









Videos



Problems



Quiz



Next >>

Previous Greater Element

Given an array of distinct elements, find previous greater element for every element. If previous greater element does not exist, print -1.

Examples:

```
Input : arr[] = {10, 4, 2, 20, 40, 12, 30}
Output : -1, 10, 4, -1, -1, 40, 40
```

Input : arr[] = {10, 20, 30, 40}
Output : -1, -1, -1, -1

Input : arr[] = {40, 30, 20, 10}
Output : -1, 40, 30, 20

Expected time complexity: O(n)

A **simple solution** is to run two nested loops. The outer loop picks an element one by one. The inner loop, find the previous element that is greater.

Implementation:



```
C++
        Java
// Java program previous greater element
 // A naive solution to print
 // previous greater element
// for every element in an array.
import java.io.*;
import java.util.*;
import java.lang.*;
 class GFG
 static void prevGreater(int arr[],
                         int n)
     // Previous greater for
     // first element never
     // exists, so we print -1.
     System.out.print("-1, ");
     // Let us process
     // remaining elements.
```

for (int i = 1; i < n; i++)

// Find first element on

// greater than arr[i].

// left side that is







```
int j;
        for (j = i-1; j >= 0; j--)
            if (arr[i] < arr[j])</pre>
            System.out.print(arr[j] + ", ");
            break;
        // If all elements on
        // left are smaller.
        if (j == -1)
        System.out.print("-1, ");
// Driver Code
public static void main(String[] args)
    int arr[] = \{10, 4, 2, 20, 40, 12, 30\};
    int n = arr.length;
    prevGreater(arr, n);
```

Output

```
-1, 10, 4, -1, -1, 40, 40,
```

Dash

all

Ф

Articles

 \triangleright

Videos



Problems



Quiz

<<

>>

An **efficient solution** is to use stack data structure. If we take a closer look, we can notice that this problem is a variation of stock span problem. We maintain previous greater element in a stack.

```
C++ Java
```

```
// Java program previous greater element
// An efficient solution to
// print previous greater
// element for every element
// in an array.
import java.io.*;
import java.util.*;
import java.lang.*;
class GFG
static void prevGreater(int arr[],
                        int n)
    // Create a stack and push
    // index of first element
    // to it
    Stack<Integer> s = new Stack<Integer>();
    s.push(arr[0]);
    // Previous greater for
```



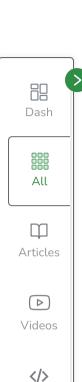


```
// first element is always -1.
                      System.out.print("-1, ");
                      // Thousance nemaining elements
 Get 90% Refund!
                                               Contests
              Tutorials
                            Jobs
                                      Practice
 Courses
                      ι
  000
                          // Pop elements from stack
  Αll
                          // while stack is not empty
                          // and top of stack is smaller
  \Box
                          // than arr[i]. We always have
Articles
                          // elements in decreasing order
                          // in a stack.
  while (s.empty() == false &&
Videos
                              s.peek() < arr[i])</pre>
                              s.pop();
  </>
Problems
                          // If stack becomes empty, then
                          // no element is greater on left
  (?)
                          // side. Else top of stack is
 Quiz
                          // previous greater.
                          if (s.empty() == true)
                              System.out.print("-1, ");
                          else
                              System.out.print(s.peek() + ", ");
                          s.push(arr[i]);
<<
   >>
```

P







Problems

(?)

Quiz

```
// Driver Code
public static void main(String[] args)
{
   int arr[] = { 10, 4, 2, 20, 40, 12, 30 };
   int n = arr.length;
   prevGreater(arr, n);
}
}
```

Output



Complexity Analysis:

- Time Complexity: O(n). It seems more than O(n) at first look. If we take a closer look, we can observe that every element of array is added and removed from stack at most once. So there are total 2n operations at most. Assuming that a stack operation takes O(1) time, we can say that the time complexity is O(n).
- Auxiliary Space: O(n) in worst case when all elements are sorted in decreasing order.

Mark as Read

Report An Issue

If you are facing any issue on this page. Please let us know.