CS 344: Design and Analysis of Computer Algorithms

Instructor: Kangning Wang

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

Acknowledgment: Our team members are Alexander Lin (al1655), Pranav Tikkawar (pt422), and Ivan Zheng (iz72) .

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. Here is my solution.
2.
3.
4.
5.
6.

7.

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. Here is my solution.
2.
3.
4.
5.
6.
7.
8.

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. Here is my solution.

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. Here is my solution.

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, ..., 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.