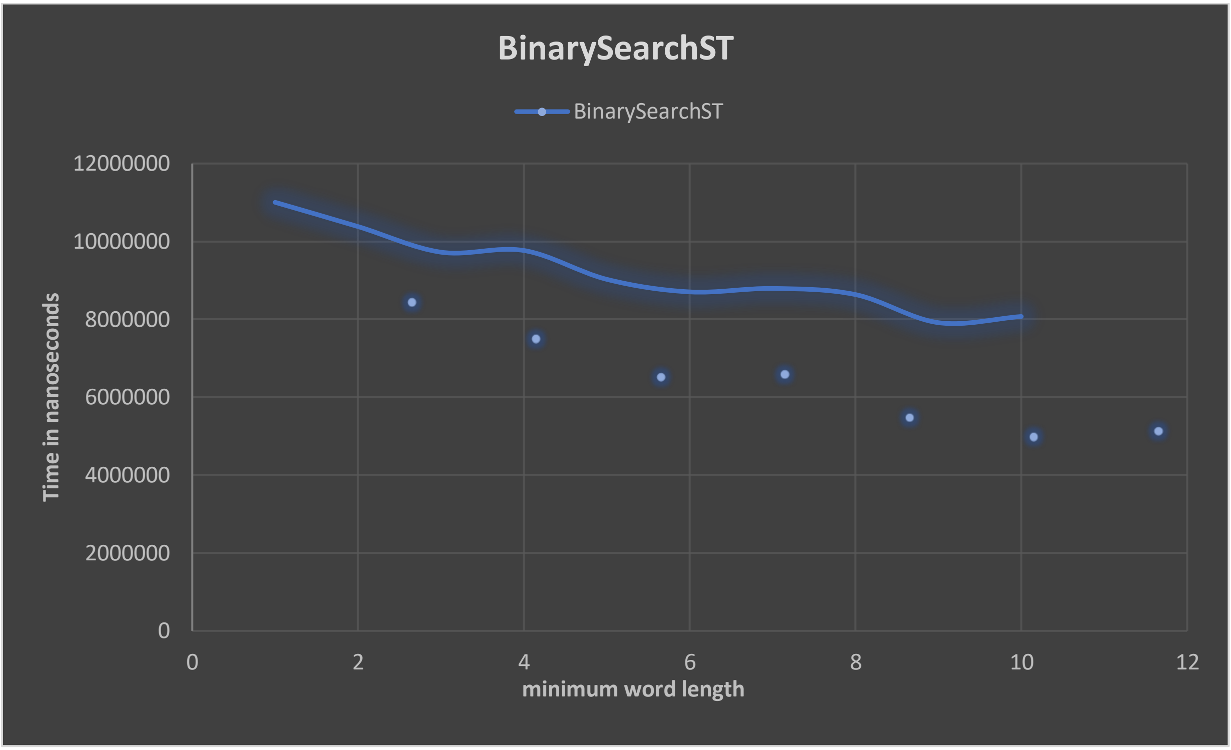
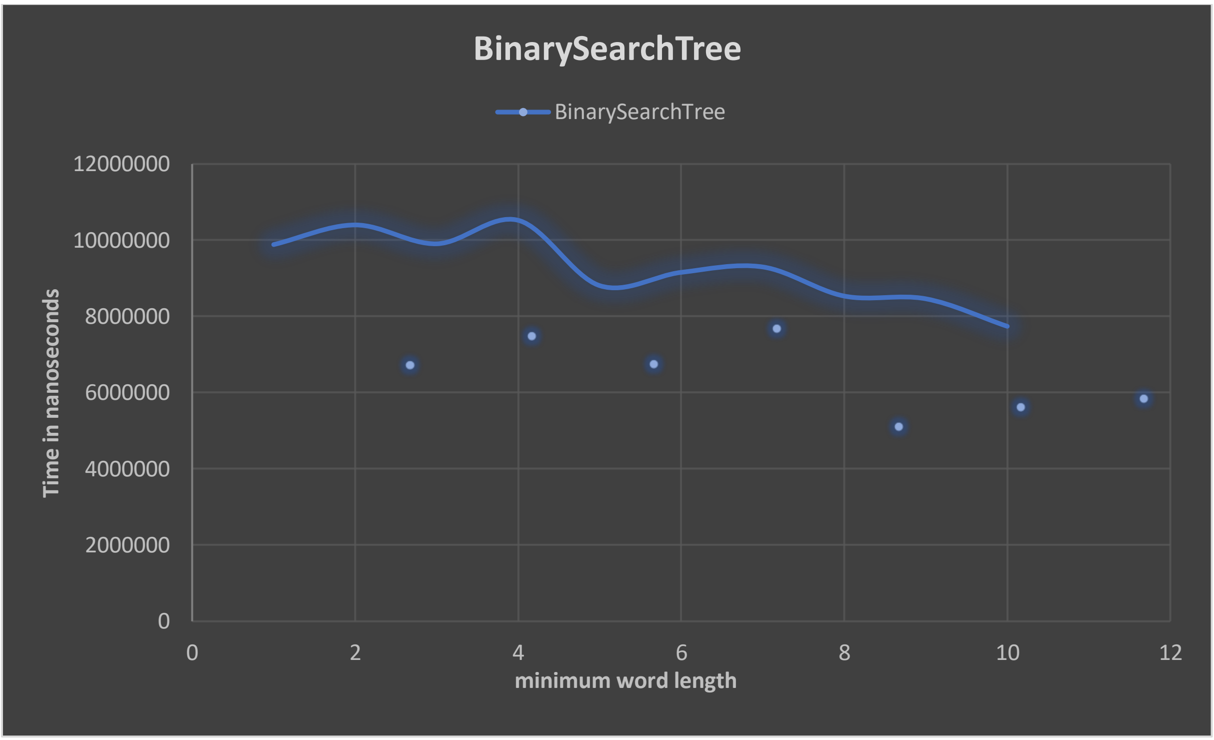
