7th of June Madina Bektayeva, 20162014

# Assignment 4: Binary Search Trees

In this assignment AVL Tree and Binary Search Tree were studied. The whole idea and algorithms were taken from " Data Structures and Algorithms in C++ ".

I am submitting this assignment at 7th of June while deadline was 4th of June with 3 days of lateness. Fortunately, I used up only one out 4 late days without penalty and hope left 3 days without penalty will cover my lateness. The reason I am so late is that I started late because of exams, but desire to make solution better made me spend much more time that I planned. Overall, this assignment helped to improve my skills a lot and understand the concept of pointers and trees.

Implementation idea was straightforward taken from lecture materials. All methods except restructure(x) got really easy in dealing with, while restructure(x) took more time since it was difficult for me to keep track of all pointers in all rotations and I usually got some segmentation fault errors and warnings.
In Performance and comparison part I created tests for 100, 1000, 10000, 100000 and 1000000 elements

```
for(int setSize = 100; setSize <= 1000000; setSize = setSize *10){
```

for each set size (100, 1000, 10000, 100000 and 1000000 elements) I fill 3 files with 3 different random sets.

```
fstream randomSetIn("RandomSet.txt", ios::in|ios::out|ios::trunc);
fstream randomSetFind("RandomSetFind.txt", ios::in|ios::out|ios::trunc);
fstream randomSetErase("RandomSetEraseFind.txt", ios::in|ios::out|ios::trunc);
```

For generation of random numbers I use seed and do srand() in 3 different ways to generate random different sets for testing insertion, find and erase and find.
Each time I run main.cpp I got different outputs in file "PerformanceComparison.txt" because I made srand dependent on current time.
In testing each insertion, find and erase and find I use clock(), only during operation I test.

```
myfileIn<<" a) Insertion: " <<'\n';
//Search Tree
fstream randomSetOut("RandomSet.txt", ios::in|ios::out);
myfileIn<<"Insertion to SearchTree a dataset of random nu
start = clock();
while (randomSetOut >> randomKey)
{
    randomSetOut >> randomValue;
    //insertion
    MySt.insert(randomKey, randomValue);
}
stop = clock();
myfileIn<< (start - stop )<< " clocks. "<<'\n';
randomSetOut.close();
```

```
myfileIn<<" b) Find: " <<'\n';
//Search Tree
fstream randomSetFindOut("RandomSetFind.txt", ios::in|ios:
myfileIn<<"Find in SearchTree elements from dataset of ran
start = clock();
while (randomSetFindOut >> randomKeyFind)
{
    randomSetFindOut >> randomValueFind;
    //find
    MySt.find(randomKeyFind);
}
stop = clock();
myfileIn<< (start - stop )<< " clocks. "<<'\n';
randomSetFindOut.close();
```

As it is seen, I take test set for insertion from RandomSet.txt and for find from RandomSetFind.txt, files with different random datasets generated using srand()

```
// generating random set for insertion
srand(time(NULL)/(i+1)*setSize); // trying random numbers
unsigned int key = rand()%setSize + 1;

srand((i*i*setSize));
float value = static_cast <float> (rand())/static_cast<float>

randomSetIn<<key;
randomSetIn<<" ";
randomSetIn<<value;
randomSetIn<<'\n';

// generating random set for find
srand(time(NULL)*time(NULL)/(i+1)*setSize);
unsigned int keyToFind = rand()%setSize + 1;

srand((i*i*i*setSize));
float valueToFind = static_cast <float> (rand())/static_cast<

randomSetFind<<keyToFind;
randomSetFind<<" ";
randomSetFind<<valueToFind;
randomSetFind<<'\n';
```

For erase and find I generate random choice between insert and find in these ratios: 30/70, 50/50, 60/40. I do this using srand and mode of division by 10:

```
srand(STseed);
int choice = rand() % 10;
if(choice <=2){
    // 30 % erase
```

So if choice is 0, 1 or 2, I do erase since these 3 elements are 30% of all possible and do find otherwise.

