Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: _____ Wisc id: _____

Asymptotic Analysis

1. *Kleinberg, Jon. Algorithm Design (p. 67, q. 3, 4).* Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $g(n)$ immediately follows function $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$.

   (a) $f_1(n) = n^{2.5}$
   $f_2(n) = \sqrt{2n}$
   $f_3(n) = n + 10$
   $f_4(n) = 10n$
   $f_5(n) = 100n$
   $f_6(n) = n^2 \log n$

   Solution:

   f_2, $[f_3, f_4, f_5]$, f_6, f_1

   or $\sqrt{2n}$, $[n + 10, 10n, 100n]$, $n^2 \log n$, $n^{2.5}$

   (b) $g_1(n) = 2^{\log n}$
   $g_2(n) = 2^n$
   $g_3(n) = n(\log n)$
   $g_4(n) = n^{4/3}$
   $g_5(n) = n^{\log n}$
   $g_6(n) = 2^{(2^n)}$
   $g_7(n) = 2^{(n^2)}$

   Solution:

   g_1, g_3, g_4, g_5, g_2, g_7, g_6

   or $2^{\log n}$, $n(\log n)$, $n^{4/3}$, $n^{\log n}$, 2^n, $2^{(n^2)}$, $2^{(2^n)}$

2. *Kleinberg, Jon. Algorithm Design (p. 68, q. 5).* Assume you have a positive, non-decreasing function f and a positive, increasing function g such that $g(n) \geq 2$ and $f(n)$ is $O(g(n))$. For each of the following statements, decide whether you think it is true or false and give a proof or counterexample.

   (a) $\log_2 f(n)$ is $O(\log_2 g(n))$

   Solution:

   Since $f(n) = O(g(n))$, $\exists c_0, n_0 > 0$ such that $\forall n \geq n_0$, $f(n) \leq c_0 \cdot g(n)$.

   So $\forall n \geq n_0$, $\log_2 f(n) \leq \log_2 c_0 + \log_2 g(n)$. Note that $\log_2 c_0$ and n_0 are constants.

   Let $n_1 = n_0$ and $c_1 = \frac{\log_2 c_0}{\log_2 g(n_0)} + 1$. So $\forall n \geq n_1$, $\frac{\log_2 c_0}{\log_2 g(n)} + 1 \leq c_1$, and thus $\log_2 c_0 + \log_2 g(n) \leq c_1 \log_2 g(n)$.

   Now, $\forall n \geq n_1$, $\log_2 f(n) \leq \log_2 c_0 + \log_2 g(n) \leq c_1 \log_2 g(n)$. Therefore, $\log_2 f(n)$ is $O(\log_2 g(n))$.

   P.S. The statement is actually false if the functions are not non-decreasing. Counterexample: $f(n) = 2(1 + \frac{1}{n})$, $g(n) = (1 + \frac{1}{n})$.

   (b) $2^{f(n)}$ is $O(2^{g(n)})$

   Solution:

   False. Counterexample: $f(n) = \log_2 n^2$, $g(n) = \log_2 n$.

   $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2\log_2 n}{\log_2 n} = 2$

   $\lim_{n\to\infty} \frac{2^{f(n)}}{2^{g(n)}} = \lim_{n\to\infty} \frac{2^{\log_2 n^2}}{2^{\log_2 n}} = \lim_{n\to\infty} \frac{n^2}{n} = \lim_{n\to\infty} n = \infty$

   (c) $f(n)^2$ is $O(g(n)^2)$

   Solution:

   Since $f(n) = O(g(n))$, $\exists c_0, n_0 > 0$ such that $\forall n \geq n_0$, $f(n) \leq c_0 \cdot g(n)$.

   Because f and g are positive, we can square both sides of the inequality: $f(n)^2 \leq {c_0}^2 \cdot g(n)^2$

   Let $n_1 = n_0$ and $c_1 = c_0^2$. So we have $\forall n \geq n_1$, $f(n)^2 \leq c_1 \cdot g(n)^2$. Therefore, $f(n)^2$ is $O(g(n)^2)$.