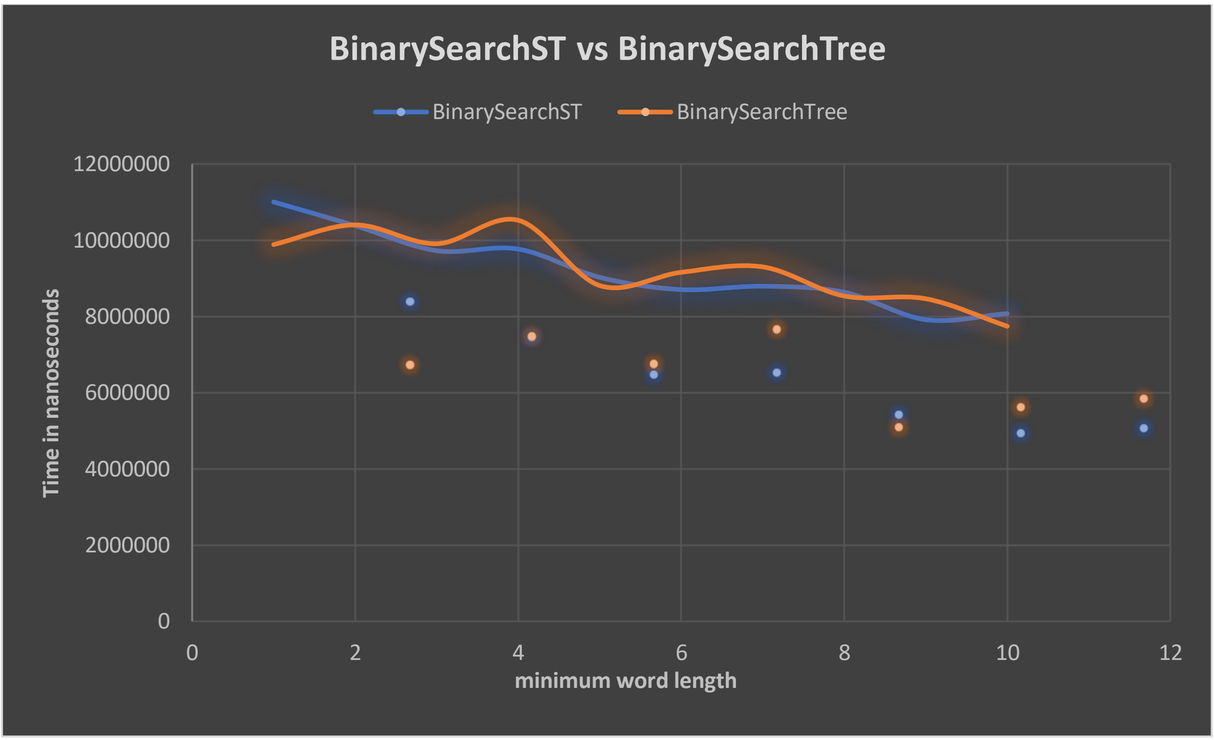
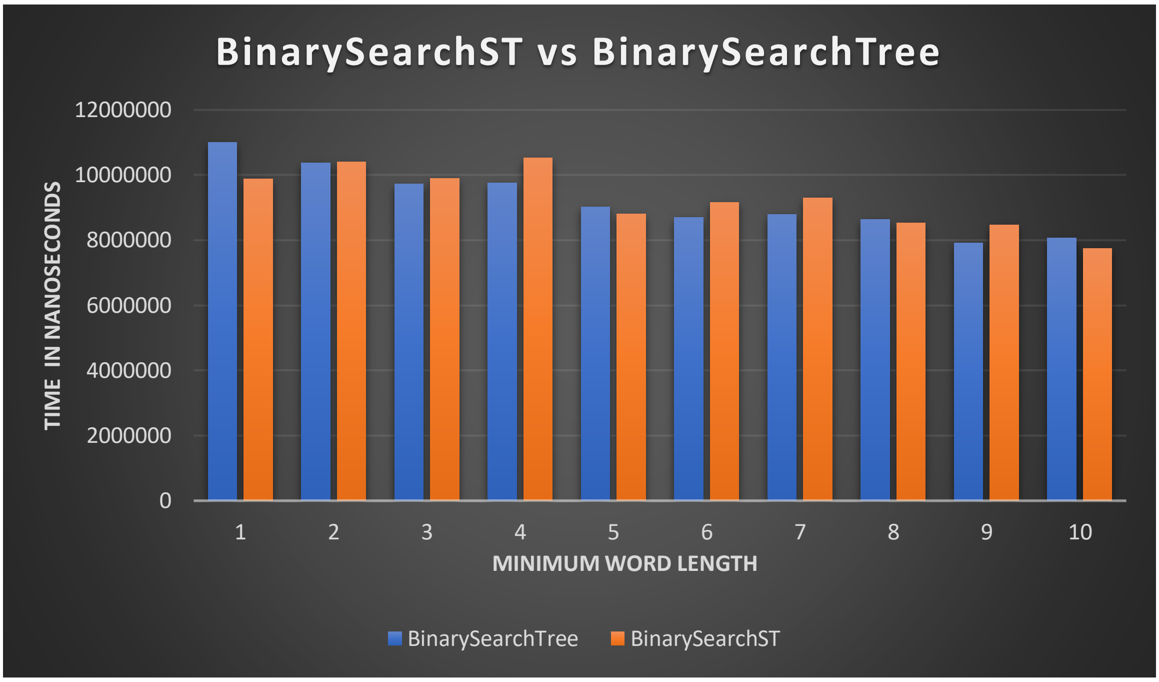

