CS 344: Design and Analysis of Computer Algorithms

Instructor: Kangning Wang

Homework 1: Due on **February 7, 2025**

Assumptions for Problems 1 and 2

In Problem 1, all named functions are strictly positive. In Problems 1 and 2, the limits in the notations $O, o, \Omega, \omega, \Theta$ are always $n \to +\infty$.

Problem 1

Is each of the following statements true or false? Give proofs for your answers.

1. If $f(n) \in O\big(g(n)\big)$, then $\log_2(10 + f(n)) \in O\big(\log_2(10 + g(n))\big)$.

2. If $f(n) \in O\big(g(n)\big)$, then $2^{f(n)} \in O\big(2^{g(n)}\big)$.

3. If $f(n) \in o\big(g(n)\big)$, then $2^{f(n)} \in o\big(2^{g(n)}\big)$.

4. If $f_1(n) \in O\big(g_1(n)\big)$ and $f_2(n) \in O\big(g_2(n)\big)$, then $f_1(n) + f_2(n) \in O\big(g_1(n) + g_2(n)\big)$.

5. We know that $f_i(n) \in O\big(g_i(n)\big)$ for every $i \in \mathbb{Z}^+$. Define the functions F, G as $F(k) = \sum_{i=1}^{k} f_i(i)$ and $G(k) = \sum_{i=1}^{k} g_i(i)$ for every $k \in \mathbb{Z}^+$. Then $F(n) \in O\big(G(n)\big)$.

6. It is possible that $f(n) \in \omega\big(g(n)\big)$ and $g(n) \in \Omega\big(f(n)\big)$.

7. It is possible that $f(n) \in \Theta\big(g(n)\big)$ and $g(n) \notin \Theta\big(f(n)\big)$.

Solution.

1. True. Suppose $f(n) \in o\big(g(n)\big)$ then there exists constants $c > 0$ and n_0 such that for all sufficiently large n, $f(n) \leq c \cdot g(n)$. We need to show that $log_2(10 + f(n)) \in O(log_2(10 + g(n)))$, meaning there exists a constant C such that $log_2(10 + f(n)) \leq Clog_2(10 + g(n))$. Since the logarithm is a monotonically increasing function, $log_2(10 + f(n)) \leq log_2(10 + cg(n)) = log_2(c) + log_2(10 + g(n))$. Letting C = 1 and $C' = log_2(c)$, $log_2(10 + f(n)) \leq Clog_2(10 + g(n)) + C'$. Thus, $log_2(10 + f(n)) \in O\big(log_2(10 + g(n))\big)$.

2. True, Suppose $f(n) \in O(g(n))$, then there exists constants $c > 0$ that for all $n > N$ $f(n) \leq c \cdot g(n)$. We need to show that $2^{f(n)} \leq d \cdot 2^{g(n)}$ for some constant $d > 0$. We can let $d = 2^c$ and thus $2^{f(n)} \leq 2^{c \cdot g(n)} = d \cdot 2^{g(n)}$.

3. True, Suppose $f(n) \in o(g(n))$ then $\forall c > 0, f(n) < c \cdot g(n)$ for all sufficiently large n. We need to show that for our choice of d that $2^{f(n)} < d \cdot o(2^{g(n)})$ we can let $c = log_2(d)$ and thus $f(n) < c \cdot g(n) \implies 2^{f(n)} < 2^{c \cdot g(n)} = d \cdot 2^{g(n)}$.

4. True. Suppose $f_1(n) \in o(g_1(n))$ and $f_2(n) \in o(g_2(n))$, then there exists constants $c_1, c_2 > 0$ and n_0 such that for all sufficiently large n, $f_1(n) \leq c_1 \cdot g_1(n)$, $f_2(n) \leq c_2 \cdot g_2(n)$. Adding these, $f_1(n) + f_2(n) \leq c_1 \cdot g_1(n) + c_2(g_2(n))$. Let C = $\max(c_1, c_2)$, $f_1(n) + f_2(n) \leq C(g_1(n) + g_2(n))$. Thus, by definition, $f_1(n) + f_2(n) \in o(g_1(n) + g_2(n))$.

