Australian National University
School of Computer Science

COMP3600/COMP6466 in 2015 – Solutions to Assignment One

Due: 23:55 Friday, September 4
Late Penalty: 5% per working day

No programming is needed for this assignment. You can submit your work electronically through Wattle. The total mark is 50. Marks may be lost for giving information that is irrelevant or for correct but sub-optimal answers. Do not forget to write down your name, student ID, and tute/lab group name in a separate cover page or the first page of your assignment.

Question 1 (3 points).

Sort the following set of functions by the order of their growth as n approaches infinity.

$$n^{2.3}\log n\log\log n,\quad n^{3/2}\log n,\quad \sqrt{n}\log^2 n,\quad n^{\log\log\log n},\quad n^{4.5},\quad 2^{1/n^2},$$

$$100^{120}\log n,\quad n^6+50n^5-2000n^{3.5},\quad 3^{0.5\log n}+5n^2,\quad n^{3/5}\log n.$$

The order of growth from smaller to larger is as follows.

$$2^{1/n^2},\quad 100^{120}\log n,\quad \sqrt{n}\log^2 n,\quad n^{3/5}\log n,\quad n^{3/2}\log n,$$
$$3^{0.5\log n}+5n^2,\quad n^{2.3}\log n\log\log n,\quad n^{4.5},\quad n^6+50n^5-2000n^{3.5},\quad n^{\log\log\log n}.$$

Question 2 (12 points).

Let $f(n)$, $g(n)$ and $h(n)$ be three positive functions. For each of the following statements, either prove that the statement is true or show that the statement is false by providing a counter-example for f, g, and h.

(a) $f(n)=\Theta(g(n))$ and $g(n)=o(h(n))$ together imply that $f(n)=O(h(n))$

The statement is true. We prove this claim as follows.

Sicne $f(n)=\Theta(g(n))$, there exists $c_1>0$, $c_2>0$, and $n_0>0$ such that for any $n\geq n_0$ we have

$$0\leq c_1\cdot g(n)\leq f(n)\leq c_2\cdot g(n). \tag{1}$$

Also, given $g(n) = o(h(n))$, then, for any constant $c > 0$ and there is a constant $n_1 > 0$ such that

$$0 \leq g(n) < c \cdot h(n). \tag{2}$$

Combining inequalities (1) and (2), we have

$$0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) < c_2 \cdot (c \cdot h(n)). \tag{3}$$

In other words, there exists constants $c' = c_2 c > 0$ and $n_2 = \max\{n_0, n_1\}$ such that inequality (3) is held. Following the big-O notation, we have $g(n) = O(h(n))$.

(b) $f(n) = \Omega(g(n))$ and $g(n) = \Theta(h(n))$ together imply that $f(n) = O(h(n))$

We prove that the statement is false, by a counter-example.

Consider $f(n) = n$, $g(n) = \log n$, and $h(n) = 12 \log n$, it is obvious that $f(n) = \Omega(g(n))$ and $g(n) = \Theta(h(n))$, but $f(n) \neq O(h(n))$.

(c) $\min\{f(n), g(n), h(n)\} = \Theta(f(n) + g(n) + h(n))$.

We prove this statement is false, by a counter-example.

Consider $f(n) = n$, $g(n) = h(n) = \log n$, it is obvious that $\min\{f(n), g(n), h(n)\} = \log n$, while $\Theta(f(n) + g(n) + h(n)) = \Theta(n)$, clearly $\log n \neq f(n) = \Theta(n) = \Theta(f(n) + g(n) + h(n))$.

(d) $a^{f(n)} = O(a^{g(n)})$ implies $g(n) = \Omega(f(n))$

We prove this statement is false by a counter-example.

Consider $f(n) = n$, $g(n) = \log n$ and $a = 1/2$. It is obvious that $a^{f(n)} = O(a^{g(n)})$, but $g(n) \neq \Omega(f(n))$.

Question 3 (10 points).

Give an asymptotic upper bound on $T(n)$ for each of the following recurrences using the $O(\,)$ notation. In each case, explain your reasoning clearly. Note that you are not allowed to use the Master theorem.

(a) $T(n) = T(n/4) + n^2 \log n$

We apply the recursive-tree (iteration) method.

