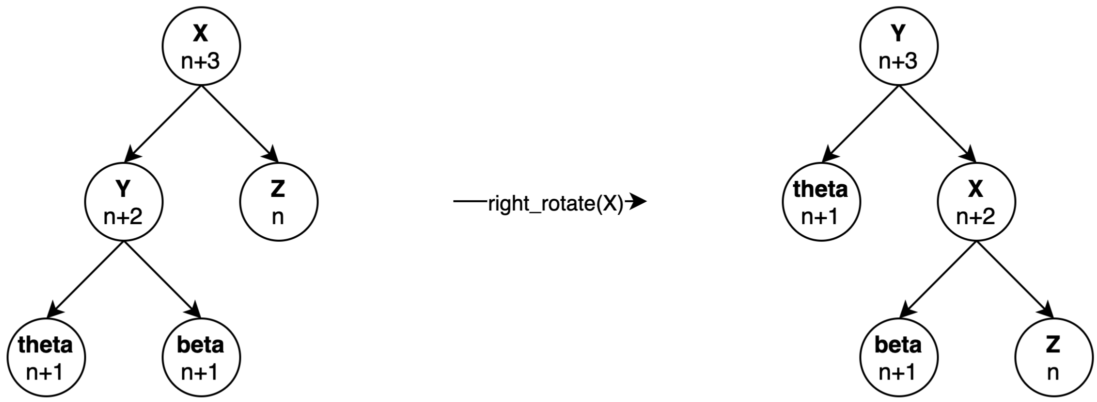
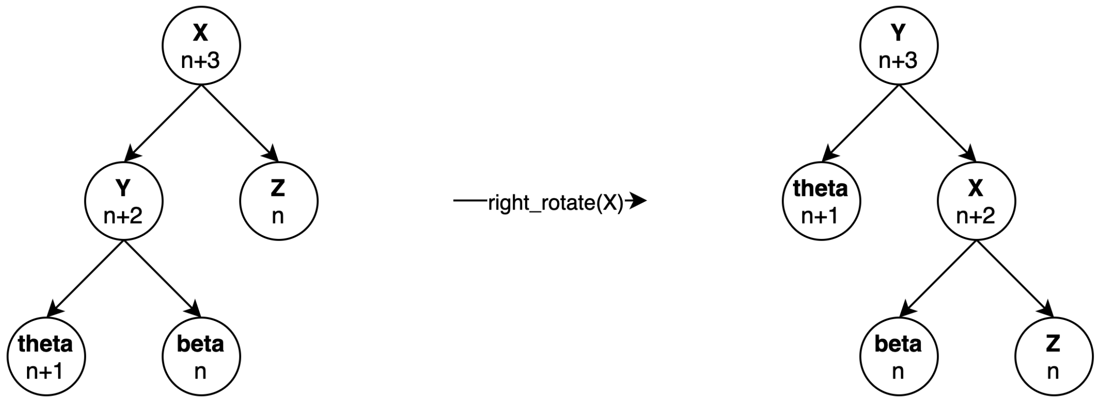
DATA-Lab5

Question 2:

