THE AUSTRALIAN NATIONAL UNIVERSITY

Second Semester 2015

COMP3600/COMP6466:Algorithms (Mid-Semester Exam Solution)

Writing Period: 2 hours duration

Study Period: 15 minutes duration

Permitted Materials: None

Answer ALL Questions

All your answers must be written in the boxes provided in this booklet. You may be provided with scrap paper for working, but it must **not** *be used to write final answers.*

There is additional space at the end of the booklet in case the boxes provided are insufficient. Label such overflow boxes with the question number. Note that the full mark is 100.

Do not remove this booklet from the examination room.

Student Number:

Official use only:

Q1 (30)	Q2 (20)	Q3 (30)	Q4 (20)	Total (100)

QUESTION 1 [30 marks]

(a) Order the following complexity classes from slowest-growing to fastest growing.

$\Theta(n^2 \log^3 n),\ \Theta(n^5),\ \Theta(n^{\log_8 7}),\ \Theta(n \log^4 n),\ \Theta(n^{\log_2 n}),\ \Theta(\sqrt{n}),\ \Theta(2^n),\ \Theta(\log \log \log n)$

Answer: $\Theta(\log \log \log n),\ \Theta(\sqrt{n}),\ \Theta(n^{\log_8 7}),\ \Theta(n \log^4 n),\ \Theta(n^2 \log^3 n),\ \Theta(n^2 \log^5 n),\ \Theta(n^5),$ $\Theta(n^{\log_2 n}),\quad \Theta(2^n).$

(b) Let $f(n)$, $g(n)$ and $h(n)$ be three positive functions. Either prove that the following statements are true, or disprove them by providing a counterexample:

(i) $\max(f(n), g(n), h(n)) = \Theta(f(n) + g(n) + h(n))$. (5 points)

Answer: First of all, we have

$$f(n) \leq f(n) + g(n) + h(n), \tag{1}$$

$$g(n) \leq f(n) + g(n) + h(n), \tag{2}$$

and

$$h(n) \leq f(n) + g(n) + h(n). \tag{3}$$

From Inequalities (1), (2), and (3), we have

$$\max\{f(n), g(n), h(n)\} \leq f(n) + g(n) + h(n). \tag{4}$$

Meanwhile,

$$f(n) \leq \max\{f(n), g(n), h(n)\}, \tag{5}$$

$$g(n) \leq \max\{f(n), g(n), h(n)\}, \tag{6}$$

and

$$h(n) \leq \max\{f(n), g(n), h(n)\}. \tag{7}$$

From Inequalities (5), (6), and (7), we have

$$f(n) + g(n) + h(n) \leq 3 \max\{f(n), g(n), h(n)\}, \tag{8}$$

In other words, from Inequality (8),we have

$$\frac{1}{3}(f(n) + g(n) + h(n)) \leq \max\{f(n), g(n), h(n)\}, \tag{9}$$

Combining both inequality (4) and inequality (9), there are positive constants $c_1 = 1/3$ and $c_2 = 1$ and $n_0 = 1$ such that for any $n \geq n_0$, the following inequalities hold.

$$0 \leq c_1(f(n) + g(n) + h(n)) \max\{f(n), g(n), h(n)\} \leq c_2(f(n) + g(n) + h(n)). \quad (10)$$

i.e, $\max(f(n), g(n), h(n)) = \Theta(f(n) + g(n) + h(n))$, the claim thus is true.

(ii) $f(n) = O(g(n))$ and $g(n) = \Omega(h(n))$ implies $f(n) = O(h(n))$. (5 points)

Answer: Let $f(n) = n \log n$, $g(n) = n^2$, and $h(n) = n$, then $f(n) = O(g(n))$ and $g(n) = \Omega(h(n))$, but $f(n) \neq O(h(n))$. The claim is false.

(c) Provide the simplest expression for $\sum_{k=1}^{n} k^{13/3}$, using the $\Theta(\,)$ notation. Explain your reasoning clearly. (4 points)

Answer: $\sum_{k=1}^{n} k^{13/3} \leq \sum_{k=1}^{n} n^{13/3} = n \cdot n^{13/3} = O(n^{16/5})$.

