Name: Ronak Pasricha

Section: 002

Github URL: https://github.com/rpasricha45/project1CS435

Name of ALL collaborators:

URLs/ISBNs for ALL consulted websites/textbooks:

(cont:)

(cont:)

(cont:)

CS 435 – Project 1: Trees

Due Dates:
Parts 1-3 due 11:59pm, February 25th.
Parts 4-7 due 11:59pm, March 4th. Part 8 due 11:59pm, March 12th.

For this project, you will be responsible for uploading all code in a Github repository. Please print this and turn in all written answers here. **Do not turn in written code here! Code must be submitted via a link to a Github repository!**

Sresht will personally review all of the code you submit, and leave comments on what you can improve on. For Project 1, you will be required to amend your code and push it back up to address those code comments. For Project 2, you will be required to give your peers code review yourself! So please pay attention to what kinds of comments I am making in code review.

Note: For this project, you may use whatever programming language you want. However, we have implemented this project in Java, Python, and C++, so using one of those languages would make it significantly easier to get help from me and the TAs.

Part:	1	2	3	4	5	6	7	8	Total

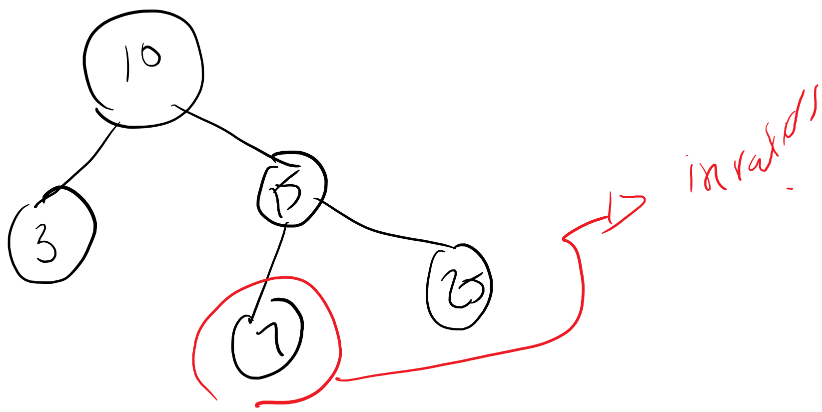
Points:	20	10	5	35	15	15	0	15	115
Score:									

1. **(20 points) Binary Search Trees**

(a) (2 points) In your own words, list the properties of a Binary Search Tree (BST).

A BSt is a tree where there are at most 2 children and the left subtree is less than the right subtree and the parent.

Notice how I mentioned the entirety of subtree b/c in Example



(b) (2 points) What are the respective asymptotic worst-case run-times of each of the following operations of a BST? Give a Θ bound if appropriate. Justify your answers. You do NOT need to do a line-by-line analysis of code.

i. Insert

ii. Delete

iii. Find-next iv. Find-prev

v. Find-min vi.

Find-max

For all let
n = number of nodes

Insert

1 find the location / where → BST log(n)
to put the node

2) insert → O(1)   O(1)+log(n)

$= \boxed{O(\log(n))}$

Delete) worst case

1) find Nodes BS (binary search)
log(n)

2) case 3 find successor() → O(h)
finding max

3) Delet the successor → O(h)

let h = hight of tree h)

worst case = O(

Find next / Find prev

They use helper functions above
so O(h)

min / max / successor / predecessor

O(N) or O(h)

where h = height of tree

(c) (8 points) (*You must submit code for part of this question!*) Use the framework below to describe how you would **recursively** implement the following methods of a Binary Search Tree. Afterwards, submit an implementation of all of the methods in your Github. Note that the Rec suffix simply means that the function is recursive.

1. insertRec
2. deleteRec
3. findNextRec
4. findPrevRec
5. findMinRec
6. findMaxRec

You must submit answers to all of the below questions for full credit!! Feel free to keep the answers as short or as long as you need to - you definitely don't need to write more than a couple of sentences for each one.

i. Repeat the question in your own words. List assumptions you make about the requirements.

ii. Enumerate edge cases that you will want to consider. In this assigment, you don't have to write code to address those as long as you call them out here. iii. Illustrate examples of input and output.

iv. Come up with an algorithm for each method. You don't need to submit anything for this part, but I strongly recommend doing this **before** starting to code.

v. Identify whether there are any issues with performance or space in your algorithm, and if so, iterate on it until it's as optimized as possible. If you're stuck here, please ask for help during Office Hours or on Slack!!!

vi. (*You must submit code for this question!*) Translate your algorithm into real code. For this exercise, please do so in an IDE and upload it to Github. Add the suffix *Rec* to all of your recursive methods. For example, your recursive implementation of insert should be called insertRec().

