***Leaders in an Array problem***

Write a program to print all the LEADERS in the array. An element is a leader if it is greater than all the elements to its right side. And the rightmost element is always a leader.

**For example:**

***Input:****arr[] = {16, 17, 4, 3, 5, 2},*  
***Output:****17, 5, 2*

***Input:****arr[] = {1, 2, 3, 4, 5, 2},*  
***Output:****5, 2*

**Naive Approach**: The problem can be solved based on the idea mentioned below:

*Use two loops. The outer loop runs from 0 to size – 1 and one by one pick all elements from left to right. The inner loop compares the picked element to all the elements on its right side. If the picked element is greater than all the elements to its right side, then the picked element is the leader.*

Follow the below steps to implement the idea:

* We run a loop from the first index to the 2nd last index.
  + And for each index, we run another loop from the next index to the last index.
  + If all the values to the right of that index are smaller than the index, we simply add the value in our answer data structure.

Below is the implementation of the above approach.

C++Java

class LeadersInArray

{

/\*Java Function to print leaders in an array \*/

void printLeaders(int arr[], int size)

{

for (int i = 0; i < size; i++)

{

int j;

for (j = i + 1; j < size; j++)

{

if (arr[i] <=arr[j])

break;

}

if (j == size) // the loop didn't break

System.out.print(arr[i] + " ");

}

}

/\* Driver program to test above functions \*/

public static void main(String[] args)

{

LeadersInArray lead = new LeadersInArray();

int arr[] = new int[]{16, 17, 4, 3, 5, 2};

int n = arr.length;

lead.printLeaders(arr, n);

}

}

**Output**

17 5 2

**Time Complexity:** O(N \* N)  
**Auxiliary Space:**O(1)

**Find Leader by finding suffix maximum:**

*The idea is to scan all the elements from right to left in an array and keep track of the maximum till now. When the maximum changes its value, print it.*

Follow the below illustration for a better understanding

**Illustration:**

*Let the array be arr[] = {16, 17, 4, 3, 5, 2}*

* *arr[] = {16, 17, 4, 3, 5,****2****} , max\_from\_right = 2 , ans[] = { 2 }*
* *arr[] = {16, 17, 4, 3,****5****, 2} , max\_from\_right = 5 , ans[] = { 2, 5 }*
* *arr[] = {16, 17, 4,****3****, 5, 2} , max\_from\_right = 5 , ans[] = { 2, 5 }*
* *arr[] = {16, 17,****4****, 3,****5****, 2} , max\_from\_right = 5 , ans[] = { 2, 5 }*
* *arr[] = {16,****17****, 4, 3, 5, 2} , max\_from\_right = 17 , ans[] = { 2, 5, 17 }*
* *arr[] = {****16****, 17, 4, 3, 5, 2} , max\_from\_right = 17 , ans[] = { 2, 5, 17 }*

Follow the steps mentioned below to implement the idea:

* We start from the last index position. The last position is always a leader, as there are no elements towards its right.
* And then we iterate on the array till we reach index position = 0.
  + Each time we keep a check on the maximum value
  + Every time we encounter a maximum value than the previous maximum value encountered, we either print or store the value as it is the leader

Below is the implementation of the above approach.

C++Java

class LeadersInArray

{

/\* Java Function to print leaders in an array \*/

void printLeaders(int arr[], int size)

{

int max\_from\_right = arr[size-1];

/\* Rightmost element is always leader \*/

System.out.print(max\_from\_right + " ");

for (int i = size-2; i >= 0; i--)

{

if (max\_from\_right < arr[i])

{

max\_from\_right = arr[i];

System.out.print(max\_from\_right + " ");

}

}

}

/\* Driver program to test above functions \*/

public static void main(String[] args)

{

LeadersInArray lead = new LeadersInArray();

int arr[] = new int[]{16, 17, 4, 3, 5, 2};

int n = arr.length;

lead.printLeaders(arr, n);

}

}

**Output**

2 5 17

**Time Complexity:** O(n)  
**Auxiliary Space:**O(1)