CSCI 570 - Spring 2022 - HW2 Solution

1 Graded Problems

1. What is the tight upper bound to the worst-case runtime performance of the procedure below?

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

Solution:

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

$$n + \lfloor n/2 \rfloor + \lfloor n/4 \rfloor + \cdots \leq (1 + 1/2 + 1/4 + \cdots) \cdot n \leq 2n = \Theta(n).$$

Rubric (5 pts):

- 2 pts: if bound correctly found as $\Theta(n)$
- 3 pts: Provides a correct explanation of the runtime
- 2 pts total if bound given is $\Theta(n \log n)$ (i.e., not a tight upper bound)

2. Arrange these functions under the O notation using only $=$ (equivalent) or $\subset$ (strict subset of):

   (a) $2^{\log n}$
   (b) 2^{3n}
   (c) $n^{n\log n}$
   (d) $\log n$
   (e) $n\log\left(n^2\right)$
   (f) n^{n^2}
   (g) $\log(\log(n^n))$

   E.g. for the function n, $n+1$, n^2, the answer should be

   $$O(n+1) = O(n) \subset O(n^2).$$

   Solution:

   First separate functions into logarithmic, polynomial, and exponential. Note that

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

   we have:

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

   - Since
     $$\log n \le 1\cdot\log(n\log n) = \log(\log(n^n)),$$
     so $\log n = O(\log(\log(n^n)))$. On the other hand,
     $$\log(\log(n^n)) = \log(n\log n) \le \log(n^2) = 2\cdot\log n,$$
     so $\log(\log(n^n)) = O(\log n)$. Thus
     $$O(\log n) = O(\log(\log(n^n))).$$
   - Since every logarithmic grows slower than every 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):

- -2 pts for each "inversion" in the order
For example, $O(2^{3n}) \subset O(n^{n^2}) \subset O(n^{n\log n})$ has one inversion and $O(n^{n\log n}) \subset O(n^{n^2}) \subset O(2^{3n})$ has two inversions
- -2 pts for any $=$ mistaken to be $\subset$ or vice versa

3. 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 you think it is true or false and give a proof or counterexample.

   (a) $f_1(n) \cdot f_2(n) = O\left(g_1(n) \cdot g_2(n)\right)$

   (b) $f_1(n) + f_2(n) = O\left(\max\left(g_1(n), g_2(n)\right)\right)$

   (c) $f_1(n)^2 = O\left(g_1(n)^2\right)$

   (d) $\log_2 f_1(n) = O\left(\log_2 g_1(n)\right)$

Solution:

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

$$f_1(n) \leq c_1 \cdot g_1(n) \text{ and } f_2(n) \leq c_2 \cdot g_2(n)$$

for n sufficiently large.

(a) True.

$$f_1(n) \cdot f_2(n) \leq c_1 \cdot g_1(n) \cdot c_2 \cdot g_2(n) = (c_1 c_2) \cdot (g_1(n) \cdot g_2(n)).$$

(b) True.

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

(c) True.

$$f_1(n)^2 \leq (c_1 \cdot g_1(n))^2 = c_1^2 \cdot g_1(n)^2.$$

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

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

Rubric (4 pts for each subproblem):

- 1 pts: Correct T/F claim
- 3 pts: Provides a correct explanation or counterexample

4. 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 G contains one.

   Solution:

   Without loss of generality assume that G is connected. Otherwise, we can compute the connected components in $O(m + n)$ time and deploy the below algorithm 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, G has a cycle and there exists an edge $e = (u, v) \in G \setminus T$. Let w be the least common ancestor of u and v. There exist a unique path T_1 in T from u to w and a unique path T_2 in T from w to v. Both T_1 and T_2 can be found in $O(m)$ time. Output the cycle e by concatenating P_1 and P_2.

Rubric (15 pts):

- No penalty for not mentioning disconnected case.
- 7 pts: for detecting whether G contains a cycle
- 5 pts: for finding (the edges in) a cycle if G contains one
- 3 pts: describing that the runtime is $O(m + n)$ in each step (and thus total)

2 Practice Problems

1. 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 $i + j - 1 = O(n)$. 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) Consider the following algorithm:

   ```
   for $i = 1, 2, \ldots, n - 1$ do
       $B[i, i + 1] \leftarrow A[i] + A[i + 1]$
   end for
   for $j = 2, 3, \ldots, n - 1$ do
       for $i = 1, 2, \ldots, n - j$ do
           $B[i, i + j] \leftarrow B[i, i + j - 1] + A[i + j]$
       end for
   end for
   ```

   It first computes $B[i, i+1]$ for all i by summing $A[i]$ with $A[j]$. 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 $O(n^2)$.

2. Solve Kleinberg and Tardos, **Chapter 3, Exercise 6.**

   Solution:

   Proof by Contradiction: assume there is an edge $e = (x, y)$ in G that does not belong to T. Since T is a DFS tree, one of x or y is the ancestor of the other. On the other hand, since T is a BFS tree, x and y differs by at most 1 layer. Now since one of x and y is the ancestor of the other, x and y should differ by exactly 1 layer. Therefore, the edge $e = (x, y)$ should be in the BFS tree T. This contradicts the assumption. Therefore, G cannot contain any edges that do not belong to T.