Sliding Window Maximum problem

Competitive Programming and Contests

Jan Krejčí

October 19, 2022

1 Implementation

In this section the implementation of three different solutions is briefly described. Valid input values *nums* and k are required.

1.1 Max Heap

In this implementation method values of current sliding window are stored in the Max Heap data structure (`std::collections::BinaryHeap`). Firstly it creates Max Heap from first k elements. In the following for cycle, it loops through all input values and adds current one to the heap. In every iteration it also checks if max value of the heap is still in the window. These two steps simulate the window movement. In the end of the loop, maximum of the window is located in the heap head so it is stored into the result vector.

All of the used heap operations are performed in the logarithmic time. Due to this, the time complexity is $\theta(n \log n)$. [CLRS22]

1.2 Binary Search Tree

Method of solution using Binary Search Tree (BST) is very similar to one described in previous subsection 1.1. Since both methods are binary trees, whole process is basically same. Only the data structure changed to `binary_search_tree::BinarySearchTree`. Following BST operations have been used:

- `tree.insert()` instead of `heap.push()`,
- `tree.extract_max()` instead of `heap.pop()`,
- `tree.max()` instead of `heap.peek()`.

1.3 Deque

The fastest linear time solution is using `std::collections::VecDeque`. There is always maximum of the current window in the head of Deque. In the loop, algorithm removes values that are no longer in the sliding window and also values which are lower than new value of the window. New value is inserted at the end of the Deque.

All of the used Deque operations (insert, delete, access to front and back) are performed in the constant-time. That means whole solution has time complexity $\theta(n)$, because it is running though all input values of the length n. [CLRS22]

2 Experiments

In this section results of multiple experiments with different implementations from previous section 1 are described.

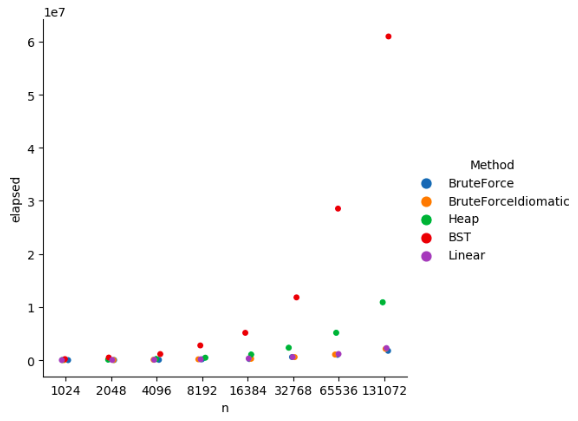
2.1 Modifying number of input numbers

With the small value of input numbers n results of experiments show that all methods have similar calculation time no matter the value k. But as soon as the value n starts to increase, there is a marked difference between methods using binary tree (Heap, BST) and others (Brute Force, Deque).

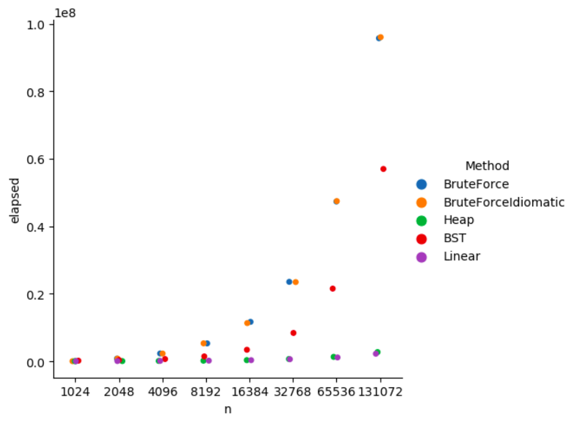
2.2 Modifying length of sliding window

With a small value of the variable k, the inner for cycle is also small in brute force method, so result is approaching to linear solution since k is constant $\theta(n * k)$ as you can see on the Figure 1 (a). Heap and BST implementations are slower because of logarithmic time operations described in subsection 1.1 and 1.2. Since sliding window is small these operations are used very often.

With a large value k, the brute force method has the real quadratic time complexity $\theta(n * k)$ as you can see on the Figure 1 (b). Since sliding window is large Heap method doesn't use its logarithmic time operations so often so the result is almost linear.



(a) Result of experiment with small sliding window, $k = 8$.



(b) Result of experiment with large sliding window, $k = 1024$.

Figure 1: Results of experiments of all implemented solutions with smallest and largest value k.

2.3 Linear

Several experiments with different parameters of the input values showed that the linear method is always among the fastest, especially with increasing input values k or n. As you can also see on the Figure 1.

2.4 BST vs Heap

The Heap differs from a Binary Search Tree. BST is an ordered data structure, the Heap is not. BST does not allow duplicate values, Heap does. Because of it, average time of insertion into a BST is $O(log(n))$, for Heap is $O(1)$. This is why BST is slower, which also corresponds to the experimental results from the Figure 1. [CLRS22]

2.5 Brute Force

Brute force implementation and 1-line idiomatic brute force appear to be almost identical after several experiments. As you can see on the Figure 2 where is result of one of them. It is because algorithm itself is the same, only the way the code is written is different.

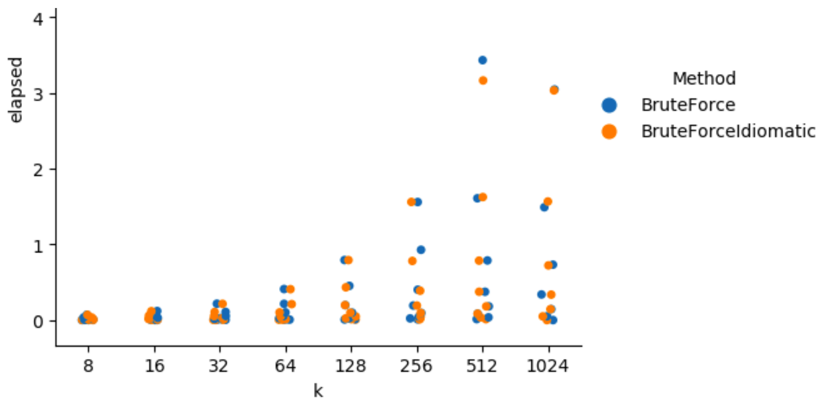


Figure 2: One of the experiment comparing brute force and its 1-line version within different values k and n.

2.6 Running time

Running time of the package run by command `cargo run` on my device is over 5 minutes. Optimized release mode run by command `cargo run --release` takes longer to compile, but the running time is much faster (13 seconds only). Compiling and running with flag `target-cpu` does not speed up program run-time.

References

[CLRS22] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.