Homework 2

Due: Friday, Sept 12, 11:59 PM PT

1. Arrange these functions under the Big-$\mathcal{O}$ notation in increasing order of growth rate, with $g(n)$ following $f(n)$ in your list if and only if $f(n) = \mathcal{O}(g(n))$; mention equality (of growth-rate) if $f(n) = \mathcal{O}(g(n))$ and also $g(n) = \mathcal{O}(f(n))$. Here, $\log(x) = \log_2(x)$, i.e., denotes the logarithm with base 2):

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

(10 points)

Solution:
$\mathcal{O}(\log(n)) = \mathcal{O}(\log(\log(n^n))) = \mathcal{O}(\log(n) + \log(\log(n)))$, $2^{\log(n)} = n^{\log 2} = n$,
$\mathcal{O}(\log(n!)) = \mathcal{O}(n\log(n^2)) = \mathcal{O}(n\log(n))$, $4^{3n} = 64^n$, $3^{4n} = 81^n$, $n^{n\log(n)}$, n^{n^2}
The final ordered list is
$\log(n) = \log(\log(n^n)) < 2^{\log(n)} < \log(n!) = n\log(n^2) < 4^{3n} < 3^{4n} < n^{n\log(n)} < n^{n^2}$

Rubrics:
- 1 point deducted for each swap needed to fix the arrangement.
- 1 point deducted for missing each equality.

2. For each of the following, indicate whether $f(n) \in \mathcal{O}(g(n))$, $f(n) \in \Theta(g(n))$, or $f(n) \in \Omega(g(n))$. Here, $\log(x) = \log_2(x)$, i.e., denotes the logarithm with base 2. (10 points)
(a) $f(n) = n!$ and $g(n) = 2^{2^n}$
(b) $f(n) = e^n$ and $g(n) = n^{\log n}$
(c) $f(n) = 2^n$ and $g(n) = 2^{3n}$
(d) $f(n) = n^2$ and $g(n) = 2^{\sqrt{\log n}}$
(e) $f(n) = \log n$ and $g(n) = \log(\log(6^{2n}))$

Solution:
(a) $f(n) \in \mathcal{O}(g(n))$
(b) $f(n) \in \Omega(g(n))$
(c) $f(n) \in \mathcal{O}(g(n))$
(d) $f(n) \in \Omega(g(n))$
(e) $f(n) \in \Theta(g(n))$

Rubrics:
- 2 points for each correct answer.

3. We have a connected graph $G = (V, E)$, and a specific vertex $u \in V$. Suppose we compute a DFS tree rooted at u, and obtain a tree T (remember that a DFS tree includes all nodes of G). Suppose we then compute a BFS tree rooted at u, and obtain the same tree T. Prove by **contradiction** that G is the same as T, that is, G cannot contain any edges that do not belong to T. (10 points)

Hint: For any edge (x, y) in G, how much can the level of x in the BFS T differ from the level of y in it? Further, what can we say about the locations of x and y relative to each other in the DFS T?

Solution:
Assume that G contains an edge $e = (x, y)$ that does not belong to T. Since T is a DFS tree and (x, y) is an edge of G that is not an edge of T, one of x or y is ancestor of the other. On the other hand, since T is a BFS tree if x and y belong to layer L_i and L_j respectively, then i and j differ by at most 1. Notice that since one of x or y is an ancestor of the other, we have that $i \neq j$ and hence i and j differ by exactly 1. However, combining that one of x or y is ancestor of the other and that i and j differ by 1 implies that the edge (x, y) is in the tree T. It contradicts the assumption that $e = (x, y)$ that does not belong to T. Thus, G cannot contain any edges that do not belong to T.

Rubrics:
- 2 points for looking into an edge that does not belong to T
- 5 points concluding that the levels differ by exactly 1
- 3 points for reaching a contradiction

4. Given an unweighted and undirected graph $G = (V, E)$ and an edge $e \in E$. Design an algorithm to determine whether the graph G has a cycle containing that specific edge e. Also, determine the time complexity of your algorithm with explanation. **Note**: To be eligible for full credits on this problem, the running time of your algorithm should be bounded by $\mathcal{O}(|V| + |E|)$. (10 points)

Solution:
Let the two nodes forming the edge e be s and t. Delete e from the graph G and run BFS or DFS starting from s. If t can be reached on running BFS or DFS from s, then there is a cycle in the graph containing edge e.
Time complexity: $\mathcal{O}(|V| + |E|)$. This is because BFS or DFS takes $\mathcal{O}(|V| + |E|)$ time.

Rubrics:
- 2 points for correct algorithm
- 2 points for each correct time complexity.

Ungraded Problems:

1. Given functions f_1, f_2, g_1, g_2 such that $f_1(n) = \mathcal{O}(g_1(n))$ and $f_2(n) = \mathcal{O}(g_2(n))$. For each of the following statements, decide whether you think it is true or false, and give a proof or counterexample. (12 points)

(a) $f_1(n) + f_2(n) = \mathcal{O}(\max(g_1(n), g_2(n)))$

(b) $f_1(n) \cdot f_2(n) = \mathcal{O}(g_1(n) + g_2(n))$

(c) $f_1(n)^2 = \mathcal{O}(g_1(n)^2)$

(d) $\log f_1(n) = \mathcal{O}(\log(g_1(n)))$

Solution:

(a) True.

