Nickolas Komarnitsky u0717854

04/13/2017

CS2420 Assignment-11-Heaps

A heap is a balanced tree that has very quick operations. At best case it is O(N), worst is only log(n). Building a heap will always run at log(N) since it has to go through many more operations the bigger the input array. A priority queue can be used abstractly at an amusement park, club, etc. They have people without the special pass, who have the same priority, then those with the special pass who have a higher priority. In a computer system they are concretely used because they are based purely on the number that is in the system. They rely only on the numbers, the “bouncer” can’t just let someone in ahead of another. In a computer system, they enter at the back and then based on that priority they move ahead of the others and end up exactly where they are. There is no line skipping.

1. Timing Experiment

For my Timing Experiment I am trying to determine the runtime complexity of a min heap. The Mathematical analysis tells me that the heap should have an average complexity of O(N), with a worst-case complexity of log(N). The build from array function I expect to be log(N), taking much longer for a bigger given array.

My testing was setup to determine as many cases as I could. My first test was on the insertion. My first test was to add items into an array in order from 1-1000000 incremented by 100, timing the add function each time. The second test was again to a size of 1000000, but adding in the same number each time. The third was to a size of 1000000 as well, but adding in random integers from 1 to 1000000.

Then I tested the deque function. For the first test, I created an heap of size 1000000, with the items inserted in order, then timed the deque function until there was only 1 element left in the heap. The second test was with the same, but the same item was inserted into the heap beforehand. The final test I tested deque on an heap that had items inserted with random values. For this test, I generated an heap of the desired size with random integers being put in. For each size, I ran this 5 times, to make up for the randomness.

Then I got the heap sort and build from array done. I ran a test on build from array, with the given array being in order first. The second test was with the array having the same number in all spots. The final was an array with the numbers randomized. For all of these I timed the method.

The graphs below show time to size, with the size on the x-axis and time on the y-axis. For the insertion tests, almost all of them came out with a constant time, with a few outliers in the data. They showed a run time of O(c) for all insertions. The deque was the same, taking O(c) for each test. The Build from array function was log(N) for each test, with it taking longer the larger the given array.

1. Software Engineering

This assignment took me about 10 hours, although a lot of that was just writing the timing tests. Non-heap wise, I figured out multithreading in javaFX much better and even got a decent GUI made that can show exactly how far along my tests are. This GUI can also have as many tests as needed added into it. The compare method is to determine what child is smaller. This allows for you to move an item up the heap as needed. The heap returns an array of Objects instead of Type because it is safer, as object has more functionality than type does. We use SupressWarnings because the editor will highlight warnings on type. We generate it in the toString method because it creates the heap, not the array so having it be made in a dot readable style makes the most sense. The test\_lots\_of\_insertions\_deletions\_peeks is a powerfull tool because it allows you to make sure that your code will accept a lot of data inserted and removed and that this is occurring properly.