On the other hand,

$$\begin{aligned}
\sum_{k=1}^{n} k^{13/3} &= \sum_{k=1}^{\lceil n/2 \rceil - 1} k^{13/3} + \sum_{k=\lceil n/2 \rceil}^{n} k^{13/3} && (11)\\
&\geq \sum_{k=\lceil n/2 \rceil}^{n} k^{13/3} && (12)\\
&\geq (n - \lceil n/2 \rceil + 1)(\lceil n/2 \rceil)^{13/3} && (13)\\
&\geq (n - (n/2 + 1) + 1)(\lceil n/2 \rceil)^{13/3} && (14)\\
&\geq (n/2)(\lceil n/2 \rceil)^{13/3} && (15)\\
&\geq (n/2)(n/2)^{13/3} && (16)\\
&= (n/2)^{16/3} && (17)\\
&= \Omega(n^{16/3}), && (18)
\end{aligned}$$

where the lower bound in (15) follows from the fact that there are more than $n/2$ additive terms in (12) and the smallest term is $(\lceil n/2 \rceil)^{13/3}$.

The asymptotic upper and lower bounds together imply $\sum_{k=1}^{n} k^{13/3} = \Theta(n^{16/3})$.

(d) Using the $O(\,)$ notation, give asymptotic upper bounds for $T(n)$ in the following recurrences. Note that you are **not** allowed to use the Master theorem.

(i) $T(n) = 3T(n/11) + n\log n$. (Notice that $\frac{n}{11^k}\log\frac{n}{11^k} \leq \frac{n}{11^k}\log n$ for any integer $k \geq 0$.) (6 (COMP3600) and 4 (COMP6466 and Honours)

Answer: Assume that $T(n)$ is defined for every positive integer. We apply the iteration method. It is obvious that $k = \log_{11} n$ when $(\frac{1}{11})^k n = 1$. Then,

$$\begin{aligned}
T(n) &= 3T(n/11) + n\log n \\
&= 3^2T((\frac{1}{11})^2 n) + n\log n + 3(n/11)\log(n/11) \\
&\leq 3^3T((\frac{1}{11})^3 n) + n\log n + 3(n/11)\log n + 3^2(n/11^2)\log n \\
&= \cdots \\
&\leq 3^kT(1) + n\log n\sum_{i=0}^{k-1}(\frac{3}{11})^i \\
&= 3^{\log_{11} n}T(1) + n\log n\Theta(1) \\
&= o(n) + \Theta(n\log n), \quad \text{since } 3^{\log_{11} n} = n^{\log_{11} 3} \text{ and } \log_{11} 3 < 1
\end{aligned}$$

using the fact that a geometric sum is Θ(largest term). Thus, the answer is $T(n) = O(n\log n)$.

(ii) $T(n) = T(n/7) + n^3$. Assume that n is a power of 7 and $T(n)$ is constant for $n \leq 14$. (6 (COMP3600) and 4 (COMP6466 and Honours))

Answer: We apply the iteration method (apply the recurrence to itself repeatedly). We assume that n is a power of 7, and once $T(n)$ is written as a sum, we assume that the same form can be obtained for every positive integer n (to get an asymptotic upper bound). It is obvious that $k = \log_7 n$ when $(\frac{1}{7})^k n = 1$. Then,

$$\begin{aligned}
T(n) &= T(n/7) + n^3 \\
&= T((\frac{1}{7})^3 n) + n^3 + (n/7)^3 \\
&= \cdots \\
&= T(1) + ((1/7)^3)^{k-1}n^3 + ((1/7)^3)^{k-2}n^3 + ((1/7)^3)^{k-3}n^3 + \ldots + n^3) \\
&= T(1) + n^3\sum_{i=0}^{k-1}((1/7)^3)^i \\
&= T(1) + n^3\Theta(1) \\
&= \Theta(n^3),
\end{aligned}$$

using the fact that a geometric sum is Θ(largest term). Since we only need to give an asymptotic upper bound, the answer is $T(n) = O(n^3)$.