$f_1(n) + f_2(n) \le c_1 g_1(n) + c_2 g_2(n) \le (c_1 + c_2)(g_1(n) + g_2(n))$
$\le 2(c_1 + c_2)\max(g_1(n), g_2(n))$

(b) False.

Let $f_1(n) = f_2(n) = g_1(n) = g_2(n) = n$.
$f_1(n) \cdot f_2(n) = n^2$, $g_1(n) + g_2(n) = 2n \Longrightarrow n^2 \notin \mathcal{O}(n)$

(c) True.

$f_1(n)^2 \le (c_1\, g_1(n))^2 = c_1^2 g_1(n)^2$

(d) False.

Consider $f_1(n) = 2$ and $g_1(n) = 1$. Then $\log f_1(n) = 1 \neq \mathcal{O}(\log g_1(n)) = \mathcal{O}(0)$.

Rubrics:

- 2 points for each True/False answer.
- 2 points for each correct explanation.

2. Consider the following prime filtering algorithm that outputs all the prime numbers in $2, \ldots, n$ (the pseudo code is presented in Algorithm 1). (10 points)

Algorithm 1 Prime Filtering

```
1: Input: a positive integer n ≥ 2
2: initialize the Boolean array isPrime such that isPrime(i) = True for i = 2, ..., n
3: for i = 2 ... n do
4:    for j = 2 ... ⌊n/i⌋ do
5:       if i × j ≤ n then
6:          isPrime(i × j) ← False
7:       end if
8:    end for
9: end for
```

(a) Please prove this algorithm is correct (that is, a positive integer k that $2 \le k \le n$ is a prime if and only if $isPrime(k) =$ **True**). (5 points)

(b) Please calculate the time complexity under the Big-$\mathcal{O}$ notation. (5 points)

Solution:

Part a:
A prime number (or a prime) is a natural number greater than 1 that is not a product of two smaller natural numbers. That is, for any prime number p, there does not exist two positive integers $x, y \geq 2$ such that $p = x \cdot y$. On the other hand, for any non-prime number z, there exists two positive integers $x, y \geq 2$ such that $z = x \cdot y$. Thus, any non-prime number z would be removed by Line 6 of Algorithm 1 when $i = x$ and $j = y$, and the prime numbers would be filtered.

Part b:
By direct calculation, we have

$$\sum_{i=2}^{n} \left\lfloor \frac{n}{i} \right\rfloor \leq \sum_{i=2}^{n} \frac{n}{i} = n \cdot \left(\sum_{i=1}^{n} \frac{1}{i} - 1 \right) = \mathcal{O}(n \log n)$$

where the last step follows from the fact that $\sum_{i=1}^{n} \frac{1}{i} = \mathcal{O}(\log n)$. The sum $\sum_{i=1}^{n} \frac{1}{i}$ is called the k-th Harmonic Number, which is important in various branches of number theory. To show that $\sum_{i=1}^{n} \frac{1}{i} = \mathcal{O}(\log n)$, one can simply verify the following inequalities:

$$\int_{1}^{n+1} \frac{dx}{x} = \sum_{i=1}^{n} \int_{i}^{i+1} \frac{dx}{x} \leq \sum_{i=1}^{n} \frac{1}{i} \leq 1 + \sum_{i=2}^{n} \int_{i-1}^{i} \frac{dx}{x} = 1 + \int_{1}^{n} \frac{dx}{x}$$

Following the Newton-Leibniz formula, one can show that $\int_{1}^{n+1} \frac{dx}{x} = \ln(n+1)$, which concludes the proof of $\sum_{i=1}^{n} \frac{1}{i} = \mathcal{O}(\log n)$.

Rubrics:
- Part a: 5 points for correct proof.
- Part b: 5 points for correct time complexity.

3. A directed graph $G = (V, E)$ is **singly connected** if G contains at most one simple path from u to v for all vertices $u, v \in V$. Construct an algorithm using DFS to determine whether or not a directed graph is singly connected. Also, determine the time complexity of your algorithm and justify your answer. (10 points)

Solution:
Run a DFS from each vertex in the graph, keeping track of the type of edges encountered during the search. Specifically, check for forward or cross edges in the same component, as their presence indicates multiple paths between some vertices. If any such edges are found, the graph is not singly connected. If all DFS runs complete without detecting forward or cross edges, the graph is singly connected.
Time complexity: $\mathcal{O}\big(|V|(|V| + |E|)\big)$. Since DFS from each vertex takes $\mathcal{O}(|V| + |E|)$ time, and we perform DFS for $\mathcal{O}(|V|)$ vertices.

Rubrics:
- 8 points for correct algorithm
- 2 points for each correct time complexity.