Median of sliding window in an array

Difficulty Level : Expert • Last Updated : 13 Jan, 2020

Given an array of integer **arr[]** and an integer **k**, the task is to find the median of each window of size **k** starting from the left and moving towards the right by one position each time.

Examples:

Input: *arr[] = {-1, 5, 13, 8, 2, 3, 3, 1}, k = 3*
Output: *5 8 8 3 3 3*

Input: *arr[] = {-1, 5, 13, 8, 2, 3, 3, 1}, k = 4*
Output: *6.5 6.5 5.5 3.0 2.5*

Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.

Approach: Create a pair class to hold the items and their index. It also implements the comparable interface so that compareTo() method will

items in single remove() call if we only check for the value.

The idea is to maintain two sorted sets (minSet and maxSet) of Pair objects of length **(k / 2)** and **(k / 2) + 1** depending on whether **k** is even or odd, minSet will always contain the first set of numbers (smaller) of window **k** and maxSet will contain the second set of numbers (larger).

As we move our window, we will remove elements from either of the sets (log n) and add a new element (log n) maintaining the minSet and maxSet rule specified above.

Below is the implementation of the above approach:

```
// Java implementation of the approach
import java.util.TreeSet;

public class GFG {

    // Pair class for the value and its index
    static class Pair implements Comparable<Pair> {
        private int value, index;

        // Constructor
        public Pair(int v, int p)
        {
            value = v;
            index = p;
        }

        // This method will be used by the treeset to
        // search a value by index and setting the tree
        // nodes (left or right)
        @Override
        public int compareTo(Pair o)
        {

            // Two nodes are equal only when
            // their indices are same
```

```
            return Integer.compare(index, o.index);
        }
        else {
            return Integer.compare(value, o.value);
        }
    }

    // Function to return the value
    // of the current object
    public int value()
    {
        return value;
    }

    // Update the value and the position
    // for the same object to save space
    public void renew(int v, int p)
    {
        value = v;
        index = p;
    }

    @Override
    public String toString()
    {
        return String.format("(%d, %d)", value, index);
    }
}

// Function to print the median for the current window
static void printMedian(TreeSet<Pair> minSet,
                        TreeSet<Pair> maxSet, int window)
{

    // If the window size is even then the
    // median will be the average of the
    // two middle elements
    if (window % 2 == 0) {
        System.out.print((minSet.last().value()
                          + maxSet.first().value())
                         / 2.0);
```

```
        else {
            System.out.print(minSet.size() > maxSet.size()
                                ? minSet.last().value()
                                : maxSet.first().value());
            System.out.print(" ");
        }
    }

    // Function to find the median
    // of every window of size k
    static void findMedian(int arr[], int k)
    {
        TreeSet<Pair> minSet = new TreeSet<>();
        TreeSet<Pair> maxSet = new TreeSet<>();

        // To hold the pairs, we will keep renewing
        // these instead of creating the new pairs
        Pair[] windowPairs = new Pair[k];

        for (int i = 0; i < k; i++) {
            windowPairs[i] = new Pair(arr[i], i);
        }

        // Add k/2 items to maxSet
        for (int i = 0; i < k / 2; i++) {
            maxSet.add(windowPairs[i]);
        }

        for (int i = k / 2; i < k; i++) {

            // Below logic is to maintain the
            // maxSet and the minSet criteria
            if (arr[i] < maxSet.first().value()) {
                minSet.add(windowPairs[i]);
            }
            else {
                minSet.add(maxSet.pollFirst());
                maxSet.add(windowPairs[i]);
            }
        }
```

```
            // Get the pair at the start of the window, this
            // will reset to 0 at every k, 2k, 3k, ...
            Pair temp = windowPairs[i % k];
            if (temp.value() <= minSet.last().value()) {

                // Remove the starting pair of the window
                minSet.remove(temp);

                // Renew window start to new window end
                temp.renew(arr[i], i);

                // Below logic is to maintain the
                // maxSet and the minSet criteria
                if (temp.value() < maxSet.first().value()) {
                    minSet.add(temp);
                }
                else {
                    minSet.add(maxSet.pollFirst());
                    maxSet.add(temp);
                }
            }
            else {
                maxSet.remove(temp);
                temp.renew(arr[i], i);

                // Below logic is to maintain the
                // maxSet and the minSet criteria
                if (temp.value() > minSet.last().value()) {
                    maxSet.add(temp);
                }
                else {
                    maxSet.add(minSet.pollLast());
                    minSet.add(temp);
                }
            }

            printMedian(minSet, maxSet, k);
        }
    }

    // Driver code
```

```
        int k = 3;

        findMedian(arr, k);
    }
```

Output:

```
1 8 2 7 3 6 4 5
```


Previous

Median of sliding window in an array | Set 2

Next

Java SAX Library

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Median of sliding window in an array | Set 2**

05 **Window Sliding Technique**
16, Apr 17

(Maximum of all subarrays of size k) using stack in O(n) time
25, Apr 19

03 Sliding Window Maximum : Set 2
05, Apr 20

04 Sliding Window Maximum (Maximum of all subarrays of size k)
27, May 11

04, Aug 21

07 Maximum possible sum of a window in an array such that elements of same window in other array are unique
04, Aug 17

08 Minimum window size containing atleast P primes in every window of given range
08, Apr 20

Article Contributed By :

Rahul_Titare
@Rahul_Titare

Vote for difficulty

Current difficulty : Expert

Easy | Normal | Medium | Hard | Expert

Article Tags : median-finding, sliding-window, Arrays, Binary Search Tree, Dynamic Programming, Java Programs, Sorting

Practice Tags : sliding-window, Arrays, Dynamic Programming, Sorting

Improve Article

Report Issue

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

Load Comments

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

feedback@geeksforgeeks.org

NodeJS