Sliding Window Maximum (Maximum of all subarrays of size k)

Given an array and an integer k, find the maximum for each and every contiguous subarray of size k.

Examples :

Input :
arr[] = {1, 2, 3, 1, 4, 5, 2, 3, 6}
k = 3
Output :
3 3 4 5 5 5 6

Input :
arr[] = {8, 5, 10, 7, 9, 4, 15, 12, 90, 13}
k = 4
Output :
10 10 10 15 15 90 90

Recommended: Please solve it on "*PRACTICE* " first, before moving on to the solution.

Method 1 (Simple)
Run two loops. In the outer loop, take all subarrays of size k. In the inner loop, get the maximum of the current subarray.

C/C++

```
#include<stdio.h>

void printKMax(int arr[], int n, int k)
{
    int j, max;
```

```
    for (int i = 0; i <= n-k; i++)
    {
        max = arr[i];

        for (j = 1; j < k; j++)
        {
            if (arr[i+j] > max)
               max = arr[i+j];
        }
        printf("%d ", max);
    }
}


int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;
    printKMax(arr, n, k);
    return 0;
}
```

Java

```
//Java Program to find the maximum for each and every contiguous subarray of size k

public class GFG
{
    // Method to find the maximum for each and every contiguous subarray of size k.
    static void printKMax(int arr[], int n, int k)
    {
        int j, max;

        for (int i = 0; i <= n - k; i++) {

            max = arr[i];

            for (j = 1; j < k; j++)
            {
                if (arr[i + j] > max)
                    max = arr[i + j];
            }
            System.out.print(max + " ");
        }
    }

    // Driver method
    public static void main(String args[])
    {
        int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int k = 3;
        printKMax(arr, arr.length, k);
    }
}

//This code is contributed by Sumit Ghosh
```

Python3

```
# Python program to find the maximum for
# each and every contiguous subarray of
# size k

# Method to find the maximum for each
# and every contiguous subarray of s
# of size k
def printMax(arr, n, k):
    max = 0

    for i in range(n - k + 1):
        max = arr[i]
        for j in range(1, k):
            if arr[i+j] > max:
                max = arr[i + j]
        print(str(max) + " ", end = "")

# Driver method
if __name__=="__main__":
    arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    n = len(arr)
    k = 3
    printMax(arr, n, k)

# This code is contributed by Shiv Shankar
```

C#

```
// C# program to find the maximum for
// each and every contiguous subarray of
// size kusing System;
using System;

class GFG
{
    // Method to find the maximum for
    // each and every contiguous subarray
    // of size k.
    static void printKMax(int []arr, int n, int k)
    {
        int j, max;

        for (int i = 0; i <= n - k; i++) {

            max = arr[i];

            for (j = 1; j < k; j++)
            {
                if (arr[i + j] > max)
                    max = arr[i + j];
            }
            Console.Write(max + " ");
```

```
        }
    }

    // Driver method
    public static void Main()
    {
        int []arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int k = 3;
        printKMax(arr, arr.Length, k);
    }
}

// This Code is Contributed by Sam007
```

PHP

```
<?php
// PHP program to find the maximum
// for each and every contiguous
// subarray of size k

function printKMax($arr, $n, $k)
{
    $j; $max;

    for ($i = 0; $i <= $n - $k; $i++)
    {
        $max = $arr[$i];

        for ($j = 1; $j < $k; $j++)
        {
            if ($arr[$i + $j] > $max)
            $max = $arr[$i + $j];
        }
        printf("%d ", $max);
    }
}

// Driver Code
$arr = array(1, 2, 3, 4, 5,
             6, 7, 8, 9, 10);
$n = count($arr);
$k = 3;
printKMax($arr, $n, $k);

// This Code is Contributed by anuj_67.
?>
```

Output :

```
 3 4 5 6 7 8 9 10
```

Time Complexity : The outer loop runs n-k+1 times and the inner loop runs k times for every iteration of outer loop. So time complexity is O((n-k+1)*k) which can also be written as O(nk).

Method 2 (Use Self-Balancing BST)

1) Pick first k elements and create a Self-Balancing Binary Search Tree (BST) of size k.

2) Run a loop for i = 0 to n – k

…..a) Get the maximum element from the BST, and print it.

…..b) Search for arr[i] in the BST and delete it from the BST.

…..c) Insert arr[i+k] into the BST.