In total I got these results:

---

Perfomance Comparison:

Test set:100.
a) Insertion: Insertion to SearchTree a dataset of random numbers with size 100 takes 0 clocks.
Insertion to AVLTree a dataset of random numbers with size 100 takes 0 clocks.
b) Find: Find in SearchTree elements from dataset of random numbers with size 100 takes 0 clocks.
Find in AVLTree elements from dataset of random numbers with size 100 takes 0 clocks.
c) Erase and Find: Erase and Find in SearchTree elements from dataset of random numbers with size 100 and ratio 30/70 takes 0 clocks.
Erase and Find in AVL Tree elements from dataset of random numbers with size 100 and ratio 30/70 takes 0 clocks.
Erase and Find in SearchTree elements from dataset of random numbers with size 100 and ratio 50/50 takes 0 clocks.
Erase and Find in AVL Tree elements from dataset of random numbers with size 100 and ratio 50/50 takes 0 clocks.
Erase and Find in SearchTree elements from dataset of random numbers with size 100 and ratio 60/40 takes 0 clocks.
Erase and Find in AVL Tree elements from dataset of random numbers with size 100 and ratio 60/40 takes 0 clocks.

Test set:1000.
a) Insertion: Insertion to SearchTree a dataset of random numbers with size 1000 takes 0 clocks.
Insertion to AVLTree a dataset of random numbers with size 1000 takes 0 clocks.
b) Find: Find in SearchTree elements from dataset of random numbers with size 1000 takes 10000 clocks.
Find in AVLTree elements from dataset of random numbers with size 1000 takes 0 clocks.
c) Erase and Find: Erase and Find in SearchTree elements from dataset of random numbers with size 1000 and ratio 30/70 takes 0 clocks.
Erase and Find in AVL Tree elements from dataset of random numbers with size 1000 and ratio 30/70 takes 0 clocks.
Erase and Find in SearchTree elements from dataset of random numbers with size 1000 and ratio 50/50 takes 10000 clocks.
Erase and Find in AVL Tree elements from dataset of random numbers with size 1000 and ratio 50/50 takes 0 clocks.

Erase and Find in SearchTree elements from dataset of random numbers with size 1000 and ratio 60/40 takes 0 clocks.
Erase and Find in AVL Tree elements from dataset of random numbers with size 1000 and ratio 60/40 takes 10000 clocks.

Test set:10000.
a) Insertion: Insertion to SearchTree a dataset of random numbers with size 10000 takes 20000 clocks.
Insertion to AVLTree a dataset of random numbers with size 10000 takes 30000 clocks.
b) Find: Find in SearchTree elements from dataset of random numbers with size 10000 takes 20000 clocks.
Find in AVLTree elements from dataset of random numbers with size 10000 takes 10000 clocks.
c) Erase and Find: Erase and Find in SearchTree elements from dataset of random numbers with size 10000 and ratio 30/70 takes 40000 clocks.
Erase and Find in AVL Tree elements from dataset of random numbers with size 10000 and ratio 30/70 takes 30000 clocks.
Erase and Find in SearchTree elements from dataset of random numbers with size 10000 and ratio 50/50 takes 40000 clocks.
Erase and Find in AVL Tree elements from dataset of random numbers with size 10000 and ratio 50/50 takes 40000 clocks.
Erase and Find in SearchTree elements from dataset of random numbers with size 10000 and ratio 60/40 takes 30000 clocks.
Erase and Find in AVL Tree elements from dataset of random numbers with size 10000 and ratio 60/40 takes 30000 clocks.