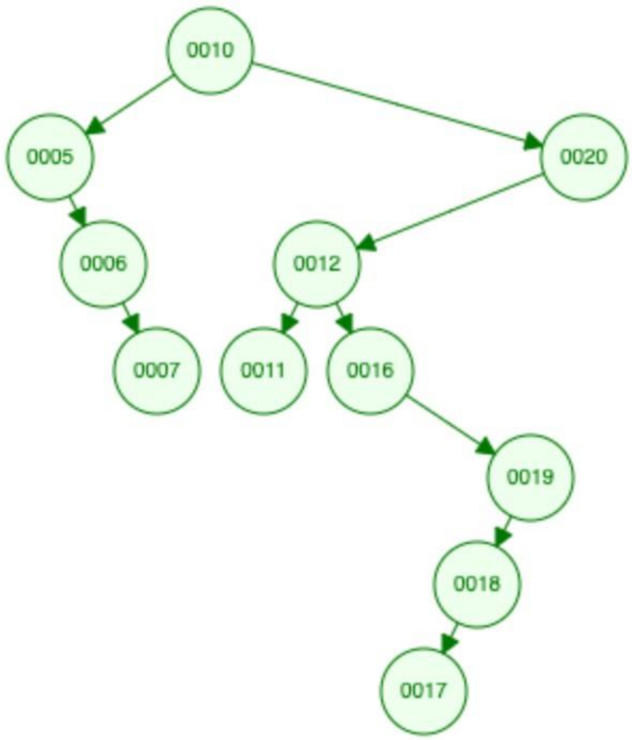
vii. What are problems/trade-offs with your current method? How might you optimize it to prevent those issues? You don't have to optimize them here, but you must enumerate them.

(d) (8 points) (*You must submit code for this question!*) Submit an implementation of the following **iterative** methods in a Binary Search Tree. You do not need to submit written answers to the framework from above, but it might be useful for you to consider the answers to those questions when writing code. Note that the Iter suffix simply means that the function is iterative. **Keep in mind that an iterative solution cannot make a single recursive call!**

1. insertIter
2. deleteIter
3. findNextIter
4. findPrevIter
5. findMinIter
6. findMaxIter

2. **(10 points) Sort It!**

(a) (1 point) In the following BST, what is the sorted order of elements, from lowest value to highest value? Write your answer as a comma-separated list.



(b) (4 points) In your own words, describe an algorithm that uses the properties of a BST to take in a list of unsorted elements and output a list of sorted elements.

[7, 6, 5, 10, 11, 12, 16, 19,
18, 17, 20]

A
Visit (left) -> Root -> Visit (Right)

Inorder Traversal

(c) (5 points) (*You must submit code for this question!*) Implement the algorithm that you described above in sort().

3. **(5 points) Arrays of Integers**

(a) (2 points) (*You must submit code for this question!*) Implement a function getRandomArray(n) where the output is an array of size n, and contains distinct random numbers (in other words, no two numbers in the array should be the same number). Math.rand() might be useful here.

(b) (3 points) (*You must submit code for this question!*) Implement a function getSortedArray(n) where the output is an array of size n. The 0^{th} element should be equal to n, the 1^{st} element should be equal to n-1, and so on.

This concludes the set of problems that must be completed and turned in by 11:59pm on Tuesday, February 25th. The rest of this project must be completed and turned in by 11:59pm on Wednesday, March 4th.

4. **(35 points) Balanced Binary Search Trees**

(a) (2 point) In your own words, list the properties of a **Balanced** Binary Search Tree (BBST). Use terminology discussed in class.

(b) (3 points) What are the respective asymptotic worst-case run-times of each of the following operations of a BBST? Give a Θ bound if appropriate. Do not forget to include the complexity of the rebalancing operation where needed. Justify your answers. You do NOT need to do a line-by-line analysis of code.

i. Insert

ii. Delete

iii. Find-next iv. Find-prev

v. Find-min vi.

Find-max (c)

(15 points)

(*You must*

submit code for

this question!)

Submit an

implementation of the following **recursive** methods in an AVL Tree. You do not need to submit written answers to the framework from above, but it might be useful for you to consider the answers to those questions when writing code. Note that the Rec suffix simply means that the function is iterative. **Keep in mind that a recursive solution must always make at least one recursive call to itself!** You will need to use your implementatio n of Node from the previous question.

