Sample Solution to Problem Set 1

(The solutions to the exercises from the book are adapted from those provided by the authors.)

1. (4 points) Chapter 1, Exercise 1, page 22.

Answer: False. Consider the example with two men M and M' and two women W and W' with the following preference lists.

- M prefers W to W'.
- M' prefers W to W'.
- W prefers M' to M.
- W' prefers M' to M.

There are no pairs at all in which the man and the woman rank each other first.

2. (12 points) Chapter 1, Exercise 4, page 23.

Answer: The algorithm is similar to the Gale-Shapley algorithm. We say that a student is either "committed" to a hospital or "free." A hospital either has available positions or is "full." The algorithm is as follows.

while some hospital h_i has available positions
    h_i offers a position to the next student s_j on its preference list
    if s_j is free
      s_j accepts the offer
    else (s_j is committed to a hospital h_k)
      if s_j prefers h_k to h_i
        s_j remains committed to h_k
      else
        s_j becomes committed to h_i
        the number of available positions in h_i decreases by 1
        the number of available positions in h_k increases by 1

Termination. The algorithm terminates in $O(nm)$ iterations because each hospital offers a position to a student at most once, and in each iteration, some hospital offers a position to some student.

Every position is filled. Suppose there are p_i positions available at hospital h_i. When the algorithm terminates, all positions are filled, because any hospital that did not fill all its available positions must have made an offer to every student; but then, all of these students would be

committed to some hospital, contradicting the assumption that the total number of students is at least the number of positions.

Stability. For the first kind of instability, suppose there are students s and s' and hospital h satisfying the instability condition. If h prefers s' to s, then h would have offered a position to s' before it offered one to s; from then on, s' would have a position at *some* hospital, and so would not be free at the end – a contradiction.

For the second kind of instability, suppose h, h', s, and s' satisfy the instability condition. Then, h must have offered a position to s', for otherwise it has p_i residents all of whom it prefers to s'. Moreover, s' must have rejected h in favor of some h'' which s/he preferred; and the final commitment of s', namely h', must be more preferred than h, contradicting the instability condition.

3. (10 points) Recursion and induction in binary codes

Digital transmission protocols transmit signals using binary codes. In order to minimize the effect of errors, it is often useful to select a code such that "similar" signals use "similar" codewords.

One such code is a list of 2^n n-bit strings in which each string (except the first) differs from the previous one in exactly one bit. Let us call such a list a *twiddling list* since we go from one string to the next by just flipping one bit.

Consider the following recursive algorithm for listing the n-bit strings of a twiddling list. If $n = 1$, the list is 0,1. If $n > 1$, first take a twiddling list of $(n-1)$-bit strings, and place a 0 in front of each string. Then, take a second copy of the twiddling list of $(n-1)$-bit strings, place a 1 in front of each string, reverse the order of the strings and place it after the first list. So, for example, for $n = 2$, the list is 00,01,11,10, and for $n = 3$, we get 000,001,011,010,110,111,101,100.

Prove by induction on n that (a) every n-bit string appears exactly once in the list generated by the algorithm, and (b) each string (except the first) differs from the previous one in exactly one bit.

Answer:

(a) The proof is by induction on n. For the base case, we consider $n = 1$. The twiddling list is $0, 1$, which contains every 1-bit string exactly once.

For the induction step, we assume that every m-bit string appears exactly once in the list generated for $n = m$; let this list be L_m. The first half of the list L_{m+1} generated for $n = m+1$ is L_m with a 0 added to the front of each string. Since every string in L_m is unique (by the induction assumption), every string in the first half of L_{m+1} is unique. Similarly, using the induction assumption, every string in the second half of L_{m+1} is unique. Furthermore, each string in the first half is different than each string in the second half. Since each half has 2^m strings, we have a total of 2^{m+1} strings in L_{m+1}, each different from the other. Thus, every $m+1$-bit string appears exactly once in L_{m+1}.

(b) Using induction on n, we prove that each string (except the first) differs from the previous one in exactly one bit. For the base case, we consider $n = 1$. The twiddling list is $0, 1$, for which the claim is obviously true.

For the induction step, we assume that the claim is true for the m-bit twiddling list L_m generated by the algorithm. The first half of the list L_{m+1} generated for $n = m + 1$ is

L_m with a 0 added to the front of each string. Since the claim is true for the list L_m (by induction), every string in the first half of L_{m+1} except the first differs from the previous one in at least one bit. The same holds for the second half of L_{m+1}. We only need to verify the claim for the first string of the second half of L_{m+1}. This string differs from the last string of the first half of L_{m+1} in the first bit only. Hence, the induction step has been established, completing the proof.

4. (12 points) Chapter 2, Exercise 6, page 68.

