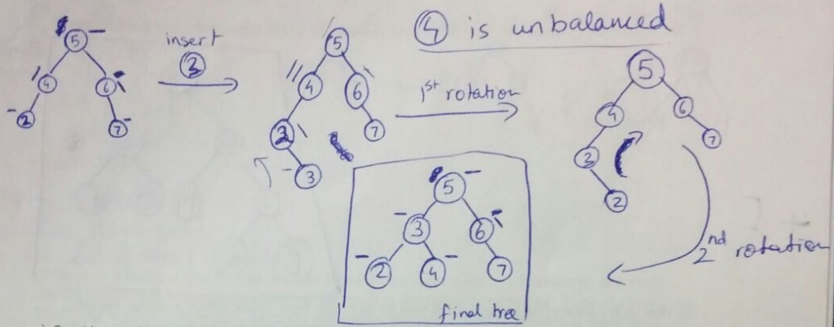
b) Draw a 5-node AVL tree with integers at the nodes, to which adding an integer results in the root being unbalanced, needing 2 rotations to rebalance. Show the original tree with integers in the nodes, mark the balance factors at the nodes, show where the insertion is done, fix balance factors as needed. Then apply the rotations at the root, one rotation at a time, and show the final tree with integers and balance factors.



+8

c) Consider two binary search trees A and B. (They are NOT AVL trees, just plain BSTs.) BST A has x nodes, and BST B has y nodes. Assume there are no duplicates in the entire set of items in A and B. Describe the fastest algorithm to output the combined set of items in A and B in sorted (ascending) order. You can use additional array space if you need to, but no other data structure. Derive the worst case big O running time of your algorithm. (If your algorithm is not the fastest, it will get at most 3 points.)

Algorithm: do inorder traversal of both and store their elements in two seperate arrays. ~~compare their individual elements one by one.~~ Sort them using merge sort and output the results.

+6

Time complexity ⇒ traversals = $O(x) + O(y) = \cancel{O(x)}O(\max(x, y))$

merge sort = ~~$O(\max(x\log y,$~~

$O((x+y)\log(x+y))$

Therefore worst case = $O((x+y)\log(x+y))$

$O(x+y)$

25

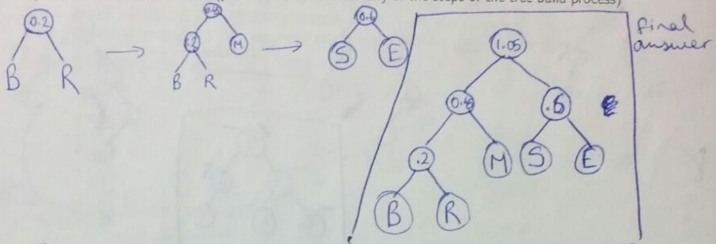
2. **Huffman Coding (25 pts, (5+7)+5+8)**

Show your work for each of the following. An answer without a clear sequence of derivation steps will not get any credit.

a) Given the following set of character-probability pairs:

```
(B,0.07), (R,0.13), (M,0.25), (S,0.27), (E,0.33)
```

i) Show the final Huffman tree (you don't need to show any of the steps of the tree build process)



+5

ii) How many total units of time did it take to build this tree? Count ONLY enqueue, dequeue, creating a leaf node, creating a new tree out of two subtrees, and picking the minimum of two probabilities. (Ignore all other operations) Each of these takes unit time. All queues are initially empty.

from the point where every node has been enqueued in Queue A	Step 1	2	3	4
Enqueue	1	1	1	1
Dequeue	2	2	2	2
create node	0	0	0	0
tree	1	1	1	1
min	0 (since selection is from arranged queue)	1 (comparing M and 0.2 only M is dequeued from arranged queue)	2 (S & E compared to 0.45)	1
	5	6	7	5

+7

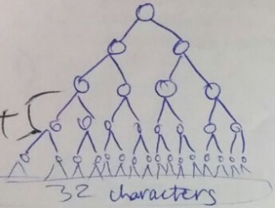
creating leaf nodes = 5

Enqueue them = 5

total ⇒ 5 + 5 + 5 + 6 + 5 + 7 − 4 + 1

= <u>30</u>

b) Given a document with 32 distinct characters, with equal number of occurences of each, what would be the ratio of the length of the Huffman coded document to the original, if each character in the input was coded in 8 bits? Show your analysis.



5 bits for each character since 5 branches lead to leaf nodes

Therefore ratio = $\boxed{\frac{5}{8}}$

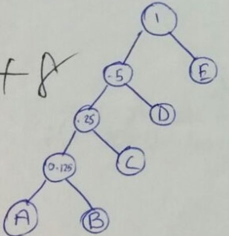
Since they have equal probabilities, they would be at the end of tree (leaf nodes)

c) Suppose a Huffman tree is to be built for 5 characters. Give a set of 5 characters, and their distinct probabilities (no duplicates), that would result in the tallest possible tree. Derive the average code length: write the expression, you don't have to simplify it down to a single value.

0.0625	0.0625	0.125	0.25	0.5
A	B	C	D	E

= $\boxed{1}$

+8



avg code length.

$1(0.5) + 2(0.25) + 3(0.125) + 2\cdot 4(0.0625)$

15

3. **BST k-th Smallest (25 pts; 15+10)**

Suppose each binary search tree node is modified by adding a field that stores the number of nodes in its LEFT subtree. This modification is useful to answer the question: what is the k-th SMALLEST element in the binary search tree? ($k = 1$ means smallest element, $k = 2$ means second smallest element, etc.)

You are given the following enhanced BSTNode class definition:

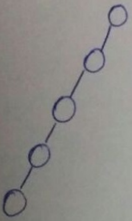
```
public class BSTNode<T extends Comparable<T>> {
    T data; BSTNode<T> left, right;
    int leftSize;  // number of nodes in left subtree
    ...
}
```

(a) Implement the following recursive method to return the k-th smallest element in a given enhanced BST. The method should throw a NoSuchElementException if k is out of range. You may implement helper methods if necessary.

```
public static <T extends Comparable<T>>
T kthSmallest(BSTNode<T> root, int k) throws NoSuchElementException {
    if ( root == null || k <= 0)
        throw new NoSuchElementException();

+15  if ( root.leftSize == k-1)
        return root;

    if ( root.leftSize >=k)
  return kthSmallest ( root.left , k-1);

    return kthSmallest (root.right , k - root.leftSize - 1);

}
```

(b) What is the *worst* case number of units of running time (not big O) for your implementation, for a tree with n nodes? Derive as follows. First, for each possible worst case scenario, draw the tree shape and specify the associated k value. Second, state the basic unit time operations you are counting (ignore time for pointer checks and assignments). Third, show how many times each of these operations are done for any ONE of the worst case scenarios. Lastly, add up the operations to get a number that is a function in either n or k, or both.

1)



K	~~time~~
1	~~5~~
2	[illegible]
3	[illegible]
4	[illegible]
5	

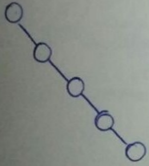
(ii) counting comparisons of k with root. leftSize
for k = 1, number of comparisons highest, therefore worstcase.

(iii) 2 comparisons for each upto last and 1 comparison for last

$\therefore \; 2(n-1)+1 = \boxed{O(n)}$

+10

2)



(ii) same as ① but in this case k = n

(iii) 2 comparisons for each upto last and 1 comparison for last

$\therefore \; 2(n-1)+1 = \boxed{O(n)}$

worst case = $f(n) = 2(n-1)+1$