Abel Gonzalez, ezv100
CS 2123.003

Final Project Write-Up

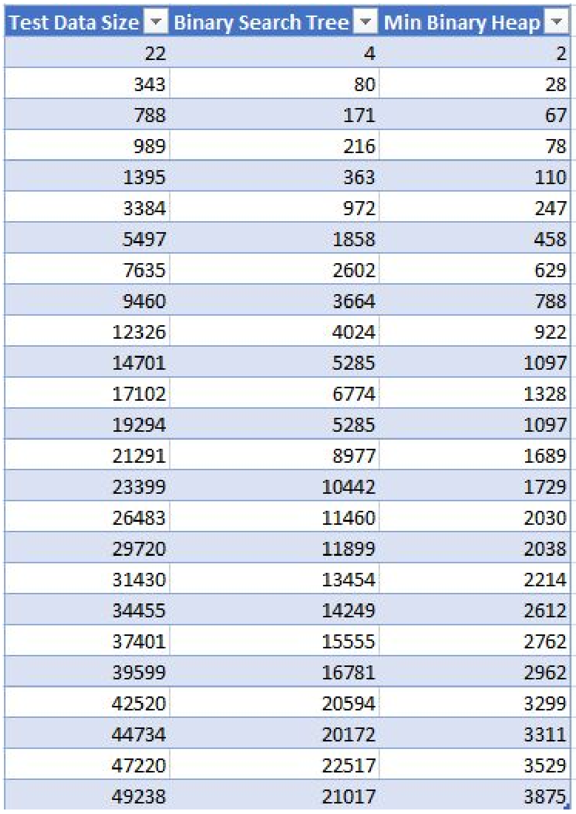
The final project topic I chose was “Binary Search Tree Vs a Min Heap.” With this topic, the objective was comparing the time it took to insert a set of data into a binary search tree and a min binary heap, then finding the minimum of each data set in each structure. In the following analysis, I will break down my program and explain how it accomplishes the topic’s desired goal, the obstacles I encountered, and an analysis of the two structures’ speed when the test data size increases.

For this project, I was tasked with tracking the time it took a binary search tree and a min binary heap to insert data into its structure and find the minimum value. To accurately accomplish this comparison, I needed to use randomized data, track the amount of time the structure took, and display the results as well as an average of the tests. So, to start, I used an array to hold the test data. The array got the job done and worked well, so I stuck to sweet and simple. I then randomized the size of the array between ten and fifty thousand, then by running the array through a for loop, initialized each integer in the array with a random value between one and fifty thousand. Once the randomized test data has been created, the tests of each structure can start. I began by starting a timer and creating the binary search tree with the first value of the array, and then used a for loop to insert each other element of the array into the tree. When finding the minimum value of the data set, I utilized the logic of the structure. A binary search tree decides where to place a node based on the node’s value compared to its own; node values that are smaller than the current go left while node values that are larger go right. With this in mind, I used a recursive function to traverse to the left most node within the binary search tree and return the value stored. After the minimum value is found in the binary search tree, the timer ends, and the function repeats two more times to increase the test amount. The test times are then averaged and the results are displayed. The program starts a second timer and inserts the same data into a min heap. For this structure, the logic is different; once the data has all been entered into the min binary heap, the minimum value is the root, or element zero in the array. So, to find the minimum, I used a function to return the value that was element zero in the array. After the tests are complete, the function repeats two more times and displays the results. All of this logic is then put in a function and called by main in a for loop. Isolating the logic into a function and using a for loop allows more control over the amount of times it is called and can be used to create multiple tests of different data sizes.

