COSC 320 – 001
Analysis of Algorithms
2020 Winter Term 2

Project Milestone 1
Wall Street Bets Data Scraper

Group Lead: Tahmeed Hossain
Group Members: Barret Jackson, Omar Mourad, Kenji Niyokindi

Abstract

For this milestone, we wanted to focus on the most appropriate algorithm for sorting data input from the Reddit scraper as it comes. With a max heap, we are able to quickly sort through stock symbols' popularity values and place them into an array and a database (takes in stock symbol as key and assigns it a popularity value). Instead of sorting all stock inputs, we need to only focus on the top 3, hence finding the top 3 will be extremely fast using a heap. This milestone includes a rough pseudo code that displays how our algorithm will function as well a detailed analysis of our sorting algorithm.

Problem Formulation

We will take an input of stock tickers, filtered by taking the reddit data and sorting, to find values that resemble stock ticker substrings. The input will be sorted in order of a weight value calculated by the number of comments and votes. We will then store these values in a max heap, a binary tree in which the value in each internal node is greater than the values of that children node. This allows us to access the root with the time complexity of the getmax operation at O(1). Insertion into a heap will have a worst-case time complexity of O(log n).

Pseudo-code

1. Data dump of all posts from r/wallstreetbets on day x
2. for (each post):
    a. search for stock symbol pattern
    b. if stock symbol pattern in dictionary: (heap data structure, scores initialized as 0)
        i. increment counter for
        { “index”: 1,
        “stock”: {
                “symbol”: XYZ,

            }
        }
    else
        add symbol to heap

stock xyz mentioned -> XYZ daily score += 3 + num_comments + num_upvotes
Look for symbol in dictionary (dictionary, key is symbol, value is daily score)
Update value of symbol in heap
Look for top 3 scores -> for scores in top 3: calc %change over following week (take top node of heap and its two child nodes)
XYZ on day 0: $5
for days 0-6: max_price(xyz), min_price(xyz)
%gain = (max-day_0)/day_o
%low = (min-day_0)/day_o

max(abs(%gain),abs(%loss)) -> effect

Pseudo code for max heap operations

```
maxHeapify (A,i) //helper function to convert array to heap
        l = left(i)
        r = right(i)
        biggest = i
        if l <= A.heap-size and A[l] > A[i]
                biggest = l
        if r <= A.heap-size and A[r] > A[biggest]
                biggest = r
        if largest != i
                swap(A[i], A[biggest])
                maxHeapify(A, biggest)
buildMaxHeap(A)
        A.heap-size = A.length
        for i = floor(A.length/2) downto 0
                maxHeapify(A,i)
//modified from CLRS, pp 154-157
```

Algorithm Analysis

The main algorithm we will be using will be to build a heap data structure in order to find the top three scores for a given day. First, we will prove the correctness of the build heap function:
Since we are concerned with the top three scores for a given day, we will be using a max heap, stored as an array (actually we will be using Python dictionaries, with the stock symbols as keys and the daily scores as the values, but the logic remains the same). To build a max heap, we use a recursive algorithm, with an important property known as the max-heap property (CLRS pg. 152), given by:We

For all nodes i in the array A that are not root nodes, A[parent(i)] >= A[i]

In other words, the value associated with the parent is always greater than or equal to the value of its child nodes. We must now define two properties of A: A.length represents the length of the array, while A.heap-size, where 0 <= A.heap-size <= A.length, represents the values in the array that are part of the heap. The buildMaxHeap algorithm starts at A[floor(A.length/2)] and iterates until i<0. Our loop invariant is therefore:

At the beginning of the for loop on each iteration, all nodes from i..n represent the roots of a heap.

Initial Condition

Before the algorithm has run, every node in the array is the root of a heap of size 1. Therefore, all nodes from floor(A.length/2)..n are roots of a heap, so the loop invariant holds.

Maintenance

Since the maxHeapify function places the largest element in {i, left(i), right(i)} in the position of the parent node, and the for loop decrements which element is being iterated on, this means that all nodes from i..n are roots of heaps, which preserves our loop invariant.

Termination

At termination, i<0, all nodes including the final node where i=0 are roots of a max heap, so our loop invariant is maintained. This completes the proof of correctness.

Time Complexity

There are two main operations of max heap; maxHeapify and buildMaxHeap. The maxHeapify operation has a time complexity of O(log n), whereas the buildMaxHeap operation has a running time of O(n) for an array with n elements. Generally, the worst-case running time for the max heap is then calculated as O(nlogn), however, a tighter bound can be found whenever the heap of size n has a height h = floor(log n) and at most, nodes = ceiling(n/2^(h+1)) of any height h (CLRS pg. 157).

In our case, the tighter bound properties hold as they generally hold for creating heaps out of unordered arrays. Stock's popularity values are stored initially in an unordered array, in which max heap operations are used to sort the array from highest values to lowest.

Furthermore, with the tighter bound properties, the maxHeapify operation is O(h) when called on a node of height h. Given that nodes = ceiling(n/2^(h+1)), the total running time for node n at height h is:

$$\sum_{h=0}^{floor(logn)} (ceiling(\frac{n}{2^{h+1}})) * O(h) = O(n * \sum_{h=0}^{floor(logn)} \frac{h}{2^h})$$

The summation portion of the right handside of the above equation can be estimated as a constant value when taking the summation from h=0 to h=∞ as it's inner value converges to 0 as h gets larger. Ultimately the summation adds up to 2. Therefore, the right handside can be further simplified to **O(n),** which is the time complexity for an unordered array (CLRS pg. 157-159).

Unexpected Cases/Difficulties

There are multiple different unexpected cases and difficulties that we encountered when considering our algorithm analysis. The most prominent being the level of specification in regards to stock mentions in a post. Whether a post mentions the same stock twice, multiple stock tickers are included in the same post, or negative mentions (posts that recommend selling stocks), the level of specification of when to increment the popularity value of the stock within our database needs to be refined. With this in mind, we intend to focus on incrementing only once per stock symbol per post, regardless of the context the ticker is being used in.

Another hurdle we had to consider were dot tickers which resemble venture stocks. Venture stocks are stocks that are not available on the global stock market yet, however they are still available for purchase. These stocks may be difficult to monitor as the dictionary dataset that we will use to compare (from Yahoo Finance) may not recognize them as stock symbols yet. In order to alleviate this issue, we will only focus our algorithm to recognize stocks that are available on the global stock market and are mentioned in our dictionary of stocks (Yahoo Finance resource).

Task Separation and Responsibilities

Tahmeed: Helped formulate problem into algorithmic problem. Broke down the data input and the data processing of the first algorithm. Write project formulation and address run-time complexities. Design implementation strategy for creating the initial reddit dataset.

Barret: Helped formulate problem into algorithmic problem. Elucidated algorithm's proof of correctness, the implementation of the heap data structure and how every insertion of an element will run through the binary tree. Wrote pseudo code for max heap operations.

Omar: Unexpected cases/difficulties, report abstract, and report editing, algorithm time complexity.

Kenji: Helped formulate the problem into algorithmic problem. Will be impl