Assume that $T(n)$ is defined for every positive integer number n. Note that $k = \log_4 n$ when $n/4^k = 1$.

$$\begin{aligned}
T(n) &= T(n/4) + n^2 \log n \\
&= T(n/4^2) + (n/4)^2 \log(n/4) + n^2 \log n \\
&= T(n/4^3) + (n/4^2)^2 \log(n/4^2) + (n/4)^2 \log(n/4) + n^2 \log n \\
&= \cdots \\
&= T(n/4^k) + \sum_{i=1}^{k} (n^2/4^{2(i-1)}) \log(n/4^{i-1}) \\
&\leq T(1) + \sum_{i=1}^{k} (n^2/4^{2(i-1)}) \log n, \quad \text{since } \log(n/4^{i-1}) \leq \log n \\
&= T(1) + n^2 \log n \sum_{i=1}^{k} 1/4^{2i-2} \\
&= T(1) + n^2 \log n \Theta(1) \\
&= \Theta(n^2 \log n),
\end{aligned}$$

Use the facts that $\log(n/4^{2(i-1)}) \leq \log n$ for any positive integer i and that a geometric sum is $\Theta(\textit{largest term})$. Thus, we obtain the asymptotic upper bound of $T(n) = O(n^2 \log n)$. As the definition of $T(n)$ immediately gives $T(n) \geq n^2 \log n$, we can be sure that our asymptotic upper bound is best possible.

(b) $T(n) = 2T(3n/5) + n^2$.

We apply the iteration method. Assume that n is a positive integer, and

$k = \log_{5/3} n$ when $(3/5)^k n = 1$. Then,

$$\begin{aligned}
T(n) &= 2T(3n/5) + n^2 \\
&= 2^2 T((\frac{3}{5})^2 n) + 2(\frac{3n}{5})^2 + n^2 \\
&= 2^2 T((\frac{3}{5})^2 n) + 2n^2 \frac{3^2}{5^2} + n^2 \\
&= 2^3 T((\frac{3}{5})^3 n) + 2^2 n^2 \frac{3^4}{5^4} + 2n^2 \frac{3^2}{5^2} + n^2 \\
&= \cdots \\
&= 2^k T(1) + n^2 \sum_{t=0}^{k-1} 2^t \frac{3^{2t}}{5^{2t}} \\
&= 2^k T(1) + n^2 \sum_{t=0}^{k-1} \frac{2^t 3^{2t}}{5^{2t}}. \\
&= 2^k T(1) + n^2 \sum_{t=0}^{k-1} \frac{18^t}{25^t}.
\end{aligned}$$

Consider now this form of $T(n)$ only for positive integer n. The summation is a geometric series, so its sum is $\Theta(largest\_term) = \Theta((\frac{18}{25})^k) = \Theta((\frac{18}{25})^{\log_{5/3} n}) = \Theta(1)$ since $\log_{5/3} n \leq \infty$. Also, $2^k = 2^{\log_{5/3} n} = n^{\log_{5/3} 2} \approx n^{1+\alpha}$ with $0 < \alpha \leq 1$. Therefore, $T(n) = 2^{\log_{5/3} n}\Theta(1) + n^2\Theta(1) = \Theta(n^2)$, which implies $T(n) = O(n^2)$. Notice that $f(n)^{\log_a b} = b^{\log_a f(n)}$.

(c) ((Honours & COMP6466 only) $T(n) = 3T(n/13) + T(n/7) + n$

We use mathematical induction, we assume that $T(n') = cn'$ for all $n' \leq n$ where $c > 0$ is a constant.

Induction Step:

$$\begin{aligned}
T(n) &= 3T\left(\frac{n}{13}\right) + T\left(\frac{n}{7}\right) + n \\
&\leq 3c\left(\frac{n}{13}\right) + c\left(\frac{n}{7}\right) + n \\
&\leq \frac{34cn}{91} + n \\
&= cn - \frac{57cn}{91} + n \\
&\leq cn
\end{aligned}$$

whenever $-\frac{57cn}{91} + n$ is negative, then $c \geq 91/57$ (for any positive c).

We have proved that there is a positive constant $c \geq 91/57$ such that $T(n) \leq cn$ for all positive values of n, this implies $T(n) = O(n)$. As the same asymptotic lower bound can be achieved for $T(n)$, we can be sure that our upper bound is best possible.

Question 4 (8 points).

Provide the simplest expression for each of the following sums, using the $\Theta(\,)$ notation. In each case, explain your reasoning clearly.

(a) $\sum_{k=1}^{n} k^{17/7}$.

$\sum_{k=1}^{n} k^{17/7} \leq \sum_{k=1}^{n} n^{17/7} = n \cdot n^{17/7} = n^{24/7}$, thus, $\sum_{k=1}^{n} k^{17/7} = O(n^{24/7})$.

