1. (a) Prove that if \( f(n) \) and \( g(n) \) are polynomials with the same degree, then \( f(n) \in \Theta(g(n)) \).

We prove this by induction on the degree \( d \) of \( f \) and \( g \). If \( d = 0 \), meaning that \( f \) and \( g \) are nonzero constants, then \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = \frac{f(n)}{g(n)} \) is a nonzero constant, so \( f \in \Theta(g) \).

Now assume \( d > 0 \) and this is true for smaller degrees. The polynomials \( f' \) and \( g' \) have degree \( d - 1 \) and so \( f' \in \Theta(g') \). Moreover, \( \lim_{n \to \infty} f(n) = \lim_{n \to \infty} g(n) = \infty \), so we can use L’Hospital’s rule to compute

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

which is a nonzero constant. Thus \( f \in \Theta(g) \).

(b) Prove that for every real number \( k \), \( n^k \in o(2^n) \).

We prove this by induction on \( \lceil k \rceil \) (you can only do induction on natural numbers). For \( \lceil k \rceil \leq 0 \), we have \( n^k \leq 1 \) so \( \lim_{n \to \infty} \frac{n^k}{2^n} \leq \lim_{n \to \infty} 1/2^k = 0 \). Now suppose it is true for any \( k' \) with \( \lceil k' \rceil = d - 1 \) for some natural number \( d > 0 \) and let \( \lceil k \rceil = d \). Then \( \lim_{n \to \infty} n^k = \lim_{n \to \infty} 2^n = \infty \), so we can use L’Hospital’s rule to show

\[
\lim_{n \to \infty} \frac{n^k}{2^n} = \lim_{n \to \infty} \frac{kn^{k-1}}{2^n \ln(2)} = 0
\]

since \( \lceil k - 1 \rceil = d - 1 \).

2. (a) In this question only you may use any theorems proved in class, but say how they are being used and show that any hypotheses to the theorem are true. Find a \( \Theta \) estimate for \( T(n) \) if:

\[
T(n) = 8T(\lceil n/3 \rceil) + n^2 \log(n).
\]

This recurrence satisfies the hypotheses of the third case of the Master Theorem on divide and conquer recurrences. In the notation of the text, \( a = 8 \), \( b = 3 \), and \( f(n) = n^2 \log(n) \).

Thus \( \log_b(a) = \log_3(8) < 2 \), so \( f(n) \in \Omega(n^{\log_b(a)+\epsilon}) \) for \( \epsilon = 2 - \log_3(8) \). Also, \( af(n/b) = 8(n/3)^2 \log(n/3) < (8/9)n^2 \log(n) = (8/9)f(n) \), so the stability condition holds with \( c = 8/9 \).

Therefore \( T(n) \in \Theta(n^{2 \log(2)}) \).

(b) Explain why the Master Theorem for divide and conquer recurrences cannot be used to solve the recurrence

\[
S(n) = S(n/2) + \log(n).
\]

In this case we have \( a = 1 \), \( b = 2 \), and \( f(n) = \log(n) \) so \( \log_b(a) = 0 \). Thus \( f(n) \in \omega(n^{\log_b(a)}) \), so the hypotheses of cases (2) and (3) do not hold, but \( f(n) \not\in O(n^{\log_b(a)+\epsilon}) \) for any positive epsilon, so the hypothesis of case (1) does not hold.
(c) Explicitly (no big-O) solve the recurrence in part (b) when \( n \) is a power of 2.

Using recursion trees: there is 1 node at depth \( d \)th, with weight \( \log(n/2^d) = \log(n) - d \). The tree has height \( r = \log(n) \). There is one leaf, with weight \( S(1) \). Thus

\[
S(n) = S(1) + \sum_{d=0}^{r-1} r - d = S(1) + \sum_{d=1}^{r} d = S(1) + \frac{\log(n)(\log(n) + 1)}{2} \in \Theta(\log(n)^2).
\]

3. Consider the following problem:

**Instance:** A list \( X = x_1, \ldots, x_n \) of integers  

**Output:** The index of the largest integer in \( X \).

(a) Give a pseudocode description of an efficient iterative (nonrecursive) algorithm that solves this problem.

\[
\text{MAX}(X[1, \ldots, n]) \{
\text{m} = 1 \\
\text{i} = 2 \\
\text{while } (i < n+1) \{
\text{if } (X[i] > X[m]) \\
\text{m} = i \\
\text{i} = i+1 \\
\text{\}} \\
\text{return(m)} \\
\}
\]

(b) Give a formal correctness statement for your algorithm and prove the algorithm is correct using the method of invariants.