5. True, Suppose $f_i(n) \in O(g_i(n))$ for every $i \in \mathbb{Z}^+$, then there exist constants $c_i > 0$ such that for all $n > N_i$, $f_i(n) \leq c_i \cdot g_i(n)$. Let $c = \max c_1, c_2, ..., c_k$ and $N = \max N_1, N_2, ..., N_k$, then for all $n \geq N$, $F(n) = \sum_{i=1}^{n} f_i(i) \leq \sum_{i=1}^{n}(c \cdot g_i(i)) = c \cdot \sum_{i=1}^{n} g_i(i) = c \cdot G(n)$, thus $F(n) \in O(G(n))$.

6. False. Suppose $f(n) \in \omega(g(n))$ and $g(n) \in \Omega(f(n))$, then by definition of ω, $\forall c > 0$ $f(n) > c \cdot g(n)$ for all $n > N$. By definition of Ω, $\exists d > 0$ such that $g(n) > d \cdot f(n)$ for all $n > M$. If we let $K = max(N, M)$, then for all $n > k$ $f(n) > c \cdot g(n) > c \cdot d \cdot f(n)$, which implies that $1 > c \cdot d$, which is a contradiction.

7. False. Suppose $f(n) \in \Theta(g(n))$, then by definition, there exists positive constants c_1, c_2, and n_0 such that for all sufficiently large n, $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$. Dividing, $\frac{1}{c_2} f(n) \leq g(n) \leq \frac{1}{c_1} f(n)$. This satisfies the definition of $g(n) \in \Theta(f(n))$, meaning that $f(n) \in \Theta(g(n)) \iff g(n) \in \Theta(f(n))$.

Problem 2

Evaluate and simplify each of the following functions of n using the Θ notation. For example, $2n^2 + 1$ should be simplified into $\Theta(n^2)$. Clearly explain each of your answers.

1. $3^{3n} + 344^n + n^{344}$.

2. $2^n + 6^{\sqrt{n}}$.

3. $4n + n \ln \sqrt{n}$.

4. $\sum_{k=1}^{n} k^2$.

5. $\sum_{k=1}^{n} k \ln^2 k$.

6. $\sum_{k=1}^{n} 4^k$.

7. $\sum_{k=1}^{n} e^{\sqrt{k}}$.

8. $\sum_{k=1}^{n} (\sin k)^8$.

Solution.

1. $\Theta(344^n)$ because exponentials grow faster than polynomials and 3^{3n} simplifies to 27^n, which is slower than 344^n.

2. $\Theta(2^n)$ because the exponential grows faster than a sub-exponential

3. $\Theta(n \ln n)$ because we can simplify $n \ln \sqrt{n}$ to $\frac{1}{2} n \ln n$.

4. $\Theta(n^3)$ because the sum of squares of the first n natural numbers is $\frac{n(n+1)(2n+1)}{6}$ and the dominating term is $\frac{2n^3}{6} = \frac{n^3}{3}$.

5. $\Theta(n^2 \ln^2(x))$, We can approximate this sum with an integral which is $\int x \ln^2 x \, dx$. After some integration, $\frac{x^2(2\ln^2(x) - 2\ln(x) + 1)}{4}$ which is $\Theta(n^2 \ln^2 n)$.

6. $\Theta(4^n)$ because the sum of a geometric series is $\frac{a_1(1-r^n)}{1-r}$, and since $r = 4$ in this case the dominating term is 4^n.