Also, $\sum_{k=1}^{n} k^{17/7} \geq \sum_{k=\lceil n/2 \rceil}^{n} k^{17/7} \geq \sum_{k=\lceil n/2 \rceil}^{n} (n/2)^{17/7}$
$= (n - \lceil n/2 \rceil + 1)(\frac{n}{2})^{17/7} \geq (\frac{n}{2})^{24/7}$, i.e, $\sum_{k=1}^{n} k^{17/7} = \Omega(n^{24/7})$.

Thus,

$$\sum_{k=1}^{n} k^{17/7} = \Theta(n^{24/7}).$$

(b) $\sum_{k=1}^{n} k^7/2^k$.

Let's define $a_k = \frac{k^7}{2^k}$, so $\sum_{k=1}^{n} \frac{k^7}{2^k} = \sum_{k=1}^{n} a_k$.

Now, we have

$$\frac{a_{k+1}}{a_k} = \frac{(k+1)^7/2^{k+1}}{k^7/2^k} = \frac{1}{2}(\frac{k+1}{k})^7 \leq \frac{1}{2}(\frac{11}{10})^7 \approx 0.974 < 1$$

when $k \geq 10$.

Let $r = \frac{1}{2}(\frac{11}{10})^7$. When $k \geq 10$, we have $a_{k+1} \leq r \cdot a_k$, which means $a_{k+1} \leq r \cdot a_k \leq r^2 \cdot a_{k-1} \leq r^3 \cdot a_{k-2} \leq \ldots \leq r^{k-9} \cdot a_{10}$ for $k \geq 10$. Thus, $a_k \leq r^{k-10} \cdot a_{10}$ when $k \geq 10$, and so

$$\begin{aligned}
\sum_{k=1}^{n} a_k &= a_1 + a_2 + a_3 + a_4 + \ldots + a_9 + \sum_{k=10}^{n} a_k \\
&\leq a_1 + a_2 + a_3 + a_4 + \ldots + a_9 + \sum_{k=10}^{n} r^{k-10} \cdot a_{10} \\
&= a_1 + a_2 + a_3 + a_4 + \ldots + a_9 + a_{10} \sum_{k=10}^{n} r^{k-10}.
\end{aligned}$$

As $\sum_{k=10}^{n} r^{k-10}$ is a geometric sum, it is $\Theta(\text{largest term}) = \Theta(a_{10}) = \Theta(1)$. Moreover, as the first nine terms have constant values, $\sum_{k=1}^{n} a_k \leq \Theta(1) + \Theta(1) \cdot \Theta(1) = \Theta(1)$. Finally, $\sum_{k=1}^{n} a_k \geq a_1 = 1/2 = \Theta(1)$. Thus, $\sum_{k=1}^{n} \frac{k^7}{2^k} = \Theta(1)$.

Question 5 (17 points)

Q5.(a) (10 points) Suppose you are running a small consulting business - just you, two associates and some rented equipment. Your clients are distributed between the East Coast and the West Coast in Australia, and this leads to the following question.

Each month, you can either run your business from an office in Sydney or from an office in Perth. In month i, you will incur an *operating cost* of S_i if you run the business out of Sydney; you will incur an operating cost of P_i if you run the business out of Perth. However, if you run the business out of one city in month i, and then out of the other city in month $i+1$, then you incur a fixed *moving cost* of M to switch base offices.

Given a sequence of n months, *a plan* is a sequence of n locations - each one equal to either Sydney or Perth - such that the ith location indicates the city in which you will be based in the ith month. The *cost* of a plan is the sum of the operating costs for each of the n months, plus a moving cost of M for each time you switch cities. The plan can begin in either city.

The Problem: Given a value for the moving cost M and sequences of operating costs $S_1, S_2, \ldots, S_n$ and $P_1, P_2, \ldots, P_n$, find an optimal plan for the problem, where a plan is optimal if its cost is the minimum one.

Devise an efficient algorithm that takes values for n, M, and sequences of operating costs $S_1, S_2, \ldots, S_n$ and $P_1, P_2, \ldots, P_n$, and returns an optimal plan and the cost of the plan. Analyze the running time of your algorithm. (*Hint: using Dynamic Programming and following the four steps of DP design*)

Following the four steps in the design of dynamic programming algorithms, we proceed as follows.

Step 1: As the problem stated, the business must be run either in Sydney or Perth. The operating cost at a city in month i is either C_i for staying at the same city, i.e., $C_i = P_i$ or $C_i = S_i$, depending on whether the business stayed at Perth or Sydney in month $i-1$, or $C_i + M$ when it moved to here from another city. The most cost effective price in month i is thus staying at which city will lead to the least accumulated cost from month 1 to month i.