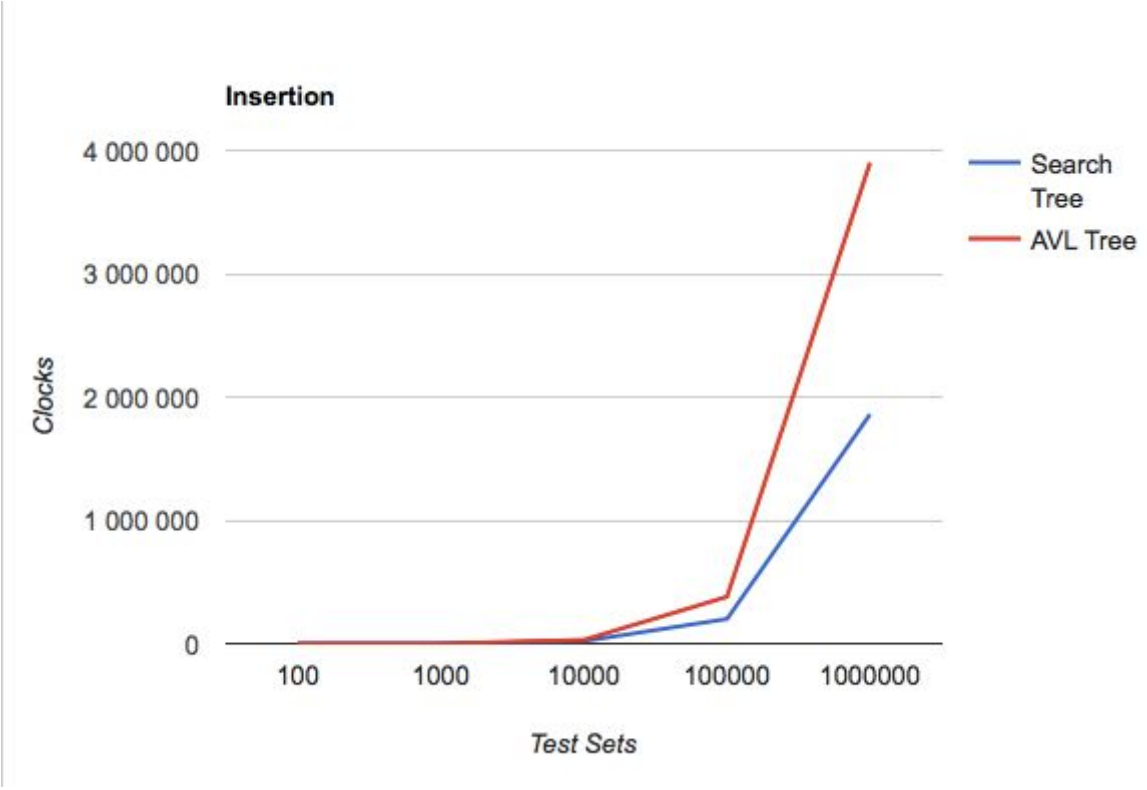
Test set:100000.
a) Insertion: Insertion to SearchTree a dataset of random numbers with size 100000 takes 200000 clocks.
Insertion to AVLTree a dataset of random numbers with size 100000 takes 380000 clocks.
b) Find: Find in SearchTree elements from dataset of random numbers with size 100000 takes 190000 clocks.
Find in AVLTree elements from dataset of random numbers with size 100000 takes 170000 clocks.
c) Erase and Find: Erase and Find in SearchTree elements from dataset of random numbers with size 100000 and ratio 30/70 takes 380000 clocks.
Erase and Find in AVL Tree elements from dataset of random numbers with size 100000 and ratio 30/70 takes 390000 clocks.
Erase and Find in SearchTree elements from dataset of random numbers with size 100000 and ratio 50/50 takes 380000 clocks.
Erase and Find in AVL Tree elements from dataset of random numbers with size 100000 and ratio 50/50 takes 370000 clocks.
Erase and Find in SearchTree elements from dataset of random numbers with size 100000 and ratio 60/40 takes 370000 clocks.
Erase and Find in AVL Tree elements from dataset of random numbers with size 100000 and ratio 60/40 takes 360000 clocks.

Test set:1000000.
a) Insertion: Insertion to SearchTree a dataset of random numbers with size 1000000 takes 1860000 clocks.
Insertion to AVLTree a dataset of random numbers with size 1000000 takes 3900000 clocks.
b) Find: Find in SearchTree elements from dataset of random numbers with size 1000000 takes 2090000 clocks.

