CMSC 270 Exam

Individual Exam 2

Name: Mohamed Khalil Ben Nasr IHRTLUC

Instructions

This is a paper computer science exam. The **only** things you should have on your desk are a pen or pencil and as much scratch paper as you desire.

Do not turn this exam over until 1:50 PM.

How to Answer Questions

For each question, write the number of the question at the top left corner of the page and circle it.

*Start a **new page** for each question, even if you have plenty of room left on the last page.*

Turn In

Turn in your exam to me. **Staple your answers, in order, to this piece of paper.**

you may include any scratch paper you want at the back of your exam in case you want to show more of your reasoning, but you should clearly label it 'scratch'.

Reminders

Here are some tips and reminders if you find yourself stuck or confused.

- **Start small.** See if there is any part of the problem where you can get a "foothold". Often writing your first instinct and going back to edit your thinking is less daunting than staring at a blank page.
- **Write it out.** Feel free to sketch things out on your paper if you feel like you know how it *should* work, but aren't sure.
- **You will have other opportunities to demonstrate your understanding of the material.** This is not your one and only chance to succeed in this class, and this isn't meant to be high-stakes.

1. The worst-case (big-Oh) time complexity for sorting via insertion sort is $O(n^2)$. The worst-case time complexity for merge sort is $O(n \log n)$. Therefore, why would you ever choose to use insertion sort over merge sort? Give at least two good reasons.

2. Is it possible to perform insertion sort on a binary search tree? If so, describe how. If not, explain why not.

3. In both a binary search tree and a red-black tree, searching for a particular element takes $O(h)$ time, where h is the height of the tree. Why, then, is it much more common to use red-black trees for efficient data structure implementations even though they are more complex?

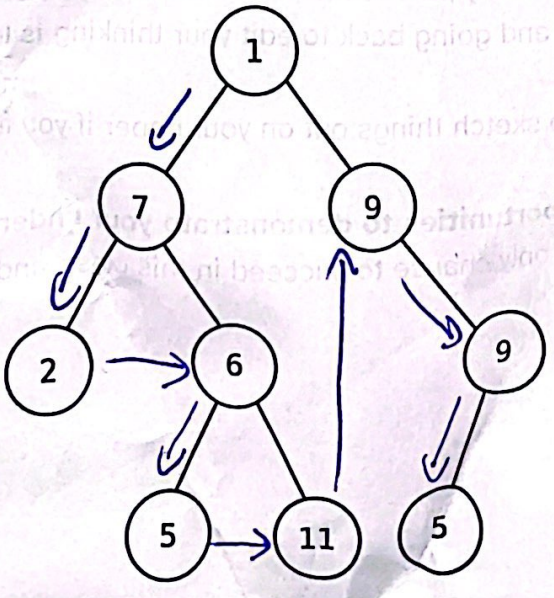
4. It is possible to traverse a binary search tree using a very simple recursive algorithm:

```
traverse(u)
    if u == nil then return
    traverse(u.left)
    traverse(u.right)
```

Is this a breadth-first or depth-first traversal? Explain.

★ **Bonus:** *Most practical applications use an iterative traversal, not recursive; why?*

5. Given the following binary tree, list the elements in order of first encounter for **(a)** a breadth-first traversal and **(b)** a depth-first traversal, starting at the root.



We would choose to use insertion sort over merge sort when:

✓ the number of elements we are sorting is small

✗ we need to do the sorting only once (merge sort is still better for single sort on large data)

However, if the number of elements we are sorting is large and/or we need to sort multiple times, merge sort is way more efficient and the best sorting algorithm to be used.

~~(2) I think it is impossible to perform insertion sort on a binary search tree. Indeed, it is extremely inefficient as it would have a time complexity of $O(n^2)$ and would make us shift the order of the nodes after every insertion sort. As a result, we always implement~~

~~I think it is impossible to perform insertion sort on a binary search tree because the elements~~

