Australian National University
School of Computer Science

COMP3600/COMP6466 in 2016 – Solutions to Assignment One

Due: 23:55 Friday, September 2
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).

L'Hôpital's Rule states that given two differentiable functions f and g, if $\lim_{n\to+\infty} f(n) = +\infty$, $\lim_{n\to+\infty} g(n) = +\infty$, $\lim_{n\to+\infty} \frac{f^{'}(n)}{g^{'}(n)}$ exists, and $g^{'}(n) \neq 0$, then

$$\lim_{n\to+\infty} \frac{f(n)}{g(n)} = \lim_{n\to+\infty} \frac{f^{'}(n)}{g^{'}(n)},$$

where $f^{'}(n)$ denotes the derivative of $f(n)$. Using the L'Hôpital's Rule, show that

$$100n^3 + n^2 = o(5^n).$$

$$\lim_{n\to+\infty} \frac{100n^3+n^2}{5^n} = \lim_{n\to+\infty} \frac{300n^2+2n}{5^n \ln(5)} = \lim_{n\to+\infty} \frac{600n+2}{5^n(\ln(5))^2} = \lim_{n\to+\infty} \frac{600}{5^n(\ln(5))^3} = 0.$$

Question 2 (9 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 $f(n)$, $g(n)$, and $h(n)$.

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

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

Also, given $g(n) = o(h(n))$, then, for any constant $c_2 > 0$ and there is a constant $n_1 > 0$ such that

$$0 \leq g(n) < c_2 \cdot h(n). \tag{2}$$

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

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

In other words, there exists constants $c' = c_1 c_2 > 0$ 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 $g(n) = O(h(n))$ together imply that $f(n) = \Theta(h(n))$

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

Consider $f(n) = n$, $g(n) = \log n$, and $h(n) = 100 \log n$, it is obvious that $f(n) = \Omega(g(n))$ and $g(n) = O(h(n))$, but $f(n) \neq O(h(n))$,
thus, $f(n) \neq \Theta(h(n))$.

(c) $f(n) < g(n),\ \forall n \geq 5$ implies $a^{f(n)} = o(a^{g(n)})$ $(a > 0)$.

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

Consider $f(n) = n$, $g(n) = 10n$ and $a = 1/2$. It is obvious that $a^{f(n)} \neq o(a^{g(n)})$.

Question 3 (8 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) $T(n) = T(\lceil n/4 \rceil) + n \log n$

We use mathematical induction, starting with an educated guess that the answer might be $T(n) = O(n \log n)$.

Base Case: We can assume that there is a positive constant c such that $T(n) \leq cn \log n$ for all integer numbers $0 < n \leq 16$.

Induction Hypothesis: Assume that for the same positive constant c, and all $0 < n' < n$, we have

$$T(n') \leq cn' \log n'.$$

Induction Step: By applying the recurrence, we have

$$\begin{aligned}
T(n) &= T(\lceil n/4 \rceil) + n \log n \\
&\leq c \cdot (\lceil n/4 \rceil \log(\lceil n/4 \rceil) + n \log n \quad \text{by the induction assumption} \\
&\leq c \cdot (n/4 + 1) \log(n/4 + 1) + n \log n \\
&\leq c \cdot (n/4 + 1) \log n + n \log n, \quad \text{as } \log(n/4 + 1) \leq \log n \text{ if } n \geq 2 \\
&= cn \log n/4 + c \log n + n \log n \\
&= cn \log n + (-3cn \log n/4 + c \log n + n \log n) \\
&\leq cn \log n
\end{aligned}$$

as long as $-3cn \log n/4 + c \log n + n \log n \leq 0$, that is,

$$\begin{aligned}
&- 3cn \log n/4 + c \log n + n \log n \leq 0 \\
\Leftrightarrow &4c + 4n \leq 3cn \\
\Leftrightarrow &n \geq \frac{4c}{3c - 4}
\end{aligned}$$

So, with a constant $c \geq 4$ that is large enough to work in the base case and such that $-3cn \log n/4 + c \log n + n \log n \leq 0$, we obtain $T(n) \leq cn \log n$ for all positive integer n. Considering now $T(n)$ for integer values of n, we get $T(n) = O(n \log n)$. We also have $T(n) \geq n \log n$ immediately from the definition of $T(n)$, so we can be sure that our upper bound is best possible, i.e., $T(n) = \Theta(n \log n)$.