**Correctness:**

\[
P : 1 \leq m \leq n \land \forall i \in \{1, \ldots, n\} : X[m] \geq X[i].
\]

**Invariant:**

\[
R : 1 \leq m \leq i - 1 \land \forall j \in \{1, \ldots, i - 1\} : X[m] \geq X[j].
\]

**Proof:**

i. Initially \( m = 1 \) and \( i = 2 \) so \( R \) becomes

\[
1 \leq 1 \leq 1 \land \forall j \in \{1\} : X[1] \geq X[j],
\]

which is true.
ii. Suppose $R$ is true at the start of the loop and the loop is entered. Let $i_0$ and $m_0$ be the values of $i$ and $m$ at the start of the loop body, and let $i + 1$ and $m_1$ be their values after the loop body executes. Then $1 \leq i_0 \leq n$. $R$ says that $1 \leq m_0 \leq i_0 - 1$ and for every $j \in \{1, \ldots, i_0 - 1\}$: $X[m_0] \geq X[j]$. We have $i_1 = i_0 + 1$, and $m$ can only change to $i_0$. Thus $1 \leq m_1 \leq i_0 - 1$ or $m_1 = i_0$. Therefore $1 \leq m_1 \leq i_1 - 1$. Also, $X[m_1] = \max(X[m_0],X[i_0]) \geq X[m_0] \geq X[j]$ for every $j \in \{1, \ldots, i_0 - 1\}$, so $X[m_1] \geq X[j]$ for every $j \in \{1, \ldots, i_1 - 1\}$.

iii. If the loop halts and $R$ is true, then $i = n + 1$ so $R$ says that $1 \leq m \leq n$ and for every $j \in \{1, \ldots, n\}$: $X[m] \geq X[j]$. This is $P$.

iv. The loop eventually halts because $i$ is increased by 1 at each iteration.

4. Give a high level description of a data structure for maintaining a set of integers with the usual operations $INSERT$, $DELETE$, and the new operation $STAT(k$: integer): return the integer with rank $k$ (the $k$th largest integer).

You may describe your data structure by saying how to modify data structures we have discussed in class, and just say briefly in words how $INSERT$ and $DELETE$ must be modified. Give pseudocode for $STAT(k)$. All operations should take time $O(\log(n))$, where $n$ is the number of integers in the set.

Use a red-black tree with the additional information in each node $v$ consisting of the number of nodes in the right subtree, $v.rnum$. $INSERT$ and $DELETE$ are modified to adjust these counts. In $INSERT$, after the new node $v$ is inserted and the red-black property restored, traverse the path from $v$ to the root. Whenever a link is followed that is a right child link, increment the count in the parent. $DELETE$ is similar, but with decrements instead of increments.

For $STAT$, suppose we know that the desired node is in the subtree rooted at a node $v$. If $k \leq v.rnum$, then search the right subtree for the $k$th largest. If $k = v.rnum + 1$, then return $v$. Else search the left subtree for the $(k - 1 - v.rnum)$th largest.

\[
STAT(r,k) \{
    \text{if (} k \leq v.rnum \text{)}
    \quad \text{return(STAT(r.right,k))}
    \text{else if (} k = v.rnum+1 \text{)}
    \quad \text{return(v)}
    \text{else if (} v.left = NIL \text{)}
    \quad \text{return(error)}
    \text{else return(STAT(v.left,k-(v.rnum)-1))}
\}
\]

5. (a) Give asymptotic estimates for the worst case complexity and average case complexity for basic QuickSort.
Worst: $\Theta(n^2)$. Average: $\Theta(n \log(n))$.

(b) State a sharp lower bound on the number of comparisons used by a comparison based sorting algorithm.
$\Omega(n \log(n))$.

(c) Let $H$ be a min-heap with $n$ elements, stored in an array in the usual way. Give a pseudocode description of an efficient implementation of the operation $\text{INSERT}(H, x)$ that inserts item $x$ in heap $H$.

```c
INSERT(H, n, x) H is a heap with n elements {
    n = n+1
    H[n] = x
    i = n
    while (i > 1 and H[i] < H[i/2]) {
        swap(H[i], H[i/2])
        i = i/2
    }
}
```

6. Give a high level description of a data structure that supports the following operations:

- $\text{Insert}(x)$: insert key $x$ in the structure only if it is not already there.
- $\text{Delete}(x)$: delete key $x$ from the structure if it is there.
- $\text{Next}(x)$: return a pointer to the smallest key in the structure that is larger than $x$ (or a NIL pointer if $x$ is the largest key in the structure).

You may refer to structures and algorithms we have described in class in describing $\text{Insert}$ and $\text{Delete}$. All operations should take time $O(\log(n))$.

Use a red-black tree. $\text{Insert}$ and $\text{Delete}$ are unchanged. For $\text{Next}$, first find $x$ at node $v$ using the usual search operation on search trees. If $v$ has a right child $u$, follow left child pointers from $u$ until you reach a node $w$ with no left child. Return $w$. Otherwise, traverse the path from $v$ to the root until a node is found that is the left child of its parent $p$. Return $p$. If no node is found that is a left child, return NIL.

7. Insert in order into a red-black tree: 7,3,2,9,11,15. Show the tree after each insertion.

Sorry, I cannot draw the trees.

8. (a) Describe an efficient algorithm that determines whether a given undirected graph $G = (V, E)$ is a tree.

Let $n = |V|$ and $e = |E|$. $G$ is a tree if and only if it is connected and $e = n - 1$. So first check whether $e = n - 1$. Then do DFS from one starting node and check whether every node is marked.
(b) Analyze the running time of your algorithm if the graph is represented by an adjacency list and if the graph is represented by an adjacency matrix.

Adjacency list: counting edges can be done by traversing every edge list in time $O(n + e)$. DFS takes time $O(n + e)$. Checking whether every node is marked takes time $O(n)$. Total: $O(n + e)$.

Adjacency matrix: counting edges can be done by traversing row in time $O(n^2)$. DFS takes time $O(n^2)$. Checking whether every node is marked takes time $O(n)$. Total: $O(n^2)$. 