        {
            maxDist = Math.max( maxDist, j-i );
            j++;
        }
        else // if(rightMax[j] < leftMin[i])
        {    i++;
        }
     }
     return maxDist;
}

// Driver Code
var arr = [ 34,8,10,3,2,80,30,33,1];
var n = arr.length;
var maxDiff = maxIndexDiff(arr, n);
document.write(maxDiff);

// This code is contributed by Shubham Singh

</script>
```

Output

```
6
```

Auxiliary Space: O(n)

Using leftMin[]: We can also do this using leftMin[] array only , where leftMin[i] = min element of the subarray arr[0...i]

C++

```
#include <bits/stdc++.h>
using namespace std;

/* For a given array arr[],
   returns the maximum j – i such that
   arr[j] > arr[i] */
int maxIndexDiff(int arr[], int n)
{
    int leftMin[n] ;
    leftMin[0] = arr[0];
    for(int i = 1 ; i<n; i++)
        leftMin[i] = min(leftMin[i-1], arr[i]);


    //leftMin[i] = min{ arr[0...i] }


    int maxDist = INT_MIN;
    int i = n-1, j = n-1;

    while(i>=0  &&  j>=0)
    {
        if(arr[j] >= leftMin[i])
        {
            maxDist = max(maxDist, j-i);
            i--;
        }
        else
            j--;
    }

    return maxDist;
```

```
// Driver Code
int main()
{
    int arr[] = { 34,8,10,3,2,80,30,33,1};
    int n = sizeof(arr) / sizeof(arr[0]);
    int maxDiff = maxIndexDiff(arr, n);
    cout << maxDiff;
    return 0;
}

// This code is contributed by Sourashis Mondal
```

Java

```
import java.util.*;
class GFG
{

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

    int []leftMin = new int[n];
    leftMin[0] = arr[0];
    for(int i = 1; i < n; i++)
      leftMin[i] = Math.min(leftMin[i - 1] , arr[i]);

    // leftMin[i] = min{ arr[i...(n-1] }

    int maxDist = Integer.MIN_VALUE;
    int i = n - 1, j = n - 1;
    while(i >= 0 && j >= 0)
    {
      if(arr[j] >= leftMin[i])
      {
        maxDist = Math.max( maxDist, j - i );
        i--;
      }
```

```
    return maxDist;
  }

  // Driver Code
  public static void main(String[] args)
  {
    int arr[] = {34, 8, 10, 3, 2, 80, 30, 33, 1};
    int n = arr.length;
    int maxDiff = maxIndexDiff(arr, n);
    System.out.print(maxDiff);
  }
}

// This code is contributed by Shubham Singh
```

Python3

```
# For a given array arr[],
#   returns the maximum j – i such that
#   arr[j] > arr[i] */
def maxIndexDiff(arr, n):

    leftMin = [0]*n
    leftMin[0] = arr[0]
    for i in range(1,n):
        leftMin[i] = min(leftMin[i-1], arr[i])

    # leftMin[i] = min arr[0...i]
    maxDist = - 2**32
    i = n-1
    j = n-1

    while(i>=0  and  j>=0):

        if(arr[j] >= leftMin[i]):
            maxDist = max(maxDist, j-i)
            i-=1
        else:
            j-=1
```

```
arr = [34,8,10,3,2,80,30,33,1]
n = len(arr)
maxDiff = maxIndexDiff(arr, n)
print(maxDiff)

# This code is contributed by Shubham Singh
```

C#

```
using System;

public class GFG{

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

    int []leftMin = new int[n];
    leftMin[0] = arr[0];
    int i,j;
    for( i = 1; i < n; i++)
      leftMin[i] = Math.Min(leftMin[i - 1] , arr[i]);

    // leftMin[i] = min{ arr[i...(n-1] }

    int maxDist = Int32.MinValue;
    i = n - 1;
    j = n - 1;
    while(i >= 0 && j >= 0)
    {
      if(arr[j] >= leftMin[i])
      {
        maxDist = Math.Max( maxDist, j - i );
        i--;
      }
      else
        j--;
    }
```

