Student Number: ____________________

The University of Melbourne
School of Computing and Information Systems

COMP20007
Design of Algorithms
Sample Exam 2017

Exam duration: Three hours

Reading time: Fifteen minutes

Number of sections: Three

Total Marks: 60

Length: This paper has 6 pages including this cover page.

Authorized materials: No materials are authorized. Calculators are *not* permitted.

Instructions to students:

- This is a sample exam paper. The cover sheet of the real paper will be available on the LMS.

Section A (15 marks)

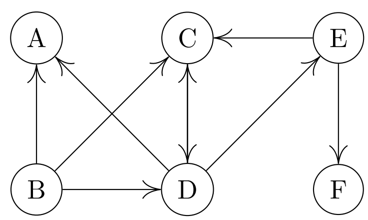
Question 1 ($5 \times 1 = 5$ marks)

Answer True or False for each of the following statements.

(a) Assuming random pivot-selection, the time complexity of quicksort is $\Theta(n \log n)$.

(b) If $f(n) = 3n^2$ and $g(n) = (n \log(n))^2$, then $f \in O(g)$.

(c) In *any* DFS of the following graph, the highest post-number will be assigned to the node labelled B.



(d) Any algorithm for a problem in the complexity class NP can be used to find a solution to any problem in the class NP-Complete.

(e) In an AVL tree, a median of all elements in the tree is always in the root node or in one of the root's two child nodes.

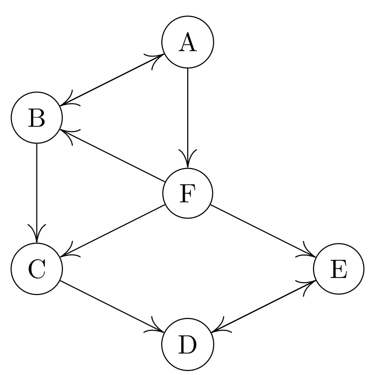
Question 2 ($2 + 3 = 5$ marks)

Question 2.1 (2 marks)

In the context of a directed graph, what is the meaning of the term 'strongly-connected component'?

Question 2.2 (3 marks)

Identify 3 strongly-connected components from the following graph:



Question 3 (2 + 3 = 5 marks)

Define the *Triangle Sum* of an array A of n elements to be the sum of its elements multiplied by decreasing positive integers, as follows:

$$n \cdot A[0] + (n-1) \cdot A[1] + \ldots + 2 \cdot A[n-2] + 1 \cdot A[n-1]$$

The following recursive C function finds the Triangle Sum of an array `A` of size `n`.

```
int triangle_sum(int A[], int n) {
    /* base case: empty array */
    if (n == 0) {
        return 0;
    }

    /* recursive case: */
    /* first, add all of the elements together once */
    int whole_array_sum = 0;
    int i;
    for (i = 0; i < n; i++) {
        whole_array_sum += A[i];
    }

    /* next, find the triangle sum of the first n-1 elements */
    int first_part_triangle_sum = triangle_sum(A, n-1);

    /* the triangle sum of this array will be the sum of these two numbers */
    return whole_array_sum + first_part_triangle_sum;
}
```

Question 3.1 (2 marks)

Write a recurrence relation, including a base case, describing the running time of this algorithm on an input array of n elements.

Question 3.2 (3 marks)

Solve this recurrence relation, to arrive at a tight big O bound on the running time on this algorithm on an input array of n elements.

Section B (30 marks)

Question 4 (3 + 3 = 6 marks)

Question 4.1 (3 marks)

Clearly draw the de Bruijn graph formed from the following reads assuming kmers of length 2.

`ADR, DEA, REA, DRE, REA, EAD, DDE, ADS, DRE, ADD, EAD, EAD`

Question 4.2 (3 marks)

Based on the de Bruijn graph from Question 4.1, reconstruct a valid string which would produce these reads.

Question 5 (4 + 4 = 8 marks)

Consider the limited alphabet and relative frequency distribution given below:

symbol	`A`	`S`	`G`	`_`	`E`
frequency	3	3	2	1	1

Question 5.1 (2 × 2 = 4 marks)

Based on these frequency counts, for each of the following algorithms, what is the length of the longest codeword generated by the algorithm?