BinarySearchST
BinarySearchST
12000000
10000000
8000000
6000000
4000000
2000000
0
Time in nanoseconds
0
2
4
6
8
10
12
minimum word length


BinarySearchTree
BinarySearchTree
12000000
10000000
8000000
6000000
4000000
2000000
0
Time in nanoseconds
0
2
4
6
8
10
12
minimum word length


BinarySearchST vs BinarySearchTree
BinarySearchST
BinarySearchTree
Time in nanoseconds
12000000
10000000
8000000
6000000
4000000
2000000
0
0
2
4
6
8
10
12
minimum word length


BinarySearchST vs BinarySearchTree
TIME IN NANOSECONDS
12000000
10000000
8000000
6000000
4000000
2000000
0
1
2
3
4
5
6
7
8
9
10
MINIMUM WORD LENGTH
BinarySearchTree
BinarySearchST

The tests were done by running each value three times and and the mean was picked. The conclusion is that the execution times of unbalanced BinarySearchTree and a BinarySearchST are very similiar. When the test values are up around minLength 8-10 it appears that the BinarySearchTree are a bit faster but no values were constant so it's hard to tell. I would conclude that they are the same, at least for this test.

Binary search Tree:

- Search: θ(log(n)), Worst case (O(n)) for unbalanced BST,

- Insert of node: θ(log(n)) , Worst case (O(n)) for unbalanced BST

- Deletion of node: θ(log(n)), , Worst case (O(n)) for unbalanced BST

Binary Search on sorted array:

- Search: O(log(n))

- Insertion of node: O(n)

- Deletion of node: O(log(n)) + O(n).

So if you want quick inserts and deletes then binary search tree is the choice. If you don't need the inserts and deletes to be super fast than a binary search array is better because it takes up less memory.