Exam 1

● Graded

Total Points

105.5 / 110 pts

Question 1

AVL Tree Insertion

12 / 12 pts

✓ **+ 2 pts** Insert 60 as left child of 66

✓ **+ 2 pts** Checked: 60, 66, 48, 68, 40
Checking 60 at the beginning and/or 40 at the end are not required.

✓ **+ 2 pts** Node 68 is imbalanced and must be rebalanced.

✓ **+ 3 pts** Left-right rotation (also OK if they say double rotation) is needed.

✓ **+ 3 pts** Rotation is done correctly.

+ 0 pts No credit

+ 0 pts Flag for cheating - if they insert 10 instead

Question 2

Move Chemical (Coding)

20 / 20 pts

- ✓ **+ 3 pts** check if source == destination and return 1
- **+ 2 pts** iterate through source correctly (using iterators)
- **+ 2 pts** iterate through destination correctly (using iterators)
- **+ 2 pts** check for name match correctly
- ✓ **+ 3 pts** Structure your code to ensure you have the iterator to the chemical in source when you try to move it to dest/targ. Grader: see additional notes.
- **+ 3 pts** use std::move correctly to invoke std::string's move semantics
- ✓ **+ 3 pts** clean up (erase or swap/pop) after move
- ✓ **+ 2 pts** return correct value
- ✓ **+ 9 pts** Correct use of std::find or std::find_if in place of rubric items 2-5
- **+ 6 pts** Mostly correct use of std::find or std::find_if in place of rubric items 2-5
- **+ 3 pts** Somewhat correct use of std::find or std::find_if in place of rubric items 2-5
- **+ 0 pts** No credit
- **− 1 pt** Minor but not trivial syntax errors
- **− 2 pts** Moderate syntax errors
- **− 3 pts** Major syntax errors
- **+ 0 pts** Flag for cheating - if they write "targ" instead of "dest"

Question 3

Vector vs List in C++

6 / 6 pts

- ✓ **+ 6 pts** 2 right examples, including: subscript notation / [] operator; .at(); O(1) advance multiple positions; difference between iterators / iterator arithmetic; comparison of iterators; resize(); capacity(); reserve(); shrink_to_fit()
- **+ 3 pts** 1 right example
- **+ 0 pts** 0 right examples

Question 4

AVL Tree of AVL Trees (Algorithm Analysis) — 25 / 25 pts

- ✓ **+ 3 pts** O(log n) complexity/operations for insert
- ✓ **+ 3 pts** But each operation includes a comparison for searching
- ✓ **+ 3 pts** In this case, each comparison takes O(log n) for findMin
- ✓ **+ 3 pts** So you must multiply the comparisons in T by the complexity of findMin in A1, A2, ..., An
- ✓ **+ 3 pts** Total O(log n * log n) = O(log^2 n)
- ✓ **+ 3 pts** Improve by tracking min for each AVL tree (this can be included in the complexity of each insert/delete without changing the complexity)
- ✓ **+ 3 pts** Improved complexity O(log n)
- ✓ **+ 4 pts** Move semantics are needed because when you insert an element of T, which is an entire AVL tree (an object), you'll probably be moving it. Otherwise, you'll be copying an entire tree every insert, which is O(n).
- **+ 2 pts** Half credit for move semantics explanation
- **+ 0 pts** No credit

Question 5

Complexities of Basic Operations — 8 / 10 pts

- **– 2 pts** Insert in splay tree (amortized): O(log n)
- **– 2 pts** Access vector (worst): O(1)
- **– 2 pts** Find in sorted linked list (average): O(n)
- **– 2 pts** Delete from AVL tree (worst): O(log n)
- ✓ **– 2 pts** Find in binary search tree (worst): O(n)
- **– 0 pts** All correct