Step 2: Let $P_P(i)$ be the minimum accumulate cost from month 1 to month i, assuming that the business stays at Perth in month i. Similarly we define $P_S(i)$ and

assume that the business stays at Sydney in month i, $1 \leq i \leq n$. We thus have

$$P_P(i) = \begin{cases} P_1, & i = 1 \\ \min\{P_S(i-1) + M + P_i, P_P(i-1) + P_i\}, & \text{otherwise} \end{cases}$$

Similarly, we have

$$P_S(i) = \begin{cases} S_1, & i = 1 \\ \min\{P_P(i-1) + M + S_i, P_S(i-1) + S_i\}, & \text{otherwise} \end{cases}$$

Thus, $\min\{P_P(n), P_S(n)\}$ is the optimal solution.

Step 3: The pseudo-code for the recurrence. It is easy to solve the recurrence. We find $P_P(1), P_S(1), P_P(2), P_S(2), \ldots, P_P(i), P_S(i), \ldots, P_P(n), P_S(n)$, Clearly, the running time is $O(n)$.

Step 4: Because this problem requires to find an optimal solution, in Step 3, we use an array $I()$ to keep the information of which city resulting in the minimum value of $P_P(i)$ (or $P_S(i)$). If $P_P(i)$ is obtained by moving from Sydney, then $I_P(i) = S$; otherwise, $I_P(i) = P$. Thus, the optimal solution can be found, starting from the city in month n, perform backtracking to identify the city of month $n-1$, then the city of month $n-2$, and so on, using the array $I()$.

Q5.(b) (7 points) John is considering opening a series of restaurants along Hume Highway. The n possible locations are along a straight line, and the distance of these locations from the start of Hume Highway are, in kilometers and in increasing order, $k_1, k_2, \ldots, k_n$. The constraints are as follows.

- At each location, John may open at most one restaurant. The expected profit from opening a restaurant at location i is p_i, where $p_i > 0$ and $i = 1, 2, \ldots, n$.
- Any two restaurants should be at least l kilometers apart, where l is a positive integer.

Devise an efficient algorithm to compute the maximum expected total profit subject to the given constraints, and analyze the time complexity of your proposed algorithm. (*Hint: using Dynamic Programming and following the four steps of DP design*)

Following the four steps in the design of dynamic programming algorithms, we proceed as follows.

Let $P(i)$ the maximum profit collected so far after having considered possible locations from location 1 to location i inclusive with $1 \leq i \leq n$.

If John is going to open a restaurant at location j with $1 \leq j \leq n$, then

$$P(j) = \max_{0 \leq i < j} \begin{cases} 0, & \text{if } j = 0 \\ P(i) + p_j, & \text{if } k_j - k_i \geq l \end{cases}$$

Initially, $P(0) = 0$. The problem thus is to calculate $\max\{P(i) \mid 1 \leq i \leq n\}$, which is the optimal solution. It is easy to solve the recurrence, the calculation order is $P(0), P(1), P(2), \ldots, P(n)$, Clearly, the calculation of $P(j)$ takes $O(j)$ time, Thus, the running time of the algorithm is $O(n^2)$. Because this problem is to find an optimal solution, it does not require to construct the optimal solution, Step 4 is omitted.

Bonus Question (5 points)

`Warning:` *the following questions are designed for people who are capable of doing some extra research-related work. Only you have finished all questions* **correctly** *and are willing to challenge yourself, you can proceed the following questions.*

BQ 1. Suppose that function $g(n)$ returns an integer and its calculation takes $\Theta(n^2 \log^3 n)$ time for input size n. Determine the running time of the following function $f(n)$ in terms of input size n, using the $O(\ )$ notation. Provide your solution in the simplest possible form (2 points).

```
int  f (int n)
{
    int i, k;
    int sum = 0;
    if    (n < 100)
          return 10 * n^2;
    else
          for   (i = 0; i < n ; i++){
                k = i;
                while  (k >= 27) {
                       sum = sum + g(k);
                       k = ⌊5k/9⌋;
                }
          }
          return sum;
}
```

$$T(n) = \begin{cases} \Theta(1), & n < 100 \\ \sum_{i=100}^{n} \sum_{j=0}^{-\log_{5/9} i} \Theta([(5/9)^j \cdot i]^2 \log^3 [(5/9)^j \cdot i]), & \text{otherwise,} \end{cases}$$

while $\sum_{j=0}^{-\log_{5/9} i} \Theta(((5/9)^j i)^2 \log^3((5/9)^j i)) \leq \sum_{j=0}^{-\log_{5/9} i} O(((5/9)^j i)^2 \log^3 i) = O(i^2 \log^3 i)$,

$$T(n) = \Theta(1) + \sum_{i=100}^{n} O(i^2 \log^3 i) = O(n^3 \log^3 n)$$

.