(b) $T(n) = 2T(3n/4) + n^2$

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

$k = \log_{4/3} n$ when $(3/4)^k n = 1$. Then,

$$\begin{aligned}
T(n) &= 2T(3n/4) + n^2 \\
&= 2^2 T((\frac{3}{4})^2 n) + 2(\frac{3n}{4})^2 + n^2 \\
&= 2^2 T((\frac{3}{4})^2 n) + 2n^2 \frac{3^2}{4^2} + n^2 \\
&= 2^3 T((\frac{3}{4})^3 n) + 2^2 n^2 \frac{3^4}{4^4} + 2n^2 \frac{3^2}{4^2} + n^2 \\
&= \cdots \\
&= 2^k T(1) + n^2 \sum_{t=0}^{k-1} 2^t \frac{3^{2t}}{4^{2t}} \\
&= 2^k T(1) + n^2 \sum_{t=0}^{k-1} \frac{2^t 3^{2t}}{4^{2t}}. \\
&= 2^k T(1) + n^2 \sum_{t=0}^{k-1} \frac{18^t}{16^t}.
\end{aligned}$$

Consider now this form of $T(n)$ only for positive integer n. The summation is a geometric series, so its sum is

$$\begin{aligned}
& n^2 \sum_{t=0}^{k-1} \frac{18^t}{16^t} \\
&= n^2 \sum_{t=0}^{k-1} (\frac{9}{8})^k \\
&= n^2 \cdot \frac{(9/8)^{\log_{4/3} n} - (9/8)}{(9/8) - 1}) \\
&= n^2 \cdot (8n^{\log_{4/3}(9/8)} - 9), \quad \text{as } f(n)^{\log_a b} = b^{\log_a f(n)}. \\
&= 8n^{2+\log_{4/3}(9/8)} - 9n^2. \\
&= 8n^{3+\epsilon} - 9n^2, \text{where } \epsilon = \log_{4/3}(9/8) - 1 > 0
\end{aligned}$$

Also, $2^k = 2^{\log_{4/3} n} = n^{\log_{4/3} 2} \approx n^{1+\alpha}$ with $0 < \alpha \leq 1$. Therefore, $T(n) = 2^{\log_{4/3} n}\Theta(1) + n^2 \cdot (8n^{\log_{4/3}(9/8)} - 9) = O(n^{3+\epsilon})$, where ϵ is a constant with $0 < \epsilon \leq 1$.

(c) ((Honours & COMP6466 only) $T(n) = 3T(n/12) + T(\lfloor n/9 \rfloor) + n$

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

Induction Step:

$$\begin{aligned} T(n) &= 3T(n/12) + T(\lfloor n/9 \rfloor) + n \\ &\le 3c \cdot (\frac{n}{12}) + c \cdot (\frac{n}{9}) + n, \quad \text{as } \lfloor n/9 \rfloor \le n/9 \\ &= \frac{13cn}{36} + n \\ &= cn + (-\frac{23cn}{36} + n) \\ &\le cn, \end{aligned}$$

whenever $(-\frac{23cn}{36} + n) \le 0$, then $c \ge 36/23$ (for any positive c).

We have proved that there is a positive constant $c \ge 36/23$ such that $T(n) \le cn$ 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) $\displaystyle\sum_{k=1}^{n} k^{9/5}$.

$\sum_{k=1}^{n} k^{9/5} \le \sum_{k=1}^{n} n^{9/5} = n \cdot n^{9/5} = n^{14/5}$, thus, $\sum_{k=1}^{n} k^{9/5} = O(n^{14/5})$.

Also, $\sum_{k=1}^{n} k^{9/5} \ge \sum_{k=\lceil n/2 \rceil}^{n} k^{9/5} \ge \sum_{k=\lceil n/2 \rceil}^{n} (n/2)^{9/5}$
$= (n - \lceil n/2 \rceil + 1)(\frac{n}{2})^{9/5} \ge (\frac{n}{2})^{14/5}$, i.e, $\sum_{k=1}^{n} k^{9/5} = \Omega(n^{14/5})$.

