Colby Sawyer
B01204512

Design and Analysis of Algorithms

You have to solve the following problems and submit them. Although preferred, you are not required to type set the answers. If you choose to write your answers down, make sure that your writing is legible though. In either case, you should submit a single document in pdf format using CANVAS.

Try to write down your answers in a clear manner. Understandability of the solution is as desirable as correctness. Show all intermediate steps and justify any auxiliary claims that you make.

This homework is based on the material in Chapters 1 and 2 of the textbook. There are **seven** problems in this homework and they are worth 100 points in total.

1. In the national resident matching problem, the situation was the following. There were *m* hospitals, each with a certain number of available positions for hiring residents. There were *n* medical students graduating in a given year, each interested in joining one of the hospitals. Each hospital had a ranking of the students in order of preference, and each student had a ranking of the hospitals in order of preference. We will assume that there were more students graduating than there were slots available in the *m* hospitals. The interest, naturally, was in finding a way of assigning each student to at most one hospital, in such a way that all available positions in all hospitals were filled. (Since we are assuming a surplus of students, there would be some students who do not get assigned to any hospital.) We say that an assignment of students to hospitals is stable if neither of the following situations arises.

   First type of instability: There are students *s* and s^0, and a hospital *h*, so that

   - *s* is assigned to *h*, and • s^0 is assigned to no hospital,

     and

   - *h* prefers s^0 to *s*.

   .

   Second type of instability: There are students *s* and s^0, and hospitals *h* and h^0, so that

- *s* is assigned to *h*, and
- s^0 is assigned to h^0, and
- *h* prefers s^0 to *s*, and
- s^0 prefers h to h^0.

. So we basically have the Stable Matching Problem, except that (i) hospitals generally want more than one resident, and (ii) there is a surplus of medical students. Come up with an algorithm to find a stable assignment of students to hospitals. Briefly sketch a proof of correctness of your algorithm. Hence, infer that there is always a stable assignment of students to hospitals, no matter what the preferences of students and hospitals are. (20 Pts.)

Algorithm:

```
While (a hospital $h_n$ is not fully assigned):
    $h_n$ offers a position to the first student on their ranking $S_x$
    if ($S_x$ is unassigned):
        $S_x$ accepts the position at $h_n$.
    Else
        if ($S_x$ is already committed to another hospital $h_m$ AND prefers $h_m$):
            $S_x$ remains assigned to $h_m$.
            $h_n$ moves on to the next student in their ranking
        Else:
            $S_x$ is reassigned to $h_n$.
            $h_m$ now has one more open position.
            $h_n$ now has one less open position.
```

Colby Sawyer
B01204512

2. Prove by **algebraic manipulation** that $4n^3 - 3n^2 + 8n + 9 = \Theta(n^3)$. (15 Pts.)

Proof:

$f(n) = 4n^3 + 3n^2 + 8n + 9$

Claim: $f(n) = O(n^3)$

Find c and n_0

$f(n) <= cn^3$ for all $n >= n_0$

$4n^3 – 3n^2 + 8n + 9 <= cn^3$

$(4n^3 – 3n^2 + 8n + 9 <= cn^3)/n^3$

$4 – (3/n) + (8/n^2) + (9/n^3) <= c$

FOUND if $n_0 = 1$

$C >= 4 – 3 + 8 + 9$ or 18

$4n^3 – 3n^2 + 8n + 9 <= 18n^3$

Or $f(n) = O(n^3)$

We can ignore the 18 as it's a constant and can conclude that $f(n) = O(n^3)$

Colby Sawyer
B01204512

3. Prove by **mathematical induction** that $n^2 = O(2^n)$. [**Hint:** First, play with the functions and make a guess about c and n_0. Then, prove that your guess is correct by mathematical induction] (15 Pts.)

Proof:

Claim: $n^2 = O(2^n)$

If $c >= 2$

$n^2 <= c\,2^n$ for all $n >= n_0$

$n^2 <= c2^n$

Plug in $c = 2$

$n^2 <= 2(2^n)$

if $n_0 = 1$

$n_0^2 <= 2(2^{n_0})$ equals $1^2 <= 2(2^1)$ or $2 <= 4$

Therefore we can conclude that $n^2 < (2^n)$ for all $n > 1$;

$n^2 = O(2^n)$

Colby Sawyer
B01204512

4. Prove by **using calculus** that $n^3 = O(3^n)$. [**Hint:** Need to use L'Hospital's rule to evaluate the limits involved]. (15 Pts.)

