Maximum in Window

Ask Doubt

Time-Limit: 1 sec Score: 100

Difficulty :

Memory: 256 MB Accepted Submissions: 100

Relevant For: AZ-201 AZ-202

Description

You have given an array A of size N and a positive integer K ($\leq N$). $A_1, A_2, ..., A_N$ are the elements of the array.
Let $B_i = max(A_i, A_{i+1}, ..., A_{i+K-1})$, for $1 \leq i \leq N - K + 1$.
Find B_i values for all is such that $1 \leq i \leq N - K + 1$.

Input Format

The first line contains T, the number of test cases.
The first line of each test case contains N, the number of integers in an array A and K.
The second line of each test case contains N space-separated integers $A_1, A_2, ..., A_N$.

Output Format

For each test case, print array B as $B_1\ B_2 ...\ B_{N-K+1}$ in a new line.

Constraints

$1 \leq T \leq 100000$
$1 \leq K \leq N \leq 100000$
$-10^9 \leq A_i \leq 10^9$
Sum of N over all test cases $\leq 5 * 10^5$.

Sample Input 1

Copy

```
4
```

9 3

9 3

1 2 3 1 4 5 2 3 6

5 5

1 -4 3 -3 -9

4 1

-3 1 -8 3

5 2

-1 -1 -1 -1 -1

Sample Output 1 Copy

3 3 4 5 5 5 6

3

-3 1 -8 3

-1 -1 -1 -1

Note

Explanation 1:
B_1 = max(1, 2, 3) = 3
B_2 = max(2, 3, 1) = 3
B_3 = max(3, 1, 4) = 4
B_4 = max(1, 4, 5) = 5
B_5 = max(4, 5, 2) = 5
B_6 = max(5, 2, 3) = 5
B_7 = max(2, 3, 6) = 6

Explanation 2:
B_1 = max(1, -4, 3, -3, -9) = 3

Hint 1

Solving with brute force in $O(N * K)$ is easy, but it's not enough to pass within a time limit

Solving with brute force in $O(N*K)$ is easy, but it's not enough to pass within a time limit.
Observe that B_i and B_{i+1} differs by only two elements. A_i is present in B_i while A_i is absent in B_{i+1}.
Similarly for A_{i+K} is present in B_{i+1} and absent in B_i.

Solution Approach

Maintain one multiset with a sliding window. *That's it! Can you do it in O(N)*? How about using a Double-ended queue?

Complete Approach

Solution 1:
$B_i = max\ (A_i, ..., A_{i+K-1})$
$B_{i+1} = max\ (A_{i+1}, ..., A_{i+K})$
If we maintain a multiset M initialize with $\{A_i, ..., A_K\}$.
For B_1, the answer is the max number present in the current multiset M.
For B_2, as mentioned in hints add remove A_1 and add A_{K+1} in the multiset M, and now find the max number in the current multiset M.
Keep on doing this for all B_is.
Time complexity: *O(N*logK)*
Space complexity: *O(K) extra space*

Solution 2:
Create a Deque, Q of capacity K, that stores only useful elements of the current window of k elements.
An element is useful if it is in the current window and is greater than all other elements on the left side of it in the current window.
Process all array elements one by one and maintain Q to contain useful elements of the current window and these useful elements are maintained in sorted order.
The element at front of the Q is the largest and element at the rear of Q is the smallest of the current window.
Algorithm:

1. Create a deque to store K elements.
2. Run a loop and insert the first K elements in the deque. While inserting the element if the element at the back of the queue is smaller than the current element removes all those elements and then insert the element.
3. Now, run a loop from K to end of the array.
4. Print the front element of the array
5. Remove the element from the front of the queue if they are out of the current window.
6. Insert the next element in the deque. While inserting the element if the element at the back of the queue is smaller than the current element removes all those elements and then insert the element.
7. Print the maximum element of the last window.

Time complexity: *O(N)*
Space complexity: *O(K) extra space.*

Editorial Code

Copy

Solution 1:

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

#define ll long long int
#define LD long double

const int N = 100010;

int inf = 1e9;
int mod = 1e9 + 7;

signed main() {
    //freopen("IN", "r", stdin);
    //freopen("OUT", "w", stdout);

    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    int T;
    cin >> T;
    while (T--) {
        int n, k;
        cin >> n >> k;
        int A[n];

        for (int i = 0; i < n; i++)
            cin >> A[i];

        multiset<int> M;

        for (int i = 0; i < k; i++)
            M.insert(A[i]);

        for (int i = 0; i <= n - k; i++) {
            cout << *(M.rbegin()) << " ";
            M.erase(M.find(A[i]));
            if(i + k < n) M.insert(A[i + k]);
        }
        cout << "\n";
    }
    return 0;
}
```

Solution 2:

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

#define ll long long int
```

```cpp
#define LD long double

const int N = 100010;

int inf = 1e9;
int mod = 1e9 + 7;

void add(deque<int> &d, int x) {
    while(!d.empty() && d.back() < x)
        d.pop_back();
    d.push_back(x);
    return;
}

void rem(deque<int> &d, int x) {
    assert(!d.empty());
    if(d.front() == x) d.pop_front();
    return;
}

signed main()
{
    //freopen("IN", "r", stdin);
    //freopen("OUT", "w", stdout);

    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    int T; cin >> T;
    while(T--) {
        int n, k; cin >> n >> k;
        int A[n];

        for(int i = 0; i < n; i++)
            cin >> A[i];

        deque<int> d;

        for(int i = 0; i < k; i++)
            add(d, A[i]);

        for(int i = 0; i <= n - k; i++) {
            cout << d.front() << " ";
            rem(d, A[i]);
            if(i + k < n) add(d, A[i + k]);
        }

        cout << "\n";
    }
    return 0;
}
```