1. (20 points)

(a) Give a \( \Theta \) estimate for the solution to the recurrence

\[
T(n) = 7T(\lceil n/5 \rceil + 4) + 3n^2.
\]

You may quote theorems, but you must justify their use.

**Solution:**

We want to make this into a divide and conquer recurrence. So for some \( a \) to be defined later let

\[
S(n) = T(n+a) = 7T(\lceil (n+a)/5 \rceil + 4) + 3(n+a)^2 = 7S(\lceil (n+a)/5 \rceil + 4 - a) + 3(n+a)^2 = 7S(\lceil (n+20 - 4a)/5 \rceil) + 3(n+a)^2.
\]

So choose \( a = 5 \). Then

\[
S(n) = 7S(\lceil n/5 \rceil) + 3(n+5)^2.
\]

This is a divide and conquer recurrence which, by the MT, implies

\[
S(n) \in O(n^2).
\]

So

\[
T(n) \in O((n-5)^2) = O(n^2).
\]

(b) Prove or disprove: if \( f(n) \in O(g(n)) \), then \( 2^{f(n)} \in O(2^{g(n)}) \).

This is false. For example, let \( f(n) = 2n \) and \( g(n) = n \). Then

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} 2 = 2 < \infty,
\]

so \( f \in O(g) \). But

\[
\lim_{n \to \infty} \frac{2^{f(n)}}{2^{g(n)}} = \lim_{n \to \infty} 2^n = \infty,
\]

so \( 2^{f(n)} \not\in O(2^{g(n)}) \).

2. (20 points)

(a) Consider the problem:

**Instance:** Two lists of integers, \( K = x_1, \ldots, x_n \) and \( L = y_1, \ldots, y_k \), with \( k \leq n \).

**Problem:** Find integers \( i_1, \ldots, i_k \) so that \( y_1 = x_{i_1}, \ldots, y_k = x_{i_k} \).

We can solve this by doing a sequential search in \( K \) for each \( y_i \). Describe another method that, for large values of \( k \), is faster than repeated sequential search.

**Solution:**

Concatenate the lists and sort the concatenation. Scan the list one more time looking for neighboring \( x_i \)s and \( y_j \)s. Time: \( O((n+k) \log(n+k)) = O(n \log(n)) \).

Or, sort the first list and do a binary search for each \( y_j \). Time: \( O((n+k) \log(n)) = O(n \log(n)) \).

(b) For what values of \( k \) and \( n \) is your solution to part (a) faster than sequential search? Justify your claim.

**Solution:**

Sequential search is \( O(nk) \), so the new method is faster if \( k \in \omega(\log(n)) \).

3. (20 points)

(a) Suppose that \( H[1, \ldots, n] \) is an array containing a Min-Heap. Give pseudocode for an algorithm \text{Extract-Min}(H, n) \) that removes the smallest element from the heap \( H \) of size \( n \) and returns its value. Analyze the time complexity of your algorithm. Do not call any subroutines unless you give the pseudocode for them. Explain your algorithm.

**Solution:**
Extract-Min(H,n) H is a heap with n elements {
    if (n=0)
        return (error)
    x = H[1]
    H[1] = H[n--]
    i = 1
    while (2i+1 <= n and (H[i] > H[2i] or H[i] > H[2i+1]) {
        j = (H[2i]< H[2i+1]) ? 2i : 2i+1
        swap(H[i],H[j])
        i = j
    }
    if (2i = n and H[i] > H[2i])
        swap(H[i],H[2i])
    return(x)
}

(b) Analyze the worst case time complexity of your algorithm.
Solution:
The variable i at least doubles at each iteration, and it halts when 2i + 1 > n, so there are at most log(n) – 1 iterations. Thus the time is \(O(\log(n))\).

4. (20 points)
(a) Let \(T\) be a binary search tree and let \(x\) be a node in \(T\). Give pseudocode for an algorithm that finds the node \(y\) in \(T\) with the largest key smaller than \(x.key\). Explain your algorithm.
Solution:
If \(x\) has a left child, then \(y\) is the largest node in the left subtree of \(x\), found by going to the left child of \(x\) then taking right branches as long as possible.
If \(x\) has no left child, then \(y\) is the nearest ancestor \(z\) of \(x\) such that \(x\) is in the right subtree of \(z\).

Predecessor(x)
{
    if (x.left=NIL)
        y = x.parent
        v = x
        while (y.left = v) {
            v = y
            y = y.parent
        }
        return(y)
    } else {
        y = x.left
        while (y.right != NIL)
            y = y.right
        return(y)
}

(b) Analyze the worst case time complexity of your algorithm.
Solution:
In the worst case the algorithm traverses a path from leaf to root and back to a leaf, so the time is in \(O(h)\), where \(h\) is the height of \(T\).

5. (20 points) Describe and analyze the time complexity of an efficient algorithm that solves the following problem. Give a verbal description of your algorithm. Pseudocode is not required. You may use algorithms described in class without giving details of the algorithms.
**Instance:** An undirected graph $G = (V,E)$.

**Question:** Does $G$ have a cycle?

**Solution:**
Run DFS on $G$. If at any point we are at a node $u$ and discover an edge $(u,v)$ where $v$ is marked and is not the parent of $u$ in the DFS tree, then report that there is a cycle and halt.

Note that if there are at least $n = |V|$ edges, then $G$ must have a cycle, so we can first check whether $|E| \geq n$. This makes the time $O(n)$.

6. Extra Credit: The distance $d(u,v)$ between two nodes $u$ and $v$ in a tree is the length of the (unique) simple path from $u$ to $v$. The diameter of a tree $T$ is the maximum over all pairs of nodes $u,v$ in $T$ of $d(u,v)$. Describe an efficient algorithm that finds the diameter of a given binary tree $T = (V,E)$. Your algorithm should run in time $O(n)$, where $n = |V|$.

**Solution:**
The longest simple path in $T$ is either in one of the subtrees of the root $u$, or is a path from a leaf of one subtree to $u$, followed by a path from $u$ to a leaf in the other subtree. Thus we can recursively compute the diameter of a tree as

$$diameter(u) = \max(diameter(leftsub(u)), diameter(rightsub(u))),$$

$$height(leftsub(u)) + height(rightsub(u)) + 2).$$

(we have to interpret the height of an empty tree as $-1$ for this to work).

Thus we want a recursive algorithm that returns both the diameter and height of the tree rooted at a given node $u$. The height of the tree rooted at $u$ is one more than the maximum of the heights of its subtrees. Here is the pseudocode:

```plaintext
(int,int) = Diameter(u) { /* returns (diameter(u),height(u)) */
    if (u=NIL) return(-1,-1)
    (a,b) = Diameter(left(u))
    (c,d) = Diameter(right(u))
    y = 1 + max(b,c)
    x = max(a,c,b+d+2)
    return(x,y)
}
```

If the time complexity is $T(n)$ for a tree with $n$ nodes, then $T(n)$ satisfies the recurrence

$$T(n) \leq \max\{T(i),T(n-i-1) : 0 \leq i \leq n-1\} + c, \quad T(0) = c.$$ 

By induction we have $T(n) \leq cn$, so $T(n) \in O(n)$. 

3