Homework 1 Solution

Problem 1

For each pair of functions in the list below, indicate their asymptotic relation (O, Ω, Θ). No justification is needed.

- $f_1(n) = n, f_2(n) = 2^{((\log_2 n)^{10})}$
- $f_1(n) = n^{100}, f_2(n) = 2^{((\log_2 n)^{10})}$
- $f_1(n) = n^{0.1}, f_2(n) = 2^{((\log_2 n)^{10})}$
- $f_1(n) = n, f_2(n) = 2^{((\log_2 n)^{0.01})}$
- $f_1(n) = n^{100}, f_2(n) = 2^{((\log_2 n)^{0.01})}$
- $f_1(n) = n^{0.1}, f_2(n) = 2^{((\log_2 n)^{0.01})}$

Solution:

$n = O(2^{((\log_2 n)^{10})})$
$n^{100} = O(2^{((\log_2 n)^{10})})$
$n^{0.1} = O(2^{((\log_2 n)^{10})})$
$n = \Omega(2^{((\log_2 n)^{0.01})})$
$n^{100} = \Omega(2^{((\log_2 n)^{0.01})})$
$n^{0.1} = \Omega(2^{((\log_2 n)^{0.01})})$

Problem 2

Prove that in any tree with n vertices, the number of nodes with degree 8 or more is at most $(n-1)/4$.

Solution:

Since a tree with n vertices contains $n-1$ edges, we have

$$\sum_v \deg(v) = 2(n-1).$$

On the other hand,

$$8 \cdot |\{v : \deg(v) \geq 8\}| = \sum_{v:\deg(v)\geq 8} 8 \leq \sum_v \deg(v).$$

So we have

$$|\{v : \deg(v) \geq 8\}| \leq 2(n-1).$$

Problem 3

Give an algorithm to detect whether a given undirected graph contains a cycle. If the graph contains a cycle, then your algorithm should output one cycle (it need not output all cycles in the graph, just one of them). The running time of your algorithm should be $O(m+n)$ for a graph with n nodes and m edges.

Solution:
An $O(m+n)$ running time algorithm:

1. Run DFS on the graph, and let T denote the DFS tree.
2. If $G = T$ (all the edges of G are in T), then return No.
3. Otherwise, let (u, v) be an arbitrary edge of G that is not in T.
4. Find the path from v to u in T, let P denote this path.
5. Return $P \cup \{(v, u)\}$.

Problem 4

Given a connected graph G with n vertices. We say an edge of G is a *bridge* if the graph becomes a disconnected graph after removing the edge. Give an $O(m+n)$ time algorithm that finds all the bridges. (Partial credits will be given for a polynomial time algorithm.)
(*Hint: Use DFS.*)

Solution:
An $O(nm)$ running time algorithm:

1. Enumerate every edge (u, v) in the tree
   (a) Delete edge (u, v) in the graph.
   (b) Run BFS with start vertex u.
   (c) If the BFS visited vertex v, then (u, v) is a bridge.
   (d) Add edge (u, v) back to the graph.

An $O(n+m)$ running time Algorithm:

1. Run DFS with an arbitrary vertex s as the root. For every vertex v, set $number(v)$ to be integer i if v is the i-th discovered vertex in the DFS.
2. For $i = n, n-1, \ldots, 1$
   (a) Let v be the vertex such that $number(v) = i$ and set $back(v) = i$
   (b) For each edge (v, u) which is a DFS tree edge, such that u is a child of v, set
   $$back(v) \leftarrow min\{back(v), back(u)\}$$
   (c) For each edge (v, u) which is not a DFS tree edge, set
   $$back(v) \leftarrow min\{back(v), number(u)\}$$

3. For each vertex v

   (a) If $v \neq s$ and $back(v) = number(v)$, then (v, u) is a bridge, where u is the parent of v.

Problem 5

Given a tree T with n vertices. Give an $O(n)$ time algorithm that finds a vertex v such that all connected components of $T - v$ (tree T removing vertex v and all the edges connected to v) has at most $n/2$ vertices. Justify the running time bound of your algorithm. (Partial credits will be given for a polynomial time algorithm.)

Solution:
An $O(n^2)$ running time algorithm:

1. Enumerate every vertex v in the tree

   (a) Delete vertex v and all of its incident edges from the graph.

   (b) Run BFS on every connected component of the resulted graph, and count the number of vertices for each connected component.

   (c) If all the connected components have size at most $n/2$, then output vertex v and exit.

   (d) Reverse all the changes made at step (a) (add vertex v and all its incident edges back to the graph).

An $O(n)$ running time algorithm:

1. Run BFS with an arbitrary vertex s as the root. For every vertex v, set $number(v)$ to be integer i if v is the i-th discovered vertex in the BFS.

2. For $i = n, n-1, \ldots, 1$,

   (a) Let v be the vertex such that $number(v) = i$ and set $size(v) \leftarrow 1$

   (b) For each edge (v, u) such that $number(u) > number(v)$, set $size(v) \leftarrow size(v) + size(u)$

3. For each vertex v,

   (a) If both conditions below hold, then return vertex v and exit, otherwise do nothing.

      i. $n - size(v) \leq n/2$

      ii. For every edge (v, u) satisfying $number(u) > number(v)$, $size(u) \leq n/2$.