THE AUSTRALIAN NATIONAL UNIVERSITY

Second Semester 2016

COMP3600/COMP6466:Algorithms (Solutions to Questions in Mid-Semester Exam)

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. (4 points)

$\Theta(n^3 \log^3 n),\ \Theta(n^{2.5}),\ \Theta(n^{\log_4 5}),\ \Theta(\log n \log^4 n),\ \Theta(n^{\log_2 n}),\ \Theta(n^6),\ \Theta(3^n),\ \Theta(n \log\log\log n)$

Answer: $\Theta(\log n \log^4 n),\ \Theta(n \log\log\log n),\ \Theta(n^{\log_4 5}),\ \Theta(n^{2.5}),\ \Theta(n^3 \log^3 n),\ \Theta(n^6),\ \Theta(n^{\log_2 n}),$ $\Theta(3^n)$.

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

Answer: First of all, as $f(n) \geq 0$, $g(n) \geq 0$ and $h(n) \geq 0$, we have

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

and

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

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

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

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

In other words, there are positive constants c and n_0 such that for any $n \geq n_0$, the following inequalities hold.

$$0 \leq \max\{f(n), g(n), h(n)\} \quad \leq \quad c(f(n) + g(n) + h(n)). \tag{5}$$

Take, for example, $c = 1$ and $n_0 = 1$.
So, $\max\{f(n), g(n), h(n)\} = O(f(n) + g(n) + h(n))$.
The claim is true.

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

Answer: Let $f(n) = n \log n$ and $g(n) = n^2$, $h(n) = n$; 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 $\sum_{k=1}^{n} k^{17/5}$, using the $\Theta(\,)$ notation. Explain your reasoning clearly. (5 points)

Answer: $\sum_{k=1}^{n} k^{17/5} \leq \sum_{k=1}^{n} n^{17/5} = n^{22/5}$, thus, $\sum_{k=1}^{n} k^{17/5} = O(n^{22/5})$.
Also, $\sum_{k=1}^{n} k^{17/5} \geq \sum_{k=\lceil n/2 \rceil}^{n} k^{17/5} \geq \frac{n}{2}(\frac{n}{2})^{17/5} = (\frac{n}{2})^{22/5}$, i.e, $\sum_{k=1}^{n} k^{17/5} = \Omega(n^{22/5})$.
Thus,

$$\sum_{k=1}^{n} k^{17/5} = \Theta(n^{22/5}).$$

(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) $T(n) = 4T(n/5) + n \log n$. (Notice that $\frac{n}{5^k} \log \frac{n}{5^k} \leq \frac{n}{5^k} \log n$ for any integer $k \geq 0$.) (5 points)

Answer: Assume that $T(n)$ is defined for every positive integer. We apply the iteration method (apply the recurrence to itself repeatedly). It is obvious that $k = \log_5 n$ when $(\frac{1}{5})^k n = 1$. Then,

$$\begin{aligned}
T(n) &= 4T(n/5) + n \log n \\
&= 4^2 T((\frac{1}{5})^2 n) + n \log n + 4(n/5) \log(n/5) \\
&\leq 4^3 T((\frac{1}{5})^3 n) + n \log n + 4(n/5) \log n + 4^2 (n/5^2) \log n \\
&= \cdots \\
&\leq 4^k T(1) + n \log n \sum_{i=0}^{k-1} (\frac{4}{5})^i \\
&= 4^{\log_5 n} T(1) + n \log n \Theta(1) \\
&= o(n) + \Theta(n \log n),
\end{aligned}$$

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

(ii) $T(n) = T(\lceil n/6 \rceil) + 3T(\lfloor n/11 \rfloor) + n^2$. Assume that $T(n)$ is constant for $n \leq 66$. [6 points (COMP3600) and 4 points (COMP6466 and Honours)

Answer: We use the substitution method (i.e., mathematical induction), starting with the guess that the answer might be $T(n) = O(n^2)$.

We prove that there is a positive constant c such that $0 \leq T(n) \leq cn^2$ for all positive integers of n (which implies $T(n) = O(n^2)$). The result holds for $n \leq 66$ as $T(n)$ is constant for these values. (This is the base case.)

