Homework #2

You do not need to turn in these problems. The goal is to be ready for the in-class quiz that will cover the same or similar problems, and to prepare you for the exams.

Problem 1: Truthfulness in Stable Matching

For this problem, we will explore the issue of truthfulness in the Stable Matching Problem and specifically in the Gale-Shapley algorithm. The basic question is: can a man or a woman end up better off by lying about his or her preferences? More concretely we suppose each participant has a true preference order. Now consider a woman w. Suppose w prefers man m to m_0, but both m and m_0 are low on her list of preferences. Can it be the case that by switching the order of m and m_0 on her list of preferences (i.e., by falsely claiming that she prefers m_0 to m) and running the algorithm with this false preference list, w will end up with a man m_{true} that she truly prefers to both m and m_0? (We can ask the same question for men but will focus on the case of women for the purposes of this question.) Resolve this question by doing one of the following two things:

Give a proof that, for any set of preference lists, switching the order of a pair on the list cannot improve a womans partner in the Gale-Shapley algorithm; or

Give an example of a set of preference lists for which there is a switch that would improve the partner of a woman who switched preferences.

Solution

It is possible for a woman to "game" the system.
Assume we have three men, m_1 to m_3 and three women w_1 to w_3 with the following (true) preferences:

- m_1: w_3, w_1, w_2
- m_2: w_1, w_3, w_2
- m_3: w_3, w_1, w_2
- w_1: m_1, m_2, m_3
- w_2: m_1, m_2, m_3
- w_3: m_2, m_1, m_3

First consider a possible execution of Gale-Shapley with these true preference lists. First m_1 proposes to w_3 then m_2 proposes to w_1. Then m_3 proposes to w_2 and w_1 and gets rejected, finally proposes to w_2 and is accepted. This execution forms pairs (m_1, w_3), (m_2, w_1), and (m_3, w_2), thus pairing w_3 with m_1, who is her second choice.
Now consider what happens when w_3 pretends that her preferences are m_2, m_3, m_1 instead. Then the execution might unfold as follows. Man m_1 proposes to w_3, m_2 to w_1, then m_3 to w_3. She accepts the proposal, leaving m_1 alone. m_1 proposes to w_1, which causes her to leave m_2, who consequently proposes to w_3 (which is exactly what w_3 wants).

Problem 2: Understanding Stable Matching Optimality

We say a man m is a *valid partner* for woman w if there exists some stable matching in which they are matched. A *man-optimal* stable matching is the one in which every man receives his best valid partner. A *woman-pessimal* stable matching is the one in which every woman receives her worst valid partner.

Recall from class that the Gale-Shapley algorithm returns the man-optimal stable matching. Prove that this stable matching is also woman-pessimal. (You may use the man-optimality of Gale-Shapley without proof.)

Solution

Suppose that the Gale-Shapley algorithm returns a stable matching S with a pair (m, w) such that m is not the worst valid partner of w. Then there is a stable matching S' in which w is paired with a man m' whom she likes less than m. It is obvious then that in S', m must also be paired with another woman $w' \neq w$. Note though that we obtained $(m, w) \in S$ using the Gale-Shapely algorithm, so it follows that w is the best valid partner for m, making m prefer w to w'. So it must be that (m, w) is an instability in S', contradicting our finding that S' was in fact stable.

Problem 3: Stable Marriage Runtime

Prove that the number of proposals made in the course of the Gale-Shapley algorithm is $\Omega(n^2)$

in the worst case.

Solution

Remember we need to show that the **worst case** input, the input causing the longest computational running time, makes a run time of at least n^2 (asymptotically). Therefore it is sufficient to show example of inputs that have a running time in $\Omega(n^2)$.
Suppose there are n men and women each, and every man ranks all the women in the same order. Since each man has to end up with a different woman, and since they all have the same preference list, each man must end up with a different rank woman in his preference list. (Eg, no two men can end up with the woman they ranked "5".) If a man ends up with woman he ranked number i, he must have proposed i times in the Gale-Shapley algorithm (and was rejected $i-1$ times). It follows that overall there are $\sum_{i=1}^{n} i$ proposals. Recall that for any integer k, $\sum_{j=1}^{k} j = \frac{k(k+1)}{2}$. Therefore the amount of proposals, $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n \in \Omega(n^2)$, as we needed to show.

Problem 4: Properties of Big-Oh

(a) Prove that for any constant $d > 0$, $f(n) = O(d \cdot g(n))$ iff $f(n) = O(g(n))$. (Note that this justifies dropping multiplicative constants in O notation. We won't say something like: the algorithm runs in $O(2n^2)$ time.)

