Project #2

Max-Heap

CS 241 Section 1

Matt Dohlen

[matt.dohlen@gmail.com](mailto:matt.dohlen@gmail.com)

05/09/2014

Section 1. Project description

The program gives the user the option of creating 20 sets of random data or 1 set of ordered data. After the user selects which option they would like to use the program inserts the data into a heap using 2 methods; sequential insertion and optimum insertion. For the random sets the program accumulates the number of swaps and reports the average amount of swaps over 20 sets for both sequential and optimum insertion. For the ordered set, the program inserts using both methods and reports the amount of swaps for each while also displaying the first ten elements in the heap, deleting ten elements, and then displaying again.

Section 2. Project specification

In this project, the heap class takes care of all number generation, heap initialization, and heap operations. The heap is implemented using an integer array plus an integer to keep track of heap size and an integer to accumulate the amount of swaps mad by the heaps operations. The main procedures used were the Insert() function, SwapUp() function which swapped a child up if necessary, SwapDown() which did the opposite, and OptSwap() function which was used to initialize the heap optimally (which utitlized the SwapDown() function). Both swap functions would increment the numSwaps member function of the heap class allowing me to find the average swaps of the 20 random and ordered sets.

Section 3. Testing methodology

This program was an easy one to build in phases. I started by creating the bare essential functions of the heap class like the various swap functions and insert and delete functions. I then built the functions that would create the data to insert in to the heap. I was able to test each of these functions as I made them to ensure correctness. Although I am not sure the amount of swaps is absolutely correct I am confident that my code is correct and was able to accurately count the amount of swaps. All in all, this data structure was pretty easy to implement and I did not encounter very many errors.

Section 4. Lessons learned

I’d say the biggest lesson learned from this project was in the analysis of the efficiency of the data structure. The ordered sequential insertion of the data is a worst case scenario which would require the largest amount of swaps which is estimated at O(nlogn) which for this case is about 664. Analyzing this before I coded my algorithm helped to alleviate any mystery of what the output of my program would be for that set of data. I was also able to analyze the average and worst cases for the other sets of data. This project also provided more practice coding data structures in java which is much needed by me.

Section 5. Analysis of output

The expected complexity for the two methods of insertion are: Sequential insertion worst case of O(nlogn), average case for large ranged random set of data is O(n) and optimum insertion has a worst case of O(n). My data seems to support these figures very well. For random sequential insertion the average of 20 sets of insertion ranged from 105 – 120 swaps. This is a very good indicator of the average case for this method which ideally would be 100. For random optimum insertion the data ranged from around 67 – 70. Because the data inserted into the heap is of a large range and random it is obvious that the amount of swaps would be significantly less than the worst case. This is why the data for the ordered integers inserted sequentially comes very close to the worst case of O(n) at a value of 96 exactly. As stated earlier the worst case of sequential insertion is O(nlogn). Although my data didn’t quite reach the 664 mark it was a significantly larger number at 480 which was about 4x my average case. So overall my data did reflect my predictions of complexity.