MIDTERM SAMPLE SOLUTION

1. Provide solutions (using big-Oh or big-Theta) for the following recurrence relations.
   (a) $T(n) = 7 \cdot T(\frac{n}{3}) + n \lg n$
   (b) $T(n) = 314 \cdot T(\frac{n}{313}) + 2^{315} \cdot n$
   (c) $T(n) = 25 \cdot T(\frac{n}{5}) + n^2$

   {sol'n} The answers are $\Theta(n^{\log_7 7})$, $\Theta(n^{\log_{313} 314})$, and $\Theta(n^2 \lg n)$, respectively.

2. Into an initially empty AVL tree, insert the following values:
   62, 47, 32, 15, 26, 30, 27, 28, 29, 10, 5.

   {sol'n} See the attached graphics below.

3. Insert the values above into an initially empty 2-3-4 tree.
   {sol'n} See the attached graphics below.

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

   (a) for $i = n$ downto 1 {
       j = i
       while (j>=1) {
         sum++
         j=j/313
       }
   }

   (b) for $i = 1$ to $n^3$
       for $j = 1$ to $i \cdot n$
         sum++

   {sol'n} The first piece of code is $\Theta(n \lg n)$ - note that the inner loop runs for $\log_{313} i = \Theta(\lg i)$ steps.

   In part (b), the maximum value of $i$ is $n^3$, and so the inner loop runs for at most $n^4$ steps. With the outer loop running $n^3$ times, the maximum total is $O(n^3 \cdot n^4) = O(n^7)$.

   A more accurate accounting would be
   \[
   \sum_{i=1}^{n^3} i \cdot n = n \cdot \sum_{i=1}^{n^3} i = n \cdot \frac{n^3(n^3 - 1)}{2} = \frac{n^7 - n^4}{2} = \Theta(n^7).
   \]
5. Write a recursive routine which, given an integer \( k \), prints the keys of all nodes at height \( k \). They can be printed in any order. The fields of each node are called key, lchild, and rchild.

\{sol’n\} This is similar to the balance factor problem of HW3 - in this case rather than calculating the balance factor, we compare our height to \( k \). The initial call should be `getHeights(T.root, k)`.

```
procedure getHeights(node p, int k) returns int

    if (p==null) return -1

    lHeight = getHeights(p.lchild, k)
    rHeight = getHeights(p.rchild, k)
    thisHeight = max(lHeight, rHeight) + 1

    if (thisHeight==k)
        print p.key

    return thisHeight
```

The idea is to calculate the height of each node - get the heights of the left and right children, add one to the maximum of those two values. At this point, we compare our height to \( k \), and print the key if equal. We will necessarily be doing a postorder traversal since we can’t know the height of the current node before we know the heights of the children.
**Question 2:** build AVL tree

After the insertion of 26:

![AVL tree diagram](attachment:avl_tree_26.png)

After 30:

![AVL tree diagram](attachment:avl_tree_30.png)

After 29:

![AVL tree diagram](attachment:avl_tree_29.png)
After 10

And the final tree
Question 3: *build 2-3-4 tree*

The final tree, using the bottom-up technique described in class, is