Solution

Note that we are asked to prove an *if and only if* statement, so we must prove both directions.
Suppose $f(n) = O(g(n))$. Then we know that $\exists c, n_0 \in \mathbb{R} : \forall n \geq n_0 : f(n) \leq cg(n)$. Let $c' = d^{-1}c$, then it should be obvious that $\forall n \geq n_0 : f(n) \leq c'dg(n) = cg(n)$, satisfying the necessary conditions for $f(n) = O(dg(n))$.
Now for the other direction. Suppose $f(n) = O(dg(n))$. Then $\exists c, n_0 \in \mathbb{R} : \forall n \geq n_0 : f(n) \leq cdg(n)$. So let $c' = cd$, and it follows that $\forall n \geq n_0 : f(n) \leq c'g(n) = cdg(n)$, satisfying our definition for $f(n) = O(g(n))$.

(b) Prove that if $f(n) = O(h_1(n))$ and $g(n) = O(h_2(n))$ then $f(n)+g(n) = O(\max\{h_1(n), h_2(n)\})$. (Note that this property is used all the time in algorithms (usually implicitly!): If your algorithm first does something that's $O(n)$ and then something that's $O(n^2)$, we claim that the overall algorithm is $O(n^2)$.)

Solution

Suppose $f(n) = O(h_1(n))$ and $g(n) = O(h_2(n))$, then there exists $c_1, c_2, n_1, n_2 \in \mathbb{R}$ such that $f(n) \leq c_1 h_1(n)$ for all $n \geq n_1$, and $g(n) \leq c_2 h_2(n)$ for all $n \geq n_2$. Let $n' = max\{n_1, n_2\}$ and $c' = max\{c_1, c_2\}$. Then we can sum the two relations to obtain $f(n)+g(n) \leq c_1 h_1(n) + c_2 h_2(n) \leq c'(h_1(n) + h_2(n))$ for all $n \geq n'$. Recall that $\forall x, y \in \mathbb{R} : x + y \leq 2max\{x, y\}$. Then it follows that $f(n) + g(n) \leq c'(2max\{h_1(n), h_2(n)\}) = 2c'max\{h_1(n), h_2(n)\}$ for all $n \geq n'$, and we can conclude that $f(n) + g(n) = O(\max\{h_1(n), h_2(n)\})$.

Problem 5: Efficiency

For a given problem, suppose you have two algorithms: A_1 and A_2 with worst-case time complexity of $T_1(n)$ and $T_2(n)$, respectively. For each part, answer the following four questions:

- Is $T_1(n) = O(T_2(n))$?
- Is $T_1(n) = \Omega(T_2(n))$?
- Is $T_1(n) = \Theta(T_2(n))$?
- If your goal is to pick the fastest algorithm for large n, would you pick A_1 or A_2?

(a) $T_1(n) = 5\sqrt{n} + \log_2 n + 3$ and $T_2(n) = 100\log_2 n + 25$

Solution

- Is $T_1(n) = O(T_2(n))$? **NO**
- Is $T_1(n) = \Omega(T_2(n))$? **YES**
- Is $T_1(n) = \Theta(T_2(n))$? **NO**
- If your goal is to pick the fastest algorithm for large n, would you pick A_1 or A_2? $\mathbf{A_2}$.

(b) $T_1(n) = 5\sqrt{n} + \ln n$ and $T_2(n) = \sqrt{n} \cdot \log_2 n$

Solution

- Is $T_1(n) = O(T_2(n))$? **YES**
- Is $T_1(n) = \Omega(T_2(n))$? **NO**
- Is $T_1(n) = \Theta(T_2(n))$? **NO**
- If your goal is to pick the fastest algorithm for large n, would you pick A_1 or A_2? $\mathbf{A_1}$

(c) $T_1(n) = 5\log_{10} n - 20$ and $T_2(n) = \frac{1}{2}\log_2 n$

Solution

- Is $T_1(n) = O(T_2(n))$? **YES**
- Is $T_1(n) = \Omega(T_2(n))$? **YES**
- Is $T_1(n) = \Theta(T_2(n))$? **YES**
- If your goal is to pick the fastest algorithm for large n, would you pick A_1 or A_2? $\mathbf{A_2}$

(d) $T_1(n) = 10n^{2/4}$ and $T_2(n) = \sqrt{n} \cdot \log_2 n$

Solution

- Is $T_1(n) = O(T_2(n))$? **YES**
- Is $T_1(n) = \Omega(T_2(n))$? **NO**
- Is $T_1(n) = \Theta(T_2(n))$? **NO**
- If your goal is to pick the fastest algorithm for large n, would you pick A_1 or A_2? $\mathbf{A_1}$.