Answer:

(a) The outer loop runs for n iterations. The inner loop runs for at most n iterations. In a single iteration of the inner loop, the addition takes at most n steps, and the storage at most 1 step. So the total number of steps is at most $n \cdot n \cdot (n+1) = n^3 + n^2$, which is $O(n^3)$. Note that this is an upper bound only.

(b) Using the same logic as above, we can say the outer loop runs for n iterations, the inner loop runs for at least 1 iteration, and the addition and storage take at least 2 steps. So the total time is at least $n \cdot 1 \cdot 2 = \Omega(n)$. This is much too weak a bound since there is a big gap between the upper and lower bounds. So we need to strengthen one of the two.

Let us consider the inner loop again. It runs for $n-1$ iterations the first time, $n-2$ the second time, and so on. So during at least half of the iterations of the outer loop, the inner loop runs at least $n/2$ iterations. Furthermore, the number of additions done inside an inner loop is also quite often as high as $n/2$. So this suggests that $\Omega(n^3)$ is more likely to be a lower bound as well.

Let us make a more precise calculation. Using the ranges of the variables i and j, the total number of steps can be written as

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} (j - i + 1)$$

($j - i$ for the additions and 1 for the storage). Let us focus on the inner sum first. For $i = 1$, it is $2+3+\ldots+n$. For $i = 2$, it is $3+4+\ldots+n$. So for general i, it is $(i+1)+(i+2)+\ldots+n$. Using the arithmetic progression formula or noting that this is the sum of the first n numbers minus the sum of the first i numbers, we get $n(n+1)/2 - i(i+1)/2$.

So our desired sum is now

$$\begin{aligned}
\sum_{i=1}^{n} \left(\frac{n^2+n}{2} - \frac{i^2+i}{2} \right) &= \frac{n^3+n^2}{2} - \frac{1}{2}\sum_{i=1}^{n} i^2 - \frac{1}{2}\sum_{i=1}^{n} i \\
&= \frac{n^3+n^2}{2} - \frac{n(n+1)(2n+1)}{12} - \frac{n(n+1)}{4} \\
&= \Omega(n^3)
\end{aligned}$$

after simplifications and elementary algebra.

(c) The given algorithm is clearly inefficient since during an iteration of the inner loop it often repeats several of the additions that have already been done in the previous iterations. Consider the following algorithm.

```
for i from 1 to n
    B[i, i+1] ← A[i] + A[i+1]
for k from 2 to n − 1
    for i from 1 to n − k
        j ← i + k
        B[i, j] ← B[i, j − 1] + A[j]
```

This algorithm works since the values $B[i, j-1]$ were already computed in the previous iteration of the outer for loop, when k was $j-1-i$, since $j-1-i < j-i$. The running time of the algorithm is $O(n)$ for the first for loop and $O(n^2)$ for the second set of loops, giving a total of $O(n^2)$ for the running time. Note that it is also $\Omega(n^2)$ since the algorithm computes at least $n(n-1)/2$ values.

5. (10 points) Ordering functions

Arrange the following functions in order from the slowest growing function to the fastest growing function. Briefly justify your answers. (*Hint:* It may help to plot the functions and obtain an estimate of their relative growth rates. In some cases, it may also help to express the functions as a power of 2 and then compare.)

$$n^{1/3} \quad n + \lg n \quad n(\lg n)^5 \quad 2^{\sqrt{\lg n}} \quad \lg n$$

Answer: The order from slowest growing to the fastest growing function is: $\lg n$, $2^{\sqrt{\lg n}}$, $n^{1/3}$, $n + \lg n$, and $n(\lg^5 n)$. The informal reasoning behind this is as follows. The function $\lg n$ clearly grows slowly than $n + \lg n$ (which essentially grows as n), which in turn grows slowly than $n \lg^5 n$. Where do we place $2^{\sqrt{\lg n}}$ and $n^{1/3}$? The function $n^{1/3}$ is a polynomial and grows faster than $\lg n$ and clearly slower than $n + \lg n$ since the exponent on n is $1/3 < 1$.

That leaves $2^{\sqrt{\lg n}}$. The function $\lg n$ can be written as $2^{\lg \lg n}$ and $n^{1/3}$ as $2^{(\lg n)/3}$. Comparing the three functions $2^{\sqrt{\lg n}}$, $\lg n$, and $n^{1/3}$ is equivalent to comparing $\sqrt{\lg n}$, $\lg \lg n$ and $(\lg n)/3$. Since $\sqrt{\lg n}$ is greater than $\lg \lg n$ and smaller than $(\lg n)/3$, we can place $2^{\sqrt{\lg n}}$ between the other two functions.

While a justification based on plots and/or informal arguments was acceptable for this homework, you must be prepared to provide formal proofs for your claims such as what follows.

