1. BST/AVL Tree (25 pts; 5+5+7+8)

For a) and b):

Consider two binary search trees A and B. (These are NOT AVL trees, just plain binary search trees.) BST A has m nodes, and BST B has n nodes. Assume neither tree has duplicates. We wish to find the common items in A and B. For each of the following, derive the running time as asked, with reasoning. No reasoning = no credit.

a) We will perform an inorder traversal of A, and for each item encountered, we will perform a search in B. What is the **worst-case** big O running time of this approach?

8 -0

It would be $O(m \cdot n)$

The worst case would be that every node in A would have to compare with every node of BST B. BST A has m nodes and BST B has n nodes. so if BST A has to check every node of BST B then be $O(m \cdot n)$

b) We will perform an inorder traversal of A, appending the items to an output array as they are encountered. We will do the same for B, appending to a second output array. Assume there is enough space in each array to hold all items that are appended. We will then find the common items in these two output arrays. What is the **worst-case** big O running time?

It would be $O(m \cdot n)$

$O(m+n)$

If both BST are put into a array being traversed in older then out put array size would be M while output2 would be ~~size~~ length n. we were to assume that neither array had any ~~it~~ item in common it would force one array to go through the other array its own array length times so ~~to~~ it is $O(m \cdot n)$

-3

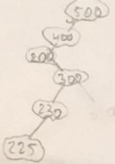
~~4~~

c) Suppose a BST (not AVL) stores some integers in the range 1 to 500. We will perform a search for the value 225, and keep a list of the values encountered on the search path through the BST. For each of the following sequences of values, say whether or not it is a possible search path. If yes, show the actual path (with branches), if not, explain why.

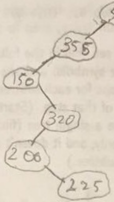
i) 500, 400, 200, 300, 230, 350, 225 yes it is Possible

No.

-2



ii) 500, 355, 150, 320, 200, 225 yes. it is Possible.

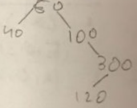


-0 ✓

iii) 50, 100, 300, 40, 120 ✓

no it is not a Possible Search Path because the number does not exist. and the search fails after it look at the right ~~[illegible]~~ Subtree of 120 and finds null.

∴ it was a failed search



d) Words are read from an input file, converted to lowercase, and inserted into an AVL tree one word at a time. Each node of the tree stores a word, along with a count of its occurrences in the file, and the tree is ordered alphabetically by words. If there are k distinct words, and n words in all in the file, what would be the worst case big O time to store all the words in the tree? Count word comparisons (each is unit time) ONLY. Show your work.

we have n words and k distinct words. n - k = common words that have occurrences already in the AVL tree.

to insert a word into ~~a~~ Avl it takes O log(n)

It would take O(log n) to store all the words,

✗

-4

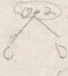
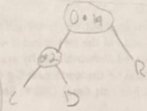
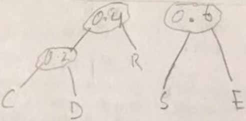
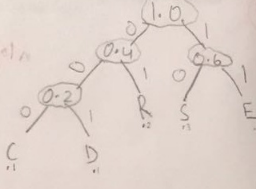
n log k.

2. **Huffman Coding (25 pts, 10+10+5)**

Given the following set of character-probability pairs:

(C,0.1), (D,0.1), (R,0.2), (S,0.3), (E,0.3)

(a) Build a Huffman tree for this character set. Fill in the following table to show the queue L, which will start with the leaf nodes for the symbols, and the queue T, which will contain the subtrees as they are built. Draw the tree shape for each subtree in T. Each row of the table must show the contents of the queues at the end of that step. (Start by filling in the queue L contents in the first line.) The last step should have a single tree (final Huffman tree) in the queue T. (Ties in probability values are broken arbitrarily, and it doesn't matter which dequeued node goes left and which goes right when building a subtree.)

Step	Queue L	Queue T
1	(C,0.1),(D,0.1),(R,0.2),(S,0.3) (E,0.3)	Empty
2	(R,0.2)(S,0.3)(E,0.3)	
3	(S,0.3),(E,0.3)	
4	Empty	
5	Empty	 1.0 0 1 0.4 0.6 0 1 0 1 0.2 R .2 S .3 E .3 0 1 C .1 D .1

+10

b) Assume that enqueue, dequeue, creating a leaf node, creating a new tree out of two subtrees, and picking the minimum of two probabilities all take unit time. Ignore the time for all other operations. How many total units of time (exact number, not big O) did it take to build your tree in part (a)? Show your work.

	enque	deque	new leaf	new tree out of	two smallest	total
1	5	0	0	0	0	5
2	1	2		0	1	4
3	1	2	1	0	1	5
4	1	2	0	0	1	3
5	1	2	0	1	1	5

+7

7 5 3

c) (This part is not related to the specific example of parts (a) and (b).)
Suppose a character string of length n is encoded to k bits using Huffman coding. Consider decoding this back to the original character string. Briefly describe the decoding process. How many units of time (NOT big O) would the decoding take? Derive your answer, starting with specifying what unit time operation(s) you are counting.

It word take big ~~[illegible]~~)

~~A unit~~

n units of time

0 1 0 1 0 /
1 2 3 4 5 k

+1

(n+k) units of time.

3. **Hash Table (25 pts;7+10+8)**

You are given the following classes:

```
class LLNode {
    String key; String value;
    int hashCode; LLNode next;
    LLNode(String k, String v,
           int h, LLNode nxt) {
        key=k; value=v;
        hashCode=h; next=nxt;
    }
}
```

```
class Hashtable {
    LLNode[] table;
    int numValues;
    float loadFactorThreshold;
    Hashtable(float loadFactorThreshold) {...}
}
```

a) Implement a method in the `Hashtable` class to insert a key-value pair into the hash table, using the function h **mod** N to map a hash code h to a table location. N is table size (capacity):

```
// inserts (key,value) into hash table,
// calls rehash method (part b) if load factor threshold is exceeded
// Note: String class implements the hashCode method
public void insert(String key, String value) {
LLNade[] X = new LLNode();
 X. Key = key;
 X. value = value;
 int index = Integer.ParseInt(X.Key) %
```

2

b) Also implement a `rehash` method, which doubles the table size when expanding it. Your implementation MUST NOT end up creating any new nodes–it should ONLY recycle the nodes already in the table.

```
private void rehash() {
```

0

c) Suppose you insert 125 integer keys into a hash table with an initial capacity of 25 and a load factor threshold of 2. The hash code is the key itself, and the function key **mod** table_capacity is used to map a key to a table position. Derive the total units of time that will be used to insert all keys, ONLY counting one unit of time each to do the mapping, insert an entry into a linked list, and check load factor against threshold. Assume a rehash doubles the table capacity, and the load factor is checked AFTER an entry is mapped and inserted into a chain. Show work.

50 + 25 · 2 = 50

50 entries until load factor met.

8 more 50 insert 50

100 + 100 +

6

150 + 150 + 75

$= \boxed{375 \text{ units of work.}}$