SAMPLE SOLUTION TO MIDTERM TEST

1. Let \( f(n) = 14n^4 + 5n^3 + 2n \log n + 15 \). Indicate whether each of the following statements is true or false.

(a) \( f(n) = O(2^n) \)
(b) \( f(n) = O(n^4) \)
(c) \( f(n) = o(n^4) \)
(d) \( f(n) = 14n^4 + o(n^4) \)
(e) \( f(n) = \Theta(n^3) \)
(f) \( f(n) = \Omega(n^3) \)

(Sol’n)

(a) true
(b) true
(c) false
(d) true
(e) false
(f) true

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

\[ 10, 4, 12, 3, 2, 5, 6, 7, 8, 9. \]

(Sol’n)
See figure 1 (at bottom) for the final tree.

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

(Sol’n)
See figure 2 for the final tree.
4. Into an initially empty red-black tree, insert the following values:

\[ 12, 8, 5, 10, 15, 17. \]

\[(Sol'n)\]

See figure 3 for the final tree.

\[\square\]

5. What are the run-times of the following pieces of code?

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

(b) for i = 1 to n
    for j = 1 to i*i
        sum++

\[(Sol'n)\] For part (a), the inner loop takes time (roughly) \(\log_3 i \leq \log_3 n^2 = 2\log_3 n = O(\log_3 n) = O(\log n)\). The total is therefore \(O(n^2\log n)\).

Part (b) is pretty clearly \(O(n^3)\).

\[\square\]

6. Suppose we have a BST \(T\), with root \(T.root\), and \(n\) nodes. Each node \(p\) of the tree has fields \(p.lchild\), \(p.rchild\), \(p.value\), and \(p.bf\), where the last stands for balance factor. This balance factor of node \(p\) is defined to be \(\text{height}(p.lchild)-\text{height}(p.rchild)\). All the fields of each node have been filled except for this balance factor, which is empty.

Write a \(\Theta(n)\) time method (in pseudo-code) which will calculate the balance factors of all the nodes of the tree, and store the balance factor of node \(p\) in the \(p.bf\) field.

You do not have access to a height function (\textit{hint}: write one and modify it). It is possible the tree is not an AVL tree - you do not need to check whether it is.

\[(Sol'n)\] In class we saw a simple \(\Theta(n)\) height function (initial call \(\text{calcHeight}(T.root)\)):

\begin{verbatim}
procedure calcHeight(node p) returns int
    if p=null return -1
    else {
        leftHt = calcHeight(p.lchild)
        rightHt = calcHeight(p.rchild)

        return 1 + max(leftHt, rightHt)
    }
\end{verbatim}

One approach is to use this to fill in all the balance factors (initial call \(\text{fillBalanceFactors}(T.root)\)):
procedure fillBalanceFactors(node p)
    if p != nil
        p.bf = calcHeight(p.lchild) - calcHeight(p.rchild)
        fillBalanceFactors(p.lchild)
        fillBalanceFactors(p.rchild)

While this works, the problem is that it makes repeated height calls, and is potentially $O(n^2)$
time. Better (also as mentioned in class) would be to modify the height function:

procedure calcHeightAndBF(node p) returns int
    if p=null return -1
    else {
        leftHt = calcHeightAndBF(p.lchild)
        rightHt = calcHeightAndBF(p.rchild)

        p.bf = leftHt - rightHt

        return 1 + max(leftHt, rightHt)
    }

Figure 1: The final AVL tree of problem 2.
Figure 2: The final 2-3-4 tree of problem 3.

Figure 3: The final red-black tree of problem 4.