(iii) $T(n) = T(\sqrt{n}) + \log\log n$. Assume that $T(n)$ is constant for $n \leq 16$. (**COMP6466 and Honours students only**). (4 points)

Answer: We make use of a transformation method to prove this. Let $n = 2^m$ then, $m = \log n$ and $\log m = \log\log n$.
Let $T(n) = S(m)$, then $T(n) = T(\sqrt{n}) + \log\log n$ can be rewritten into

$$S(m) = S(m/2) + \log m. \tag{19}$$

The recurrence (19) can be solved, using either the inductive method or the recursive-tree method, which is

$$S(m) = c' \log^2 m,$$

where $c' > 0$ is a constant, assuming $m \geq 1$, i.e., $n \geq 16$. Thus, $T(n) = S(m) = c' \log^2 m = c' \log^2(\log n)$.
When $n \neq 2^m$, assume that $2^{m-1} \leq n < 2^m$. Let $n' = 2^m$, we then have $T(n) \leq T(n') = c' \log^2(\log n')$. Thus, $T(n) = O(\log^2(\log n))$.

QUESTION 2 [20 marks]

(a) Show that under the comparison computational model, the lower bound of sorting problem is $\Omega(n \log n)$, where n is the number of elements to be sorted and $n! = \sqrt{2\pi n} \cdot (\frac{n}{e})^n \cdot (1 + \Theta(1/n))$ by Stirling's approximation. (5 points)

Answer: To sort n elements in increasing (or decreasing) order, there are $n!$ possibilities ($n!$ arrangements of n elements in the sorted sequence). If we use a binary decision tree to represent this sorting procedure, it can be seen that each node (corresponding to one comparison between two elements) in the tree has two children (the left child $\leq$ and the right child $>$). Each leaf node in the decision tree represents a comparison sequence of the n elements (from the root to the leaf, thereby a sorted sequence), and the number of nodes in the path from the tree root to the leave node is the number of comparisons needed to get that sorted sequence. Then, the longest path from the root of the binary decision tree to one of the leaves is the worst time complexity (the number of comparisons needed for sorting any n elements). To show the lower bound on sorting, we need to construct a binary decision tree which contains at least $n!$ leaves such that the longest path from the tree root to any leaves as short as possible. Let h be the longest path in such a tree constructed, then the tree can contain at most 2^h leaves, i.e.,

$$2^h \geq n!$$

We then have $h \geq \log n! = \log \sqrt{2\pi n} \cdot (\frac{n}{e})^n \cdot (1 + \Theta(1/n)) = \Omega(n \log n)$. In other words, the minimum number of comparisons for sorting n elements is at least $\Omega(n \log n)$.

(b) Propose an algorithm to identify the minimum and the maximum elements from a set of n distinct elements such that the number of comparisons by the algorithm is no more than $3\lfloor n/2 \rfloor$. (5 points).

Answer: Let $a_1, a_2, \ldots, a_n$ be the n elements.

We first assume that n is even. if not, we deal with the first $n-1$ elements ($n-1$ is even). We then have the following algorithm.

Step 1, partition the n elements into $n/2$ groups with group i containing two elements a_{2i-1} and a_{2i} for all i with $1 \leq i \leq n/2$.

Step 2, compare the two elements for each group, the smaller one is added to a subset S_1 and the larger one is added to another subset S_2.

Step 3, Find the minimum element from S_1 and the maximum element from S_2.

Thus, the total number of comparisons are $n/2$ $(Step\ 2) + (n/2-1)$ $(Step\ 3) + (n/2-1)$ $(Step\ 3) = 3(n/2) - 2 \leq 3\lfloor n/2 \rfloor$.

If n is odd, the last element will be compared with both found the minimum and maximum values, this takes 2 extra comparisons. Thus, the total number of comparison is $(n-1)/2$ $(Step\ 2) + ((n-1)/2-1)$ $(Step\ 3) + ((n-1)/2-1)$ $(Step\ 3) + 2 = 3((n-1)/2) - 2 + 2 = 3(n-1)/2 \leq 3\lfloor n/2 \rfloor$.

