DIRECTIONS:

- Closed book, closed notes except for one 8.5×11 sheet.
- Time limit 80 minutes.
- Calculators not allowed.
- Do not turn the page until I tell you to.
- Good luck!

Solutions

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/ 21</td>
</tr>
<tr>
<td>2</td>
<td>/ 40</td>
</tr>
<tr>
<td>3</td>
<td>/ 16</td>
</tr>
<tr>
<td>4</td>
<td>/ 24</td>
</tr>
<tr>
<td>5</td>
<td>/ 12</td>
</tr>
<tr>
<td>6</td>
<td>/ 21</td>
</tr>
<tr>
<td>7</td>
<td>/ 6</td>
</tr>
<tr>
<td>Total</td>
<td>/ 140</td>
</tr>
</tbody>
</table>
1. (21 points) Circle True (T) or False (F).

No justification is required.

(a) Let \( G = (V, E) \) be a connected undirected graph. Let \( L(v) \) for \( v \in V \) be the level of \( v \) in the BFS tree rooted at some node \( s \). Then for all \( (u, v) \in E \), we have \(|L(u) - L(v)| \leq 1\). \( \bigcirc \) \( \bigcirc \) T F

(b) Suppose that in the DFS of a directed graph there is a non-tree edge. Then it must connect an ancestor and a descendant. \( \bigcirc \) \( \bigcirc \)

(c) Let \( G = (V, E) \) be a connected, directed graph. If the edge \( (u, v) \) is a cross edge in the depth-first search, then the dfs-number of \( u \) is less than the dfs-number of \( v \). \( \bigcirc \) \( \bigcirc \)

(d) If a directed graph is not strongly connected, then there is a pair of nodes \( x \) and \( y \) for which there is no path from \( x \) to \( y \). \( \bigcirc \) \( \bigcirc \)

(e) Let \( G \) be a directed graph. If there is a path from \( u \) to \( v \) then there is also a path from \( v \) to \( u \). \( \bigcirc \) \( \bigcirc \)

(f) If a directed graph \( G \) has a cycle then it cannot have a topological ordering. \( \bigcirc \) \( \bigcirc \)

(g) Let \( G \) be a directed graph. Construct a graph \( G' \) as follows: \( G' \) has a vertex for each strongly connected component of \( G \). There is an edge from \( S \) to \( T \) in \( G' \) if and only if some vertex \( u \in S \) has an edge to a vertex \( v \in T \). \( G' \) is acyclic. \( \bigcirc \) \( \bigcirc \)
2. (40 points)

For each pair of functions $f(n)$ and $g(n)$ below, circle all of the answers A through D that apply.

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$g(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$500n^2 - 42n$</td>
<td>$3n^2$</td>
</tr>
<tr>
<td>$500n^2 - 42n$</td>
<td>$3n^3$</td>
</tr>
<tr>
<td>$n \log n$</td>
<td>$\sqrt{n}$</td>
</tr>
<tr>
<td>$n \log n$</td>
<td>$n$</td>
</tr>
<tr>
<td>$n^{10}$</td>
<td>$10^n$</td>
</tr>
<tr>
<td>$n^{100}$</td>
<td>$1.01^n$</td>
</tr>
<tr>
<td>$n^{1/3}$</td>
<td>$2^{\sqrt{\log n}}$</td>
</tr>
<tr>
<td>$n^{2+\sin n}$</td>
<td>$n^3$</td>
</tr>
<tr>
<td>$(\log n)^{\log n}$</td>
<td>$2^{(\log_2 n)^2}$</td>
</tr>
<tr>
<td>$2^{\log_2 n}$</td>
<td>$2^{3\log_2 n}$</td>
</tr>
</tbody>
</table>

- A $f(n) = O(g(n))$
- B $f(n) = \Omega(g(n))$
- C $f(n) = \Theta(g(n))$
- D $f(n) = o(g(n))$
3. (16 points)

For each of the recurrences listed below, indicate which of the following algorithms has a running time that satisfies this recurrence. (Enter a number in the box. For example if binary search has that recurrence, enter A in the box next to the recurrence.)

A. Binary search
B. Finding inversions in a list of $n$ distinct integers.
C. Strassen's matrix multiplication algorithm
D. Karatsuba multiplication.