3. *Kleinberg, Jon. Algorithm Design (p. 68, q. 6).* You're given an array A consisting of n integers. You'd like to output a two-dimensional n-by-n array B in which $B[i, j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$ — that is, the sum $A[i] + A[i + 1] + ... + A[j]$. (Whenever $i \geq j$, it doesn't matter what is output for $B[i, j]$.) Here's a simple algorithm to solve this problem.

```
for i = 1 to n
  for j = i + 1 to n
    add up array entries A[i] through A[j]
    store the result in B[i, j]
  endfor
endfor
```

(a) For some function f that you should choose, give a bound of the form $O(f(n))$ on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm).

Solution:

$O(n^3)$

(b) For this same function f, show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asympto tically tight bound of $\Theta(f(n))$ on the running time.)

Solution:

We have a triple-nested loop, in which the outer loop is $\Omega(n)$, the middle loop is $\Omega(n)$, and the inner loop is $\Omega(n)$. The other work within the loops is constant, so this algorithm is $\Omega(n^3)$.

(c) Although the algorithm provided is the most natural way to solve the problem, it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $O(g(n))$, where $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$.

Solution:

Initialize $B[1, 1] = A[1]$, then fill in the first row as follows:

```
for i = 2 to n
   B[1,i] = B[1,i-1] + A[i]
endfor
```

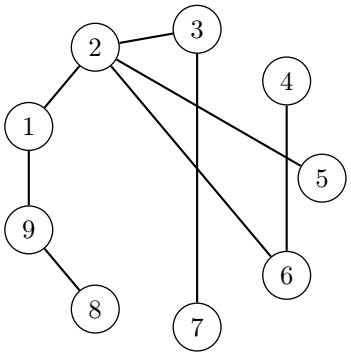
To fill in the rest of the array:

```
for row = 2 to n
   for col = row + 1 to n
      B[row,col] = B[row-1,col] - A[row-1]
   endfor
endfor
```

The first loop runs in $O(n)$ time and the second loop runs in $O(n^2)$ time, so the overall runtime for this algorithm is $O(n^2)$.

Graphs

4. Given the following graph, list a possible order of traversal of nodes by breadth-first search and by depth-first search. Consider node 1 to be the starting node.



Solution:

Breadth-First Search: in the groups, order doesn't matter
1, [2, 9], [3, 5, 6, 8] , [4, 7]

Depth-First Search: not an exhaustive list of solutions
1, 9, 8, 2, 3, 7, 6, 4, 5
1, 2, 3, 7, 6, 4, 5, 9, 8
1, 2, 6, 4, 5, 3, 7, 9, 8

5. *Kleinberg, Jon. Algorithm Design (p. 108, q. 5).* A binary tree is a rooted tree in which each node has at most two children. Show by induction that in any binary tree the number of nodes with two children is exactly one less than the number of leaves.

Solution:

In a binary tree with n nodes, let t_n be the number of nodes with two children and l_n be the number of leaves.

WTS: In a binary tree with n nodes, $t_n = l_n - 1$.

Base Case: A binary tree with 1 node has $l_1 = 1$ and $t_1 = 0$, so it holds that $t_1 = l_1 - 1$.

Inductive Hypothesis: In a binary tree with k nodes, $t_k = l_k - 1$.

Inductive Step: In a binary tree with *k+1* nodes, consider an arbitrary leaf c. Since the tree has more than 1 node, c has a parent, p.

Consider the tree of k nodes that results from removing c:

1. If c was the only child of p, p is now a leaf in the place of c, and $l_{k+1} = l_k$. Additionally, the number of two-child nodes did not change, so $t_{k+1} = t_k$. Since by the inductive hypothesis, $t_k = l_k - 1$, we have $t_{k+1} = l_{k+1} - 1$.

2. If p had another child in addition to c, then removing c decrements the number of two-child nodes, so $t_{k+1} - 1 = t_k$. Additionally, p is not a leaf, so the number of leaves also decrements: $l_{k+1} - 1 = l_k$. Since by the inductive hypothesis, $t_k = l_k - 1$, we have $t_{k+1} - 1 = l_{k+1} - 1 - 1 \implies t_{k+1} = l_{k+1} - 1$.