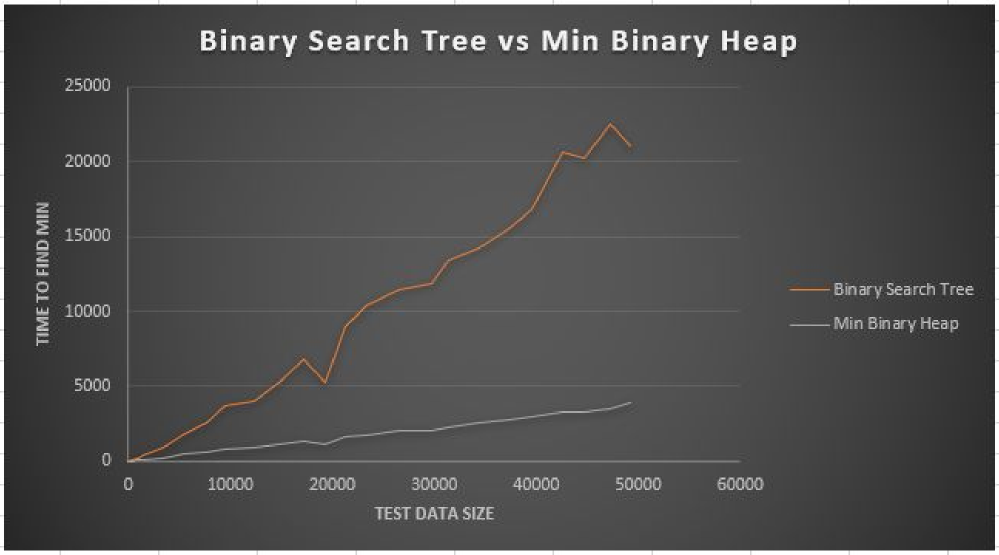
Throughout this project, I was met with multiple obstacles, even when I began. The first and most important part of this project is to correctly implement each structure. I was experienced with the binary search tree and implementing it proved to be a simple task. However, I had no prior experience with coding a binary heap and had limited knowledge of what it required, so it was difficult to build the header and source file for the structure. I solved this problem through research and implementation. My second issue was a memory error with valgrind. To utilize the binary search tree structure, I originally malloced each node before inserting it into the tree. This caused a major leak in the program because there were more malloc calls than free. I instead used a temporary pointer to insert each node into the binary search tree to fix this error. After making this edit, I used a postorder traversal to free each malloced node from the bottom up. This type of traversal is useful to free the tree because it allows the program to free each node's left and right before it, thus each node is cleared.

After fixing my issues and overcoming my obstacles, I was able to correctly build the program and return accurate results. Then, through the use of varying test data sizes, I was able to build the following graph. Although my program shows how long a structure took to find the minimum value in seconds in the graph, I multiplied the result by 10^6 to better visualize the time complexity. The X-axis shows the size of the array used for test data and the Y-axis shows the resulting runtime from the data size. This graph visualizes the runtime complexity of both structures and allows it to be more accurately depicted. With the orange line, the binary search tree, it is clear as the data size increases, the time it takes for the structure to find the minimum value significantly increases causing the slope of the line to grow larger. However, with the grey line, min binary heap, the runtime is barely affected by the data size. Although it does increase as the data size increases, it has a very small slope. Through the analysis of the graph and its data, I believe the binary search tree structure to be closer to O(n log n) and the min binary heap structure to be O(log n).

In conclusion, through the data resulting from the program and the graph, it is clear that the min binary heap is faster than the binary search tree at inserting data into the structure and finding the minimum value. Although both structures find the minimum significantly fast with large data sizes, the min binary heap is faster with larger amounts of data and continues to be the preferable structure as the data size increases.



Test Data Size	Binary Search Tree	Min Binary Heap
22	4	2
343	80	28
788	171	67
989	216	78
1395	363	110
3384	972	247
5497	1858	458
7635	2602	629
9460	3664	788
12326	4024	922
14701	5285	1097
17102	6774	1328
19294	5285	1097
21291	8977	1689
23399	10442	1729
26483	11460	2030
29720	11899	2038
31430	13454	2214
34455	14249	2612
37401	15555	2762
39599	16781	2962
42520	20594	3299
44734	20172	3311
47220	22517	3529
49238	21017	3875



Graph 1.) The above graph displays the time to find min vs test data size.