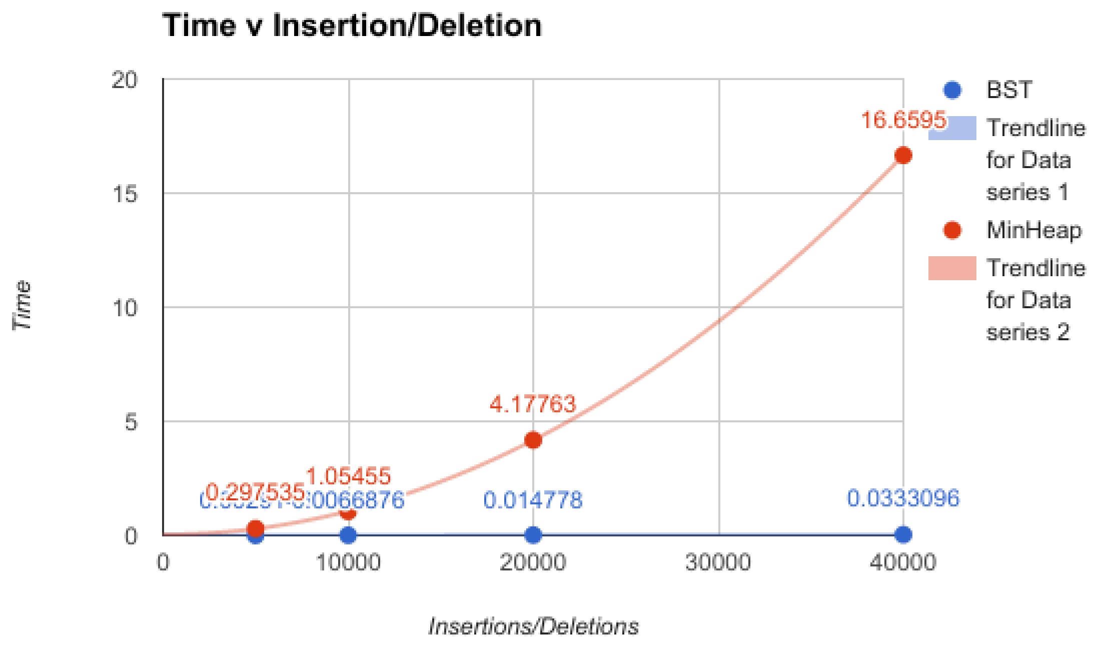
Daniel Murga

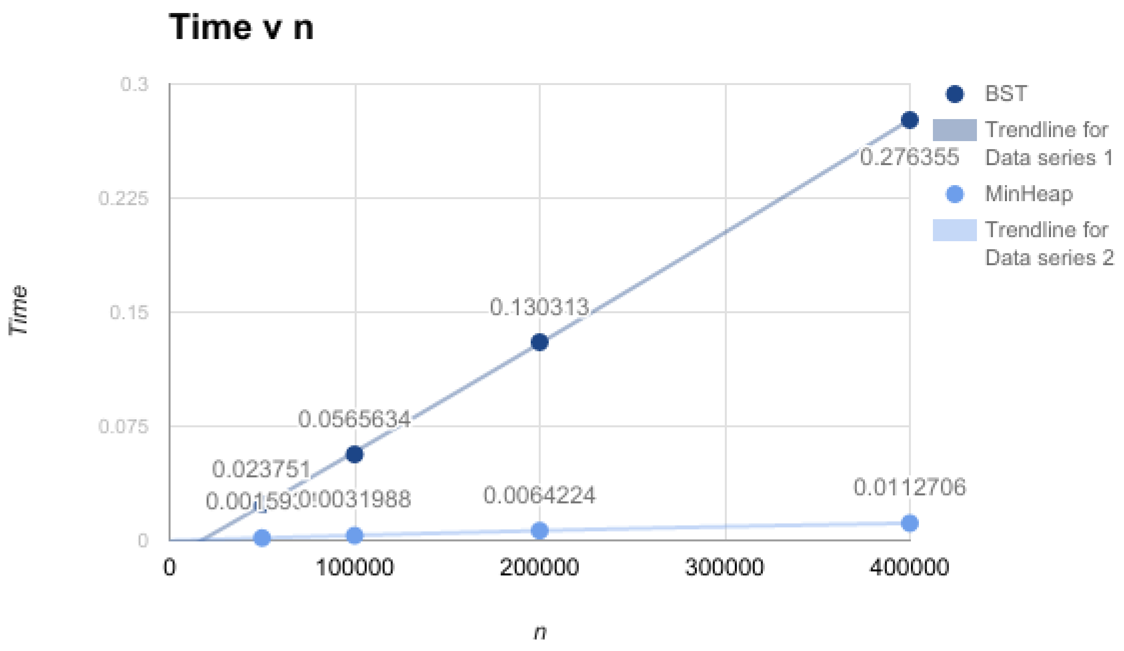
1. When creating the binary search tree and min heap, a for loop is used based on n until 400,000, incrementing n by n = n * 2. Within the loop, a nested for loop from 1 to 5 sets the seed using srand(i) and gets random numbers and places them into an array. It then starts the timer, a for loop calls insert with each value on the binary search tree, populates the tree, and calculates the duration. The program starts timing, builds the heap using the build function, and stops the timer. To check deletion and insertion time, it creates an array containing operations to perform and random values if needed. The time to complete these operations It then outputs the results to the terminal and calculates the average with the stored durations for each table.
2. To generate the data for the test, a for loop from 1 to 5 sets the seed in srand(i) based on the value of i. The next loop iterates from 0 to n and populates an array called random with values using rand(). To time the deletion and insertion operations, it creates an operations array to determine which operation to perform and an array for a random value if needed to insert or delete. A for loop to .10 * n checks each operation and executes the appropriate command. Each iteration doubles the size of n until 400,000.
3. From the results, the average time to build the binary search tree would double as n doubles, starting at around 0.02, 0.05, 0.13 to 0.27 seconds at max n. The average time to build a heap is much lower and less in growth, going from 0.001, 0.003, 0.006, until 0.01 to build it at n = 400,000. When timing the duration of the same operations performed on the heap and tree, the tree performs the remove and deleteMax operations much faster than the min heap. Not only is this due to the way these operations are handled in the heap, but it may also result from an incorrect or unnecessary heapifying operation done in the code. Opposite from the build time, the heap operation time is exponential and grows much faster as the heap grows.
4. When building heaps and binary search trees, using the bottom up method to build the heap is exponentially faster than continually inserting into a binary search tree, and the heap only has to heapify values that break its min rule, while the tree must do at least lgn comparisons. When performing operations on the tree and heap, the tree is much faster at performing operations like remove and deleteMax because these operations do not require as much computing as they do in the min heap because it is not equipped to efficiently execute these operations.

Daniel Murga

BUILD	BST	MinHeap
50000	0.023751	0.0015934
100000	0.0565634	0.0031988
200000	0.130313	0.0064224
400000	0.276355	0.0112706
OPERATIONS	BST	MinHeap
5000	0.0029146	0.297535
10000	0.0066876	1.05455
20000	0.014778	4.17763
40000	0.0333096	16.6595



Daniel Murga


Time v n
Time
n
BST
Trendline for Data series 1
MinHeap
Trendline for Data series 2