## **Exercise 11E. (Voluntary exercise, Binary search tree, 1p)**

We studied the principles of binary search in the class. Implement the class binary search tree. It means that you have to define a class BinSTree and write the following operation functions (in this exercise only these three are required):

```
contstructor (initializes the tree)
insert_to_tree (insert an item to binary search tree)
isInTree (finds out whether the item is in tree or not)
```

Our search tree stores integer numbers, but remember that the implementation should be independent of the item type.

Test your search tree in the following way. Write a program, that generates 10000 integer random numbers between [0, 10000) and stores them to the binary search tree. Because numbers to be inserted are generated randomly, there is no need to balance the tree. After inserting the numbers, the program enters the loop, where it asks the user to enter integer numbers one at a time. When a number is entered, the program finds out whether the number is in tree or not. It prints the result of the search. It is interesting to know, how many comparisons are needed in each case. Calculate inside the isInTree function, how many comparisons were needed before the item was found or it became clear that it is not in the tree. Print this information to the screen at the end of function isInTree.

Remark1. The goal of the exercise is not only to get the program to work. It is also to learn to understand the concept of binary search tree a bit more thoroughly. When you run the program try to search for different kinds of numbers from the whole range. Enter numbers which are in the tree and numbers which are not. What is the largest number of comparisons needed and what is the smallest. What does it mean?

Hint. C-standard library (stdlib.h) contains a pseudo-random generator srand() and rand() functions<sup>1</sup>. srand() is used to initialize the generator (if you don't initialize the generator, it will always generate the same "random" sequence). rand() returns the random integer value between 0 – MAX RAND.

 $<sup>^1</sup>$  Note that in MS-Windows environment we have the function rand\_s() which gives statistically better results than rand() function. If you want to use that function, please refer Visual Studio help how to use that function.

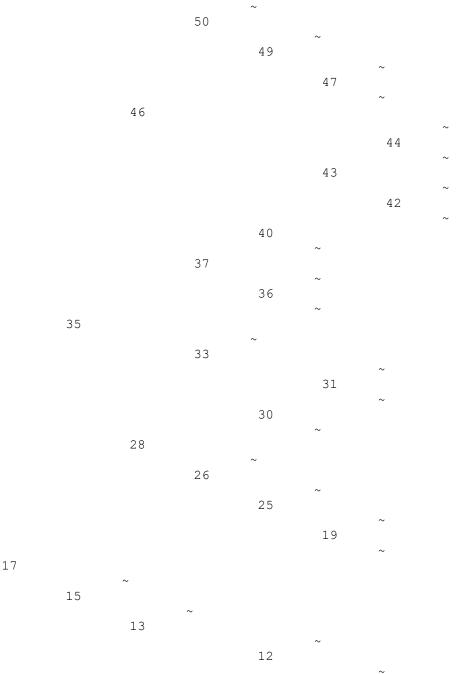
Here is a demo output of one solution (note that the number of items inserted to the tree is smaller than required to be able to show the tree in graphical format, and note that the queries to the search tree are automated):

BST exerciser (51 items stored to the tree, 51 items searched for)

Numbers inserted to the BST:

17 15 35 28 46 37 33 40 15 13 26 43 37 30 8 17 31 2 50 25 2 12 3 9 26 36 50 42 0 35 15 28 49 3 28 44 17 8 1 43 19 47 46 28 25 31 26 9 44 37 44

Generated three in graphical format (root is at the left):



9

## Lab. exercise 11 TX00EW61 Data Structures and Algorithms

28.02.2025 JV

found 6 comparision needed found 7 comparision needed found 5 comparision needed 2: found 6 comparision needed 4: not found 6 comparision needed 5: not found 6 comparision needed 6: not found 6 comparision needed 7: not found 6 comparision needed found 4 comparision needed 8: found 6 comparision needed 10: not found 6 comparision needed 11: not found 6 comparision needed 12: found 5 comparision needed 13: found 3 comparision needed 14: not found 3 comparision needed 15: found 2 comparision needed 16: not found 2 comparision needed 17: found 1 comparision needed 18: not found 6 comparision needed 19: found 6 comparision needed 20: not found 6 comparision needed 21: not found 6 comparision needed 22: not found 6 comparision needed 23: not found 6 comparision needed 24: not found 6 comparision needed 25: found 5 comparision needed 26: found 4 comparision needed 27: not found 4 comparision needed 28: found 3 comparision needed 29: not found 5 comparision needed 30: found 5 comparision needed found 6 comparision needed 32: not found 6 comparision needed 33: found 4 comparision needed 34: not found 4 comparision needed found 2 comparision needed found 5 comparision needed found 4 comparision needed 38: not found 5 comparision needed 39: not found 5 comparision needed found 5 comparision needed 41: not found 7 comparision needed found 7 comparision needed 43: found 6 comparision needed found 7 comparision needed 44: 45: not found 7 comparision needed

## **METROPOLIA**

Information Technology

## Lab. exercise 11 TX00EW61 Data Structures and Algorithms

Page 4

28.02.2025 JV

```
46: found 3 comparision needed 47: found 6 comparision needed 48: not found 6 comparision needed 49: found 5 comparision needed 50: found 4 comparision needed
```

Theoretical number of comparisions needed is 5.7 Average number of actual comparisions is 4.7 Min depth 2, Max depth 7