Question 6

Move Assignment — 7.5 / 10 pts

- **+ 5 pts** a) If you don't check for self-assignment, and you delete the target before writing to it, you end up deleting and losing any dynamically allocated data members.
- ✓ **+ 2.5 pts** Half credit for above - answer almost but not completely right
- ✓ **+ 5 pts** b) If you don't delete the target before writing to it, you overwrite any pointers to dynamically allocated memory, losing access to that memory and causing a memory leak.
- **+ 2.5 pts** Half credit for above - answer almost but not completely right
- **+ 0 pts** No credit
- **+ 0 pts** Flag

Question 7

B-Trees

6 / 6 pts

✓ **+ 3 pts** B-trees are used when random access times are much greater than sequential access times, typically due to the data not being able to fit completely in memory. Examples include but are not limited to disk I/O or accessing data over a network or at a different physical location.

+ 1.5 pts Half credit for above - answer almost but not completely right

✓ **+ 3 pts** B-trees "flatten" the tree by keeping an array of pointers to intervals within the ordered data, effectively making it an m-ary tree rather than binary. This reduces the depth of the tree, trading sequential accesses through the array of intervals for random accesses required to follow a "child" pointer.

+ 1.5 pts Half credit for above - answer almost but not completely right

+ 0 pts No credit

Question 8

Remove Range (Coding)

15 / 15 pts

✓ **+ 3 pts** start at first element in range by using lower_bound.

erase each element in range and move the iterator forward by:

✓ **+ 6 pts** by using return value of .erase() - most efficient

+ 4 pts by using a 2nd iterator so one keeps the place of the element to be erased while the other moves forward (make sure they actually implement this correctly)

+ 3 pts by repeatedly calling lower_bound again after each erase – inefficient

+ 2 pts by using the return value of .erase() and also incrementing the iterator - this will skip values.

+ 0 pts By trying to increment the iterator after erasing it - this won't work.

+ 0 pts No credit

✓ **+ 3 pts** stop iterating when you reach a value > y or end of set.

✓ **+ 3 pts** return correct count.

+ 0 pts No credit

− 1 pt Minor but not trivial syntax errors

− 2 pts Moderate syntax errors

− 3 pts Major syntax errors

Question 9

Asymptotic Complexity Classes

6 / 6 pts

+ 3 pts Θ(n)

✓ **+ 3 pts** Θ(n log n)

+ 3 pts Θ(n $\sqrt{n}$)

✓ **+ 3 pts** Θ(n log log n)

+ 3 pts Θ(n log* n)

+ 3 pts Other correct answer that is not on the order of any of the above

+ 0 pts No correct answers

CSCI 335 Exam 1
Fall 2024

Version A

Instructions:

Write your name and EmplID clearly.

When you receive this exam, check to make sure anyone sitting immediately adjacent to you has a different version. If you have the same version as a neighbor, notify a TA.

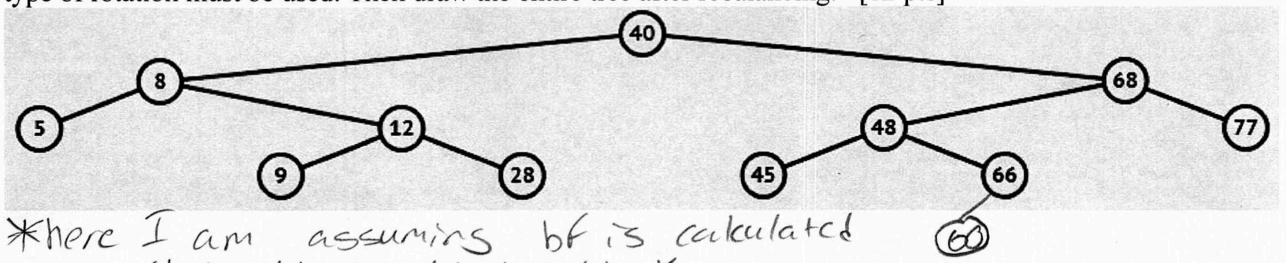
All phones, earbuds, watches, and other electronic devices must be put away in your bag.

