Cody Wu
133001517

PA3 Report

Introduction

In this programming assignment, a min heap priority queue will be implemented in three different ways. The first two ways are through a sorted and unsorted array. The third way is an array representation of a binary heap. These will then be used to test for different complexities.

Theoretical Analysis:

The unsorted array has an insert time complexity of O(1) and sort time complexity of $O(n^2)$. The unsorted array has such a fast insert as it does not have to insert in a certain spot. It is able to simply insert it to the end of the array. The sort time complexity is longer as it has to insert into a queue then search for the max or min. Since it is unsorted, it has to search each element in the array. These complexities are the average for the unsorted array. The advantage of the unsorted array is that the insert is easy to implement and is extremely fast. However, it takes more time to delete and sort.

The sorted array has an insert time complexity of O(n) and sort time complexity of $O(n^2)$. The sorted array must find the spot in the array to insert. This means that if the element is not being inserted at the end, the array must shift elements to the right. This depends on the number of elements, n. The sort time complexity is the same as the unsorted array as it requires inserting which has a greater time complexity. These complexities are the average for the sorted array. The advantage of the sorted array is that it has a quicker delete and is able to more easily find the max or min. The disadvantage is that it has a larger time complexity for insert.

The binary heap has an insert time complexity of O(log n) and sort time complexity of O(n log n). The binary heap has to insert the element in the last spot of the array which also corresponds with the last spot in the binary tree. To keep the heap order, the new element might have to go through bubble up. It does not have to go through each index of the array as each spot in the array corresponds to a certain spot in the tree. This allows for a faster insert of O(log n). The sort takes a little longer than insert as it must insert into the heap and remove it while using heapify to maintain heap order. This leads to O(n log n). These complexities are the average for the binary

heap. The advantage of the binary heap is that it has faster insert and sort complexities compared to the sorted and unsorted array. The disadvantage is that it is more complex to set up.

Experimental Setup

Once the code passed all the tests, a script was written to test insert and sort. Using a system clock, the time before and after were recorded and subtracted to find the runtime. The input sizes were 10, 100, 1,000, 10,000, and 100,000. These sizes allow for analysis of the different implementations when dealing with a small and large amount of elements, n. By comparing runtime, the effectiveness of each implementation can be compared.

Experimental Results

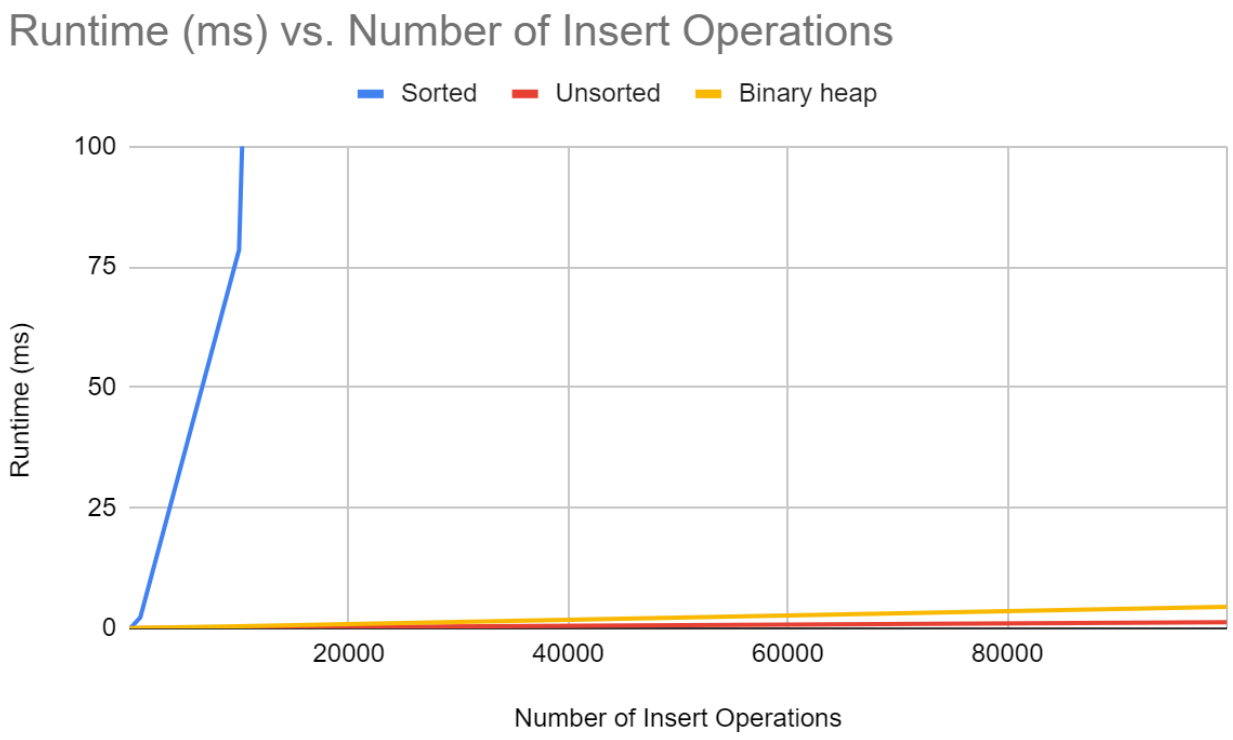
The unsorted array performed the best during the insert test while the binary heap performed the best during the insert and remove (sort) test. The number of inputs does affect the runtime. However, the amount of inputs did not allow one implementation to beat the other.