Time Complexity: Time Complexity of step 1 is O(kLogk). Time Complexity of steps 2(a), 2(b) and 2(c) is O(Logk). Since steps 2(a), 2(b) and 2(c) are in a loop that runs n-k+1 times, time complexity of the complete algorithm is O(kLogk + (n-k+1)*Logk) which can also be written as O(nLogk).

Method 3 (A O(n) method: use Dequeue)

We create a Dequeue, *Qi* of capacity k, that stores only useful elements of current window of k elements. An element is useful if it is in current window and is greater than all other elements on left side of it in current window. We process all array elements one by one and maintain *Qi* to contain useful elements of current window and these useful elements are maintained in sorted order. The element at front of the *Qi* is the largest and element at rear of *Qi* is the smallest of current window. Thanks to Aashish for suggesting this method.

Following is the implementation of this method.

C++

```
#include <iostream>
#include <deque>

using namespace std;

// A Dequeue (Double ended queue) based method for printing maixmum element of
// all subarrays of size k
void printKMax(int arr[], int n, int k)
{
    // Create a Double Ended Queue, Qi that will store indexes of array elements
    // The queue will store indexes of useful elements in every window and it will
    // maintain decreasing order of values from front to rear in Qi, i.e.,
    // arr[Qi.front[]] to arr[Qi.rear()] are sorted in decreasing order
    std::deque<int>  Qi(k);

    /* Process first k (or first window) elements of array */
    int i;
```

```
    for (i = 0; i < k; ++i)
    {
        // For very element, the previous smaller elements are useless so
        // remove them from Qi
        while ( (!Qi.empty()) && arr[i] >= arr[Qi.back()])
            Qi.pop_back();  // Remove from rear

        // Add new element at rear of queue
        Qi.push_back(i);
    }

    // Process rest of the elements, i.e., from arr[k] to arr[n-1]
    for ( ; i < n; ++i)
    {
        // The element at the front of the queue is the largest element of
        // previous window, so print it
        cout << arr[Qi.front()] << " ";

        // Remove the elements which are out of this window
        while ( (!Qi.empty()) && Qi.front() <= i - k)
            Qi.pop_front();  // Remove from front of queue

        // Remove all elements smaller than the currently
        // being added element (remove useless elements)
        while ( (!Qi.empty()) && arr[i] >= arr[Qi.back()])
            Qi.pop_back();

         // Add current element at the rear of Qi
        Qi.push_back(i);
    }

    // Print the maximum element of last window
    cout << arr[Qi.front()];
}

// Driver program to test above functions
int main()
{
    int arr[] = {12, 1, 78, 90, 57, 89, 56};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;
    printKMax(arr, n, k);
    return 0;
}
```

Java

```
//Java Program to find the maximum for each and every contiguous subarray of size k

import java.util.Deque;
import java.util.LinkedList;

public class SlidingWindow
{

    // A Dequeue (Double ended queue) based method for printing maixmum element of
```

```
    // all subarrays of size k
    static void printMax(int arr[],int n, int k)
    {
        // Create a Double Ended Queue, Qi that will store indexes of array element
        // The queue will store indexes of useful elements in every window and it w
        // maintain decreasing order of values from front to rear in Qi, i.e.,
        // arr[Qi.front[]] to arr[Qi.rear()] are sorted in decreasing order
        Deque<Integer> Qi = new LinkedList<Integer>();

        /* Process first k (or first window) elements of array */
        int i;
        for(i = 0; i < k; ++i)
        {
            // For very element, the previous smaller elements are useless so
            // remove them from Qi
            while(!Qi.isEmpty() && arr[i] >= arr[Qi.peekLast()])
                Qi.removeLast();   // Remove from rear

            // Add new element at rear of queue
            Qi.addLast(i);
        }

        // Process rest of the elements, i.e., from arr[k] to arr[n-1]
        for( ;i < n; ++i)
        {
            // The element at the front of the queue is the largest element of
            // previous window, so print it
            System.out.print(arr[Qi.peek()] + " ");

            // Remove the elements which are out of this window
            while((!Qi.isEmpty()) && Qi.peek() <= i-k)
                Qi.removeFirst();

            // Remove all elements smaller than the currently
            // being added element (remove useless elements)
            while((!Qi.isEmpty()) && arr[i] >= arr[Qi.peekLast()])
                Qi.removeLast();


            // Add current element at the rear of Qi
            Qi.addLast(i);

        }

        // Print the maximum element of last window
        System.out.print(arr[Qi.peek()]);
    }

    // Driver program to test above functions
    public static void main(String[] args)
    {
        int arr[]={12, 1, 78, 90, 57, 89, 56};
        int k=3;
        printMax(arr, arr.length,k);
    }

}
//This code is contributed by Sumit Ghosh
```

