# Solution to CSE531 Midterm

November 3, 2014

## Problem 1

Give a True or False answer to each of the following statements. Here $f(n)$ and $g(n)$ are two positive functions defined over positive integers.

### (a)

**TRUE**.

*Proof.*

If $f(n)$ or $g(n)$ is a constant function then this is trivial. Otherwise:

$$\begin{aligned}
2^{f(n)} = \Theta(2^{g(n)}) &\Rightarrow \exists c_1, c_2, n_0 > 0, \forall n \geq n_0, c_1 2^{g(n)} \leq 2^{f(n)} \leq c_2 2^{g(n)} \\
&\Rightarrow \exists c_1, c_2, n_0 > 0, \forall n \geq n_0, 2^{g(n)+\log(c_1)} \leq 2^{f(n)} \leq 2^{g(n)+\log(c_2)} \\
&\Rightarrow \exists c_1, c_2, n_0 > 0, \forall n \geq n_0, g(n) + \log(c_1) \leq f(n) \leq g(n) + \log(c_2) \\
&\Rightarrow \exists d_1, d_2, n_0' > 0, \forall n \geq n_0', d_1 g(n) \leq f(n) \leq g(n) + \log(c_2)
\end{aligned}$$

$$\text{where } d_1 = \begin{cases} 1 & \text{if } \log(c_1) \geq 0 \\ c_1 & \text{o.w.} \end{cases}, d_2 = \begin{cases} c_2 & \text{if } \log(c_2) \geq 0 \\ 1 & \text{o.w.} \end{cases}$$

$$\text{and } n_0' = \max(n_0, \min\{n \mid d_1 g(n) \leq g(n) + \log(c_1)\}, \min\{n \mid d_2 g(n) \geq g(n) + \log(c_2)\})$$

(Note: the value for $n_0'$ can be computed since $d_1$ and $d_2$ guarantee $d_1 g(n) \leq g(n) + \log(c_1)$ and $d_2 g(n) \geq g(n) + \log(c_2)$ for sufficient values of $n$)

$\therefore f(n) \in \Theta(g(n)) \Rightarrow g(n) \in \Theta(f(n))$ □

### (b)

**TRUE**.

*Proof.*

If $f(n)$ or $g(n)$ is constant a function then this is trivial. Otherwise:

$$\begin{aligned}
2^{f(n)} = O(g(n)) &\Rightarrow \exists c, n_0 > 0, \forall n \geq n_0, 2^{f(n)} \leq cg(n) \\
&\Rightarrow \exists c, n_0 > 0, \forall n \geq n_0, f(n) \leq \log(cg(n)) \\
&\Rightarrow \exists c, n_0 > 0, \forall n \geq n_0, f(n) \leq \log(c) + \log(g(n)) \\
&\Rightarrow \exists d, n_0' > 0, \forall n \geq n_0, f(n) \leq d \log(g(n)) \\
&\text{where } d = \begin{cases} 1 & \text{if } \log(c_1) \leq 0 \\ 2 & \text{o.w.} \end{cases}, n_0' = \begin{cases} n_0 & \text{if } \log(c) \leq 0 \\ \min\{n \mid g(n) \geq \log(c)\} & \text{o.w.} \end{cases}
\end{aligned}$$

$\therefore f(n) \in O(\log(g(n)))$ □

## (c)

**FALSE**.

*Proof.* Counter-example: let $f(n) = 2^{2n}$, let $g(n) = 2^n$, and let $h(n) = 2^n$. So $\log f(n) = 2n$, $\log g(n) = n$, and $\log h(n) = n$. Since $2n = O(n)$ we satisfy the first part. However,

$$\lim_{n\to\infty} \frac{f(n)}{h(n)} = \lim_{n\to\infty} \frac{2^{2n}}{2^n} = \lim_{n\to\infty} 2^n = \infty.$$