Assume that for some positive constant c (the same that was used in the base case), we have

$$T(n') \leq cn'^2$$

for all n' such that $0 < n' < n$. (This is the induction hypothesis.) Then, applying the recurrence for any $n \geq 66$, we have

$$\begin{aligned}
T(n) &= T(\lceil n/6 \rceil) + 3T(\lfloor n/11 \rfloor) + n^2 \\
&\leq c(\lceil n/6 \rceil)^2 + 3c(\lfloor n/11 \rfloor)^2 + n^2 \\
&\leq c[(n/6) + 1]^2 + 3c(n/11)^2 + n^2 \\
&= c(n/6)^2 + cn/3 + c + 3c(n/11)^2 + n^2 \\
&= \frac{229cn^2}{4256} + n^2 + cn/3 + c \\
&= cn^2 + (n^2 + cn/3 + c - \frac{(4256 - 229)cn^2}{4256}) \\
&\leq cn^2,
\end{aligned}$$

as long as $(n^2 + cn/3 + c - \frac{(36\times121-229)cn^2}{36\times121})$ is no greater than zero, then the above inequality holds, i.e.,

$$n^2 + cn/3 + c - \frac{(36 \times 121 - 229)cn^2}{36 \times 121}) \leq 0,$$

which is equivalent to the following inequality

$$3 + 1/n \leq (\frac{4029}{4256} - 1/c)n, \tag{6}$$

When $c \geq 4256$ and $n_0 \geq 6$, inequality (6) always holds.

In other words, there are constants $c = 4256 > 0$ and $n_0 \geq 6$, for any $n \geq n_0$, we have $T(n) \leq cn^2$. By induction, we have $T(n) = O(n^2)$.

(iii) $T(n) = T(\sqrt{n}) + \log^2 n$. Assume that $T(n)$ is constant for $n \leq 16$. (**COMP6466 and Honours students only**). (2 points)

Answer: We use the substitution method (i.e., mathematical induction), starting with the guess that the answer might be $T(n) = O(\log^2 n)$.

We prove that there is a positive constant c such that $0 \leq T(n) \leq c \log^2 n$ for all positive integers of n (which implies $T(n) = O(\log^2 n)$). The result holds for $n \leq 4$ as $T(n)$ is constant for these values. (This is the base case.)

Assume that for some positive constant c (the same that was used in the base case), we have

$$T(n') \leq c \log^2 n'$$

for all n' such that $0 < n' < n$. (This is the induction hypothesis.) Then, applying the recurrence for any $n \geq 4$, we have

$$\begin{aligned} T(n) &= T(\sqrt{n}) + \log^2 n \\ &\leq c \log^2 \sqrt{n} + \log^2 n \\ &= \frac{c}{4} \log^2 n + \log^2 n \\ &= c \log^2 n + (\log^2 n - \frac{3c}{4} \log^2 n) \\ &\leq c \log^2 n, \end{aligned}$$

as long as $(\log^2 n - \frac{3c}{4} \log^2 n)$ is negative, then the above inequality holds, i.e., $c \geq 4/3$.
By induction, we have $T(n) = O(\log^2 n)$.

QUESTION 2 [20 marks]

(a) You are given two sorted lists of size m and n. Give an $O(m + n)$ time algorithm for computing the kth smallest element in the union of the two lists, where k is a given value with $1 \leq k \leq m + n$. (6 points)

Answer: Let A and B be the two sorted lists of size m and n respectively. Then, it takes $O(m + n)$ time to merge them into a sorted list C of size $m + n$ due to the fact that both A and B are sorted lists. Thus, the kth smallest element in C is the element with index k, which can be identified in $O(1)$ time. Thus, the time complexity of this algorithm is $O(m + n) + O(1) = O(m + n)$.

(b) 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. (6 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 $\Omega(n \log n)$.

(c) There is an algorithm for the selection problem whose time complexity is expressed by the following recurrence.

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n \leq 100 \\ T(\lceil n/17 \rceil) + T(25n/34 + 18) + O(n) & \text{if } n > 100 \end{cases}$$

Show that the algorithm takes linear running time (**Hint:** use mathematical induction). (8 points)

Answer: We can prove by the method of substitution that there is a constant c such that for any rational number $n > 0$, $T(n) \leq cn$. Choose c large enough that $T(n) \leq cn$ for $n \leq 100$.