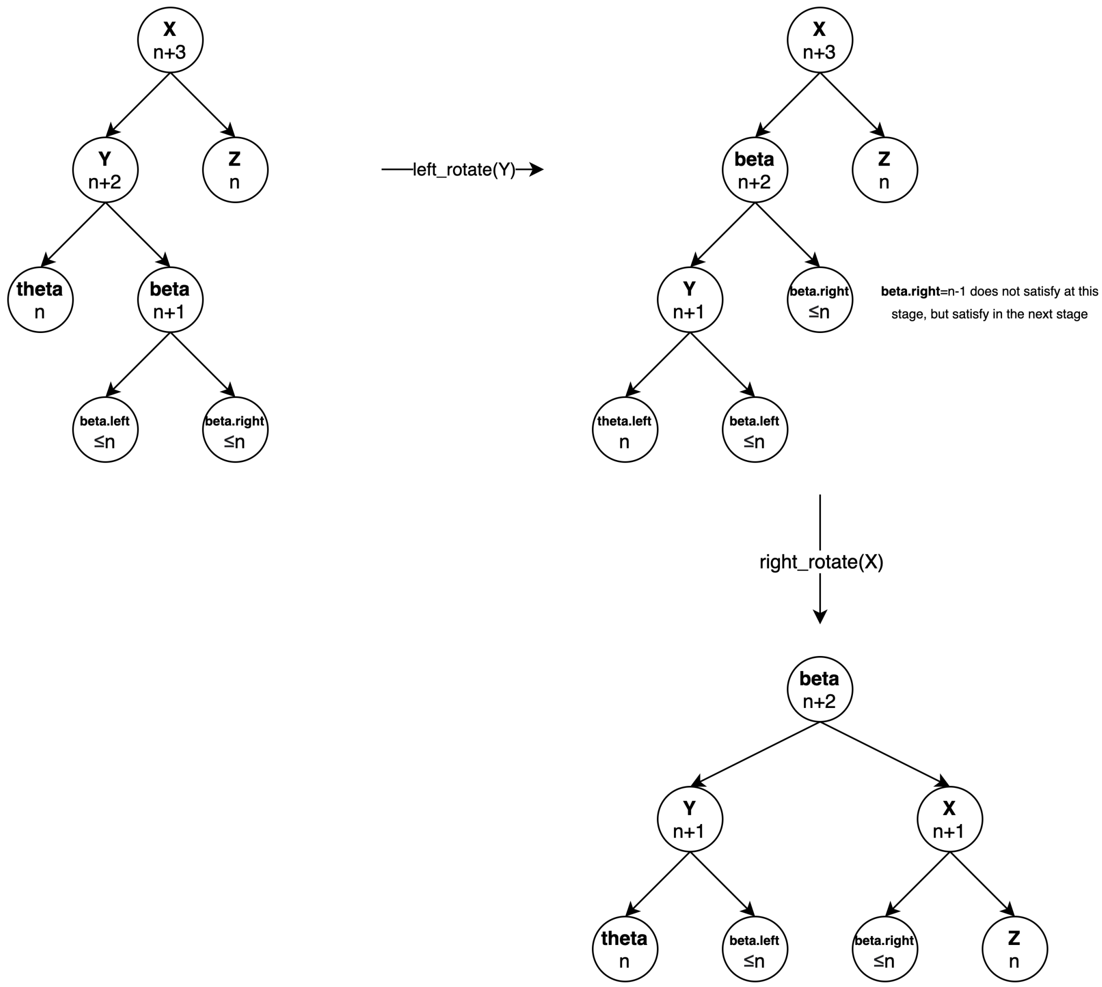
Case 1: The left child of the node to balance has greater height than the right one



