Homework 1: H-index LeetCode Challenge

Hayden C. Daly

September 11, 2020

1 Approach

Started with the naive approach of using a nested `for` loop checking if there exists `N` values equal to or larger than each citation `C`. This produced $O(n^2)$ time-complexity which could immediately improved. To reduce the repeated work in iterations, the `citations vector<int>` was sorted using the built in `std::sort` method which has $O(nlogn)$ time complexity.

```cpp
class Solution {
public:
    int hIndex(vector<int>& citations) {
        int max = 0;
        int len = citations.size();
        sort(citations.begin(), citations.end());
        for (int i = len - 1; i >= 0; i--) {
            if (len - i <= citations[i] && len - i > max) {
                max = len - i;
            }
        }
        return max;
    }
};
```

Listing 1: Submitted Algorithm

Having the `citations vector<int>` sorted allowed for iteration checking that the amount of greater than or equal to citations falls within a calculated range with $O(n)$ time-complexity. This range has a lower-bound of the pre-existing value for the max and an upper-bound of the citations of the index `i`. If within the range, the max is set to size of the validated set of citations.

There are improvements to be made for the complexities, such as negating the max variable. However, this approach came intuitively, performed wellm and has good readability so satisfied me. I would like to see if I could implement a binary based approach which would work off a midpoint in the vector but was not certain on the precise implementation.

2 Proof of Completion

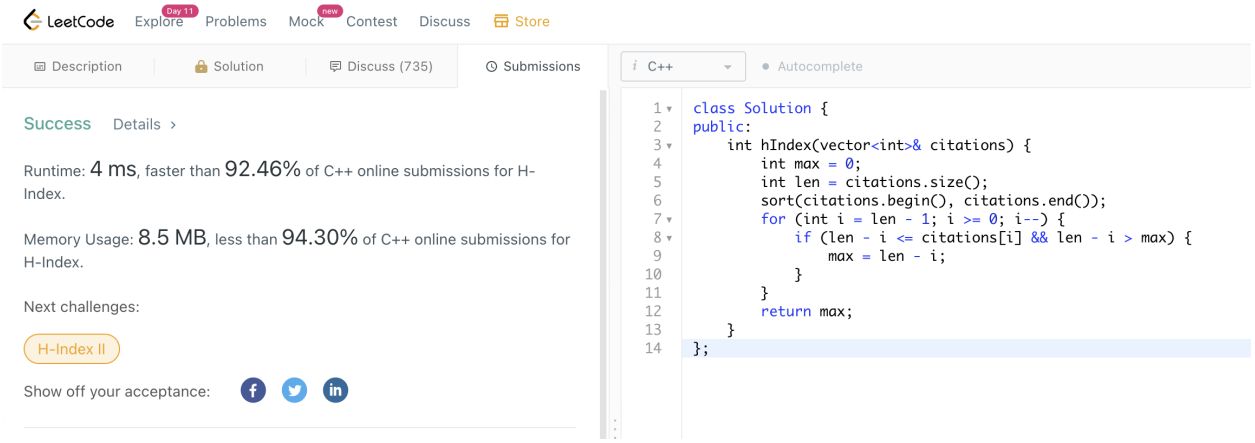

Figure 1: LeetCode Submission Proof