(c) In the linear-time selection algorithm in the textbook, the elements in the input set are divided into groups of size 5. Show that the algorithm still runs in linear time if the input elements are divided into groups of size 23 instead.

(i) Provide a recurrence of time complexity of this modified selection algorithm $T(n)$, assuming there are n elements in the input set. (5 points).

Answer: Recall that we need to find a pivot element x to partition the set into three disjoint subsets R_1, R_2 and R_3. We assume that the ith smallest element that we are looking for is in R_3, we now show that the size of R_1 is lower bounded. Otherwise (i.e., the element is in R_1 or R_2), we can show that the size of R_3 is lower bounded as well.

$$|R_1| \geq 12(\frac{\lceil n/23 \rceil}{2} - 2) \geq \frac{6n}{23} - 24.$$

Thus, the size of R_3 is

$$|R_3| = n - |R_1| - |R_2| \leq n - |R_1| \leq \frac{17n}{23} + 24.$$

We thus have

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n \leq n_0 \\ T(\lceil n/23 \rceil) + T(17n/23 + 24) + O(n) & \text{otherwise,} \end{cases}$$

where n_0 is a given constant, which will be determined in the next subquestion.

(ii) Show that the algorithm takes linear running time (**Hint:** use mathematical induction). (5 points)

Answer: We can prove by the substitution method that there is a constant c such that for any positive integer $n > 0$, $T(n) \leq cn$.
Choose a c large enough that $T(n) \leq cn$ for $n \leq 200$.

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n \leq 200 \\ T(\lceil n/23 \rceil) + T(17n/23 + 24) + O(n) & \text{if } n > 200 \end{cases}$$

For $n > 200$, the recurrence says

$$\begin{aligned} T(n) &\leq T(\lceil n/23 \rceil) + T(17n/23 + 24) + an && (20) \\ &\leq c\lceil n/23 \rceil + 17cn/23 + 24c + an && (21) \\ &\leq cn/23 + c + 17cn/23 + 24c + an && (22) \\ &= 18cn/23 + 25c + an && (23) \\ &= cn + (-5cn/23 + 25c + an) && (24) \end{aligned}$$

The expression $-5cn/23 + 25c + an$ is negative if $n > 200$ and $c \geq \frac{23*a*n}{5n-25*23} = \frac{23an}{5n-575} = \frac{4600a}{425}$ when $n_0 = 200$. Therefore, for some c, $T(n) \leq cn$ always.

QUESTION 3 [30 marks]

(a) Given a sequence of n distinct positive integers: $a_1, a_2, \ldots, a_n$,

(i) devise a dynamic programming algorithm to find a *longest decreasing subsequence* from the sequence. (10 points)

Answer: Let $a_1, a_2, \ldots, a_n$ be the given sequence of n distinct positive integers. The key property is: for any j, the longest decreasing sequence ending with a_j consists of either a_j only or a longest decreasing sequence ending with some earlier element a_i that is larger than a_j with $i < j$. This gives a recurrence. Denote by $l(j)$ the length of the longest decreasing sequence ending with a_j for all j with $1 \leq j \leq n$. Then, for $1 \leq j \leq n$,

$$l(j) = \max_{i<j \& a_i \geq a_j} \{1, l(i) + 1\},$$

and

$$l(1) = 1.$$

Define another one dimension index array I as follows.
$I(j) = i_0$ if $l(j) = l(i_0) + 1$ and $a_{i_0} \geq a_j$ with $1 \leq i_0 < j \leq n$, otherwise, $I(j) = j$.

The length of the longest decreasing subsequence of the given sequence thus is $\max_{1 \leq i \leq n}\{l(i)\}$.

(ii) Analyze the time complexity of the proposed algorithm. (5 points)

Answer: The algorithm implementation is quite straightforward. We calculate $l(1), l(2), \ldots, l(n)$, then find an index j such that $l(j)$ is the maximum one. Clearly, this algorithm takes $\sum_{j=1}^{n} O(j) = O(n^2)$ time as the calculation of each $l(j)$ takes $O(j)$ time.

