CSCI 270
HW2 Solutions

1. Arrange these functions under the O notation using only = (equivalent) or ⊂ (strict subset of):
   a) $2^{\log n}$
   b) 2^{3n}
   c) $n^{n \log n}$
   d) $\log n$
   e) $n\log(n^2)$
   f) n^{n^2}
   g) $\log(\log(n^n))$

All logs are base 2.

Solution:
First separate the functions into logarithmic, polynomial, and exponential

$$2^{\log n} = n,\ n^{n \log n} = 2^{n(\log n)^2},\ n^{n^2} = 2^{n^2 \log n}$$

Logarithmic: $\log n$, $\log(\log(n^n))$
Polynomial: $2^{\log n}$, $n \log(n^2)$
Exponential: 2^{3n}, $n^{n\log n}$, n^{n^2}

- Since

$$\log n \le 1. \log(n \log n) = \log(\log(n^n))$$

so $\log n = O(\log(\log(n^n)))$

$$\log(\log(n^n)) = \log(n \log n) \le \log(n^2) = 2\cdot\log n$$

so $\log(\log(n^n)) = O(\log(n^2))$

$$O(\log(\log(n^n))) = O(\log(n^2))$$

- Since every logarithmic grows slower than a polynomial

$$O(\log(\log(n^n))) \subset O(2^{\log n})$$

$2^{\log n} = O(n) \subset O(n \log n) = O(2 \cdot n\log n) = O(n \log n^2)$. Thus

$$O(2^{\log n}) \subset O(n \log n^2)$$

- Since every exponential grows faster than every polynomial

$$O(n \log(n^2)) \subset O(2^{3n})$$

- Since

$$O(3n) \subset O(n \log(n^2)) \subset O(n^2 \log n),$$

so

$$O(2^{3n}) \subset O(2^{n(\log n)^2}) = O(n^{n \log n}) \subset O(2^{n^2 \log n}) = O(n^{n^2})$$

Therefore

$O(\log n) = O(\log(\log(n^n))) \subset O(2^{\log n}) \subset O(n \log n^2) \subset O(2^{3n}) \subset O(n^{n \log n}) \subset O(n^{n^2})$

Rubric (10 pts):

- 1 pt for correctly placing each of the 7 functions. '=' in place of '⊂' or vice versa is an incorrect placement
- 3 pts for brief justification

2. Given functions f_1, f_2, g_1, g_2 such that $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$. For each of the following statements, decide whether it is true or false and briefly explain why.

    a) $f_1(n)/f_2(n) = O(g_1(n)/g_2(n))$
    b) $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$
    c) $f_1(n)^2 = O(g_1(n)^2)$
    d) $\log_2(f_1(n)) = O(\log_2(g_1(n)))$

Solution:

By definition, there exists $c_1, c_2 > 0$ such that

$$f_1(n) \leq O(g_1(n)) \text{ and } f_2(n) \leq O(g_2(n))$$

for a sufficiently large n

a) False

$$f_1(n) = n^3$$

$f_2(n) = n$
$g_1(n) = n^3$
$g_2(n) = n^2$

b) True

$$
\begin{aligned}
f_1(n) + f_2(n) &\le c_1 \cdot g_1(n) + c_2 \cdot g_2(n) \\
&\le (c_1 + c_2)(g_1(n) + g_2(n)) \\
&\le 2 \cdot (c_1 + c_2)\max(g_1(n), g_2(n)))
\end{aligned}
$$

c) True

$$f_1(n)^2 \le (c_1 \cdot g_1(n))^2 = c_1{}^2 \cdot g_1(n)^2$$

d) False. Consider $f_1(n) = 2$ and $g_1(n) = 1$

$$\log_2 (f_1(n)) = 1 \ne O(\log_2(g_1(n)) = O(0)$$

Rubric:

- 1 pt: Correct T/F claim
- 2 pts: Providing the correct explanation or counterexample

3. Given an undirected graph G with n nodes and m edges, design an O(m+n) algorithm to detect whether G contains a cycle. Your algorithm should output a cycle if there is one.

Solution:
Without loss of generality assume that G is connected. Otherwise, we can compute the connected components in O(m+n) and deploy our algo on each component.
Starting from an arbitrary vertex s, run BFS to obtain a BFS tree T, which takes O(m+n) time. If G = T, then G is a tree and has no cycles. Otherwise, there is a cycle and there exists an edge e = (u,v) belonging to G but not T. Let w be the least common ancestor of u and v. There exists a unique path T_1 in T from u to w and a unique path in T_2 from w to v. Both T_1 and T_2 can be found on O(m) time. Output the cycle by concatenating T_1 and T_2.