Case 1.1: h(theta)=h(beta)=n+1

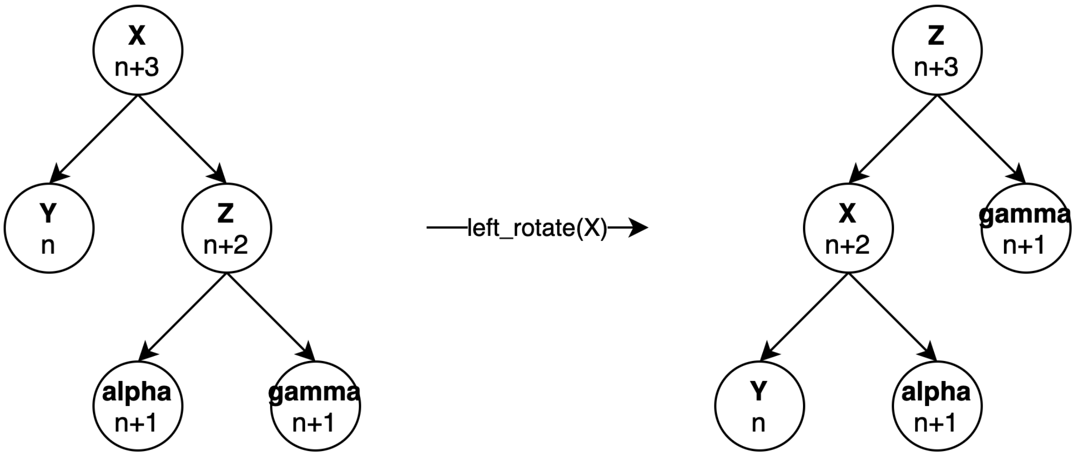


Case 1.2: h(theta)=n+1; h(beta)=n

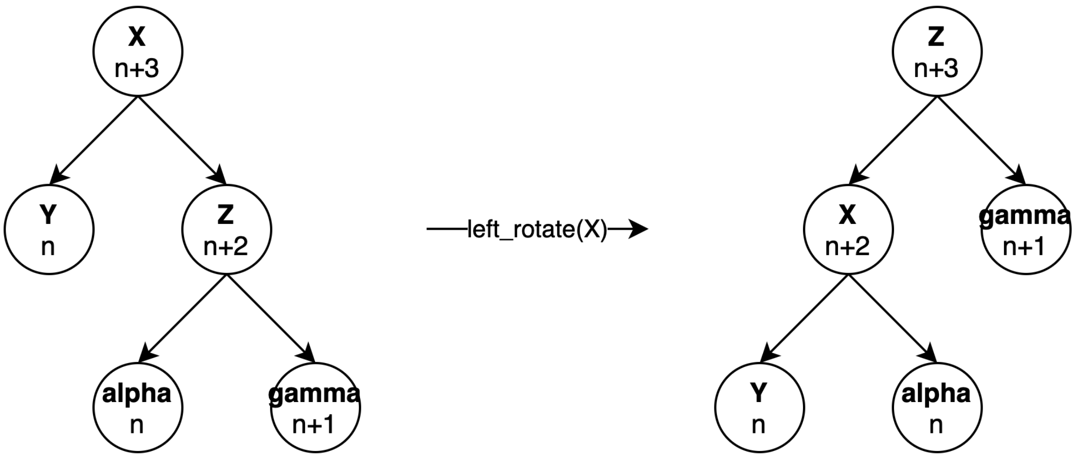