Python3

```
# Python program to find the maximum for
# each and every contiguous subarray of
# size k

from collections import deque

# A Deque (Double ended queue) based
# method for printing maximum element
# of all subarrays of size k
def printMax(arr, n, k):

    """ Create a Double Ended Queue, Qi that
    will store indexes of array elements.
    The queue will store indexes of useful
    elements in every window and it will
    maintain decreasing order of values from
    front to rear in Qi, i.e., arr[Qi.front[]]
    to arr[Qi.rear()] are sorted in decreasing
    order"""
    Qi = deque()

    # Process first k (or first window)
    # elements of array
    for i in range(k):

        # For every element, the previous
        # smaller elements are useless
        # so remove them from Qi
        while Qi and arr[i] >= arr[Qi[-1]] :
            Qi.pop()

        # Add new element at rear of queue
        Qi.append(i);

    # Process rest of the elements, i.e.
    # from arr[k] to arr[n-1]
    for i in range(k, n):

        # The element at the front of the
        # queue is the largest element of
        # previous window, so print it
        print(str(arr[Qi[0]]) + " ", end = "")

        # Remove the elements which are
        # out of this window
        while Qi and Qi[0] <= i-k:

            # remove from front of deque
            Qi.popleft()

        # Remove all elements smaller than
        # the currently being added element
        # (Remove useless elements)
        while Qi and arr[i] >= arr[Qi[-1]] :
            Qi.pop()
```

```
            # Add current element at the rear of Qi
            Qi.append(i)

        # Print the maximum element of last window
        print(str(arr[Qi[0]]))

# Driver programm to test above fumctions
if __name__=="__main__":
    arr = [12, 1, 78, 90, 57, 89, 56]
    k = 3
    printMax(arr, len(arr), k)

# This code is contributed by Shiv Shankar
```

Output:

```
 78 90 90 90 89
```

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 at most once. So there are total 2n operations.
Auxiliary Space: O(k)

Below is an extension of this problem.
Sum of minimum and maximum elements of all subarrays of size k.

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

Recommended Posts:

Find maximum of minimum for every window size in a given array

Maximum subarray size, such that all subarrays of that size have sum less than k

Maximum sum two non-overlapping subarrays of given size

Sum of minimum and maximum elements of all subarrays of size k.

Window Sliding Technique

Maximum possible sum of a window in an array such that elements of same window in other array are unique

Number of subarrays whose minimum and maximum are same

Count of subarrays whose maximum element is greater than k

Number of subarrays with maximum values in given range

Maximum sum of lengths of non-overlapping subarrays with k as the max element.

Partition into two subarrays of lengths k and (N - k) such that the difference of sums is maximum

Split array to three subarrays such that sum of first and third subarray is equal and maximum

Maximum size subset with given sum

Size of The Subarray With Maximum Sum

Subsequence of size k with maximum possible GCD

Improved By : vt_m

Article Tags : Arrays Queue Amazon Directi Flipkart sliding-window

Practice Tags : Amazon Flipkart Directi sliding-window Arrays Queue

8

3.5

To-do Done

Based on **406** vote(s)

Feedback Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

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

Share this post!

107 Comments **GeeksforGeeks** **Login**

Recommend 8 **Tweet** **Share** **Sort by Newest**

Join the discussion…

LOG IN WITH **OR SIGN UP WITH DISQUS**

Name

Amar Vashishth • 12 days ago

better solution explained at:

https://stackoverflow.com/a...

• Reply • Share ›

Deepak Srivastava • 22 days ago

```
int minIndex = 0;
int maxIndex = 0;
int count = 0;
int maxSoFar = Integer.MIN_VALUE;
int nextMaxSoFar=Integer.MIN_VALUE;
while (count <= ar.length) {
if (maxIndex - minIndex < k) {
if (count < ar.length && ar[count] > maxSoFar) {
nextMaxSoFar = maxSoFar;
maxSoFar = ar[count];

} else if (count < ar.length && ar[count] > nextMaxSoFar && nextMaxSoFar != maxSoFar) {
nextMaxSoFar = ar[count];
}
maxIndex++;
count++;
}
else if (maxIndex - minIndex == k) {
System.out.println(maxSoFar + " ");
if (minIndex < ar.length && ar[minIndex] == maxSoFar) {
maxSoFar = nextMaxSoFar;

}
minIndex++;
}
}
```

• Reply • Share ›

Harshit Gupta • 2 months ago

Do you guys think that this can also be implemented using a priority queue(max Heap).
Keep a window of K and keep it sorted. Keep inserting the arr[i] and popping out the element which are left. Keep showing the maxHeap[0] to get the largest of all subarrays.