(e) $T_1(n) = 10^{(2\log_{10} n)} - 2n$ and $T_2(n) = 5n^3 + 10n - 3$

Solution

- Is $T_1(n) = O(T_2(n))$? **YES**
- Is $T_1(n) = \Omega(T_2(n))$? **NO**
- Is $T_1(n) = \Theta(T_2(n))$? **NO**
- If your goal is to pick the fastest algorithm for large n, would you pick A_1 or A_2? $\mathbf{A_1}$

Problem 6: Direct proof of Θ

Prove that $\log(n!) = \Theta(n \log n)$.

Solution

We need to show the following:

1. $c_1 n \log n \geq \log(n!) \qquad c_1 > 0, n \geq n_0$

2. $c_2 n \log n \leq \log(n!) \qquad c_2 > 0, n \geq n_0$

First, let us prove the first statement, or that $\log(n!) = O(n \log(n))$ Remember that $\log(ab) = \log(a) + \log(b)$.

$$\log(n!) = \log(n) + \log(n-1) + ... + \log(1)$$

Now, $n \log(n)$ is n terms of log(n) added together. It follows that

$$\log(n) + \log(n-1) + ... + \log(1) \leq \log(n) + \log(n)... + \log(n)$$

This is true for all $n \geq 1$, because there are n terms on both the left and right sides of the equation. On the left side, every term is less than or equal to every term on the right side. Let $c_1 = 1$, $n_0 = 1$.

$$n \log n \geq \log(n!) \qquad \forall n \geq 1$$

The first statement is proven. Let's prove the second statement, that $\log(n!) = \Omega(n \log(n))$ Notice that $\log(n) + \log(n-1) + ... + \log(1)$ can be split into two sequences:

$$\log(n) + \log(n-1) + ... + \log(\frac{n}{2} + 2) + \log(\frac{n}{2} + 1)$$

and

$$\log(\frac{n}{2}) + \log(\frac{n}{2} - 1) + ... + \log(1)$$

Notice that $(\frac{n}{2}) \log(\frac{n}{2})$ has the same number of terms the first half of the equation does, and each term is smaller.

$$(\frac{n}{2}) \log(\frac{n}{2}) = \log(\frac{n}{2}) + \log(\frac{n}{2}) + ... + \log(\frac{n}{2})$$

$$\log(\frac{n}{2}) + \log(\frac{n}{2}) + ... + \log(\frac{n}{2}) \leq \log(n) + \log(n-1) + ... + \log(\frac{n}{2} + 2) + \log(\frac{n}{2} + 1)$$

This is true because there are $\frac{n}{2}$ terms on both sides of the inequality, and each term on the left is smaller than each term on the right. Since half of $\log(n!)$ expanded is greater than $(\frac{n}{2}) \log(\frac{n}{2})$,

$$(\frac{n}{2}) \log(\frac{n}{2}) \leq \log(n!)$$

However, we need to prove the inequality for $c_2 n \log n \leq \log(n!)$, not for $c_2 n \log \frac{n}{2}$.
Notice that:

$$(\frac{n}{2}) \log(\frac{n}{2}) \leq \frac{n}{4} \log(n)$$

Solution

This can be shown through algebra:

$$(\frac{n}{2})\log(\frac{n}{2}) \leq \frac{n}{4}\log(n)$$

$$\frac{n}{2}(\log(n) - \log(2)) \leq \frac{n}{4}\log(n)$$

Remember the log is base 2, so $\log(2) = 1$

$$\frac{n}{2}(\log(n)) - \frac{n}{2} \leq \frac{n}{4}\log(n)$$

$$2(\log(n)) - 2 \leq \log(n)$$

$$\log(n) \leq 2$$

This is true $\forall n \geq 4$. So, by transitivity, since

$$(\frac{n}{2})\log(\frac{n}{2}) \leq \log(n!)$$

and

$$(\frac{n}{4})\ log(n) \leq (\frac{n}{2})\log(\frac{n}{2}) \qquad \forall n \geq 4$$

then it must be true that

$$(\frac{n}{4})\ log(n) \leq \log(n!) \qquad \forall n \geq 4$$

Therefore, we have proven $\log(n!) = \Omega(n\log(n))$ and $\log(n!) = O(n\log(n))$, so

$$\log(n!) = \Theta(n\log(n))$$