Case 1.3: h(theta)=n; h(beta)=n+1

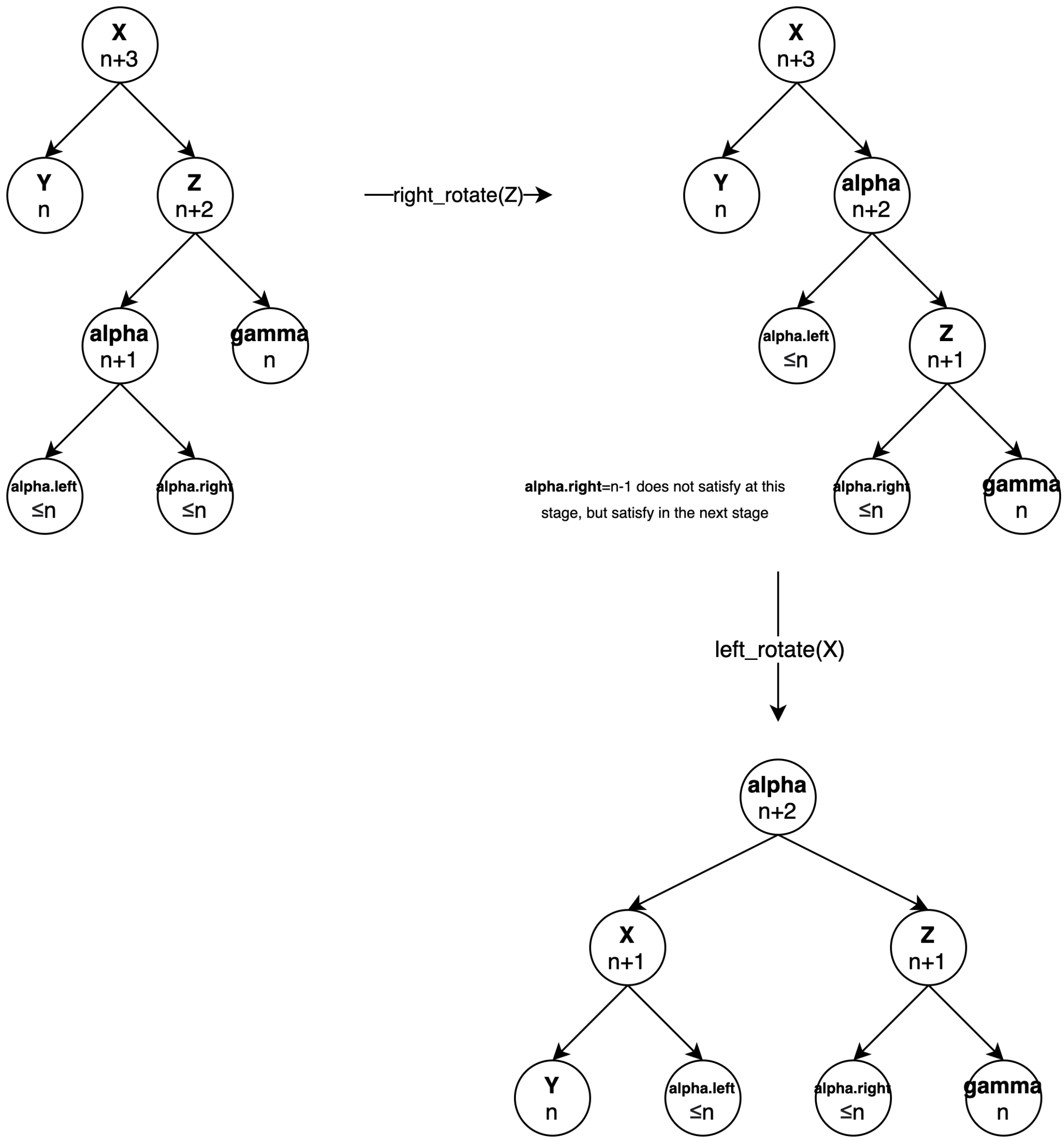
Case 2: The right child of the node to balance has greater height than the left one