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n \leq 100 \\ T(\lceil n/17 \rceil) + T(25n/34 + 18) + O(n) & \text{if } n > 100 \end{cases}$$

For $n > 100$, the recurrence says

$$\begin{aligned} T(n) &\leq T(\lceil n/17 \rceil) + T(25n/34 + 18) + an && (7) \\ &\leq c\lceil n/17 \rceil + 25cn/34 + 18c + an && (8) \\ &\leq cn/17 + c + 25cn/34 + 18c + an && (9) \\ &= 27cn/34 + 19c + an && (10) \\ &= cn + (-7cn/34 + 19c + an) && (11) \\ &\leq cn && (12) \end{aligned}$$

The expression $-7cn/34 + 19c + an$ is negative if $n > 100$ and $c \geq \frac{34*100*a}{34}$. 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 increasing 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 increasing sequence ending with a_j consists of either a_j only or a longest increasing sequence ending with some earlier element a_i that is no greater than a_j with $i < j$. This gives a recurrence. Denote by $l(j)$ the length of the longest increasing sequence ending with a_j for all j with $1 \leq j \leq n$. Then, for $1 \leq j \leq n$,

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

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 $a_{i_0} \leq a_j$ with $1 \leq i_0 < j \leq n$, otherwise, $I(j) = j$.

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

(ii) Analyze the time complexity of the proposed algorithm. (3 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 increasing subsequence. (3 points)

Answer: In order to find the subsequence, we have an index array I in step 3 of DP design 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 increasing subsequence.

Let $l(j)$ be the maximum value and k the length of the subsequence, then the longest increasing 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) You are given a string of n characters $s[1 \ldots n]$, which you believe to be a corrupted text document in which all punctuation has vanished (so that it looks something like "it-wasthebestoftimes..."). You wish to reconstruct the document using a dictionary, which is available in the form of a Boolean function `dict(·)`: for any string w,

$$dict(w) = \begin{cases} true & \text{if } w \text{ is a valid word} \\ false & \text{otherwise} \end{cases}$$

(i) Give a dynamic programming algorithm that determines whether the string $s[\cdot]$ can be reconstructed as a sequence of valid words. The running time of your algorithm should be $O(n^2)$, assuming that calls to `dict` take unit time. (10 points)

Answer: Let $s[1 \ldots j] = s_1 s_2 \ldots s_j$ be a substring of the original string of $s_1 s_2 \ldots s_n$ starting from s_1 and ending at s_j. Let $B(j)$ be a Boolean value for the substring $s[1 \ldots j]$ which is defined as follows. If $s[1 \ldots j]$ is either a word or a valid sequence of words, then $B(j) =' true'; otherwise$,B(j)='false'. Thus, when $B(j) =' true'$ then, either $s[1 \ldots j]$ itself is a word, or $s[1 \ldots j]$ is a valid sequence of words that implies there is an integer k with $1 \leq k < j$ such that substring $s[1 \ldots k]$ is a valid sequence of words and substring $s[k+1 \ldots j]$ is a word. Thus, it gives the following recurrence.

$$B(j) = \begin{cases} dict(s_j) & j = 1 \\ dict(s[1 \ldots j]) \bigvee \exists_{1 \leq k < j}(B(k) \wedge dict(s_{k+1} \ldots s_j)) & \text{otherwise} \end{cases}$$

To memorize how to obtain $B(j)$, we use another 1-D index array $I(j)$ which is defined as follows.

If $B(j)$ is true and $dict(s[1 \ldots j])$ is true, we set $I(j) = 1$; otherwise, there must exist an integer k with $1 \leq k < j$ such that both $B(k)$ and $dict(s_{k+1} \ldots j)$ are true, we set $I(j) = k + 1$.

We then solve the defined recurrence by calculating $B(1), B(2), \ldots, B(n)$ and the value of $I(1), I(2), \ldots, I(n)$ at the same time. Notice that the calculation of $B(j)$ and $I(j)$ takes $O(j)$ time. Thus, to calculate all $B(j)$ ($I(j)$) with $1 \leq j \leq n$, it takes in total $O(\sum_{j=1}^{n} O(j)) = O(n^2)$ time.

(ii) In the event that the string is valid, make your algorithm output the corresponding sequence of words (4 points).

Answer: In order to print out the corresponding sequence of words, we have an index array I in step 3 of DP design for the book-keeping purpose of calculating $B(j)$. Having both 1-D arrays $B(.)$ and $I(.)$, the following procedure can be used to find the corresponding sequence of words.

If $B(n)$ is true and $I(n) \neq 1$, then the original sequence consists of a substring $s[1 \ldots I(n) - 1]$ and a word $s[I(n) \ldots n]$, the substring $s[1 \ldots I(n) - 1]$ can then

be printed out recursively, i.e., it prints out the substring $s[1 \ldots I(I(n)-1)-1]$ and word $s[I(I(n)-1) \ldots I(n)-1]$ before the word $s[I(n) \ldots n]$. This procedure continues until the word containing s_1 is printed out.

