Rules of the Game:

- Solutions need to be submitted through Gradescope.
- If you would like to learn the LaTeX typesettting system and use it for assignment submissions, then please take a look at this template for what it can do.
- If you plan to write your answers on paper, then you will need to scan your work:

App	Platform	Description
Adobe Scan	iOS, Android	Great quality, but requires an Adobe account
Microsoft Office Lens	iOS, Android, Windows 10	Works on laptops too
Evernote Scannable	iOS	Recommended by Gradescope
Genius Scan	iOS, Android	Recommended by Gradescope

- Please consult Gradescope's Scanning Work on a Mobile Device page and watch the Submitting PDF Homework video.

Our Lateness Policy:

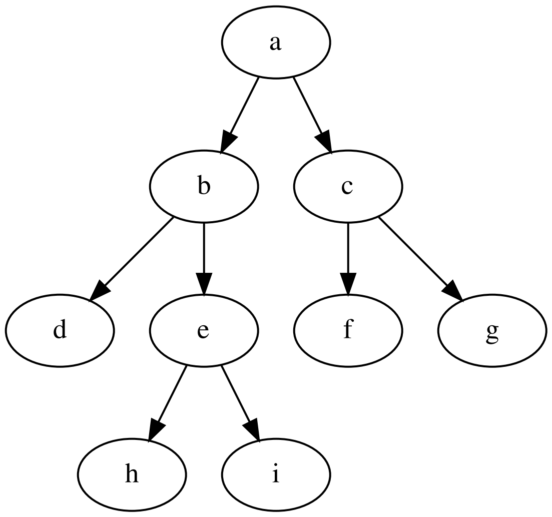
- You have one slip day for **each** assignment you can use without having to notify us. **Work submitted beyond the slip day without an academically valid excuse will not earn any points.**
- The deadline to submit documents regarding academically valid excuses is the original deadline of the assignment (i.e. not that of the slip day).
- In order to reward complying with deadlines, anyone who uses only one slip day will have a 1% added to their final score and anyone who uses zero slip days will a 2% added to their final score.

Regarding Academic Integrity:

- You may collaborate with one or two of your peers in the course to brainstorm for solving the assigned problems. However, your solution must be written up completely on your own.
- If you collaborate with others during the brainstorming part of the assignment, you must list their names at the beginning of your submission.
- You cannot use any online resources, any solutions from past semesters, etc. You are not allowed to share digital or written notes or images of your work in any form.
- Even when a question feels overwhelming, you should not compromise your academic integrity. Since the exam questions will be similar to the assigned exercises/problems, not struggling with these questions (in the context of assignments) will be counter productive with respect to grades.

QUESTIONS/PROBLEMS:

1. Suppose we have a binary search tree to which we are given a root pointer. Provide the pseudo-code for a method that traverses this tree layer by layer. For example, if we have the following tree



then a level-by-level traversal would output

a, b, c, d, e, f, g, h, i

This question can be answered in pseudo-code; that is, you do not have write/submit java code or show any evidence of a program execution. We are interested strictly in algorithmic logic.

2. Suppose the following numbers are inserted into an originally empty AVL tree: 3, 2, 1, 4, 5, 6, 7, 16, 15, 14, 13, 12, 11, 10, 8, 9. What does the final tree look like?

   The right answer to this question will consist of a series tree visualizations. You do not need to redraw if the insertion of an element does not cause any rotations (the new node can simply be appended to the previously drawn tree); you do need to redraw if the insertion of an element causes rotation(s).

3. When a new key-value pair is added to an AVL tree, we run from the newly inserted node all the way up to the root and check, every step along the way, whether there is an AVL imbalance. If we find one (i.e. we find the α node), we correct the issue based on how we arrived at α:

   (a) We arrived from the left child of the left child of α:
   Clockwise rotation at α

   (b) We arrived from the right child of the left child of α:
   Counter-clockwise rotation at α's left child and then clock-wise rotation at α.

   (c) We arrived from the left child of the right child of α:
   Clockwise rotation at α's right child and then counter-clockwise rotation at α.

   (d) We arrived from the right child of the right child of α:
   Counter-clockwise rotation at α.

   This question is about what we do next: do we have to resume running all the way up to the root or is it necessarily the case that, once we fix the problem at node α, there cannot be any other imbalance problems remaining in the tree? You should do a few examples to get a sense of the problem and, based on what you observe from the example, make a final/general argument.

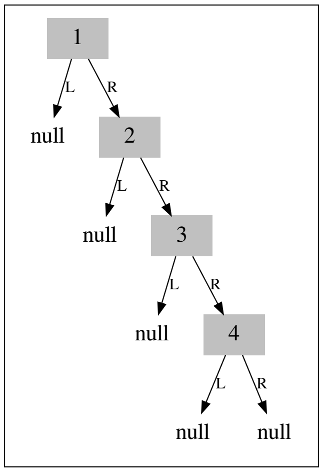


Figure 1: *A Skewed BST with as many layers as its nodes... Effectively it is a linked-list at a 45 degree inclination.*

This question is best answered with a few of the most revealing example you construct followed by a conclusion stated in clear, illustrative, non-verbose, and unambiguous terms.

