Brian Burton
u1038667
04/14/17
CS 2420
Assignment 11

Heap Analysis Document

1. Title Page

For assignment eleven, the heap data structure, I was blessed with the opportunity to work with Lisa Richardson. We were able to learn quite a bit about the heap data structure. Many of the things that we learned dealt with the important properties of a heap. Some of those properties include the following.

Each parent node has two children. In a min heap, the parent node is smaller than its children. In a max heap, the parent node is larger than its children. When you add an element to a heap, you must first add it as the parents left child, and then the right. After you add the element to the end of the heap you much bubble the element up. If the heap is a min heap, then you must compare the new element with the parent to see if it is smaller than it. If it is smaller than these two elements swap. You keep doing this until the element is no longer larger than the parent node. On the other hand, if you are implementing a max heap, then you must compare the parent node to see if it is larger than the element that was added.

To delete an element in a min heap, you must first swap the last element added with the root node. Then you must compare the new root node to its children swapping them with the smallest child only if the parent is larger. This is called bubbling down in our code. In a max heap, you also must swap the last element added with the root node. Then you must bubble it down.

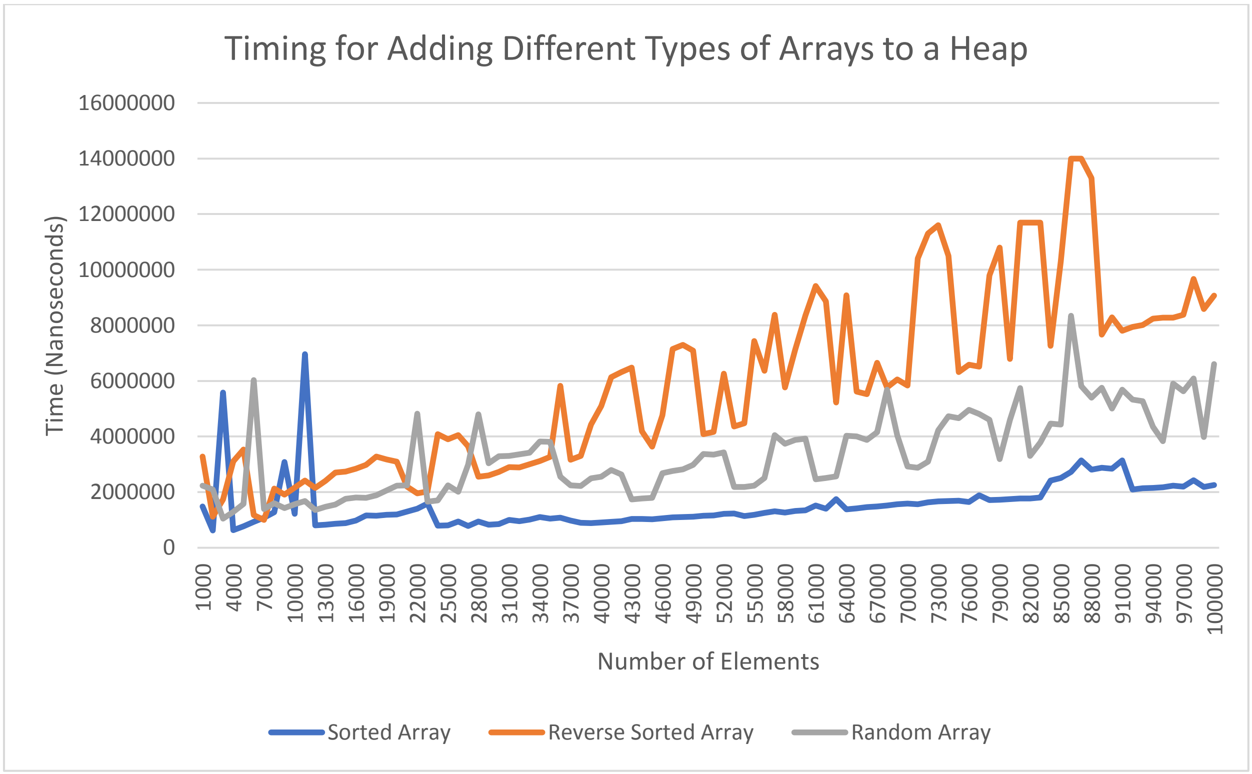
Finally, in order to sort a heap, you must first swap the last element with the root node. Then you make it so that the computer can no longer work with the last element in the heap. Then you bubble down the new root node.

In the real world, heaps can be used both abstractly and concretely. They can be used abstractly when you are making payments on a house and concretely when you are trying to store a lot of data.

2. Timing Experiment

For our analysis, we decided to focus on the worst, best, and average cases of adding to a heap. We learned in class, that adding to a min heap from a reverse sorted array is big o of NlogN. On the other hand, adding a sorted array is constant and adding a random array is close to constant. This is because half of the data will be at the bottom of the heap assuming that the last level of the heap is completed. However, what we did not discuss was how long it takes to add to a min heap. Therefore, we decided to do an analysis on this.

The way that we conducted this experiment was we created a sorted, reverse sorted, and a random array. We then walked through each of the arrays adding the current element to our min heap. As we did this, we timed it. Below is a graph of our findings.



The graph shows us that reverse sorted arrays defiantly do take longer to build because it is having to bubble up to the root node every time. The graph also shows that sorted arrays are extremely fast because we never have to bubble up. The random array is somewhere in the middle because on average each element is only having to swap twice. Therefore, the graph proves this theory.

3. Software Engineering

   It took us approximately ten to twelve hours to complete this assignment. We ran into every issue that you possible could have probably. Each method caused its own unique bugs. The best part was that each method affected each other. This meant that once we believed that we had fixed a method, we would move onto another method only to find out while debugging the new method that the old method still had a bug or two. In particular, the bubble up method and how to use the while loop in that method.

   The purpose behind the compare method was to allow our heap to be generic. Without the compare method, we would not be able to compare the current node to its parent, or children generically. The magic that it is doing is allowing us to compare any kind of data structure.

   The reason that the heap returns an array of Objects instead of an array of types is because types are objects. There really is no such thing as an array of types at run time. Only a compile time. This is why we must use Suppress Warnings.

   The reason that we generate the "DOT code" in the toString method instead of the generateDotFile method is because it is convenient. Whenever we create a heap, we can visualize it by copying the toString method output into a web browser. We also do it to abstract all of the details.

   The reason that the "test_lots_of_insertions_deletions_peeks" is a powerful testing tool is because it uses a random generator. This means that it adds random objects to the heap. The allows us to test a lot of objects without having to code all of those different possibilities.