
CS202, Spring 2023

Homework 2 - Binary Search Trees

Due: 25/03/2023

Before you start your homework please read the following instructions carefully:

FAILURE TO FULFIL ANY OF THE FOLLOWING REQUIREMENTS WILL RESULT IN A GRADE SCORE OF 0 (zero) WITHOUT ANY CHANCE OF REDEMPTION.

- See the course page for any late submission policies and Honor Code for Assignments.
- Upload your solutions in a single ZIP archive using the Moodle submission form. Name the file as studentID_name_surname_hw2.zip.
- Your ZIP archive should contain **only** the following files:
  - **studentID_name_surname_hw2.pdf**, the file containing the answers to Questions 1, and 3.
  - main.cpp, BSTNode.h, BSTNode.cpp, BST.h, analysis.h and analysis.cpp files which contain the C++ source codes and the **Makefile**.
  - Do not forget to put your name, student id, and section number in all of these files. Comment your implementation well. Add a header (see below) to the beginning of each file:

```
/**
 * Title: Binary Search Trees
 * Author : Name & Surname
 * ID: 12345678
 * Section : 1
 * Homework : 2
 * Description : description of your code
 */
```

  - Do not put any unnecessary files such as the auxiliary files generated from your preferred IDE.
- Your code must compile.
- Your code must be complete.
- Your code must run on the **dijkstra.cs.bilkent.edu.tr** server.
- For any question related to the homework contact your TA: *saeed.karimi@bilkent.edu.tr*

Question 1 (25 points)

(a) (5 points) Insert 8, 15, 10, 4, 9, 6, 16, 5, 7, 14 into an empty binary search tree in the given order. Show the resulting BST after every insertion.

(b) (5 points) What are the preorder, inorder, and postorder traversals of the BST you have after (a)?

(c) (5 points) Delete 7, 10, 8, 9, 5 from the BST you have after (a) in the given order. Show the resulting BST after every deletion.

(d) (5 points) Write a recursive pseudocode implementation for finding the minimum element in a binary search tree.

(e) (5 points) What is the maximum and minimum height of a binary search tree that contains n items ?

Question 2 (55 points)

(a) (15 points) Implement a pointer-based Binary Search Tree whose keys are integers. Implement methods for insertion, deletion and inorder traversal. Traversal method must return an ordered array of keys and set the length parameter to the length of the array. Put your code into the **`BSTNode.h`**, **`BSTNode.cpp`**, **`BST.h`**, **`BST.cpp`** files. Prototypes of the methods must be the following:

- **`void BST::insertItem(int key);`** // 5 points
- **`void BST::deleteItem(int key);`** // 5 points
- **`int* BST::inorderTraversal(int& length);`** // 5 points

(b) (20 points) Implement a method named `hasSequence` that takes a sorted array of integers as input, returns true if it is a sub-array of the inorder traversal of the BST, and false otherwise. The method should **not** traverse all nodes, but only the nodes that are required to perform this check. Your method should print the key of the node when it enters the node while traversing the tree, so that one can check which nodes are visited. Prototype of the method must be the following:

```
bool BST::hasSequence(int* seq, int length);
```

For example, for the BST in Figure 1 :

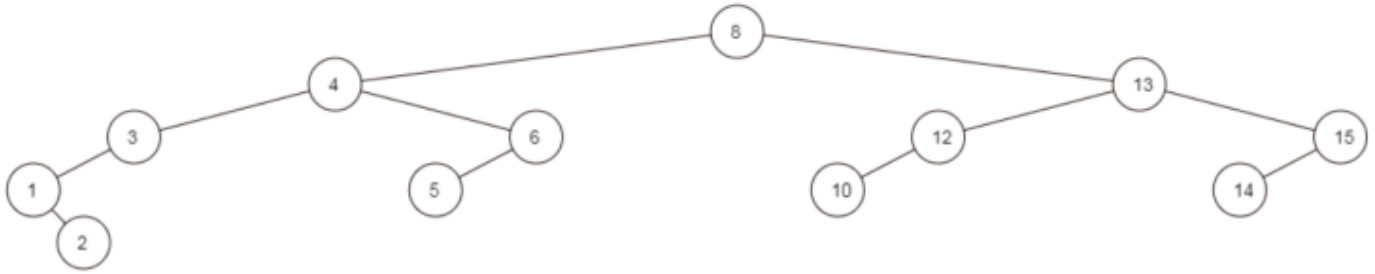

Figure 1: BST Example

- hasSequence([1, 2, 3, 4, 5, 6], 6) must return true and must not traverse the subtree rooted at 13.
- hasSequence([10, 12, 13, 15], 4) must return false since the given sequence skips 14, and must **not** traverse the subtree rooted at 4.
- hasSequence([10, 11, 12], 3) must return false since 11 does not exists in the tree, and must **not** traverse the subtrees rooted at 4 and 15.

(c) (10 points) Implement a standalone function named merge (not a member method of BST) that takes two BSTs as input and returns a new BST that contains all elements of the two given BSTs (the two trees could be merged in any way as long as the resulting one is also a valid BST). Put your code into the main.cpp file. Prototype of the function must be the following:

```
BST* merge(const BST& tree1, const BST& tree2);
```

(d) (0 points, mandatory) Add a main function to the **`main.cpp`** file which calls the functions above with proper inputs (you will need to create some trees first), and display their outputs. At the end, write a basic **`Makefile`** which compiles all your code and creates an executable file named hw2. Please make sure that your **`Makefile`** works properly, otherwise you will not get any points from Question 2.