So $f(n) = \omega(h(n)) \Rightarrow f(n) \notin O(h(n))$. □

# Problem 2

(a): Directly apply Theorem in the page 14 of Note 02:

**Theorem 1.** *If $T(n) = aT(n/b) + f(n)$, where $f(n) = \theta(n^{\log_b a}(\log n)^k)$, then*

$$T(n) = n^{\log_b a}(\log n)^{k+1}.$$

We have $a = 3, b = 9, k = 1$, $T(n) = n^{\log_b a}(\log n)^{k+1} = n^{\frac{1}{2}}(\log n)^2$.

$$\begin{aligned}
T(n) &= n^{\log_9 3}(\log n)^{1+1} \\
&= n^{\frac{1}{2}}(\log n)^2.
\end{aligned}$$

(b): Directly apply Master Theorem, where $a = 4$, $b = 3$, and $n^2 = \Omega(n^{\log_3 4})$, also $7 \times (\frac{n}{3})^2 \leq \frac{7}{9}n^2$, so Case 3 applies. So $T(n) = \theta(n^2)$.

(c): $T(n) = T(4n/7) + T(n/3) + n$.

Suppose $T(n) = cn$ with $c = 21$. Then,

Base Case $n = 1$:

$$cn = 21 > 21 \times \frac{4}{7} + 21 \times \frac{1}{3} + 1 = T(4n/7) + T(n/3) + n \Rightarrow 21 > 12 + 7 + 1.$$

Inductive case: We assume our substation is true for $n - 1$. We now want to show that it is true for $n$.

$$\begin{aligned} 21n &> \frac{4}{7} \times 21 \times n + \frac{1}{3} \times 21 \times n + n \\ \Rightarrow 21n &> 12n + 7n + n = 20n. \end{aligned}$$

So, $T(n) = \theta(n)$.

# Problem 3

The characteristic equation is

$$x^2 = 10x - 21$$

The roots for the equation is $x_1 = 3$, $x_2 = 7$. So $g(n) = A \cdot 3^n + B \cdot 7^n$ Since $g(1) = 2$ and $g(2) = 3$, we have $3A + 7B = 2$ and $9A + 49B = 3$. Solving this we get $A = \frac{11}{12}$, $B = -\frac{28}{3}$. So

$$g(n) = \frac{11}{12} \cdot 3^n - \frac{3}{28} \cdot 7^n$$

# Problem 4

Let $l$ be a leaf of $T$ and the parent be $p(l)$. If $p(l)$ has the only one leaf $l$, we can have the following observation:

**Observation 1.** *Each perfect matching of $T$ must contain the edge $(l, p(l))$.*

It's because the only way to cover a leaf $l$ is to match $l$ and $p(l)$. And, after $p(l)$ was covered, it's impossible to cover other leaves of $p(l)$. On the other hand, after removing the edge $(l, p(l))$ from $T$, $T$ will be partitioned into a set of disjoint trees $\{T_1, T_2, \cdots, T_k\}$. And, we have a key theorem as follows:

**Theorem 2.** *$T$ has a perfect matching if and only if each $T_i, 1 \leq i \leq k$, has a perfect matching.*

Our algorithm will iteratively pick up a leaf and test whether its parent $p(l)$ connects the only one leaf. If not, from the above observation, $T$ doesn't contains a perfect matching. If it is, we remove the edge and partition $T$ into a set of disjoint subtrees, $\{T_1, T_2, \cdots, T_k\}$. Then, for each subtree, do the same procedure for each $T_i, 1 \leq i \leq k$. If we can successfully remove all edges of $T$, $T$ has a perfect matching.

We will prove the correctness of our algorithm by induction theory. We assume $|T| = 2n$.

