Instructions:
Put your assignment in an envelope labeled with your name and course number and drop it in the CS2210 locker # 310 by the midnight on the due date.

1. (20 marks) Consider the algorithm $Multiply(A, n)$:

   ![Algorithm Multiply(A,n)](image)

   (a) (5 marks) Write down recurrence equations that describe the running time of algorithm $Multiply(A, n)$.

   **Solution:** $T(3) = k$, $T(n) = c + T(n - 3)$.

   (b) (15 marks) Solve these recurrence equations and give the best asymptotic (big-O) characterization of the time complexity of the algorithm.

   **Solution:** $T(n) = c + T(n - 3) = 2c + T(n - 2\cdot3) = 3c + T(n - 3\cdot3) = k\cdot c + T(n - i\cdot3)$. The unwrapping stops when $n - i \cdot 3 = 3$, that is when $i = (n - 3)/3$. Therefore $T(n) = (c(n - 3))/3 + T(3) = (c(n - 3))/3 + k$ which is $O(n)$.

2. (15 marks) Prove that in a proper binary tree of height $h$, the minimum number of nodes is $1 + 2 \cdot h$, and the maximum number of nodes is $2^{h+1} - 1$, where $h$ is the height of the tree. That is prove $1 + 2 \cdot h \leq n \leq 2^{h+1} - 1$, where $n$ is the number of nodes.

   **Solution:** The maximum possible number of nodes is proven exactly as for binary trees in lecture 6. Now, because the tree is proper binary tree, at each level bigger than zero we must have at least two nodes. Level 0 has only one node, the root. If $l(d)$ denotes the number of nodes at level $d$, then $l(d) \geq 2$ for $d > 0$. The tree has levels 0, ..., $h$, where $h$ is the tree height. Therefore the minimum number of nodes is $l(0) + l(1) + ... + l(h) \geq 1 + 2 \cdot h$, exactly what had to be shown.
3. (10 marks) Draw a binary search tree where each node stores a character key, and preorder traversal visits nodes in the order DCBFEZ, and postorder traversal visits nodes in the order BCEZFD.

Solution:

4. (10 marks) Suppose that a heap stores 1000 elements. What is its height?

Solution: A heap of height \( h \) that has all levels, including the last one, full stores \( 2^0 + 2^1 + ... + 2^h = 2^{h+1} - 1 \) elements. So if the heap had height 8, it could only store \( 2^9 - 1 = 511 \) elements, which is not enough. A heap of height 9 can store \( 2^{10} - 1 = 1023 \) elements, which is enough. So this heap must have height 9.

5. (15 marks) Let \( H \) be a heap of height \( h > 3 \). State all the levels of the heap at which the 3rd largest key element can be located. Recall that larger keys correspond to lower priority.

Solution: The third largest element can be at a leaf, the parent of a leaf, or the grandparent of a leaf. Since a leaf in a heap can be at levels \( h \) or \( h - 1 \), the third largest element can be at levels \( h, h - 1, h - 2, h - 3 \).

6. (30 marks)

(a) (20 marks) Suppose \( T \) is a tree that at each node stores a positive integer key. Let us call a node \( \text{EvenOdd} \) if the sum of keys in its left subtree is even, but the sum of the keys in its right subtree is odd. For example, in the tree above, the root node is \( \text{EvenOdd} \), since the sum of keys in its left subtree is 4 and in the right subtree is 5. None of the other nodes are \( \text{EvenOdd} \). Write an algorithm that takes as an input the root of the tree and returns the number of \( \text{EvenOdd} \) nodes. Assume that leaves are never \( \text{EvenOdd} \). For the tree above, your algorithm should return 1. If you need, you can store an additional variable \( v.temp \) at each node.
Solution: Algorithm EvenOdd(v)

if v.external()
    v.temp ← v.key
    return 0
else
    sumLeft ← 0, sumRight ← 0
    resultLeft ← 0, resultRight ← 0
    if v.hasLeft()
        resultLeft ← EvenOdd(v.left)
        sumLeft ← v.left.temp
    if v.hasRight()
        resultRight ← EvenOdd(v.right)
        sumRight ← v.right.temp
    v.temp = sumLeft + sumRight + v.key
    if sumLeft is even and sumRight is odd
        return 1 + resultLeft + resultRight
    else return resultLeft + resultRight

(b) (10 marks) Analyze the running time of your algorithm in part (a) as a function of the number of tree nodes n.

Solution: The algorithm visits each node of the tree, performing a constant amount of operations at each node, so the running time is \(O(n)\).