(e) (10 points) In this part, you will analyse the time performance of the pointer based implementation of binary search trees. Write a global function, **`void timeAnalysis()`**, which does the following:

- Creates an array of 15000 random numbers and starts inserting them into an empty pointer based BST. At each 1500 insertions, outputs the time elapsed for those insertions (use clock from **`ctime`** for calculating elapsed time).
- Shuffles the array created in the previous part. Then iterates over it and deletes the numbers from the tree. After each 1500 deletions, outputs the time elapsed for the deletion.

Add your code into **`analysis.h`** and **`analysis.cpp`** file. When **`timeAnalysis`** function is called, it needs to produce an output similar to Figure 2.

```
Part e - Time analysis of Binary Search Tree - part 1
-------------------------------------------------------
Tree Size       Time Elapsed
-------------------------------------------------------
1500              x ms
3000              x ms
...

Part e - Time analysis of Binary Search tree - part 2
-------------------------------------------------------
Tree Size       Time Elapsed
-------------------------------------------------------
13500             x ms
12000             x ms
...
```

Figure 2:

Question 3 (20 points)

After running your programs, you are expected to prepare a single page report about the experimental results that you obtained in Question 2(e). With the help of a spreadsheet program (Google sheets, Matlab or other tools), plot number of elements versus elapsed time after each 1500 insertions and deletions. On the same figure, plot number of elements versus theoretical worst time elapsed after each 1500 insertions and deletions. A sample figure is given in Figure 3. In your report, you need to discuss the following points:

- Interpret and compare your empirical results with the theoretical ones. Explain any differences between the empirical and theoretical results, if any.
- How would the time complexity of your program change if you inserted sorted numbers into it instead of randomly generated numbers?

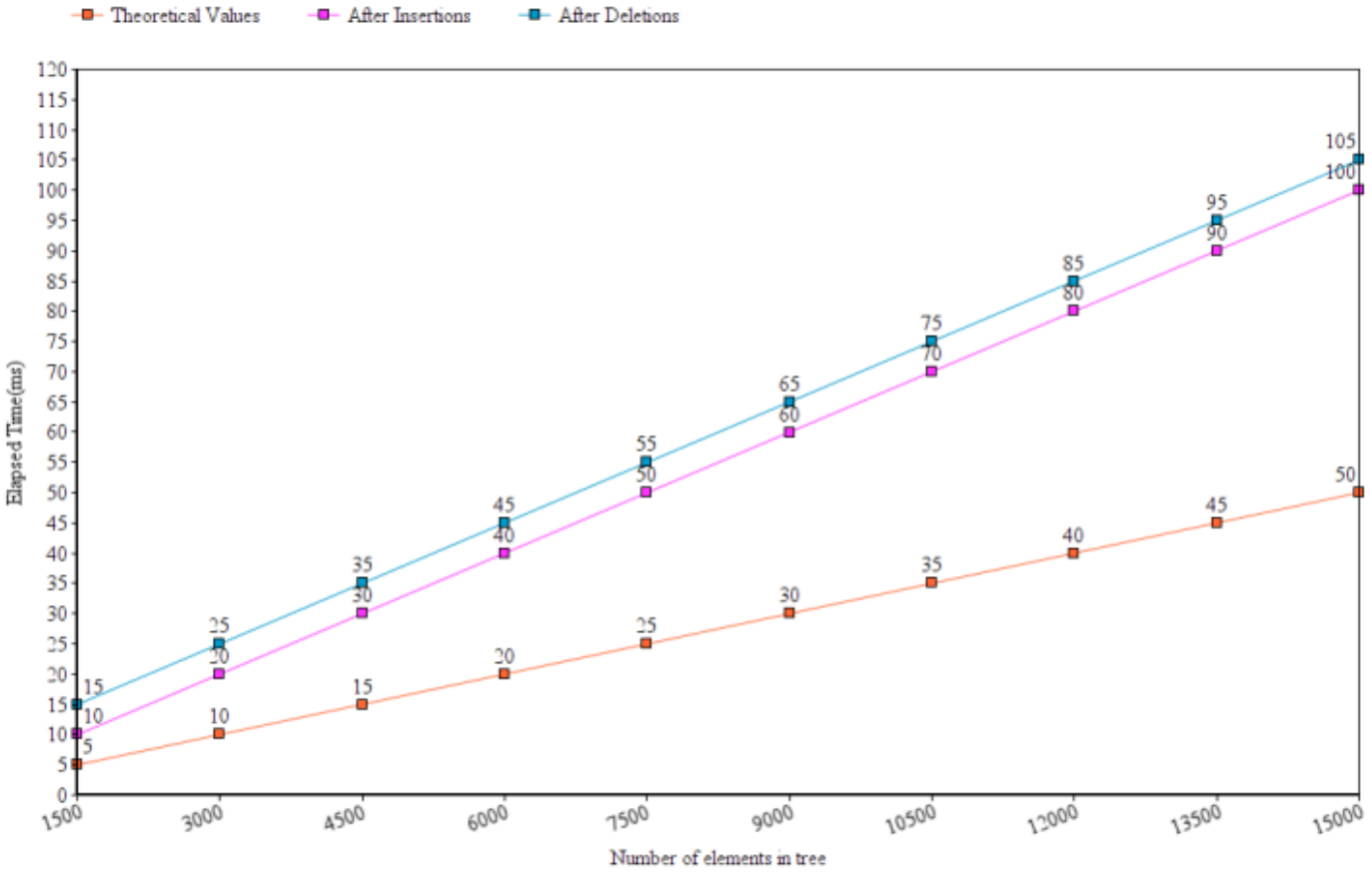

Figure 3: Sample figure for BST Performance Analysis