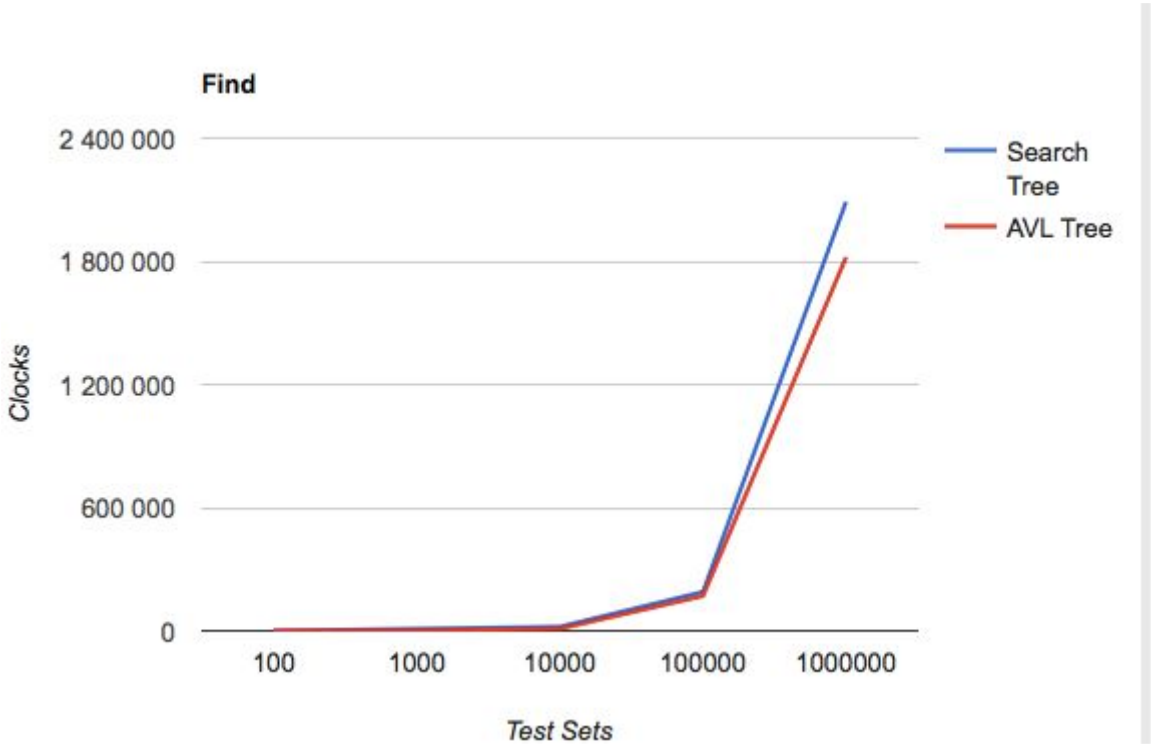
Find in AVLTree elements from dataset of random numbers with size 1000000 takes 1820000 clocks.
c) Erase and Find: Erase and Find in SearchTree elements from dataset of random numbers with size 1000000 and ratio 30/70 takes 3950000 clocks.
Erase and Find in AVL Tree elements from dataset of random numbers with size 1000000 and ratio 30/70 takes 3720000 clocks.
Erase and Find in SearchTree elements from dataset of random numbers with size 1000000 and ratio 50/50 takes 3900000 clocks.
Erase and Find in AVL Tree elements from dataset of random numbers with size 1000000 and ratio 50/50 takes 3690000 clocks.
Erase and Find in SearchTree elements from dataset of random numbers with size 1000000 and ratio 60/40 takes 3890000 clocks.
Erase and Find in AVL Tree elements from dataset of random numbers with size 1000000 and ratio 60/40 takes 3640000 clocks.