(a) $T(n) = 3T(n/2) + O(n)$.

(b) $T(n) = T(n/2) + O(1)$.

(c) $T(n) = 7T(n/2) + O(n^2)$.

(d) $T(n) = 2T(n/2) + O(n)$.
4. (24 points) 5 points each

Consider a recursive algorithm whose running time \( T(n) \) satisfies the following recurrence relation:

\[
T(n) = \begin{cases} 
5T(n/3) + n^2 & n > 1 \\
1 & n = 1 
\end{cases}
\]

Assume that \( n \) is a power of 3.

(a) How many nodes are there at level \( j \) in the recursion tree?

(b) What is the size of each subproblem at level \( j \)?

\[
\frac{n}{3^j}
\]

(c) How much work is done at level \( j \) (i.e., the total number of steps require to combine the results of recursive calls performed by algorithm at level \( j \))?

\[
5^j \left( \frac{n}{3^j} \right)^2 = n^2 \left( \frac{5}{9} \right)^j
\]

(d) What is the depth of the recursion tree?

\[
\log_3(n)
\]

(e) What is the total work done by the algorithm? Express your answer as a sum over the work at the various levels in the recursion tree.

\[
h^2 \sum_{j=0}^{\log_3 n} \left( \frac{5}{9} \right)^j
\]

(f) Compute the value of the sum from the previous part, expressing your answer in the form \( O(n^c) \) for some constant \( c \). You may use the fact that \( \sum_{j=0}^{k} \alpha^j = \frac{\alpha^{k+1} - 1}{\alpha - 1} \) and \( a^{\log_a d} = d^{\log_a a} \).

\[
\frac{n^2 \left( \frac{5}{9} \right)^{\log_3 n + 1}}{1 - \frac{5}{9}} = O(n^2)
\]
5. (12 points)

Consider running a breadth-first-search on an undirected, connected graph $G$.

Modify the pseudo-code below to determine if $G$ is bipartite or not, by filling in the line below so that it does the appropriate check and then updates the variable `bipartite` to `FALSE` if the graph is not bipartite.

```
BFS(s)
    mark s "discovered"
    L[s] = 0;
    queue = { s };
    bipartite = TRUE;
    
    while queue not empty {
        u = remove_first(queue);
        for each edge (u, x) {
            if (x is undiscovered) then {
                mark x "discovered";
                append x to queue;
                L[x] = L[u] + 1;
            } else {
                if L[u] = L[x] then bipartite = FALSE
            }
        }
    }
```
6. (21 points)

Consider running a depth-first-search on a directed graph $G$ with $n$ nodes.

Modify the pseudo-code on the next page to either output a topological order of the nodes in the graph (in the array $A[1..n]$) or else to say that the graph has a cycle (by setting the variable $\text{hasCycle}$ to $\text{TRUE}$). If the graph can be topologically sorted, the topological ordering of nodes should be output in the array $A[1..n]$ from left to right, i.e., if there is an edge from $A[i]$ to $A[j]$, then $i < j$. The code you add should only be in the spaces provided, where there is a line. You may not need to fill in all of the blank lines. Also, if you wish, you can use an additional global variable that you initialize on the blank line in Main.
Main; { 
    integer array A[1..n]; // for storing topological order
    Boolean HasCycle = FALSE; // initially assume the graph is acyclic.
    int time = 0;

    for v = 1 to n do { 
        v.dfs# = -1 // initialization: DFS of v hasn't started
        v.finish# = -1 // initialization: DFS of v hasn't finished.
    }

    int i = n;
    for u = 1 to n do {
        if u.dfs# == -1 then 
            DFS(u);
    }
}

DFS(v) { 
    v.dfs# = time++; // v's dfs number gets current time.
    // increment time.

    for each edge (v, x) { 
        if (x.dfs# == -1) { // x unvisited
            DFS(x)
        } else if (x.finish# == -1 and x.dfs# < v.dfs#) then
            HasCycle = TRUE;
    }

    v.finish# = time++; // v's finish number gets current time.
    // increment time.

    A [i] = v;

    i--; 
}
7. (6 points)

Circle the articulation points on the graph below.