What do you guys think? Any ideas?

2 • Reply • Share ›

Joey ➚ Harshit Gupta • a day ago

MaxHeap of size k will take O(n*Logk) time complexity, Since every time we have to extract large element from MaxHeap and insert new element and MxHeapify again.

• Reply • Share ›

Neil Thaker • 2 months ago

How method-3 is O(n) time complexity? Loop iteration is n times and each time in worst case we are looping k times. So it is O(n*k). Even though we remove from the Dqueue and peeking last. Correct me if I have mistaken. Thanks.

• Reply • Share ›

supernovagu ➚ Neil Thaker • 2 months ago

But after a worst case, all k elements in the dequeue will be removed. So next time, there will be no more comparison and remove for these elements. So in total, there will be no more than n peeking and comparing. So it's correct the complexity is O(n).

Reply • Share ›

Anubhav Shukla • 2 months ago

No need of using while loop to remove the nodes which are out of the window . Because either one element will be removed from the front or no element will be removed in case of arr[i] < arr[Qi.back()] . Just an if condition will suffice

Reply • Share ›

Vaibhav Aggarwal • 3 months ago

Watch this before reading article https://www.youtube.com/wat...

Reply • Share ›

Anchal Gupta • 3 months ago

I have this doubt that how come arr[ ] could take null as a value when Qi.peekLast( ) will give null value on being empty so arr[Qi.peekLast()] will give error but why is it not giving any error...? Because array will take numerical value as in input....Can anyone please explain this to me ...?

Reply • Share ›

surbhi kohli • 5 months ago

In the Deque solution ,if someone could explain the following

while((!Qi.isEmpty()) && Qi.peek() <= i-k)
Qi.removeFirst();

Reply • Share ›

surbhi kohli ➚ surbhi kohli • 5 months ago

Well,I got the logic. We check if the element at Qi.peek (array index) is less than i-k(and if that is a case, it implies that the index is not in the current window so remove that index from deque.

Reply • Share ›

Rathin Sen ➚ surbhi kohli • 4 months ago

Even I got the logic, but why a while loop?
a simple "if(Qi.peek() <= i-k)" would have sufficed, since all the entries <= i-k would have been removed in previous iterations.

Correct me if I'm wrong.

Reply • Share ›

surbhi kohli ➚ Rathin Sen • 4 months ago

Correct Observation.An if condition would be Ok

Reply • Share ›

Varan Shukla • 5 months ago

Correction: An element is useful if it is in current window and is greater than all other elements on RIGHT side of it in current window.

3 Reply • Share ›

Avinash Singhal ➚ Varan Shukla • 5 months ago

yes, it should be corrected, unnecessarily wasted my 15 minutes

Reply • Share ›

Jay • 6 months ago

Please correct me if my solution is wrong. I will first calculate the max of first k values and print it, I will also save the max among i+1 till k-1 values. Now I will loop through starting from k'th value and compare it with max and update it if the current value is greater than max. I will repeat this until I reach n-k+1 index. I'll have O(n) with O(k) auxiliary space.

Reply • Share ›

RAVIKANT TATIWAL ➚ Jay • 3 months ago

by this method you will calculate max element in array
not max in each window .
take ex. arr[]={12,1,78,90,56,89,57}

Reply • Share ›

shiva • 6 months ago

DP solution : https://ide.geeksforgeeks.o...

```
#include <iostream>
#include<limits.h>

using namespace std;
int hsh[1000][1000];

int findmax1(int arr[], int i, int k)
{
if(hsh[i][k]!=INT_MIN)
return hsh[i][k];

if(i==k)
{
hsh[i][k]=arr[i];

return hsh[i][k];
}
```

see more

Reply • Share ›

Navjot Singh • 6 months ago

Inside the second loop where we remove the elements out of window, it is enough to check with if condition only rather than while loop.