(iii) Use the pseudo-code to describe the procedure of printing out the found decreasing subsequence. (4 points)

Answer: In order to find the subsequence, we have an index array I in step 3 for the book-keeping purpose of calculating $l(j)$. Having both arrays $l(.)$ and $I(.)$, the following procedure can be used to find the longest decreasing subsequence.

Let $l(j)$ be the maximum value and k the length of the subsequence, then the longest decreasing subsequence is $a_{I^{k-1}(j)}, a_{I^{k-2}(j)}, \ldots, a_{I^1(j)}, a_j$ where $I^{k'}(j) = I(I^{k'-1}(j))$, $I^0(j) = j$ and $I^1(j) = I(j)$.

(b) We are looking at the price of a given stock over n consecutive days, numbered $i = 1, 2, \ldots, n$. For each day i, we have a price $p(i)$ per share for the stock on that day. For simplicity, we assume that the price was fixed during each day. We would like to know:

(i) How should we choose a day i on which to buy the stock and a later day $j > i$ on which to sell it, if we want to maximize the profit per share, $p(j) - p(i)$? Give an algorithm for finding the optimal numbers i and j with $1 \leq i < j \leq n$. (8 points)

Answer: Consider a day i, let $P_{min}(i)$ be the minimum price to buy the stack prior to day i, i.e.,

$$P_{min}(i) = \min\{\, p(j) \mid 1 \leq j \leq i\} = \min\{P_{min}(i-1), p(i)\}.$$

Let $P_{max}(i)$ be the maximum price to sell the stack after day i, then

$$P_{max}(i) = \max\{\, p(j) \mid i+1 \leq j \leq n\} = \max\{P_{max}(i+2), p(i+1)\}.$$

The maximum profit of buying and selling the stack thus is

$$\max\{P_{max}(i) - P_{min}(i) \mid 1 \leq i \leq n\}.$$

Initially, $P_{min}(1) = p(1)$ and $P_{max}(n) = p(n)$.
Calculate $P_{min}(2), P_{min}(3), \ldots, P_{min}(n-1), P_{min}(n)$.
Note that we keep the index $I_{min}(i)$ of the day to buy the stack when calculating $P_{min}(i)$, and clearly we have $1 \leq I_{min}(i) \leq i$.
Similarly, we can calculate $P_{max}(n-1), P_{max}(n-2), \ldots, P_{max}(2)$, and use the index array $I_{max}(i)$ of the day to sell the stack when calculating $P_{max}(i)$, then $2 \leq I_{max}(i) \leq n$.
The calculations of both $P_{min}(i)$ and $P_{max}(i)$ for all i with $1 \leq i \leq n$ takes $O(n)$ time. Assume that $P_{max}(i_0) - P_{min}(i_0) = \max\{P_{max}(i) - P_{min}(i) \mid 1 \leq i \leq n\}$. Then, the stack buying day is $I_{min}(i_0)$ while its selling day is $I_{max}(i_0)$. The identification of i_0 takes $O(n)$ time. Thus, the entire algorithm takes $O(n)$ time.

(ii) Analyze the time complexity of the algorithm. (4 points).

Answer: As the calculation of both $P_{min}(i)$ and $P_{max}(i)$ for all i with $1 \leq i \leq n$ takes $O(n)$ time. Then, identifying the maximum value of $P_{max}(i) - P_{min}(i)$ takes $O(n)$ time too, which also will identify the buying and selling days $I_{min}(i_0)$ and $I_{max}(i_0)$ to maximize the profit.

QUESTION 4 [20 marks]

(a) We use Huffman's algorithm to obtain an encoding of alphabet $\{a, b, c\}$ with frequencies f_a, f_b, f_c. In each of the following cases, either give an example of frequencies $\{f_a, f_b, f_c\}$ that would yield the specified code, or explain why the code cannot possibly be obtained (no matter what the frequencies are). (6 points)

- (I) Code: $\{0, 10, 11\}$
- (II) Code: $\{0, 1, 00\}$
- (III) Code: $\{10, 01, 00\}$

Answer:

- (I) true, $f_a = 1/2, f_b = f_c = 1/4$
- (II) false, because codeword 0 is the prefix of codeword 00, which is not allowed by Huffman's coding
- (III) false, as in the Huffman's coding, there are at most two codewords with the length 2. In this case, there are 3 codewords with length 2, there is not such a Huffman tree.

