Nick Jonas, Rahul Rangarajan, Cameron King

5/4/2020

CSC 301

Assignment 3 Report

This assignment definitely proved to be a slightly harder challenge than the previous one, as there were not only more problems, but also more problems of different types. In order to deal with this, we each took charge of an individual problem, with one person working on two. No one was working on a problem fully alone, as we tried to meet once each night and talk about problems we were having and suggest solutions for each other. We also looked over each other’s work to improve efficiency and catch bugs that the person writing it didn’t see.

For question one, we decided to simulate a heap with max heap capabilities using an array, like shown in class. We did this because the space complexity would only involve one array and the occasional resize function. We also ensured that the resize function would not only increase the array size, but also decrease it as well. Towards the end of our project we considered working with generics, as it would allow for multiple data types to be within the max heap. However due to our unfamiliarity with the concept we were unable to come up with definitive pseudocode`. If we understood hashing better and had managed our time better, we could’ve probably have finished at least the pseudocode and have begun programming by the time the assignment ended.

For question two, we decided to base the algorithm off of the InOrder algorithm for tree traversal. One of the reasons why we chose this algorithm was because we had already designed a variation of the same question in class, which instead found the minimum value of the tree. We also decided to stick with InOrder because there is no major difference in time complexity between it and the other two tree traversing algorithms we have been shown (PreOrder and PostOrder). We also kept the assumption that we were working with nodes instead of an array because it was easier to visualize, and possibly could be referenced to again one day if any one of us had to work a binary tree.

For question 3 we decided to base our algorithm off of quicksort to separate the even and odd numbers. We imagined that our array was divided into 3 sections. The even numbers will be at the beginning, the odd numbers at the end, and the unknown numbers in the middle. Going through the middle section, if the number is even, add to the end of the even section and if the number is odd, it will be added to the beginning of the odd section.

While trying to understand how to code permutation for question 4, the biggest hurdle was figuring out the pseudo code. We already knew that we needed to convert our string to a character array and then swap locations to create all the permutations, but we struggled to figure out a pattern. While attempting to figure out the pattern, I spent hours on trying to figure out patterns and often discussed these ideas with my team to discuss potential flaws in this pseudo code It took me three pages of testing in my notebook for us to realize that the key was to do the exact same swapping mechanic but to always make sure that the next character in the original array would become the first character.