1. (30 points) (Pessimistic Patricia Tries). Assume lower case Roman alphabet a..z, parent pointers, and the find() method as in the slides (page 7–8).

(a) Implement remove(String x, TrieNode p) in time proportional to length of x.

(b) Show pictures similar to the one for insert when the strings bull, stop, buy, bid, bell, bear, sell, and stock are removed in order.
2. (Optional) Give an efficient implementation of \texttt{restructure(x)} presented in \texttt{rebalance(z)}. Contrast it with the implementation discussed in class.

3. (20 points) (Vanilla BST). The notion of ‘average’ on page 12 is different from the notion of “expected” on page 13.

   (a) Prove the above statement using the first four odd numbers.

   (b) Which notion do you prefer? Why?
4. (20 points) (AVL removal) Suppose \( z \) is an unbalanced node with right child \( a \) and left child \( y \), and the removal happened in the subtree rooted at \( a \). Also assume that the two children of \( y \) are equally tall. Does it matter which child we pick as \( x \) for \( \text{restructure}(x) \) to balance \( z \)? Explain.

5. (30 points) (External Binary Search Tree) An eBST might be useful to support \( \text{rangeSearch}(a, b) \): Print ages of all employees in the interval \([a, b]\). Recall that we defined a BST with no data in the placeholder external nodes. An eBST

- Has all the \( n \) user data only in the external (leaf) nodes. Otherwise, it is similar to a BST.
- The right child pointer in external nodes are used to point to the next (in order) data (if any).
- Duplicate keys are permitted (the ‘satellite’ data such as names are expected to be different).
- Every node contains a Boolean instance variable \texttt{leaf} which is \texttt{false} for internal nodes

(a) Use \texttt{System.out.println} to complete in \( O(\log n + s) \) time (\( s \) is the size of the output)

```java
void rangeSearch(int a, int b, Node p) {
```
(b) Use a constructor of the form `Node(Key k, Node toRight, Boolean iAmLeaf)` to complete in $O(\log n)$ time

```java
Node insert(Key k, Node p) {
```

(c) (Optional) Implement `remove(Key k, Node p)`

(d) Draw four figures to show the execution of your insert code that generates the figure above.