(b) Consider a distribution over n possible outcomes, with probabilities $p_1, p_2, \ldots, p_n$.

(i) Assume that each p_i is a power of 2 (that is, of the form $1/2^k$). Suppose a long sequence of m samples is drawn from the distribution and that for all $1 \leq i \leq n$, the i-th outcome occurs exactly mp_i times in the sequence. Show that if Huffman encoding is applied to this sequence, the resulting encoding will have length $\sum_{i=1}^{n} mp_i \log \frac{1}{p_i}$. (5 points)

Answer: Following the Huffman's coding, each possible distribution corresponds to a character in the Huffman coding, and the number of appearances of the character corresponding to distribution p_i in the sample sequence is mp_i. Let $l(p_i)$ be the length of the codeword in the Huffman tree for distribution p_i, then $l(p_i) = \log \frac{1}{p_i}$. Since p_i is the power of 2, $l(p_i)$ is a positive integer. The total length of all distribution appearances in the sample sequence thus is $\sum_{i=1}^{n} mp_i l(p_i) = \sum_{i=1}^{n} mp_i \log \frac{1}{p_i}$.

(ii) Now consider arbitrary distributions - that is, the probabilities p_i are not restricted to power of 2. The most commonly used measure of the amount of randomness in the distribution is the entropy $\sum_{i=1}^{n} p_i \log \frac{1}{p_i}$. (1) For what distribution (over n outcomes) is the entropy the largest possible? and (2) for what distribution (over n outcomes) is the entropy the smallest possible? (5 points)

Answer: It can be seen that when p_i becomes smaller, the codeword for distribution p_i becomes longer, the value of $\log \frac{1}{p_i}$ becomes larger, following Huffman's coding.

(1) To maximize the entropy $\sum_{i=1}^{n} p_i \log \frac{1}{p_i}$, we must have $p_1 = p_2 = \ldots = p_n$. Thus, $n \cdot p = 1$, then $p = 1/n$, i.e.,

$$\sum_{i=1}^{n} p_i \log \frac{1}{p_i} = np \log \frac{1}{p} = \log n.$$

(2) To minimize the entropy $\sum_{i=1}^{n} p_i \log \frac{1}{p_i}$, we have $p_i = 1/2^i$, then $p_i \log \frac{1}{p_i} = \frac{i}{2^i}$, and

$$\sum_{i=1}^{n} p_i \log \frac{1}{p_i} = \sum_{i=1}^{n} \frac{i}{2^i}.$$

Let $a_i = \frac{i}{2^i}$, then $\frac{a_{i+1}}{a_i} = \frac{i+1}{2i} = 1/2(1 + 1/i) \leq 3/4$ when $i \geq 2$. Thus, $\sum_{i=1}^{n} p_i \log \frac{1}{p_i} = \sum_{i=1}^{n} a_i \leq a_1 + \sum_{i=2}^{n} a_2 (3/4)^{i-1} \leq 1/2 + a_2 \sum_{i=1}^{\infty} (\frac{3}{4})^{i-1} = 1/2 + a_2 \frac{1}{1-3/4} = 1/2 + 1/2 * (1/4) = 1/2 + 1/8 = 5/8$.

(c) Under a Huffman encoding of n symbols with frequencies $f_1, f_2, \ldots, f_n$, what is the longest a codeword could possible be? Give an example set of frequencies that would produce this case. (4 points)

Answer: The longest a codeword can be $n-1$. Let $f_{n-1} = f_n = \alpha$ with $0 < \alpha < 1$, $f_{n-i} = 2^{i-1}\alpha$ for all i with $3 \leq i \leq n-1$, where the value of α can be derived by solving the equation $\sum_{i=1}^{n} f_i = 1$.