If you're found to be in violation of these rules, or are otherwise observed trying to exchange information with another student or read their answers, your name will be taken and your exam will be carefully reviewed for signs of academic dishonesty.

Do not open this exam until instructed to do so.

Note: this exam has a total of 110 points, though the maximum score will be capped at 100.

1) Insert 60 into this AVL tree (add it on the diagram), then rebalance if necessary. List all nodes checked for balance in the order they're checked. If rebalancing is necessary, state which node must be rebalanced and what type of rotation must be used. Then draw the entire tree after rebalancing. [12 pts]



here I am assuming bf is calculated by left-height - right-height.

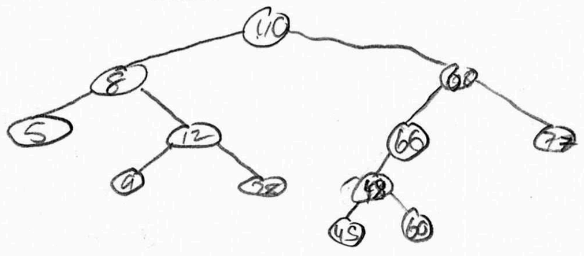
66 checked -> bf = 1 - 0 = 1 ✓

48 checked -> bf = 1 - 2 = -1 ✓

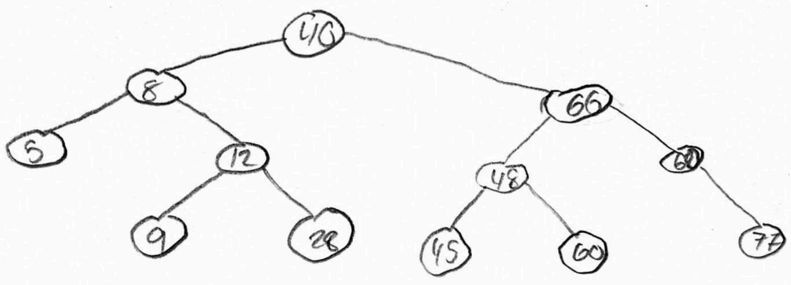
68 checked -> bf = 3 - 1 = 2 ✗ -> 68 has left-heavy sub
48 has right-heavy subtr.

68 needs to be rebalanced with a double rotation, specifically a left-right rotation.

Rotation 1



Rotation 2



2) Suppose you have two lists of chemicals – for example, an "approved" list and a "banned" list. The lists are implemented as vectors, and the chemicals as strings. You want to write a function that can move a given chemical from one list to the other.

Implement the following function that moves *chemical* from *source* to *dest* using efficient move semantics. If *chemical* is found in *source* and not found in *dest*, it is moved and 1 is returned. Otherwise, nothing is moved and 0 is returned. If *source* and *dest* are the same, 1 is returned. Iterators must be used wherever appropriate. [20 pts]

shouldn't these be references?

```
bool moveChemical (const std::string& chemical, std::vector<std::string> source, std::vector<std::string> dest){
    if ( source == dest) return 1;
    auto it_source = std::find(source.begin(), source.end(),
                                            chemical);
    auto it_dest = std::find(dest.begin(), dest.end(),
                                            chemical);
    if(it_source == source.end() || it_dest != dest.end() ) {
        return 0;
    }
    std::swap(*it_source, *source.rbegin());
    dest.push_back(std::move(*source.rbegin());
    source.pop_back();
    return 1;

}
```

3) List 2 things a C++ Vector or Vector iterator can do that a C++ List or List iterator can't. Your answers must mention some method, syntax, etc. that C++ Vectors use, not just general properties of vectors. [6 pts]

A vector iterator is random access while a list iterator is bidirectional. This means you can do iterator arithmetic with vector iterators such as it - 3; but only it--; with list iterators. Also, you can see if one vector iterator is greater than another while you can only see if two list iterators are equal or not

4) Suppose you have an AVL tree, T, whose elements are themselves AVL trees, $A_1, A_2, \ldots, A_n$. The elements of $A_1, A_2, \ldots, A_n$ are integers, and $A_1, A_2, \ldots, A_n$ are ordered within T by their own min elements.