1. insertRec
2. deleteRec
3. findNextRec
4. findPrevRec
5. findMinRec
6. findMaxRec

(d) (15 points) (*You must submit code for this question!*) Submit an implementation of the following **iterative** methods in an AVL Tree. You do not need to submit written answers to the framework from above, but it might be useful for you to consider the answers to those questions when writing code. Note that the Iter suffix simply means that the function is iterative. **Keep in mind that an iterative solution cannot make a single recursive call!** You will need to use your implementation of Node from the previous question.

1. insertIter
2. deleteIter
3. findNextIter
4. findPrevIter
5. findMinIter
6. findMaxIter

From here, we will prove the efficiency of a balanced binary search tree as it compares to an unbalanced binary search tree using by building trees using a list of integers.

---

5. **(15 points) Constructing Trees**

(a) (5 points) (*You must submit code for this question!*) Use your **recursive implementations** of your AVL Tree and BST from Parts 1 and 2 to construct trees using getRandomArray(10,000). Both trees must be made from the same array. In other words, **do not call the method twice - store the output of the method from** getRandomArray(10,000) **once and use it to construct both trees**.

(b) (5 points) Did you run into any issues? Test your code on a smaller input (say, getRandomArray(10), and see if you're still running into the same error. If it works on inputs of size 10 but not size 10,000, your code is probably fine and this is expected! Explain why you're running into issues (or might run into issues), using concepts we covered in class.

(c) (5 points) (*You must submit code for this question!*) Use your **iterative implementations** of your AVL Tree and BST from Parts 1 and 2 to construct trees from the input of your implementation of getRandomArray(10,000). Both trees must be made from the same array. In other words, **do not call the method twice - store the output of the method from** getRandomArray(10,000) **once and use it to construct both trees**.

6. **(15 points) Compare Implementations**

(a) (5 points) (*You must submit code for this question!*) Modify your iterative implementations of your methods in AVLTree and BinarySearchTree by keeping track of how many times you traverse one level down in the tree. In other words, if you go from a node to its child, add 1 to the counter.

(b) (5 points) (*You must submit code for this question!*) Construct a BST and AVLTree iteratively using getRandomArray(10000). Compare how many levels we have to traverse in the two trees. You can include a screenshot of your code's output or write it out by hand.

(c) (5 points) (*You must submit code for this question!*) Construct a BST and AVLTree iteratively using getSortedArray(10000). Compare how many levels we have to traverse in the two trees. You can include a screenshot of your code's output or write it out by hand.

7. **(13 points) Extra Credit**

Note: Extra credit problems require significantly more independent research and effort than the for-credit problems. They will also not off-set poor comprehension of previous parts of this project. It is strongly recommended to save these for last.

(a) (3 points) (*You must submit code for extra credit on this question!*) Use time packages in your respective language to quantify (in milliseconds/picoseconds) how much longer it takes to run 10,000 inserts and 10,000 deletes on a Binary Search Tree versus a Balanced Binary Search Tree.

(b) (10 points) *Warning: This problem will be very time consuming!!*
(*You must submit code for extra credit on this question!*) Use https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/ as a guide to learning about Red-Black trees (or R/B Trees), which are another self-balancing tree. Implement a R/B tree that supports the same features as the AVL tree you implemented, and compare run-times in milliseconds.

This concludes the set of problems that must be completed and turned in by 11:59pm on Wednesday, March 4^{th}.

8. **(15 points) Code Review**

   After submitting your code, Sresht will be individually reading and reviewing your code. Comments will be added to your code. At some point before March 12^{th}, it is your responsibility to amend your code and push up a new commit to the same repository. If you sufficiently address all points made in code review, you will receive full credit on this part of the project.

This page is intentionally blank. If you need extra space to answer any questions in this assignment, please use this page to do so. Please clearly label what problem and part you are answering if you use this page.

This page is intentionally blank. If you need extra space to answer any questions in this assignment, please use this page to do so. Please clearly label what problem and part you are answering if you use this page.

This page is intentionally blank. If you need extra space to answer any questions in this assignment, please use this page to do so. Please clearly label what problem and part you are answering if you use this page.

This page is intentionally blank. If you need extra space to answer any questions in this assignment, please use this page to do so. Please clearly label what problem and part you are answering if you use this page.

This page is intentionally blank. If you need extra space to answer any questions in this assignment, please use this page to do so. Please clearly label what problem and part you are answering if you use this page.