7. $\Theta(\sqrt{n}e^{\sqrt{n}})$ because for large n, $\sum_{k=1}^{n} e^{\sqrt{k}} \approx \int_1^n e^{\sqrt{x}} dx$. Using substitution, $\int e^u \cdot 2u \, du$. Using integration by parts, $\int u e^u du = u e^u - \int e^u du = u e^u - e^u$. Multiplying by 2, $\approx 2((\sqrt{n}e^{\sqrt{n}} - e^{\sqrt{n}}) - (\sqrt{1}e^1 - e^1))$. Thus, the dominant term is $2\sqrt{n}e^{\sqrt{n}}$.

8. $\Theta(n)$ because $\sin^8(x)$ is bounded by $(0, 1)$, so the sum to n is bounded by n.

Problem 3

In the stable matching problem, is it possible that there are 10! different stable matchings in an instance with 10 vertices on each side? Give a proof for your answer. (Note that we assume strict preferences, that is, no vertex is indifferent between two vertices on the other side.)

Solution.
Lets say we have a stable matching problem with 10 vertices on each side, which we will name $M = \{m_1, m_2, ..., m_{10}\}$ and $W = \{w_1, w_2, ..., w_{10}\}$.
We first propose that there are exactly 10! possible perfect matchings. This is simply a combinatorics problem: if arbitrarily match members of set M to members of set W, m_1 has 10 choices of w_n that it can pair to, m_2 has 9, m_3 has 8, and so on, making $10 * 9 * ... * 2 * 1 = 10!$ possible perfect matchings.
With this knowledge, if we prove that a single one of these perfect matchings is unstable, then we have proven that there cannot be 10! different stable matchings for this problem.
Consider a scenario where m_1 has the following preferences: $w_1 \succ w_2 \succ ...$ and m_2 has the following preferences: $w_2 \succ w_1 \succ ...$. Meanwhile, w_1 has the following preferences: $m_1 \succ m_2 \succ ...$ and w_2 has the following preferences: $m_2 \succ m_1 \succ ...$. Disregarding all other members of both sets, there exists at least one perfect matching where m_1 is paired with w_2 and m_2 is paired with w_1. This matching is unstable, so the scenario does not have 10! different stable matchings.

Problem 4

We have an $n \times n$ matrix A. The entries in the matrix are distinct real numbers. Our goal is to select n entries so that

- exactly one entry is selected in each row,

- exactly one entry is selected in each column, and

- any unselected entry in the matrix is either smaller than the selected entry in the same row or larger than the selected entry in the same column.

Prove that this is always possible by reducing our problem to the stable matching problem.

Solution.
We can map this problem to the stable matching problem with n vertices on each side. Map all rows to one side of the stable matching problem, which we will call M, such that m_i corresponds to row i in A. Map all columns to the other side, which we will call W. Then construct the preference list as follows:

For a vertice of M, which we will call m_i, we go through its corresponding row and construct m_i's preference list such that if $A_{ij} < A_{ik}$, $j \neq k$, then w_j is lower on m_i's preference list than w_k. In other words, m_i's preference list is created by sorting its elements in A from largest to smallest and then taking their column index.

For a vertice of W, which we will call w_j, we do the same process as above, except our requirement is $A_{ij} > A_{kj}$, where $i \neq k$. In other words, we sort the elements of column j from smallest to largest, then take their indices.

From this process, we create a full stable matching problem with two sides of n vertices each as well as full preference lists. We have now mapped our problem to the stable matching problem and can use the Gale-Shapley Algorithm to solve it.

Problem 5

A *clique* in an undirected graph G is a set C of vertices in G with the property that every pair of distinct vertices in C are adjacent (i.e., connected by an edge) in G.

An *independent set* in an undirected graph G is a set I of vertices in G with the property that every pair of distinct vertices in I are not adjacent in G.

Prove the following statement on Ramsey numbers: Every undirected graph of 6 vertices contains either a clique of 3 vertices or an independent set of 3 vertices (or both).