What is the big-O complexity of inserting a new element into T? Write out a full complexity analysis. What modification to AVL trees could be made to improve the complexity, and what would the resulting complexity be? Assume efficient move semantics in your answer, but also briefly explain how a lack of move semantics might affect the algorithm and complexity. [25 pts]

To start, overload the AVL comparison operators such that it returns the correct bool based on the min elements of the AVL trees. The function to get the min element of an AVL will keep traversing left until the left child is null, in which case, it will return the current element. Because we are traversing down a path, worst case time is O(log n). Therefore, comparisons are O(log n). Now, inserting into an AVL tree is normally O(log n), but since comparisons are are O(log n), the overall complexity of inserting to T is now $O(\log^2 n)$.

A good modification would be changing insert & delete of the AVL trees such that there is always a pointer to the minimum element. This would make comparisons O(1) and reduce the complexity of inserting to T to O(log n).

We have assumed efficient move semantics thus far, but if we didn't have move semantics, then we would need to deep copy the whole AVL tree when inserting to T. This will add O(n) to overall complexity making it $O(\log^2 n + n) = O(n)$.

5) Give the time complexity of each, in big O notation, with a tight bound. Assume it's a single operation (e.g. insert/delete/find 1 element) on a data structure of size n. Find is also known as "lookup" or "search". [10 pts]

Insert in a splay tree (amortized worst case)	O(log n)
Access in a vector (worst case)	O(1)
Find in a sorted linked list (average case)	O(n)
Delete from an AVL tree (worst case)	O(log n)
Find in a binary search tree (worst case)	O(log n)

6) When writing a move assignment operator according to best practices, two things you need to do include: a) check to make sure you're not performing a self-assignment, and b) delete the target before writing to it. What happens in each case if you don't perform that action? [10 pts]

If you don't do (a), then when you delete the target, you won't have anything to write from (if you are doing self-assignment). If you don't do (b), then when you overwrite the target, the previous memory target pointed to will still be in memory with no real means of being deleted.

7) In 3-5 sentences, explain the type of situation B-trees are used for, and how B-trees are designed to improve performance. [6 pts]

B-trees are designed to limit the amount of fetching the CPU has to do to memory in exchange for an increase in average number of comparisons executed. Since each node has more than one element in it, the height of a B-tree is significantly less than a regular binary-tree, thus limiting the number of fetches on average for find, insert, delete, etc. Of course, this means the CPU has to perform more comparisons, but that ok since comparisons are fast, & fetching is not as fast.

8) Complete the function removeRange so that it efficiently removes all integers with values between x and y from a given set S. It should return the number of elements removed. It should be as efficient as possible.

References for std::set::lower_bound and std::set::erase are included below, and you should use those methods in removeRange. You may assume all necessary libraries are included. [15 pts]

```
// Returns an iterator pointing to the first element in the container which is not considered to go before val
// (i.e., either it is equivalent or goes after). Returns set::end if all elements are considered to go before val.
iterator lower_bound (const value_type& val);

// Removes the element at position from the set container. Returns an iterator to the element that follows the
// last element removed (or set::end, if the last element was removed).
iterator erase (const_iterator position);

// removes all values from x to y, inclusive, from S.
// precondition: x ≤ y.
// returns the number of elements removed.
int removeRange (std::set<int>& S, int x, int y){
    auto it = S.lower-bound(x);
    int res = 0;
    while (it != S.end() && *it <= y){
        it = S.erase(it);
        res ++;
    }
    return res;
}
```

9) Give an example of two mathematical functions, f(n) and g(n), such that: [6 pts]

- $f(n), g(n) \in o(n^2)$
- $f(n), g(n) \in \Omega(n)$
- $f(n) \notin \Theta(g(n))$
- neither function contains an exponent.

$f(n) = n \log\log n$

$g(n) = n \log n$