Thus,

$$\sum_{k=1}^{n} k^{9/5} = \Theta(n^{14/5}).$$

(b) $\displaystyle\sum_{k=1}^{n} k^6/3^k$.

Let's define $a_k = \frac{k^6}{3^k}$, so $\sum_{k=1}^{n} \frac{k^6}{3^k} = \sum_{k=1}^{n} a_k$.

Now, we have

$$\frac{a_{k+1}}{a_k} = \frac{(k+1)^6/3^{k+1}}{k^6/3^k} = \frac{1}{3}(\frac{k+1}{k})^6 \le \frac{1}{3}(\frac{11}{10})^6 < 1$$

when $k \geq 10$.

Let $r = \frac{1}{3}(\frac{11}{10})^6$. When $k \geq 10$, we have $a_{k+1} \leq r \cdot a_k$, which means $a_{k+1} \leq r \cdot a_k \leq r^2 \cdot a_{k-1} \leq r^3 \cdot a_{k-2} \leq ... \leq r^{k-9} \cdot a_{10}$ 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, $\sum_{k=1}^{n} \frac{k^6}{3^k} = \Theta(1)$.

Question 5 (22 points).

Q5.(a) (12 points) Suppose we want to replicate a file over a collection of n servers, labelled $S_1, S_2, \ldots, S_n$. To place a copy of the file at server S_i results in a placement cost of c_i for an integer $c_i > 0$. Now, if a user requests the file from server S_i, and no copy of the file is present at S_i, then the servers $S_{i+1}, S_{i+2}, S_{i+3}, \ldots$ are searched in order until a copy of the file is finally found, say at server S_j with $j > i$. This results in an access cost of $j - i$. (Notice that the lower-indexed servers $S_{i-1}, S_{i-2}, \ldots$ are not consulted in this search.) The access cost is 0 if S_i does hold a copy of the file. We will require that a copy of the file be placed at server S_n, so that all such searches will terminate, at the latest at S_n.

We would like to place copies of the files at the servers so as to minimize the sum of placement and access costs. Formally, we say that a configuration is a choice, for each server S_i with $i = 1, 2, \ldots, n-1$, of whether to place a copy of the file at S_i or not. The total cost of a configuration is the sum of all placement costs of servers with a copy of the file, plus the sum of all access costs associated with all n servers.

Devise a polynomial-time algorithm to find a configuration with the minimum total cost, and analyze the running time of your algorithm. (*Hint: using Dynamic Programming and following the four steps of DP design*).

In terms of file placements, there is always a copy of the file in S_n. If we place the file at server S_i, it will incur a placement cost c_i. The structure of the optimal solution is as follows.

Let $C(i)$ be the minimum configuration cost of configuration servers $S_1, \ldots, S_i$

assuming that the file is placed at S_i. For i servers $S_1, S_2, \ldots, S_i$ to be configured with $1 \leq i \leq n$, assume that server S_k is the last server that has a copy of the file with $k < i$, and there must have a copy of the file at server S_i. Then, the configuration cost of servers $S_1, S_2, \ldots, S_i$ is $C(k) + \frac{(i-k)(i-k+1)}{2} + c_i$, where $C(k)$ the configuration cost of servers $S_1, S_2, \ldots, S_k$, and the file is placed at server S_k, there is no file placed at server from S_{k+1} to S_{i-1}, the access cost of the file at S_i from any of these servers $S_{k+1}, \ldots, S_{i-1}$ thus is $\frac{(i-k)(i-k+1)}{2}$, which is the sum of the distances between server S_i and the servers indexed between $k+1$ and $i-1$. Notice that c_i is the placement cost of the file at S_i. Since we aim to find a configuration to minimize the configuration cost, we choose a k such that the configuration cost by that k is the minimum one. Thus, we have

$$C(i) = \begin{cases} c_1, & \text{if } i = 1 \\ \min_{1 \leq k < i}\{C(k) + \frac{(i-k)(i-k+1)}{2} + c_i\}, & \text{if } i > 1 \end{cases}$$

Use the recurrence to compute $C(1), C(2), \ldots, C(n)$ in that order. Then, $C(n)$ is the answer to the problem. Clearly, the computation of all $C(i)$ takes $O(n^2)$ time.