**Algorithm 1** Perfect-Matching($T$)

```
1: Pick up a leaf l of T, check whether l is the only one leaf of p(l)?
2: if Yes then
3:     Removing the edge (p(l), l) from T.
4:     Recursively call Perfect-Matching(T_1), Perfect-Matching(T_2), ···,
   Perfect-Matching(T_k).
5:     if every T_i, 1 ≤ i ≤ k has a perfect matching then
6:         return T has a perfect matching.
7:     else
8:         return T doesn't have a perfect matching.
9:     end if
10: else
11:    return T doesn't have a perfect matching.
12: end if
```

**Base Case:** $|T| = 2$ : In this case, $T$ is an edge $(a, b)$. It's obvious to see that our algorithm select $(a, b)$ as a perfect matching of $T$.

**Inductive Case:** $|T| = 2n$ : Assume our algorithm works for any tree $T$ with $2(n-1)$ vertices.

We have two cases: Case (1): we can pick up a leaf $l$ such that $p(l)$ has the only one leaf $l$, Case (2): we can't pick up such a leaf.

Case (1): By the above theorem, we just need to test whether each $T_i, 1 \leq i \leq k$, has a perfect matching. Because each $|T_i| \leq 2(n-1), 1 \leq i \leq k$, by induction, our algorithm can test whether there is a perfect matching for each $T_i, 1 \leq i \leq k$. Hence our algorithm can test whether there is a perfect matching for $T$.

Case (2): By the above observation, $T$ doesn't have a perfect matching.

We will analyze the time complexity for our algorithm in the followings:

$$\begin{aligned} t(|T|) &= \sum_{i=1}^{k} t(|T_i|) + O(1) \\ &= \theta(|T|) \\ &= \theta(n). \end{aligned}$$

# Problem 5

We can see that the base cases for the recurrence are $C[0, j]$ and $C[i, 0]$ for $0 \leq i \leq n$ and $0 \leq j \leq m$. To compute a value $C[i, j]$ for $i, j \geq 1$ we need the value of $C[i-1, j]$, which comes from the previous row in the same column, and $C[i, j-1]$, which comes from the same row and the previous column. From this we can formulate our solution.

Begin by constructing a matrix with $n$ rows and $m$ columns ($n$ x $m$ matrix). Next, fill the first row and column of the matrix with all 1s ($C[0, j] = 1$ and $C[i, 0] = 1$ for $0 \leq i \leq n$ and $0 \leq j \leq m$). Then we can fill the remainder of the matrix, one row at a time from left to right ($j = 1$ to $m$), starting with the second row to the last row ($i = 1$ to $n$).

Each entry is computed in constant time since we use previously computed values from our matrix ($C[i-1, j]$ and $C[i, j-1]$) and perform 2 additions followed by a single comparison. We fill each row one entry at a time, so we have $n$ rows each with $m$ entries giving $nm$ entries to compute to solve for $C[n, m]$. So we have a running time of $\mathrm{T}(n, m) = nm\Theta(1)$, giving $\mathrm{T}(n, m) = \Theta(nm)$ for our algorithm to compute $C[n, m]$.

# Problem 6

Here we only want a greedy-strategy based algorithm, not a greedy algorithm that is guaranteed to produce the optimal output. As long as your algorithm is greedy-strategy based, and has a counter-example or proof depending on whether you believe it's optimal or not, you should receive full marks.

One possible (also most popular) solution is as follows: each time assign the job with the lowest cost to the employee that can achieve this cost. If the employee is not available (i.e. already assigned another job), find the job with the second smallest cost. Repeat until all jobs are assigned.

This simple algorithm does not always output the optimal solution. Consider the following counter-example: for 2 jobs and 2 employees, where $c(1, 1) = 1$, $c(1, 2) = 10$, $c(2, 1) = 5$, $c(2, 2) = 100$. The optimal solution is to assign job 1 to employee 2, and job 2 to employee 1, with a total cost of 15. But the algorithm we gave will first assign job 1 to employee 1, then there is only employee 2 left to assign job 2 to. The total cost is 101, larger than the optimal solution.

Giving a greedy-strategy based algorithm is 13 points, giving a counter-example or a proof is 7 points.