In place of
while((!Qi.isEmpty()) && Qi.peek() <= i-k)
Qi.removeFirst();

It is enough to do
if(Qi.peek() <= i-k)
Qi.removeFirst();

This works because at any point only one element can go out of the window and that is which is at the

front of the deque because we are adding only one element to deque. Also the deque will never be empty at this point. We can see that we are printing the value just before it and there is no case where it can be empty. This was just to simplify the code :)

2 ▲ | ▼ • Reply • Share ›

delta_x ➚ Navjot Singh • 4 months ago

Nice observation.
I think we can just write equality sign and it should work.
if(Qi.peek() == i-k)

What do you think?

1 ▲ | ▼ • Reply • Share ›

Navjot Singh ➚ delta_x • 3 months ago

Yes, equality sign would be more appropriate.

▲ | ▼ • Reply • Share ›

Aniket Tikalkar ➚ Navjot Singh • 5 months ago

Had the same thought, Your analysis is correct, that while loop is unnecessary!

1 ▲ | ▼ • Reply • Share ›

Debraj Ray • 6 months ago

A detailed explanation of the working of the last algorithm (using Dequeue). This is for people finding it hard to follow the explanation/code provided by the author :)

We will maintain a Queue (Q) which is sorted in descending order from front to end. For keeping it sorted, we will not actually sort, but instead, remove elements (also called useless elements) that breaks the sorting.

For example, in the window (8, 5, 10, 7) of size k = 4, following will be the way to create the Q
Q = 8 => Q = 8, 5 => Q = 10 , notice 8 and 5 are removed as they are less than 10 => Q = 10, 7

We now continue the same process as we move the window forward. However, we also start dropping elements from the Q when their indices goes out of the window frame.

We print the first element of Q and then update Q. Printed -> 10

window - 5, 10, 7, 9
Q = 10, 9 (7 dropped)

Printed -> 10 10

see more

9 ▲ | ▼ • Reply • Share ›

Hiccup Coder ➚ Debraj Ray • a month ago

This should be part of main post...

▲ | ▼ • Reply • Share ›

Akash Patel ➚ Debraj Ray • 4 months ago

thanks bro

2 ▲ | ▼ • Reply • Share ›

Navjot Singh ➚ Debraj Ray • 6 months ago
Surely better than what is given in the post.
^ | v • Reply • Share ›

TANYA SHRIVASTAVA • 7 months ago
https://ide.geeksforgeeks.o...

o(n) time..
^ | v • Reply • Share ›

anirudh chugh ➚ TANYA SHRIVASTAVA • 4 months ago
time complexity of your algo is O(n*k)
^ | v • Reply • Share ›

Hitman47 ➚ TANYA SHRIVASTAVA • 5 months ago
nice simple implementation.....
^ | v • Reply • Share ›

Vishwa Mohan • 7 months ago
Solved in O(n) times
public class MaxWindowValue {

private static void printMaxWindowValue(int[] arr , int n, int k){
Deque< Integer> q = new LinkedList<>();

for(int i=0 ;i <k ;i++){="" if(q.isempty()){="" q.add(i);="" }else{="" if(!q.isempty()="" &&="" arr[i]="">=arr[q.peekLast()]){
q.removeLast();
q.addLast(i);
}
}
}

for(int i=k ;i<n ;i++){="" system.out.println(arr[q.peekfirst()]);="" if(q.isempty()){="" q.add(i);="" continue="" ;="" }="" if(!q.isempty()="" &&="" (="" i-q.peekfirst()="" <="0)){" q.removefirst();="" }="" if(!q.isempty()="" &&="" arr[q.peeklast()]="" <="arr[i]){" q.removelast();="" q.addlast(i);="" }="" }="" system.out.println(arr[q.peekfirst()]);="" }="" public="" static="" void="" main(string="" args[]){="" int="" arr[]="{1," 2,="" 3,="" 1,="" 4,="" 5,="" 2,="" 3,="" 6};="" printmaxwindowvalue(arr,="" arr.length,="" 3);="" }="" }="">
^ | v • Reply • Share ›

rohan talwar • 8 months ago
this question can also be solve by using sliding window technique in O(n) time
^ | v • Reply • Share ›

Shivan ➚ rohan talwar • 7 months ago
This algorithm uses sliding window itself
^ | v • Reply • Share ›

kannaya_18 • 8 months ago

in method 1: (JAVA code)
why do we need to pass size of array ??

as we can calculate the size by arr.length....
Can anyone explain me why ? or any special reason ?