Solution. Suppose G is a graph of 6 vertices. We can consider a vertex $v \in G$. Clearly v can have 0 to 5 edges connected to it, in other words $0 \leq \deg(v) \leq 5$. We can consider the following cases:

If v has degree 3 or more, then we can find at least 3 neighbors of v. Let us call them x, y, z. If x, y, z have at least one edge that connects any of the two members, then the two connected members form a clique with v. If they do not have any connection between them, then x, y, z form an independent set.

If v has degree 2 or less, then we can consider any combination of three vertices that are not connected to v, which we can call a, b, c. We are guaranteed at least three of these vertices because v is degree 2 or less. If there are edges connecting every pair of distinct vertices of a, b, c, a, b, c form a clique. If there are two who do not have an edge between them, the two unconnected vertices form an independent set with v.

Problem 6

A *quasi-kernel* in a directed graph $G = (V, E)$ is an *independent set* (See Problem 5) Q of vertices with the property that for any vertex $v \in V \setminus Q$, there exists a vertex $u \in Q$, so that there is a simple path of length at most 2 from u to v.

As a non-obvious fact, there is at least one quasi-kernel in every directed graph. (You can stop reading now and try to prove it yourself.) Here is an algorithm that finds one.

Algorithm 1: Finding a quasi-kernel

Name the vertices $\{v_1, v_2, \ldots, v_n\}$;
Initially all vertices are active;
foreach i *from* 1 *to* n **do**
   if v_i *is active* **then**
      foreach *edge from* v_i *to* v_j *in the graph* G **do**
         | Deactivate v_j if $j > i$;
      end
   end
end
foreach i *from* n *down to* 1 **do**
   if v_i *is active* **then**
      foreach *edge from* v_i *to* v_j *in the graph* G **do**
         | Deactivate v_j if $j < i$;
      end
   end
end
Output the active vertices as a quasi-kernel;

Prove the correctness of Algorithm 1, and write down its running time (assuming a good implementation).

Solution.

Description of the Algorithm
The first loop of the algorithm goes through all the vertices in the graph and deactivates all the vertices that are outgoing from the current vertex if they have a higher index. The second loop goes through all the vertices in the graph in reverse order and deactivates all the vertices that are outgoing from the current vertex if they have a lower index. The vertices that are not deactivated are output as the quasi-kernel.
To prove this algorithm, we need to show that for any $v \in V \setminus Q$, there exists a vertex $u \in Q$ such that there is a simple path of length at most 2 from u to v.

Proof of the Algorithm
Let $v \in V \setminus Q$. Thus v must have been either deactivated in the first or the second loop.
If v was deactivated in the first loop, then there exists a vertex w with index lower then v's index with an edge from u to v.
If w is still active at then end of the algorithm, then w is an element of the output of the algorithm. There is a path of length 1 from $w \to v$, since w deactivated v.
If w is later deactivated in the second loop, it would have been due to another active vertex with a higher index, which we call x. Since we reached x in the second loop, it is impossible to deactivate it, as the loop will proceed down the indices past x, and a lower-index vertex cannot deactivate a higher-index vertex in the second loop. Therefore we have a path of length 2 from $x \to w \to v$.
If v was deactivated in the second loop then there exist a vertex w with index higher then v's index with an edge from u to v. Again, w cannot be deactivated if it was reached in the second loop, so there is a path of length 1 from w to v
From these cases, we have concluded that the maximum path length from a member of the output of the algorithm to a vertex $v \in V \setminus Q$ is 2, so the output is a quasi-kernel.

Running Time
There are two loops in this algorithm. The first loop goes through all the vertices and for each vertex that is still active, it goes through all the edges. The second loop goes through all the vertices and for each vertex that is still active, it goes through all the edges. The worst-case is that all vertices and edges have to be considered; improvements can be made since the presence of edges will turn off some vertices, but this correction would be some coefficient on the two terms, or constant, so it does not matter in big O. Besides, optimizing how edges turn off vertices would no longer be worst case. Thus the running time of this algorithm is $O(V + E)$ where V is the number of vertices and E is the number of edges.