Proof:

Claim: $n^3 = O(3^n)$

$f(n) = O(g(n)$

La Hopitals Rule States: $\lim_{x \to \infty} (f(n))/(g(n) = 0 \Rightarrow f(n) = O(g(n))$

Finding the limit for f(n) or n^3

$n = o(n^3)$

$\underline{\lim_{x \to \infty} ((n)/(n^3)) = \lim_{x \to \infty} ((1)/(n^2)) = 0}$

Finding the limit for g(n) or 3^n

$n = 3^n$

$\underline{\lim_{x \to \infty} ((n)/(3^n)) = 0}$

Therefore we can conclude:

$\lim_{x \to \infty} ((f(n))/(g(n))= 0$ which implies that $f(n) = O(g(n))$

and $\underline{\lim_{x \to \infty} ((n^3)/(3^n))}$ <u>or 0/0 = 0 which implies that</u> $\underline{n^3 = O(3^n)}$

given that $f(n) = n^3$ *and* $g(n) = (3^n)$

Colby Sawyer
B01204512

5. Let $f(n)$ and $g(n)$ be two non-negative functions. Then $h(n) = max(f(n),g(n))$ is defined naturally as follows.

$$h(n) = \begin{cases} f(n &),\text{ if } f(n) > g(n)\ ) \\ g(n & \text{otherwise.} \end{cases}$$

Prove that $h(n) = \Theta(f(n) + g(n))$. (15 Pts.)

Proof:

Claim: $h(n) = \Theta(f(n) + g(n))$

Given that $h(n) = max(f(n),g(n))$

Goal: $\Theta(f(n) + g(n)) = max(f(n),g(n))$

f(n) < f(n) +g(n) and g(n) <= f(n) + g(n)

$max(f(n),g(n)) \in O(f(n)+g(n))$

f(n) +g(n) < 2 $max(f(n),g(n))$

$max(f(n),g(n)) \in \Omega(f(n)+g(n))$

$max(f(n),g(n))$ = { f(n) if f(n) >= g(n)

g(n) if g(n) >= f(n)

f(n) + g(n) <= max(f(n), g(n)

Therefore we can conclude $\Theta(f(n) + g(n)) = max(f(n),g(n))$

Or written as <u>h(n) = $\Theta(f(n) + g(n))$</u>

Colby Sawyer
B01204512

6. Suppose we have two different algorithms to solve a problem. Algorithm A requires $10n^3$ instructions to solve the problem. Algorithm B requires 2^n instructions to solve problem. Suppose you code up algorithm A and run it on a computer that can execute, say million instructions per second. Suppose you code up algorithm B and run it on a computer that runs 1000 times faster than the other computer (i.e., this computer can execute billion instructions per second). Compare the running time of the two algorithms (on the respective computers) for input sizes of n = 20, n = 30, and n = 40. Express the running times in suitable units (such as seconds, minutes, hours, days or years). Assume that an year has exactly 365 days. What is the conclusion that you can draw from your results. (10 Pts.)

Computer A :	Speed: 10^6 instructions/second
Computer B:	Speed: 10^9 instructions/second
Algorithm A:	$10n^3$ instructions per problem
Algorithm B:	2^n instructions per problem

Computer A can compute 20 instructions in 0.08 seconds

Computer B can compute 20 instructions in 0.00105 seconds

Computer A can compute 30 instructions in 0.27 seconds

Computer B can compute 30 instructions in 1.07374 seconds

Computer A can compute 40 instructions in 0.64 seconds

Computer B can compute 40 instructions in 18.32 minutes

We can conclude that Computer A running Algorithm A is much more efficient than Computer B running Algorithm B. Algorithm B can process a low number of problems much more quickly than Algorithm A but when you increase the number of problems its computation time soars far above Algorithm A; meaning it is less efficient in the long -run.

Colby Sawyer
B01204512

7. Suppose f and g are functions such that $f(n) = O(g(n))$. Prove, by providing a concrete counterexample, that it does not necessarily imply $2^{f(n)} = O(2^{g(n)})$.

   (10 Pts.)

Proof:

Claim: $f(n) = O(g(n))$, then $2^{f(n)} = O(2^{g(n)})$.

Provide a counterexample:

$F(n) = 2\log(n)$

$G(n) = \log(n)$

2log(n) <= c log(n) if c>2 therefor we know $f(n) = O(g(n))$

But $2^{f(n)} = O(2^{g(n)})$ or $2^{2\log(2)} = O(2^{\log(n)})$ or $n^2 = O(n)$ is invalid;

We can conclude that $2^{f(n)} \neq O(2^{g(n)})$ since we know that $n^2 > n$.