1. Provide solutions (using big-Oh or big-Theta) for the following recurrence relations.

   (a) \( T(n) = 8 T\left(\frac{n}{2}\right) + n^2 \)
   (b) \( T(n) = 25 T\left(\frac{n}{5}\right) + n^2 \)
   (c) \( T(n) = 5 T\left(\frac{n}{2}\right) + (n \log n)^2 \)
   (d) \( T(n) = 25 T\left(\frac{n}{5}\right) + n^{\log_5 30} \)

(Sol’n)

(a) Here \( f(n) = n^2 \) and \( n^{\log_a b} = n^{\log_2 8} = n^3 \). This is case 3 of the master method, so \( T(n) = \Theta(n^3) \).

(b) And here \( f(n) = n^2 \) and \( n^{\log_a b} = n^{\log_5 25} = n^2 \), so by case 2 \( T(n) = \Theta(n^2 \log n) \).

(c) \( f(n) = (n \log n)^2 \) and \( n^{\log_a b} = n^{\log_2 5} \), so by case 1 (noting that \( \log_2 5 > 2 \)), \( T(n) = \Theta(n^{\log_2 5}) \).

(d) \( f(n) = n^{\log_5 30} \) and \( n^{\log_a b} = n^{\log_5 25} = n^2 \), by case 3 (since obviously \( \log_5 30 > \log_5 25 \)) we get \( T(n) = \Theta(n^{\log_5 30}) \).

2. Into an initially empty AVL tree, insert the following values:

   1, 2, 3, 4, 5, 6, 7, 12, 11, 10.

3. Insert the values above into an initially empty 2-3-4 tree.

4. Determine the run-time of the following two segments of pseudo-code, using big-Oh notation.

   (a) for i=1 to n*n
       for j=1 to i*i
           sum++
    
   (b) for i=1 to n*n*n
       j=1
       while (j<i) {
           sum++
           j=3*j
       }

(Sol’n)
(a) Since \( i \) can get as large as \( n^2 \), the inner loop runs for as long as \( (n^2)^2 = n^4 \) steps. The total is thus at most \( n^2 \cdot n^4 \) - in other words the runtime is \( O(n^6) \). (And in fact it is \( \Theta(n^6) \).)

(b) The inner loop uses time \( \Theta(\log_3 i) \). Since \( i \leq n^3 \), the total time is \( O(n^3 \log_3(n^3)) \). As has been mentioned, this simplifies to \( O(n^3 \log n) \).

\( \square \)

5. Given a BST \( T \) with \( n \) nodes, write a small piece of pseudo-code to determine the \textit{external} path length of \( T \). The children of each node are called \textit{lchild} and \textit{rchild}, and external nodes are indicated by \textit{null} pointers.

- Recall that the external path length \( E \) is the sum of the depths of all the external nodes.
- Think of a recursive routine that determines the depth of each node and modify it.
- Aim for \( O(n) \) time.

\( (Sol'n) \)

\[
calcE(node \ p, \ int \ depth) \ returns \ int
\{
\begin{align*}
  & \text{if} \ (p==\text{null}) \ \text{then return depth} \\
  & \text{else} \\
  & \ \ \ \ \ return \ calcE(p.lchild, \ depth+1) + calcE(p.rchild, \ depth+1)
\end{align*}
\}
\]

The initial call should be \( \text{calcE}(T.root, \ 0) \). This is \( O(n) \) since it is essentially a postorder traversal.

\( \square \)