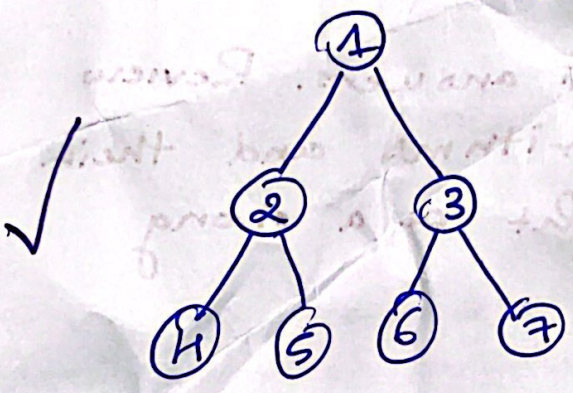
(2) I think it is not possible to perform insertion sort on a binary search tree because it is extremely inefficient as it has a time complexity of $O(n^2)$ and would require us to shift all the nodes after every insertion sort operation. It is not possible but not for the

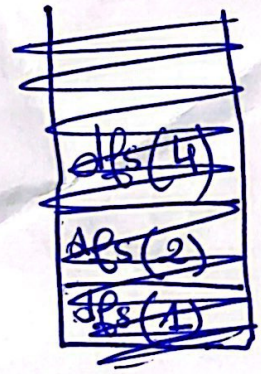
(3) I think that it is much more common to use red-black trees for efficient data structure implementations even though they are more complex because they are balanced search trees and the height of the left and right subtrees of any of their nodes differ by at most 1 which results in the height of red black trees to be closest to $\log(n+1)$ than binary-search tree. Since the search operation is dependent on that, and many oth- red-black trees become more efficient. ✓

Excellent answer.

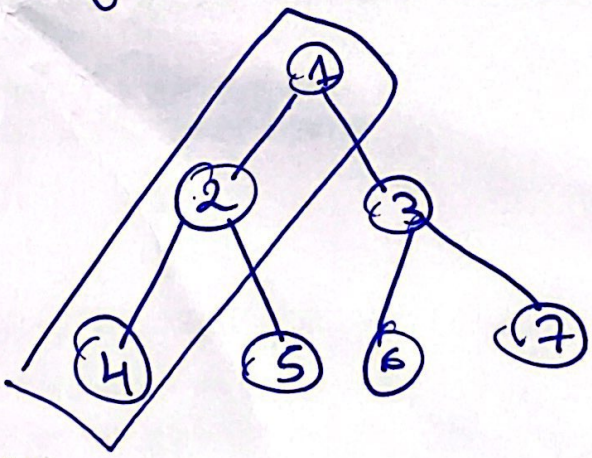
This is a depth-first traversal, because starting from the root, it traverses the left branches first, then the right branches.

Example [dfs implements a stack to traverse a binary search tree]

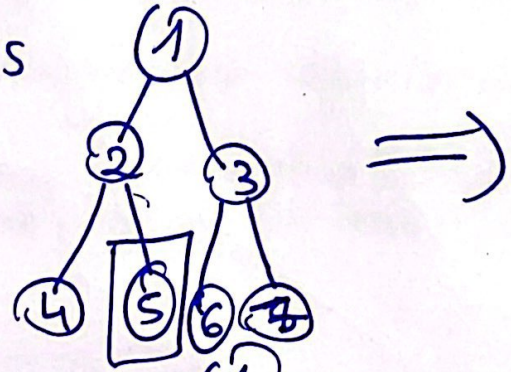




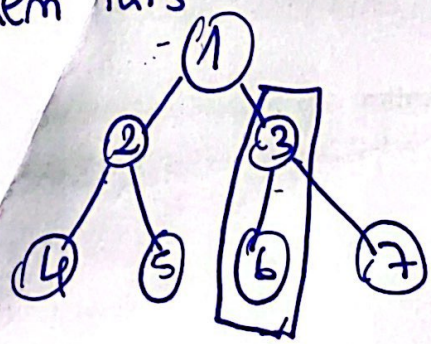
At first it visits this branch



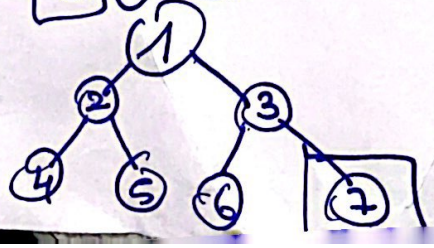
then this



then this



then this



(5)

(a) 1, 7, 9, 2, 6, 9, 5, 11, 5 ✓

(b) 1, 7, 2, 6, 5, 11, 9, 9, 5

Overall excellent answers. Review of sorting algorithms and their trade-offs might be a strong next step.