Rubric(12 pts):

- No penalty for not mentioning disconnected case
- 6 pts: For detecting the cycle in G
- 4 pts: For finding the cycle if there is one
- 2 pts: Describing the runtime as O(m+n)

4. Solve Kleinberg and Tardos, Chapter 2, Exercise 6.

Solution:

a) The outer loop runs for exactly n iterations, the inner loop runs for at most n iterations, and the number of operations needed for adding up array entries A[i] through A[j] is $j - i + 1 = O(n)$. Therefore, the running time is in $n^2 \cdot O(n) = O(n^3)$.

b) Consider those iterations that require at least n/2 operations to add up array entries A[i] through A[j]. When $i \leq n/4$ and $j \geq 3n/4$, the number of operations needed is at least n/2. So there are at least $(n/4)^2$ pairs of (i, j) such that adding up A[i] through A[j] requires at least n/2 operation. Therefore, the running time is at least $\Omega((n/4)^2 \cdot n/2) = \Omega(n^3/32) = \Omega(n^3)$.

c) Algorithm:

```
for i = 1,2….n-1 do
        B[i,i+1] = A[i] + A[i+1]
end for
for j  = 2,3….n-1 do
        for i = 1,2….n-j do
                B[i,i+j] = B[i,i+j-1] + A[i+j]
        end for
end for
```

It first computes B[i, i + 1] for all i by summing A[i] with A[i+1]. This for loop requires O(n) operations. For each j, it computes all B[i,i+ j] by summing B[i, i + j − 1] with A[i + j]. This works since the value B[i, i + j − 1] were already computed in the previous iteration. The double for loop requires $O(n) \cdot O(n) = O(n^2)$ time. Therefore, the algorithm runs in $O(n^2)$.

Rubric:

- 2 pts: Finding Upper bound
- 2 pts: Correct explanation for upper bound
- 2 pts: Finding lower bound
- 2 pts: Correct explanation for lower bound
- 3 pts: Correct algo
- 3 pts: Correct time complexity

5. What Mathematicians often keep track of a statistic called their Erdős Number, after the great 20th century mathematician. Paul Erdős himself has a number of zero. Anyone who wrote a mathematical paper with him has a number of one, anyone who wrote a paper with someone who wrote a paper with him has a number of two, and so forth and so on. Supposing that we have a database of all mathematical papers ever written along with their authors:
    a. Explain how to represent this data as a graph.
    b. Explain how we would compute the Erdős number for a particular researcher.
    c. Explain how we would determine all researchers with Erdős number at most two.

Solution:

a) Create a graph where nodes represent mathematicians and edges represent co-authorship.
b) Run BFS in that graph starting from Erdős and see at which level that particular mathematician appears in the BFS tree. That will be his/her Erdős number.
c) All mathematicians within the first 2 levels of the BFS tree (staring from Erdős) have Erdős numbers of at most two.

Rubric:
2 pts each

6. Given a DAG, give a linear-time algorithm to determine if there is a simple path that visits all vertices.

Solution:
Find a topological order. See if there is a path that goes through the nodes of the graph in that topological order. If there is, then this will be the longest path we are looking for. In fact, if this path exists, it gives us a strict precedence order from the starting node to the end node on the path, and therefore the graph can only have one topological ordering. [Any other ordering of the nodes in the graph will violate this precedence order because if we move the position of any node in that order, we will end up with at least one edge that goes in the opposite direction of the topological order.] So, if our topological order does not provide such a path, then we can conclude that such a path does not exist in the graph.

Rubric:

- 4 pts: Correct algorithm
- 4 pts: Correct explanation

Ungraded Problems

1. What is the worst-case performance of the procedure below?

```
c = 0
i = n
while i>1 do
        for j = 1 to i do
                c = c+1
        end for
        i = floor(i/2)
end while
return c
```

Provide a brief explanation for your answer.

Solution:
There are i operations in the for loop and the while loop terminates when i becomes 1. The total time is

$n + \text{floor}(n/2) + \text{floor}(n/4) + \ldots.. \leq (1 + \frac{1}{2} + \frac{1}{4} + \ldots.).n \leq 2n = O(n)$