6. *Kleinberg, Jon. Algorithm Design (p. 108, q. 7).* Some friends of yours work on wireless networks, and they're currently studying the properties of a network of n mobile devices. As the devices move around, they define a graph at any point in time as follows:

   > There is a node representing each of the n devices, and there is an edge between device i and device j if the physical locations of i and j are no more than 500 meters apart. (If so, we say that i and j are "in range" of each other.)

   They'd like it to be the case that the network of devices is connected at all times, and so they've constrained the motion of the devices to satisfy the following property: at all times, each device i is within 500 meters of at least $\frac{n}{2}$ of the other devices. (We'll assume n is an even number.) What they'd like to know is: Does this property by itself guarantee that the network will remain connected?

   Here's a concrete way to formulate the question as a claim about graphs.

   Claim: Let G be a graph on n nodes, where n is an even number. If every node of G has degree at least $\frac{n}{2}$, then G is connected.

   Decide whether you think the claim is true or false, and give a proof of either the claim or its negation.

Solution:

Proof by Contradiction:

Suppose the graph is not connected, so there exists $x, y \in V$ such that there is no path between x and y.

Let X and Y be the set of nodes reachable from x and y, respectively. Note that $X \cap Y = \emptyset$.

Since nodes in G have degree at least $\frac{n}{2}$, we have that $|X| \geq \frac{n}{2} + 1$ and $|Y| \geq \frac{n}{2} + 1$.

Therefore, $|X \cup Y| = |X| + |Y| \geq 1 + \frac{n}{2} + 1 + \frac{n}{2} = n + 2$, which is a contradiction, since the graph G only has n nodes.

Coding Question

7. Implement depth-first search in either C, C++, C#, Java, or Python. Given an undirected graph with n nodes and m edges, your code should run in $O(n + m)$ time. Remember to submit a makefile along with your code, just as with week 1's coding question.

   Input: the first line contains an integer t, indicating the number of instances that follows. For each instance, the first line contains an integer n, indicating the number of nodes in the graph. Each of the following n lines contains several space-separated strings, where the first string s represents the name of a node, and the following strings represent the names of nodes that are adjacent to node s. You can assume that the nodes are listed line-by-line in lexicographic order (0-9, then A-Z, then a-z), and the adjacent nodes of a node are listed in lexicographic order. For example, consider two consecutive lines of an instance:

   ```
   0, F
   B, C, a
   ```

   Note that 0 < B and C < a.

   Input constraints:

   - $1 \leq t \leq 1000$
   - $1 \leq n \leq 100$
   - Strings only contain alphanumeric characters
   - Strings are guaranteed to be the names of the nodes in the graph.

   Output: for each instance, print the names of nodes visited in depth-first traversal of the graph, *with ties between nodes visiting the first node in input order*. Start your traversal with the first node in input order. The names of nodes should be space-separated, and each line should be terminated by a newline.

   Sample:

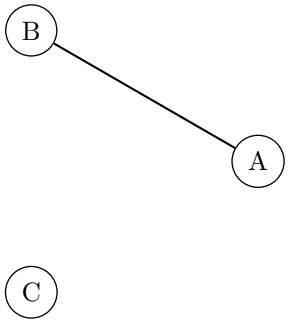
   Input:

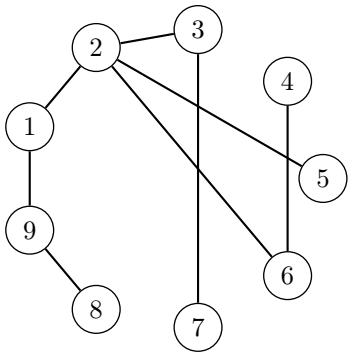
   ```
   2
   3
   A B
   B A
   C
   9
   1 2 9
   2 1 6 5 3
   4 6
   6 2 4
   5 2
   3 2 7
   7 3
   8 9
   9 1 8
   ```

   Output:

   ```
   A B C
   1 2 6 4 5 3 7 9 8
   ```

   The sample input has two instances. The first instance corresponds to the graph below on the left. The second instance corresponds to the graph below on the right.


B
A
C


3
2
4
1
5
9
6
8
7