CS 202, Spring 2019

Homework #2 – Binary Search Trees

Due Date: March 23, 2019

Important Notes

Please do not start the assignment before reading these notes.

- Before 23:55, March 23, upload your solutions in a single **ZIP** archive using Moodle submission form. Name the file as `studentID_hw2.zip`.
- Your ZIP archive should contain the following files:
  - `hw2.pdf`, the file containing the answers to Questions 1 and 3.
  - `BSTNode.h, BSTNode.cpp, BST.h, BST.cpp, main.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. Well comment your implementation. Add a header as in Listing 1 to the beginning of each file:

Listing 1: Header style

```
/**
 * Title: Binary Search Trees
 * Author: Name Surname
 * ID: 21XXXXXX
 * Section: X
 * Assignment: 2
 * Description: description of your code
 */
```

  - Do not put any unnecessary files such as the auxiliary files generated from your favorite IDE. Be careful to avoid using any OS dependent utilities (for example to measure the time).
  - Your submission for Question 1 should be clearly readable, and your submission for Question 3 should be prepared using a word processor. Combine these two parts into a single report file named `hw2.pdf`.

  - Use the exact algorithms shown in lectures.

- Although you may use any platform or any operating system to implement your algorithms and obtain your experimental results, your code should work on the **dijkstra** server (dijkstra.ug.bcc.bilkent.edu.tr). We will compile and test your programs on that server. Please make sure that you are aware of the homework grading policy that is explained in the rubric for homeworks.

- This homework will be graded by your TA, Yigit Ozen. Thus, please contact him directly for any homework related questions.

Attention: For this assignment, you are allowed to use the codes given in our textbook and/or our lecture slides. However, you ARE NOT ALLOWED to use any codes from other sources (including the codes given in other textbooks, found on the Internet, belonging to your classmates, etc.). Furthermore, you ARE NOT ALLOWED to use any data structure or algorithm related function from the C++ standard template library (STL).

Do not forget that plagiarism and cheating will be heavily punished. Please do the homework yourself.

Question 1 – 15 points

You must draw the trees in this question by hand, and embed the scans or photographs of your drawings in your report. Make sure your drawings are clear, as ambiguities would make you lose points.

(a) [*5 points*] Insert 5, 12, 7, 1, 6, 3, 13, 2, 10, 11 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 2, 7, 5, 6, 11 from the BST you have after **(a)** in the given order. Show the resulting BST **after every deletion**.

Question 2 – 70 points

(a) [*21 points*] Implement a pointer-based Binary Search Tree whose keys are integers. Implement methods for insertion, deletion, retrieval, and inorder traversal. Traversal method must return an ordered array of keys and set the length parameter to the

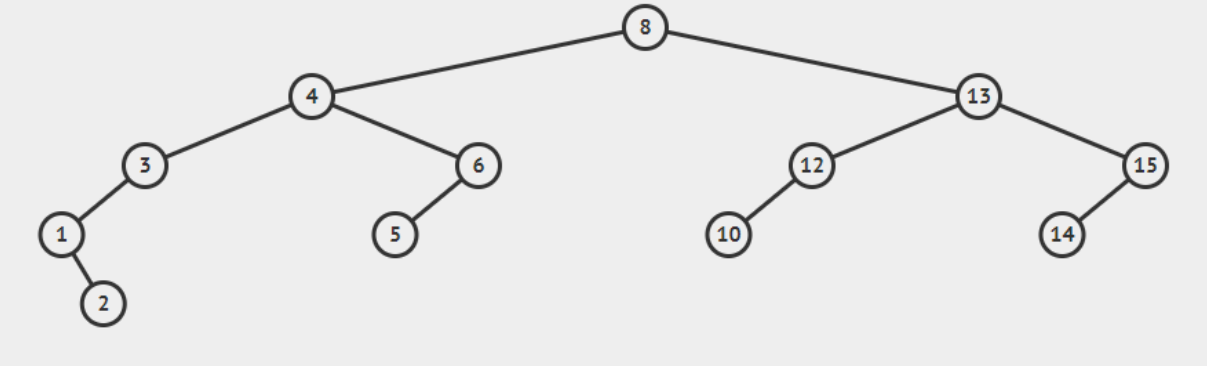
length of the array. (Note: The returned array must be dynamically allocated.) Put your code into the `BSTNode.h, BSTNode.cpp, BST.h, BST.cpp` files. Prototypes of the methods must be the following:

- `void BST::insertItem(int key); // 5 points`
- `void BST::deleteItem(int key); // 5 points`
- `BSTNode* BST::retrieveItem(int key); // 5 points`
- `int* BST::inorderTraversal(int& length); // 6 points`

(b) [*25 points*] Implement a method named `containsSequence` 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. Do 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 to 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::containsSequence(int* seq, int length);`

Figure 1: Example BST



For example, for the BST in Figure 1:

- `containsSequence([1, 2, 3, 4, 5, 6], 6)` must return true and must not traverse the subtree rooted at 13.
- `containsSequence([10, 12, 13, 15], 4)` must return false since the given sequence skips 14, and must not traverse the subtree rooted at 4.
- `containsSequence([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) [*12 points*] The level of a node is the number of nodes in the path from the node to the tree's root node. The level of the root node is 1. Implement a method named

`countNodesDeeperThan` that returns the number of nodes whose level is greater than or equal to a given number:

- `int BST::countNodesDeeperThan(int level);`

(d) [*12 points*] Implement a method named `maxBalancedHeight` that returns what would be the height of the tree if we removed the minimum number of nodes required to make the tree height-balanced. For the BST in Figure 1, removing 2 would make the tree height-balanced, and the result of `maxBalancedHeight` would be 4. (Hint: Think recursively.) Prototype of the method must be the following:

- `int BST::maxBalancedHeight();`

(e) [*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 displays 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.

Question 3 – 15 points

In this question, you need to briefly explain your implementations in Question 2, analyze the asymptotic behavior of your implementations, and derive the resulting worst-case and average-case running time and space complexities. Your answer should be written in your report. You may, if you wish, write parts of your code or pseudo-code while explaining it.