Method	**BST**	**Worst Case Time Complexity of BST**	**Hash Table**	**Worst case time complexity of HT**
Search	0.000104s	O(n)	0.000814s	O(n)
Insert	0.000136s	O(n)	0.000654s	O(n)
Delete	0.000109s	O(n)	0.000255s	O(n)
Sort	0.201072s	O(n)	0.374032s	O(n log(n))
Range query (n=10)	0.010710s	O(n)	0.015930s	O(n)
Range query (n=100)	0.085024s	O(n)	0.082073s	O(n)
Range query (n=1000)	0.557707s	O(n)	0.525916s	O(n)

In the worst case scenario, the basic functions of search, insert, and delete are O(n) because due to the BST resembling a Linked List and the Hash Table having many collision--however we have average runtimes that are seemingly better than O(n) because our input doesn’t invoke O(n) runtimes. We hypothesize that our BST has been slightly faster in general than our Hash Table implementation due to the collision property of Hash Tables, which leads to additional runtime from probing. The runtime for Hash Table sort should be O(2n + n logn) since we had to insert n elements to a vector and print the sorted elements out (2n), and had to use std sorting algorithm (n log n) to sort the vector. However, this can also be generalized as O(n log n) worst case bound as well. BST sorting, ideally, should take a little bit longer than O(n) since it takes time to traverse between stacked nodes.