---

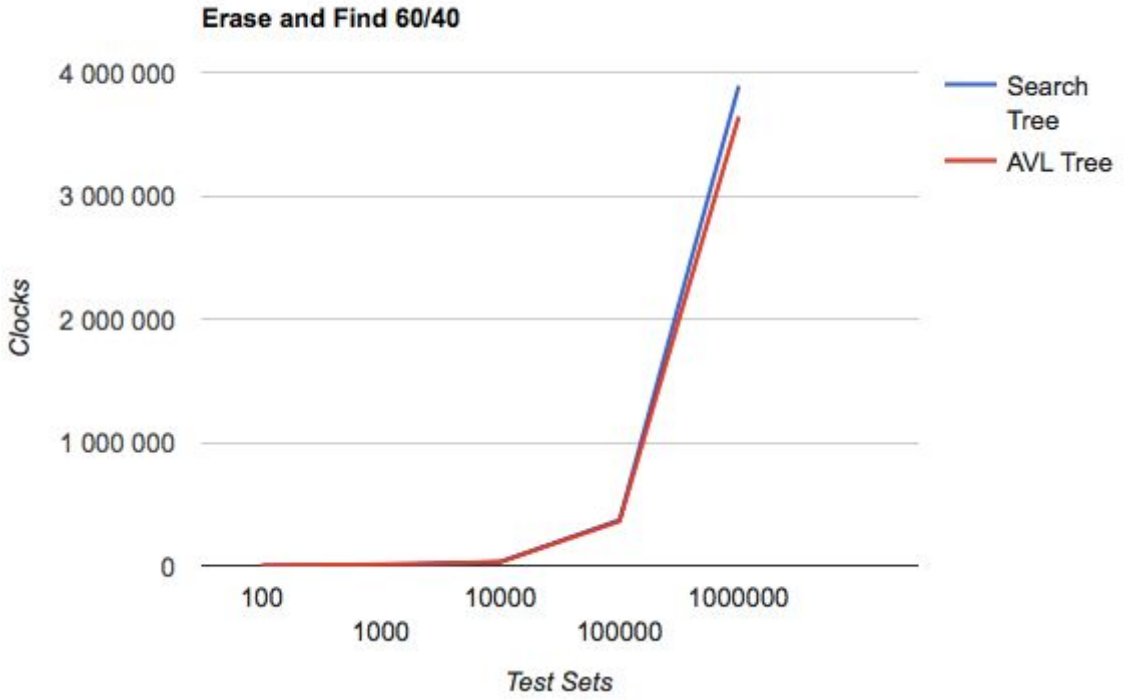
Let’s take a look at visualization for better analysis:

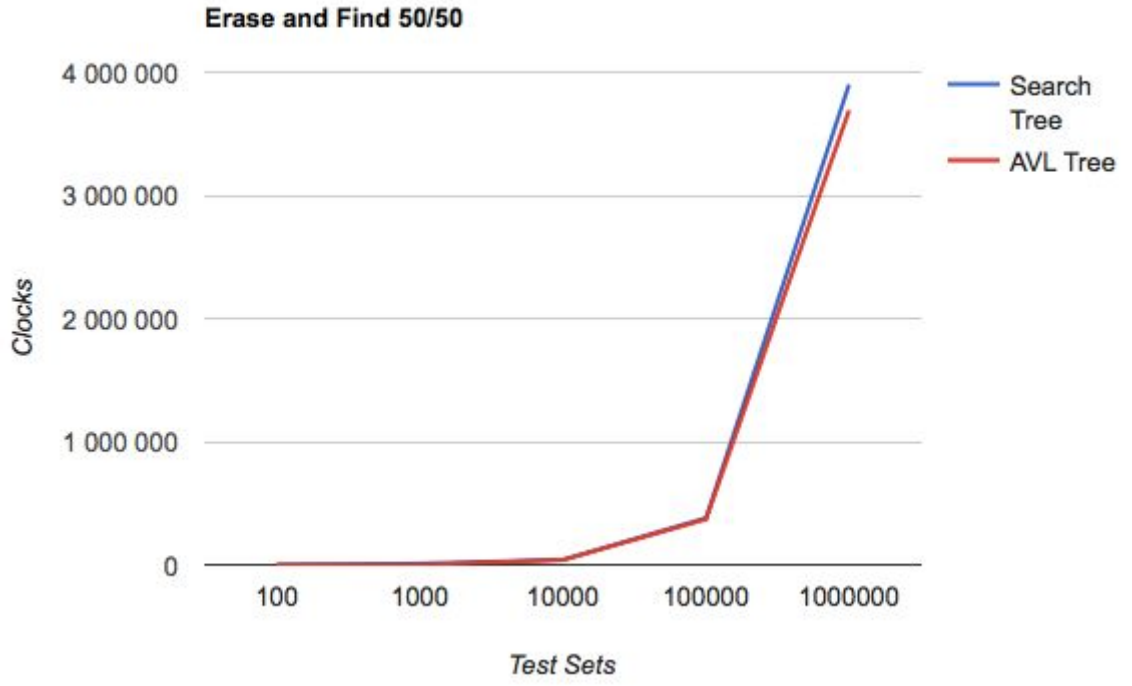


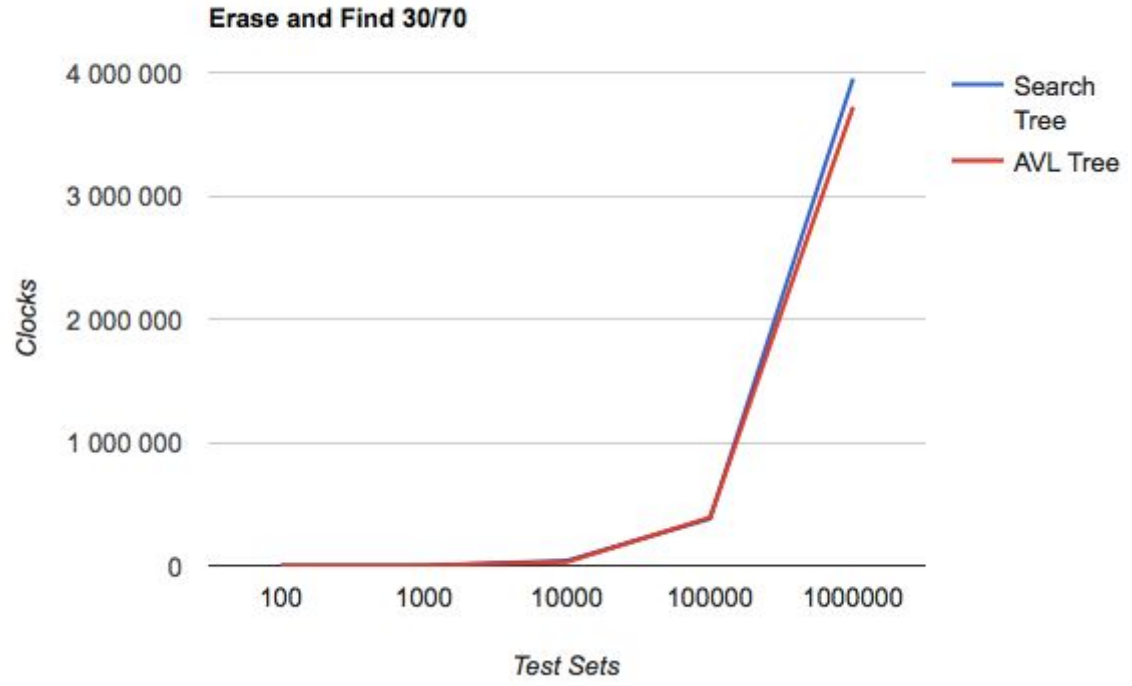

In my tests for small datasets AVL Tree and Search Tree a pretty similar, however as it could be expected Insertion to AVL tree takes much more time for large datasets. It is natural consequence because often insertion to AVL Tree require additional operations for tree rotations for keeping tree balanced.




However, find in AVL Tree in most cases better than in Search Tree. This is where balanced structure of a tree shows its  use. In my opinion, AVL Tree is perfect data structure for data you need to insert once and use a lot without modification.
Going further:






In all above conditions AVL Tree has shown better performance for erase and find.Overall improvement in find overweights all drawbacks.
In conclusion, I want to say that for small data sets both trees are pretty similar. However, for large data set, especially, if amount of find operations overweights modification operations, AVL Tree is much better than  Search Tree.