(a) Huffman's algorithm

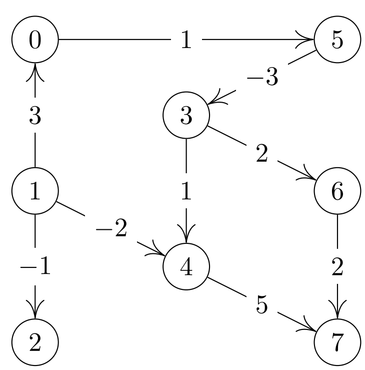
(b) The Shannon-Fano algorithm

Question 5.2 (4 marks)

Suppose messages of fixed length 4 are to be encoded using arithmetic coding according to this distribution, and that the sections of this interval are arranged in the order `A`, `S`, `G`, `_`, `E`. What decimal interval would represent the message `SAGE`?

Question 6 (3 + 3 = 6 marks)

Consider the following directed acyclic graph:



Question 6.1 (3 marks)

Give a topological ordering of this graph.

Question 6.1 (3 marks)

What is the distance array produced by running Dijkstra's algorithm on this graph, starting from node 1?

Question 7 (2 + 4 + 4 = 10 marks)

The keys I, S, O, G, R, A, M are to be stored in a tree data structure.

Question 7.1 (2 marks)

Insert these keys into an initially-empty binary search tree.

Question 7.2 (4 marks)

Insert the same keys into an initially empty AVL Tree. It is recommended to show the intermediate stages of your work rebalancing the tree.

Question 7.3 (4 marks)

Insert the same keys into an initially empty 2-3-4 Tree. It is recommended to show the intermediate stages of your work restructuring the tree.

Section C (15 marks)

Question 8 (3 + 1 + 3 = 7 marks)

Question 8.1 (3 marks)

Consider a hash table being used to store strings in C. The strings are to be hashed using a simple hash function 'sum hash', which hashes a string to the sum of the ASCII values of its characters. That is, for a string s of n characters (and a 'null character' `'\0'` at $s[n]$), the hash value $h(s)$ should be:

$$h(s) = s[0] + s[1] + \ldots + s[n-1]$$

Write a C function for an algorithm that computes the sum hash value for a string `s`. The prototype of your function should be as follows:

```
int sum_hash(char *s);
```

Question 8.2 (1 marks)

The hash function `sum_hash()` described in Question 8.1 does not do a great job of distributing common strings evenly throughout the hash table. Give an example of **three** strings that will all hash to the same value using this hash function.

Question 8.3 (3 marks)

Lets say these strings are to be stored in a hash table that uses linear probing with step size 1 to resolve collisions. The following pseudocode algorithm is suggested for removing a string `s` from a hash table `table`. It returns `true` if the string was found and removed, and `false` if it was not in the table to begin with.

Note: `table.slots` is an array of strings and `table.inuse` is a boolean array storing `true` if the hash table is currently storing a string, and `false` otherwise.

```
hash_table_remove(table, s):
    address = sum_hash(s) % table.size
    steps = 0
    while table.inuse[address] and (steps < table.size):
        if table.slots[address] == s:          // (string comparison)
            table.inuse[address] = false
            return true
        address = (address + 1) % table.size
        steps += 1
    return false
```

Unfortunately, **this algorithm is not correct**. Clearly describe the flaw in the algorithm. Include an example illustrating this flaw.

Question 9 (3 + 3 + 2 = 8 marks)

Two of the following questions ask you to 'design an algorithm'. For each question, you should give a **high-level description** of an algorithm, clearly describing your approach and any key details. You do **not** have to give detailed C code or detailed pseudocode. You do not need to analyse the complexity of your algorithms. In describing your algorithms, you may refer to any algorithms discussed in lectures or assignments.

Question 9.1 (3 marks)

Consider an array of n items containing many duplicates, such that there are only P distinct items in the entire array, with P much smaller than n. Design a hashing-based algorithm for sorting the items in this array. Your algorithm should run in $O(n + P \log P)$ time in the average case.

Question 9.2 (3 marks)

Consider another array of size n, this time containing only non-negative integers of a known maximum size Q, where Q is much smaller than n. Design an algorithm for sorting this array in worst-case $O(n + Q)$ time.

Question 9.3 (2 marks)

It seems you have just created two linear-time sorting algorithms. How do these algorithms *not* defy the $\Omega(n \log n)$ lower bound on the sorting problem discussed in lectures?

END OF EXAM