Q5.(b) (10 points) You are going on a long trip. You start on the road at mile post 0. Along the way there are n hotels, at mile posts $a_1 < a_2 < \ldots < a_n$, where each a_i is measured from the starting point. The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance a_n), which is your destination.

You would ideally like to travel 200 miles a day, but this may not be possible (depending on the spacing of the hotels). If you travel x miles during a day, the *penalty* for that day is $(200 - x)^2$. You want to plan your trip so as to minimize the total penalty – that is, the sum, over all travel days, of the daily penalties.

Devise an efficient dynamic algorithm that determines the optimal sequence of hotels at which to stop, and analyze the running time of your algorithm. Notice that you must follow the four steps of the DP design methodology.

Step 1:

Let $a_1, a_2, \ldots, a_n$ be the given sequence. The key property is: for any j, if you stop at hotel a_j, then the minimum penalty so far is the penalties from the previous hotel a_i you stopped plus the penalty applied to the distance from a_i to a_j, assuming that $a_0 = 0$, $0 \leq i < j \leq n$.

Step 2:

This gives a recurrence. Define $P(j)$ to be the penalty so far when stopping at hotel a_j, for $0 \le j \le n$. $P(0) = 0$. Then, for $1 \le j \le n$,

$$P(j) = \min_{0 \le i \le j} \{P(i) + (200 - (a_j - a_i))^2\}.$$

Step 3:

Use the recurrence to compute $P(1), P(2), \ldots, P(n)$, in that order. Then, $P(n)$ is the answer to the problem. Clearly the computation of $P(j)$ takes $O(j)$ time and the total amount of time thus is $\sum_{j=1}^{n} O(j) = O(n^2)$ time.

Step 4:

To find the detailed tour of hotels stayed, we use another table $I()$ to remember the index, i.e., for $1 \le j \le n$,

$$P(j) = \min_{0 \le i < j \le n} \{P(i) + (200 - (a_j - a_i))^2\}.$$

$$I(j) = i,$$

if $P(j) = P(i) + (200 - (a_j - a_i))^2$.

The found trip scheduling thus is

$$a_{I^k(n)}, \ldots, a_{I^3(n)}, a_{I^2(n)}, a_{I(n)}$$

where $I^{i+1}(n) = I(I^i(n))$ and $I^{k+1}(n) = 0$.

Bonus Question (5 points)

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

BQ 1. Suppose that function $g(n)$ returns an integer and its calculation takes $\Theta(n^2 \log^2 n)$ 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 $(k >= 27)$ {
$sum = sum + g(k);$
$k = \lfloor \frac{5k}{13} \rfloor;$
}
}
return $sum;$
}

$$T(n) = \begin{cases} \Theta(1), & n < 100 \\ \sum_{i=100}^{n} \sum_{j=0}^{-\log_{5/13} i} \Theta([(5/13)^j \cdot i]^2 \log^2 \ [(5/13)^j \cdot i]), & \text{otherwise,} \end{cases}$$

while $\sum_{j=0}^{-\log_{5/13} i} \Theta(((5/13)^j i)^2 \log^2((5/13)^j i)) \leq \sum_{j=0}^{-\log_{5/13} i} O(((5/13)^j i)^2 \log^2 i) = O(i^2 \log^2 i)$,

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

.

Or, a better solution is

$\sum_{j=0}^{-\log_{5/13} i} \Theta(((5/13)^j i)^2 \log^2((5/13)^j i)) \leq \sum_{j=0}^{-\log_{5/13} i} \Theta(((5/13)^j i)^2 \log^2 i) \leq (n(5/13)^j)^2 \log^2 i \leq n^2 (5/13)^{2j} \log^2 n)$,

$$T(n) \leq \Theta(1) + \sum_{i=100}^{n} n^2 (5/13)^{2j} \log^2 n) = (n^2 \log^2 n) \sum_{i=100}^{n} (5/13)^{-\log_{5/13} i} = O(n^2 \log^2 n).$$

BQ 2. (3 points)
You are a stock trader with an initial budget of $\$n_0$.
We assume that n_0 is a multiple of $1,000$ and $n_0 \geq 10,000$.
Your goal is to invest the maximum amount of money in the stock market.
Let n_i denote your budget on day i.