Or, a better solution is

$\sum_{j=0}^{-\log_{5/9} i} \Theta(((5/9)^j i)^2 \log^3((5/9)^j i)) \leq \sum_{j=0}^{-\log_{5/9} i} \Theta(((5/9)^j i)^2 \log^3 i) \leq (n(5/9)^j)^2 \log^3 i \leq n^2(5/9)^{2j} \log^3 n)$,

$$T(n) \leq \Theta(1) + \sum_{i=100}^{n} n^2(5/9)^{2j} \log^3 n) = (n^2 \log^3 n) \sum_{i=100}^{n} (5/9)^{-\log_{5/9} i} = O(n^2 \log^3 n).$$

BQ 2. Given three sequences $X = x_1, x_2 \ldots, x_n$, $Y = y_1, y_2, \ldots, y_m$, and $Z = z_1, z_2, \ldots, z_p$, we aim to find a common subsequence of these three sequences X, Y and Z with the longest length. Design an DP algorithm for the problem and analyze the time complexity of the proposed algorithm. (3 points)

Let $X_i = x_1, x_2, \ldots, x_i$, $Y_j = y_1, y_2, \ldots, y_j$ and $Z_k = z_1, z_2, \ldots, z_k$ with $0 \leq i \leq n$, $0 \leq j \leq m$ and $0 \leq k \leq p$.

Let $l(i, j, k)$ be the longest length of the common subsequence of the three sequences X_i,Y_j and Z_k. We notice that if $x_i = y_j = z_k$, then the problem becomes to find a longest common subsequence from X_{i-1}, Y_{j-1} and Z_{k-1} and put x_i as the last element, then it forms the longest common subsequence of X_i, Y_j and Z_k. Otherwise, at least one of the element of x_i, y_j and z_p should not be included in the longest common subsequence. In other words, the longest common subsequence is contained in either of the three sequences: (1) X_i, Y_j and Z_{k-1}; (2) X_i, Y_{j-1} and Z_k; and (3) X_{i-1}, Y_j and Z_k. Thus we have the following recurrence.

$$l(i,j,k) = \max \begin{cases} 0 & \text{if } i = 0, \text{ or } j = 0, \text{ or } k = 0 \\ l(i-1, j-1, k-1) + 1 & \text{if } x_i = y_j = z_k \\ l(i, j-1, k) & \text{if } y_j \neq x_i \text{ and } y_j \neq z_k \\ l(i, j, k-1) & \text{if } z_k \neq x_i \text{ and } z_k \neq y_j \\ l(i-1, j, k) & \text{if } x_i \neq y_j \text{ and } x_i \neq z_k \end{cases}$$

where $b[i, j, k]$ is used to store the index from which the value of $l(i, j, k)$ is derived.

LCS_Length(X, Y, Z)

1 for $i \leftarrow 1$ to n and $j \leftarrow 1$ to m do $l[i, j, 0] \leftarrow 0$

```
2     for i ← 1 to n and k ← 1 to p do l[i, 0, k] ← 0
3         for j ← 1 to m and k ← 1 to p do l[0, j, k] ← 0
4     for i ← 1 to n do
5         for   j ← 1 to m do
6               for   k ← 1 to p do
7                     if  x_i = y_j = z_k
8                         then  l[i, j, k] ← l[i−1, j−1, k−1] + 1;  b[i, j, k] ← " ↖ "
9               else  if  l[i−1, j, k] ≥ max{l[i, j−1, k], l[i, j, k−1]}
10                        then  l[i, j, k] ← l[i−1, j, k];  b[i, j, k] ← " ↑ X"
11                              else  if  l[i, j−1, k] ≥ max{l[i−1, j, k], l[i, j, k−1]}
12                                    then l[i, j, k] ← l[i, j−1, k];  b[i, j, k] ← " ↑ Y"
13                                    else l[i, j, k] ← l[i, j, k−1];  b[i, j, k] ← " ↑ Z"
```

where To find the LCS, it follows the path in b back from $b[m, n, p]$, where $\nwarrow$ implies that $x_i = y_j = z_k$

$\uparrow$ X implies the three strings are X_{i-1}, Y_j and Z_k

$\uparrow$ Y implies the three strings are X_i, Y_{j-1} and Z_k

$\uparrow$ Z implies the three strings are X_i, Y_j and Z_{k-1}.

The time complexity of this algorithm is to fill in all entries in the 3-dimension array $l(\cdot, \cdot, \cdot)$ with each taking $O(1)$ time, Thus, the proposed algorithm takes $O(mnp)$ time.