4. This is the deletion counterpart of question 3. Suppose a key-value pair is removed from an AVL tree. As before, we are expected to run up from the deleted node to to the root, looking for an AVL imbalance. Suppose we find one at node α and we apply a suitable rotation to correct it. Do we have to resume running all the way up to the root or is it necessarily the case that, once we fix the problem at node α, there cannot be any other imbalance problems remaining in the tree? You should do a few examples to get a sense of the problem and, based on what you observe from the example, make a final/general argument.

   This question is best answered with a few of the most revealing examples you construct followed by a conclusion stated in clear, illustrative, non-verbose, and unambiguous terms.

5. Suppose we have a binary search tree (BST) implementation without automatic height-balancing. And suppose we keep inserting and removing randomly generated keys on an instance B of this implementation. How will the height h of B relate to the number of nodes N in B? It goes without saying that h cannot be any higher than N (see figure 1). We can also show that h cannot be any lower than $\lg N$ (see figure 2). But where exactly will h settle between these two limits? At around $\lg N$? Or N? Or somewhere in between? If we can show that h is significantly less than N, then we will know in some cases, the simplest implementation of BSTs (i.e. one that does not self-balance its heights) will be good enough.

   It is evident that the dependent[1] variable of our experiment is h. But what about the independent variables[2]? Here are some candidates:

   (a) The number of times we issue insert/remove calls.
   (b) The range of the magnitudes of the keys inserted/removed.
   (c) The specific pattern of insert/remove calls.
   (d) The degree of orderliness (or lack thereof) of the keys being inserted/removed.

   For simplicity, we will focus only on 5a and 5b. For 5c, we will assume that the insert/remove calls strictly alternate. For 5d, we will assume that the inserted/removed keys are completely random. We will also assume that our BST does not store duplicates (e.g. inserting a key value of 5 to an empty tree will have an effect on the attempt but not on the second attempt).

[1] A **dependent** variable in an experiment is a measure on which the inquiry is formulated.
[2] An **independent** variable in an experiment is an "experimental dial" that influences the value of the dependent variable(s).

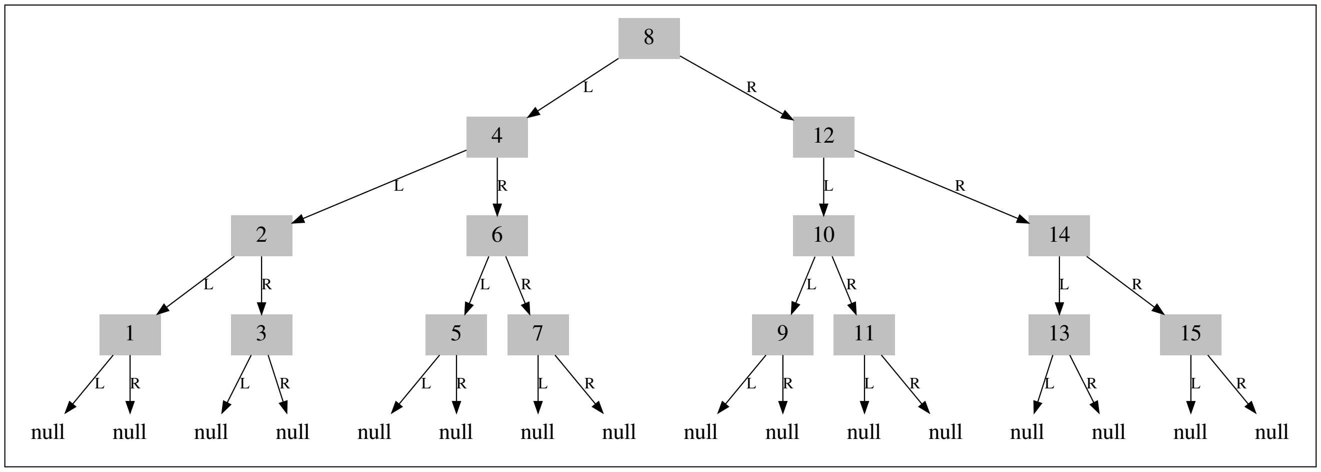


Figure 2: *A balanced BST where the number of levels is as close at it gets to the log-base-2 of the number nodes.*

This is the exploration question of the problem set and you need to do your data collection with a Java program you write. You are allowed to use the '`BST.java`' class provided by the book; you are also allowed to modify it. The ideal response will be a plot drawn from the numbers you collect and a conclusion stated in clear, illustrative, non-verbose, and unambiguous terms.