Case 2.1: h(alpha)=h(gamma)=n+1



Case 2.2: h(alpha)=n; h(gamma)=n+1



Case 2.3: h(alpha)=n+1; h(gamma)=n

Question 3:

Both bst.insert() and avl.AVL_insert() run in O(h) or O(log(n)). To compare running time of bst.insert() and avl.AVL_insert, 2 tests were performed:

1. **Test 1:** The total running time of inserting 20 new keys into the bst and avl
2. **Test 2:** Wilcoxon two-sided test at different size of input (n≤100). H0: iterative_tree_search() on bst and avl do not express significant difference in running time

⇒ As you can see in Figure 1 and result box from Wilcoxon box, there is no significant difference in running time of 2 insert function. Although AVL_insert() takes an extra step of balancing the height of the tree, but it will support the next insertion, ie. it takes less time to find the correct position for a new key.

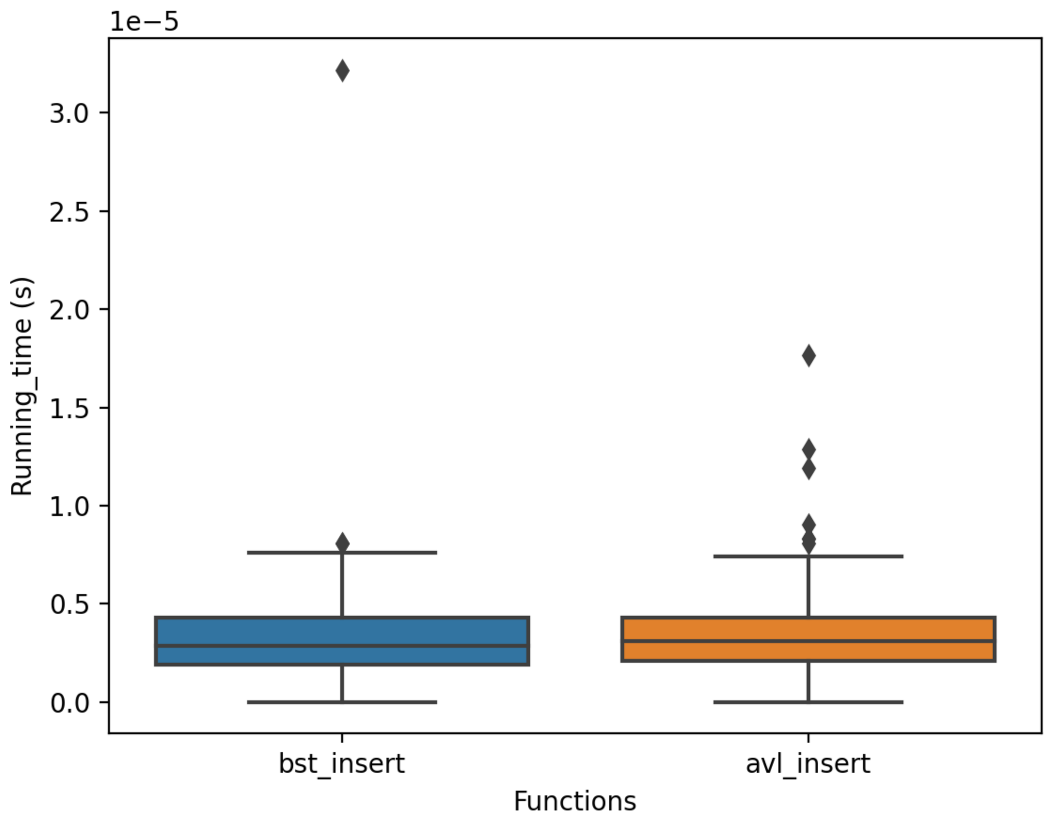


Figure 1. Total running time of bst_insert() and avl_insert(), n=20

p_value from Wilcoxon test for insert() and avl_insert()

```
Attempt no.1:
        pvalue_insert: 0.67769
Attempt no.2:
        pvalue_insert: 0.42465
Attempt no.3:
        pvalue_insert: 0.28125
Attempt no.4:
        pvalue_insert: 0.52353
```

Similarly, avl is also a bst, so iterative_tree_search() on both bst and avl run in O(h) or O(log(n)). To compare running time of the functions one 2 types of trees, 2 tests were performed:

1. **Test 1:** Running time when searching for a number does not exist in the trees when each has 20 elements.
2. **Test 2:** Wilcoxon two-sided test at different size of input. H0: iterative_tree_search() on bst and avl do not express significant difference in running time

⇒ As you can see in Figure 2 and result box from Wilcoxon box, there is no significant difference in running time of 2 insert function (H0 is not rejected). The graphs from test 1 are not consistent that the running time is less on avl or bst. In worst case where the key are mostly inserted in increasing or decreasing order, avl is more efficient. The bst might take the longer branch to find the key (Then it runs in O(n)). However, it is undeniable that there is some chance that the bst takes short branch to find the key (eg. h=2) and it might be much faster than the avl with balanced branches, which has to traverse through a branch of h=3. For example, find 0 in the trees below.

```
Example.
Normal bst
(10)---(13)---(20)---(21)
      |      |      |
      |      |      |_NIL
      |      |
      |      |_(11)
      |
      |_(5)
Tree using AVL_insert()
(13)---(20)---(21)
     |      |
     |      |_NIL
     |
     |_(10)---(11)
             |
             |_(5)
```