The market regulator has imposed the following trading rules:

- You can perform only one of the following actions per day:
  - a1. Invest half of your budget.
  - a2. Invest two thirds of your budget.
  - a3. Invest \$1,000 from your budget.
- In order to perform action $a1$, your budget n_i should be even.
- In order to perform action $a2$, your budget n_i should be a multiple of 3.
- Your final operating budget should be exactly equal to $\$10,000$.
- The fee corresponding to one transaction is \$150.

Problem: Invest your money while paying a minimal amount of fees.

Example: Let $n_0 = \$34,000$, here are two possible solutions:

Solution 1:
Day 1 : invest half of your budget, that is $n_1 = n_0/2 = 17,000$.
Day 2 : invest \$1000 from your budget, that is $n_2 = n_1 - 1,000 = 16,000$.
Day 3 : invest \$1000 from your budget, that is $n_3 = n_2 - 1,000 = 15,000$.
Day 4 : invest \$1000 from your budget, that is $n_3 = n_2 - 1,000 = 14,000$.
Day 5 : invest \$1000 from your budget, that is $n_3 = n_2 - 1,000 = 13,000$.
Day 6 : invest \$1000 from your budget, that is $n_3 = n_2 - 1,000 = 12,000$.
Day 7 : invest \$1000 from your budget, that is $n_3 = n_2 - 1,000 = 11,000$.
Day 8 : invest \$1000 from your budget, that is $n_3 = n_2 - 1,000 = 10,000$.
Total fees = total number of transactions $\times 150 = 8 * 150 = \$1200$

Solution 2:
Day 1 : invest \$1000 from your budget, that is $n_1 = n_0 - 1,000 = 33,000$.
Day 2 : invest two thirds of your budget, that is $n_2 = n_1/3 = 11,000$.
Day 3 : invest \$1000 from your budget, that is $n_3 = n_2 - 1,000 = 10,000$.
Total fees = total number of transactions $\times 150 = 3 * 150 = \$450$

(a) Write a recursive algorithm that can explore every possible solution to your problem, and explain why 3^n is an asymptotic upper bound on the running time of your recursive algorithm.

(b) Devise a greedy algorithm for the problem, and explain why $\log_3(n)$ is a lower bound on the running time of your greedy algorithm.

(c) Propose an algorithm based on Dynamic Programming and provide a tight bound on the running time of your DP algorithm.

(a) Recursive algorithm:

Algorithm 1 `Rec(n)`

```
if n = 10000 then return 0
opt ← Rec(n-1000) + 1;
if n mod 2 = 0 and n ≥ 20000 then opt ← min(opt,Rec(n/2) + 1)
if n mod 3 = 0 and n ≥ 30000 then opt ← min(opt,Rec(n/3) + 1)
return opt
```

The number of nodes at a given level i of the recursion tree is 3^i. The longest path will be the one where n is reduced by 1000 at each level, which leads to a path of length $\frac{n}{1000} - 10$, therefore the worst case running time is $\sum_{i=0}^{\frac{n}{1000}-10} 3^i = O(3^n)$.

(b) Greedy algorithm:

1. If n is a multiple of 3000, invest 2/3 of your budget, that is divide your current budget by 3.
2. Else if n is a multiple of 2000 invest 1/2 of your budget, that is divide your current budget by 2.
3. Else if $n \geq 11000$, invest 1000, that is remove 1000 from your current budget.

In the best case scenario, the input is always a multiple of 3000, therefore, the recursion stops after dividing n by 3, $\log_3(n)$ times before reaching 10000.

(c) Dynamic Programming algorithm:

Algorithm 2 `DP(n)`

```
k ← n/1000 - 10;
t ← new int[k+1];
t[0] ← 0;
for i = 1 .. k do
    t[i] ← t[i-1] + 1;
    if i mod 2 = 0 and i ≥ 10 then t[i] ← min(t[i], t[i/2] +1)
    if i mod 3 = 0 and i ≥ 20 then t[i] ← min(t[i], t[i/3] +1)
return 150 × t[k]
```

The DP algorithm clearly runs in linear time as it only goes through the `for` loop once.