We prove this using four steps: (i) $\lg n = O(2^{\sqrt{\lg n}})$, (ii) $2^{\sqrt{\lg n}} = O(n^{1/3})$, (iii) $n^{1/3} = O(n + \lg n)$, and (iv) $n + \lg n = O(n \lg^5 n)$.

(i) We use the connection between asymptotic notation and limits.

$$\begin{aligned}
\lim_{n\to\infty} \frac{\lg n}{2^{\sqrt{\lg n}}} &= \lim_{m\to\infty} \frac{m}{2^{\sqrt{m}}} \\
&= \lim_{m\to\infty} \frac{1}{2^{\sqrt{m}}/\sqrt{m}} \\
&= \lim_{m\to\infty} \frac{\sqrt{m}}{2^{\sqrt{m}}}
\end{aligned}$$

$$\begin{aligned} &= \lim_{m\to\infty} \frac{1/(2\sqrt{m})}{2^{\sqrt{m}}/(2\sqrt{m})} \\ &= \lim_{m\to\infty} \frac{1}{2^{\sqrt{m}}} \\ &= 0, \end{aligned}$$

where we replace $\lg n$ by m in the first step and use L'Hopital's rule in the second and fourth steps. The final step holds since $2^{\sqrt{m}}$ goes to infinity as m goes to infinity. Therefore, $\lg n = O(2^{\sqrt{\lg n}})$.

(ii) We again use the connection between asymptotic notation and limits.

$$\begin{aligned} \lim_{n\to\infty} \frac{2^{\sqrt{\lg n}}}{n^{1/3}} &= \lim_{n\to\infty} \frac{2^{\sqrt{\lg n}}}{2^{(\lg n)/3}} \\ &= \lim_{n\to\infty} \frac{1}{2^{(\lg n)/3-\sqrt{\lg n}}} \\ &= 0, \end{aligned}$$

since $(\lg n)/3 - \sqrt{\lg n}$ goes to ∞ as n goes to ∞. Therefore, $2^{\sqrt{\lg n}} = O(n^{1/3})$.

(iii) We again use the connection between asymptotic notation and limits.

$$\begin{aligned} \lim_{n\to\infty} \frac{n^{1/3}}{n+\lg n} &= \lim_{n\to\infty} \frac{1}{n^{2/3}+(\lg n)/n^{1/3}} \\ &= 0, \end{aligned}$$

Therefore, $n^{1/3} = O(n + \lg n)$.

(iv) We again use the connection between asymptotic notation and limits.

$$\begin{aligned} \lim_{n\to\infty} \frac{n+\lg n}{n\lg^5 n} &= \lim_{n\to\infty} \frac{1}{\lg^5 n} + \frac{1}{n\lg^4 n} \\ &= 0. \end{aligned}$$

Therefore, $n + \lg n = O(n \lg^5 n)$.

6. (3 × 4 = 12 points) Properties of asymptotic notation

Let $f(n)$, $g(n)$, and $h(n)$ be asymptotically positive and monotonically increasing functions. For each of the following statements, decide whether you think it is true or false and give a proof or a counterexample.

(a) $f(n) + g(n) = \Theta(\max\{f(n), g(n)\})$.

Answer: True. Since $f(n) + g(n) \geq \max\{f(n), g(n)\}$ for all n, we have $f(n) + g(n) = \Omega(\max\{f(n), g(n)\})$ $(c = 1, n_0 = 1)$. Since $f(n) + g(n) \leq 2\max\{f(n), g(n)\}$ for all n, we have $f(n) + g(n) = O(\max\{f(n), g(n)\})$ $(c = 2, n_0 = 1)$.

(b) If $f(n) = \Omega(h(n))$ and $g(n) = O(h(n))$, then $f(n) = \Omega(g(n))$.

Answer: True. Since $f(n) = \Omega(h(n))$, there exist positive constants c_1 and n_1 such that $f(n) \geq c_1 h(n)$ for all $n \geq n_1$. Similarly, since $g(n) = O(h(n))$, there exist positive constants c_2 and n_2 such that $g(n) \leq c_2 h(n)$ for all $n \geq n_2$. This implies that for all $n \geq \max\{n_1, n_2\}$, $f(n) \geq c_1 h(n) \geq c_1 g(n)/c_2$. Therefore, $f(n) = \Omega(g(n))$.

(c) If $f(n) = O(g(n))$, then $f(n)^2$ is $O(g(n)^2)$.

Answer: True. Since $f(n) = O(g(n))$, there exist positive constants c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$. We thus have $f(n)^2 \leq c^2 g(n)^2$ for all $n \geq n_0$. Since c^2 is a positive constant, we thus have $f(n)^2 \leq c^2 g(n)^2$.