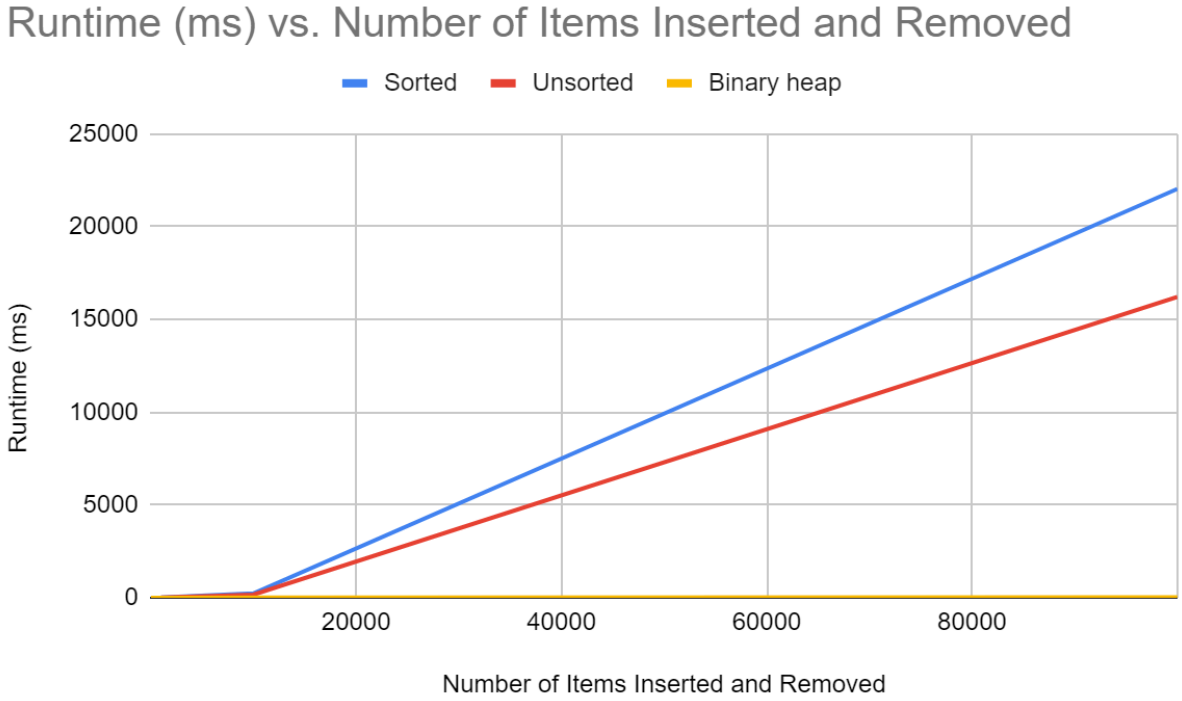
The experimental results for the insert test does follow the theoretical analysis. The unsorted array with the O(1) insert time complexity was the fastest. The binary heap with O(log n) was the second fastest while the sorted array with O(n) was the slowest. The insert and delete (sort) test had different experimental results compared to the theoretical analysis. The biggest discrepancy was between sorted and unsorted. They should have the same time complexity at $O(n^2)$. However, unsorted was significantly faster. During the unsorted delete, the unsorted array's sort is dependent on the delete function. Since the unsorted array does not need to be sorted, the delete function can be done by replacing the deleted index with the last element in the array then reducing the size to get rid of the last element. This allows for a deletion without having to shift every element. The sorted array had to shift every element which caused it to have a higher runtime. The binary heap did follow theoretical analysis of O(n log n) as it was faster than both the sorted and unsorted array ($O(n^2)$).

It can be hard to see the exact values of each implementation as the runtime can jump very high when the number of operations is also increasing by a lot. The values of each test can be seen in the table below.

Insert	Elements	Sorted	Unsorted	Binary heap
	10	0.002	0.001	0.001
	100	0.031	0.004	0.011
	1000	2.28	0.036	0.093
	10000	78.55	0.142	0.434
	100000	7095.29	1.26	4.46

Sort	Elements	Sorted	Unsorted	Binary heap
	10	0.002	0.002	0.003
	100	0.083	0.071	0.028
	1000	6.342	2.888	0.2
	10000	228.6	163.26	2.158
	100000	22037.3	16209.3	26.21




Runtime (ms) vs. Number of Items Inserted and Removed
Sorted
Unsorted
Binary heap
25000
20000
15000
10000
5000
0
Runtime (ms)
20000
40000
60000
80000
Number of Items Inserted and Removed