Reply • Share ›

Siva Subramanian • 8 months ago

Java Solution using LinkedList
private static void printMaxNumSubArray(int[] arr, int k) {
// TODO Auto-generated method stub

LinkedList<integer> list = new LinkedList<>();
int max=0;int i=0;
int counter=0;
for(i=0;i<k;i++){ if(max<arr[i]){="" max="arr[i];" }="" }="" list.add(max);="" for(;i<arr.length;i++){="" if(list.get(counter)<arr[i]){="" list.add(arr[i]);="" }="" else="" {="" max="list.get(counter);" list.add(max);="" }="" counter++;="" }="" for(int="" j="0;j<list.size();j++){" system.out.print(list.get(j));="" system.out.println("\t");="" }="" }="">

Reply • Share ›

Varun Aggarwal • 9 months ago

// Remove the elements which are out of this window
while((!Qi.isEmpty()) && Qi.peek() <= i-k)
Qi.removeFirst();

dont need while over here. if will do.

Reply • Share ›

Anshuman Kaushik • 9 months ago

Using HashMap in Java
https://ide.geeksforgeeks.o...

Reply • Share ›

shiwang • a year ago

Alternative approach using max heap..Although it would still be O(n*log k) but better than bst approach

5 Reply • Share ›

Debraj Ray ➔ shiwang • 6 months ago

Would love to see the solution

Reply • Share ›

Deepak Tiwari • a year ago

I think the explanation is confusing here, as it says that 'current window has an element A if it's greater than all elements from the left side of the window'.
However the code says that 'if An element A is greater than any the elements on it's left side of the window then remove them, otherwise if the element less than the element from it's left side then simply insert it'
By this way the greatest element would always be at the rear end, as it is greater then all the elements from the right side or it won't present in the window.

https://ide.geeksforgeeks.o...

2 ^ | v • Reply • Share ›

GOKU • a year ago

if we push the elements of array in reverse order in stack1 and pop top k no of elements and push them into another stack while maintaining maximum of those k and printing max.Then we can pop back top k-1 elements from stack2 to stack1 and repeat the process until stack1 has greater than or equal to k elements.I think it would work...

39 ^ | v • Reply • Share ›

chirag ➚ GOKU • 10 months ago

Order of that approach would be O(n*k)

^ | v • Reply • Share ›

Володимир Козубаль • a year ago

'An element is useful if it is in current window and is greater than all other elements on left side of it in current window'
is this correct ?
Can't the Qi store the window like [78, 2 , 0] ? and 0 is not greater than 2 and 78 which are to the left side ?

1 ^ | v • Reply • Share ›

Avinash Singhal ➚ Володимир Козубаль • 5 months ago

nopes, this is not correct

^ | v • Reply • Share ›

Vivek Roshan Suggala • a year ago

My Version of simple code with a Priority Queue(or a List with Comparator, sorted in descending order)

```
public static void SlidingMaximum(int[] arr,int m)
{
int j=0;
int max = Integer.MIN_VALUE;
PriorityQueue<integer> queue = new PriorityQueue<integer>(Collections.reverseOrder());
for (int i = 0; i < arr.length; i++) {
j++;
queue.add(arr[i]);
if(arr[i] > max)
{
max = arr[i];
}
if(j%m == 0)
{
System.out.println(max);
j--;
```

see more

1 ^ | v • Reply • Share ›

brajbhushan patidar • a year ago

can we use Queue in place of deque

^ | v • Reply • Share ›

Nikkitricky • a year ago

http://ide.geeksforgeeks.or...

Reply • Share ›

Nit • a year ago

Can we solve this using Kandace algorithm?

Reply • Share ›

hulk hogan • a year ago

https://ideone.com/SLa4Jq

Reply • Share ›

gagan nagpal • a year ago

1st while in 3rd method can be replaced by an if statement , as it is executed at most 1 time. There cannot be more than one elements which get out of the window at any iteration.

3 Reply • Share ›

Hemant Yadav • a year ago

```
void printKMax(int arr[], int n, int k)
{
int max = arr[0];

// find max of initial k elements.
for (int i = 1; i < k; i++) {
if (arr[i] > max)
max = arr[i];
}
printf("%d ", max);

for (int i = k; i < n; i++) {
```

A computer science portal for geeks

710-B, Advant Navis Business Park,
Sector-142, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org