QUESTION 4 [20 marks]

(a) Indicate whether each of the following properties is true for a Huffman code, where an optimal prefix code is called a Huffman code if in which no codeword is also a prefix of some other codeword. (3 points)

(i) The codewords of two least frequent symbols have the same length.

(ii) The length of the codeword of a more frequent symbol is always smaller than or equal to the length of the codeword of a less frequent one.

(iii) The length of the codeword of a more frequent symbol is always larger than that of the codeword of a less frequent one.

Answer:

- (i) true
- (ii) true
- (iii) false

(b) Suppose the symbols a, b, c, d, e occur with frequencies 3/20, 3/20, 1/10, 1/5, 2/5, respectively. The Huffman algorithm for producing a Huffman code takes as input of a document A whose alphabet C is $C = \{c_1, c_2, \ldots, c_n\}$ with frequency f_i (≥ 0) of character $c_i \in C$ for all i, $1 \leq i \leq n$ and $\sum_{i=1}^{n} f_i = 1$. The algorithm then outputs a binary codeword for each character such that no codeword is a prefix of another, and the encoding length of document A is minimal.

(i) What is the Huffman encoding of the alphabet? (10 points)

Answer: The frequencies of characters a, b, c, d, e are 3/20, 3/20, 2/20, 4/20, 8/20, respectively. The Huffman encoding of the alphabet are as follows.

- e:0
- d:111
- c:101
- b:110
- a:100

(ii) Encode string $baacdea$, using the Huffman coding. (3 points)

Answer: 110-100-110-101-111-0-100,
where "-" is used to separate the coding of characters.

(c) Prove the following two properties of the Huffman encoding scheme (4 points).

(i) If some character occurs with frequency more than 2/5, then there is guaranteed to be a codeword of length 1. (2 points)

Answer: We show this by contradiction. Assume that none of the characters has a codeword with length 1, then the length of the codeword of each character is at least 2. Let T be the Huffman tree, r the root of T, and nodes v_{ab} and v_{cd} the left and right children of node r. Further, let the left and right children of subtrees rooted at nodes v_{ab} and v_{cd} are a, b, and c, d, respectively. Notice that the subtrees rooted at a, b, c and d may and may not be leaf nodes. Let f_a, f_b, f_c and f_d be the frequencies of these four nodes. We assume that $f_a \leq f_b$ and $f_c \leq f_d$. Following the construction of the Huffman tree, we know

$$\max\{f_a, f_b\} \leq \min\{f_c, f_d\}, \tag{13}$$

and

$$f_a + f_b \geq \max\{f_c, f_d\}, \tag{14}$$

otherwise, there is a codeword with length 1 in T.

As we have $f_a \leq f_b$ and $f_c \leq f_d$, we have $f_a \leq f_b \leq f_c \leq f_d$ by inequalities (13) and (14).

It is known that $f_a + f_b + f_c + f_d = 1$ by the property of Huffman trees. Meanwhile, we have $f_a + f_b + f_c + f_d \geq f_a + f_b + 2f_c \geq 2f_a + 2f_c \geq 4f_a$, then $f_a \geq 1/4$. Since $f_a = \min\{f_a, f_b, f_c, f_d\}$, $f_a + f_b \geq 2f_a \geq 1/2$. On the other hand, we have $f_c + f_d = 1 - (f_a + f_b) \leq 1/2$. If $f_d \geq 2/5$, then $f_c \leq 1/2 - 2/5 = 1/10$. Thus, $f_c = 1/10 < 1/4 = f_a$, which contradicts the fact in inequality (13) that $f_c \geq f_a$. The claim thus is correct.

(ii) If all characters occur with frequency less than 1/3, then there is guaranteed to be no codeword of length 1. (2 points)

Answer: We show this by contradiction. We assume that there is a character c with length of 1, then it is one of the two children of the Huffman tree root. Let another non-leaf child of the tree root be v_{ab}, then the left and right children of v_{ab} are the subtrees rooted at nodes a and b respectively. Following the construction of the Huffman tree, then $\max\{f_a, f_b\} \leq f_c$, then we have $f_c + f_{ab} = f_c + (f_a + f_b) \leq f_c + 2\max\{f_a, f_b\} \leq 3f_c < 1$ as all characters occur with frequency less than 1/3. Meanwhile, we know the fact that $f_c + f_{ab} = 1$, this leads to a contradiction, thus, no character has a codeword with length 1.