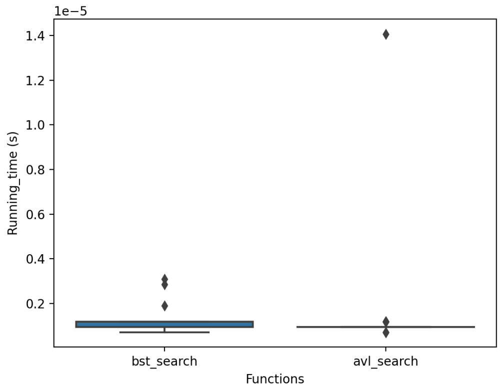


Figure 2.1 Total running time of bst_search() and avl_search(), n=20

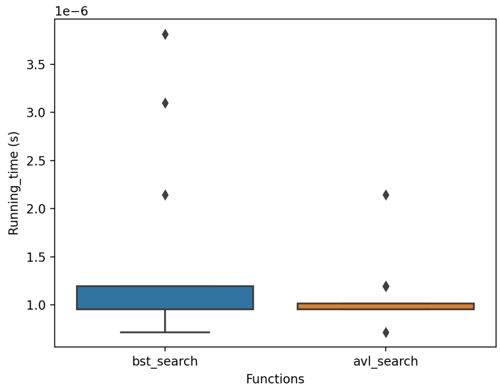


Figure 2.2 Total running time of bst_search() and avl_search(), n=20

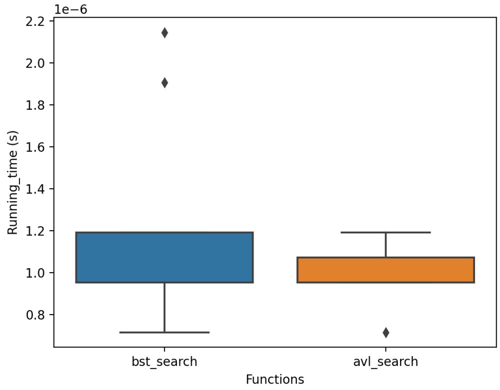


Figure 2.3 Total running time of bst_search() and avl_search(), n=20

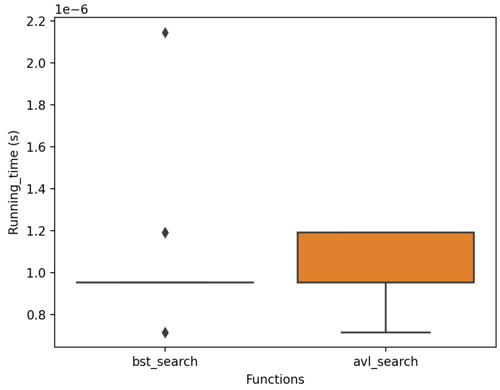


Figure 2.4 Total running time of bst_search() and avl_search(), n=20

p_value from Wilcoxon test for iterative_tree_search() on bst and avl

```
Attempt no.1:
        pvalue_search: 0.91846
Attempt no.2:
        pvalue_search: 0.93675
Attempt no.3:
        pvalue_search: 0.44168
Attempt no.4:
        pvalue_search: 0.71599
```

Again for the worst case, to compare running time of iterative_tree_search() on 2 types of trees, 2 tests were performed:

1. **Test 1:** Running time when searching for a number does not exist in the trees when each has 20 elements.

2. **Test 2:** Wilcoxon one-sided test at different size of input. H0: iterative_tree_search() on bst is less efficient in running time than avl.

⇒ Both test results agreed that iterative_tree_search() on avl works more efficiently than on bst.

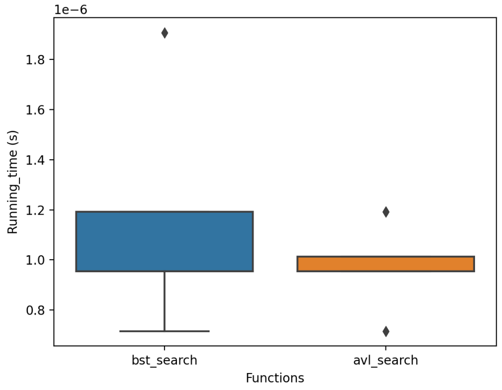


Figure 3.1 Total running time of bst_search() and avl_search(), n=20

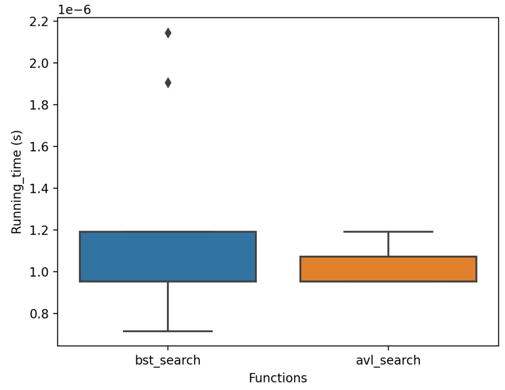


Figure 3.2 Total running time of bst_search() and avl_search(), n=20

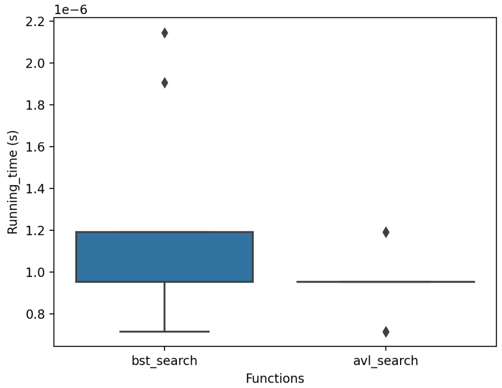


Figure 3.3 Total running time of bst_search() and avl_search(), n=20

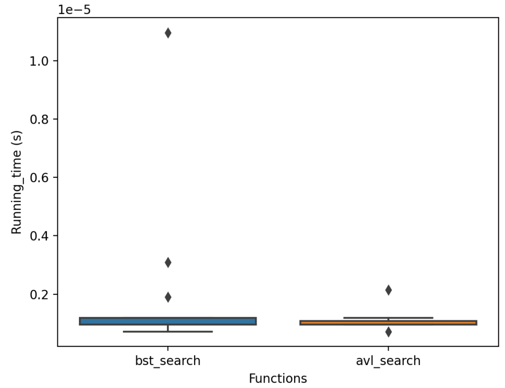


Figure 3.4 Total running time of bst_search() and avl_search(), n=20

p_value from one-sided Wilcoxon test for iterative_tree_search() on bst and avl in worst case

```
Attempt no.1:
        pvalue_search: 0.95265
Attempt no.2:
        pvalue_search: 0.9855
Attempt no.3:
        pvalue_search: 0.88238
Attempt no.4:
        pvalue_search: 0.96809
```