```
  // Driver Code
  static public void Main ()
  {
    int[] arr = {34, 8, 10, 3, 2, 80, 30, 33, 1};
    int n = arr.Length;
    int maxDiff = maxIndexDiff(arr, n);
    Console.Write(maxDiff);
  }
}

// This code is contributed by Shubham Singh
```

Javascript

```
<script>

/* For a given array arr[],
   returns the maximum j – i such that
   arr[j] > arr[i] */
function maxIndexDiff(arr, n)
{
    var leftMin = new Array(n).fill(0);;
    leftMin[0] = arr[0];
    for(var i = 1; i < n; i++){
        leftMin[i] = Math.min(leftMin[i-1] , arr[i]);
    }

    // leftMin[i] = min{ arr[i...(n-1] }
    var maxDist = Number.MIN_VALUE;
    var i = n-1;
    var j = n-1;
    while(i >= 0 && j >= 0)
    {
        if(arr[j] >= leftMin[i])
        {
            maxDist = Math.max( maxDist, j-i );
            i--;
        }
        else // if(rightMax[j] < leftMin[i])
```

```
// Driver Code
var arr = [ 34,8,10,3,2,80,30,33,1];
var n = arr.length;
var maxDiff = maxIndexDiff(arr, n);
document.write(maxDiff);

// This code is contributed by Shubham Singh

</script>
```

Output

```
6
```

Time Complexity: O(n)
Auxiliary Space: O(n)

Please suggest if someone has a better solution that is more efficient in terms of space and time.
This article is contributed by **Aarti_Rathi**. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above

between two elements such that larger element appears after the smaller number

(Maximum of all subarrays of size k)

RECOMMENDED ARTICLES

01 **Count pairs (i, j) from an array such that |arr[i]| and |arr[j]| both lies between |arr[i] - arr[j]| and |arr[i] + arr[j]|**
12, Mar 21

02 **Check whether (i,j) exists such that arr[i] != arr[j] and arr[arr[i]] is equal to arr[arr[j]]**
15, Nov 18

03 **Maximum value of |arr[0] - arr[1]| + |arr[1] - arr[2]| + ... +|arr[n - 2] - arr[n - 1]| when elements are from 1 to n**
07, Mar 19

04 **Minimize last remaining element of Array by selecting pairs such that arr[i] >= arr[j] and replace arr[i] with arr[i] - arr[j]**
27, Jan 22

05 **Count quadruples (i, j, k, l) in an array such that i < j < k < l and arr[i] = arr[k] and arr[j] = arr[l]**
17, Sep 20

06 **C++ Program to Rearrange array such that arr[i] >= arr[j] if i is even and arr[i]<=arr[j] if i is odd and j < i**
30, Dec 21

07 **Java Program to Rearrange array such that arr[i] >= arr[j] if i is even and arr[i]<=arr[j] if i is odd and j < i**
30, Dec 21

08 **Python3 Program to Rearrange array such that arr[i] >= arr[j] if i is even and arr[i]<=arr[j] if i is odd and j < i**
30, Dec 21

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Hard

Easy | Normal | Medium | Hard | Expert

Improved By : Sam007, Gautam Karakoti, ukasp, Akanksha_Rai, rathbhupendra, msg, executable, shikhasingrajput, Rajput-Ji, winter_soldier, avanitrachhadiya2155, rag2127, mank1083, decode2207, suresh07, abhijit gupta 1, 2011harshgupta, dheerwani2498, sourashis69, SHUBHAMSINGH10, simranarora5sos, shinjanpatra, pushpeshrajdx01, sachinvinod1904

Article Tags : Amazon, Snapdeal, Arrays

Practice Tags : Amazon, Snapdeal, Arrays

Improve Article | Report Issue

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

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

About Us

Careers

In Media

Contact Us

Privacy Policy

Copyright Policy

Learn

Algorithms

Data Structures

SDE Cheat Sheet

Machine learning

CS Subjects

Video Tutorials

Courses

News

Top News

Technology

Work & Career

Business

Finance

Lifestyle

Knowledge

Languages

Python

Java

CPP

Golang

C#

SQL

Kotlin

Web Development

Web Tutorials

Contribute

Write an Article

NodeJS