MONASH University

Office Use Only

Monash University

Semester One Examination Period 2014

Faculty of Information Technology

EXAM CODE:	**FIT2004**
TITLE OF PAPER:	**Algorithms and Data Structures**
EXAM DURATION:	3 hours writing time
READING TIME:	10 minutes

THIS PAPER IS FOR STUDENTS STUDYING AT:(tick where applicable)

☐ Berwick [✓ Clayton] [✓ Malaysia] ☐ Off Campus Learning ☐ Open Learning
☐ Caulfield ☐ Gippsland ☐ Peninsula ☐ Enhancement Studies ☐ Sth Africa
☐ Pharmacy ☐ Other (specify)

AUTHORISED MATERIALS

CALCULATORS ☐ YES [✓ NO]
OPEN BOOK ☐ YES [✓ NO]
SPECIFICALLY PERMITTED ITEMS ☐ YES [✓ NO]

Candidates must complete this section if required to write answers within this paper

STUDENT ID __ __ __ __ __ __ __ __ DESK NUMBER __ __ __ __

Section		Marks
1		16
2		8
3		10
4		7

Section		Marks
5		12
6		8
7		9
Total		70

Office use only

INSTRUCTIONS

- You must answer ALL the questions.
- Answers to each question should be in the space DIRECTLY BELOW the questions and (if required) on the blank page overleaf of each question.
- Script book may be used if ADDITIONAL SPACE is required for answering these questions

General exam technique

Some candidates throw marks away by not attempting all questions. Suppose you get 7/10 on a question for a 20 minutes effort. Spending another half hour on the question gets *at most* 3 more marks. On the otherhand, if you spend that time on a new question, you might get another 10 marks, or more.

Answer the question that is asked. If the question asks for Insertion sort, do not give Quicksort. Where necessary, especially where it says "No explanation, no marks", justify your answer with a clear explanation.

Some of the questions ask you to write Pseudocode. A Pseudocode is essentially a high-level description of your program, that should allow a human to understand it. If you feel more comfortable using Python/Java syntax, you are welcome to use it but don't get bogged down in syntax. What will essentially be assessed in such questions is your basic understanding of the algorithm (and not the syntactical correctness).

Write neatly. Use a new pen not an old, splodgy one.

Good Luck!

Section 1 (Short Questions) **(Section weight = 16 marks)**

This section is composed of 7 questions: (A) to (F)

(A) Consider the following binary tree of words (**depicted in an anticlockwise-rotated sense with the root node on the left and leaves on the right**):

```
           |kind------|
           |
           /
the------+
           \
           |              |roamer---+
           |              |          \
           |              |           |of-----+
           |              |                    \
           |              |                     |robot-----|
           |              /
           |yutu-----+
                      \
                      |is------+             |moon------|
                                \            /
                                 |a-------+
```

Write the string of words formed by recursively traversing this tree over each of the following traversals:

(i) Preorder/prefix traversal.
(ii) Postorder/postfix traversal.
(iii) Inorder/infix traversal.
(iv) Breadth-First traversal.

(2 marks)

(i) Prefix: the yutu is a moon roamer of robot kind.
(ii) Postfix: moon a is robot of roamer yutu kind the
(iii) Infix: a moon is yutu robot of roamer the kind
(iv) BF: the yutu kind is roamer a of moon robot

2

(B) The amount of time, T_N taken by an algorithm is typically a function of the size, N, of the input data. Assume that you wrote a program that shows the following time recurrence:

$$T_N = \begin{cases} \phi T_{N-1} + a, & \text{if } N > 0 \\ b & \text{if } N = 0, \end{cases}$$

where ϕ, a and b are constants.

(i) Work out the solution of T_N from this recurrence.

(ii) What is the time complexity of your program in Big-O notation?

(3 marks)

(i) $T_N = \phi T_{N-1} + a$

$= \phi(\phi T_{N-2} + a) + a = \phi^2 T_{N-2} + (\phi + 1)a$

$= \phi^2(\phi T_{N-3} + a) + (\phi + 1)a$

$= \phi^3 T_{N-3} + (\phi^2 + \phi + 1)a$

$\vdots$

$= \phi^N T_0 + (\phi^{N-1} + \phi^{N-2} + \dots + \phi + 1)a$

$= \phi^N b + (\phi^{N-1} + \phi^{N-2} + \dots + \phi + 1)a$ //

(ii) Exponential $\approx O(\phi^N)$

3

(C) Assume you are given an array of N integers, `a[1...N]`.

(i) Write a pseudocode to sort these numbers using insertion sort. (Don't write any function and arguments to the function in your pseudocode. Treat the array as given and just focus on writing down as pseudocode, the core essence of insertion sort of this array.)

(ii) Identify the loop invariant of your algorithm.

(iii) In Big-O notation, what is the best-case time-complexity and worst-case space-complexity of insertion sort?

(3 marks)

(i)
```
for (i=2; i<=N; i++)
{
  //INVARIANT 1: a[1... i-1] is sorted
   ai = a[i];
    j = i-1;
    while (j<=1 & a[j]>ai)
    {
     //INVARIANT 2: a[j+2, .... i] >ai
      a[j+1] = a[j];
      j--;
    }
   a[j+1] = ai;
}
```

(ii) a[1... i-1] is sorted.

(iii) Time Complexity:
O(n) - best, O(n²) - worst

Space Complexity:

O(N)

3

(D) An array `arr[1...N]` contains N characters from a binary alphabet. Write an algorithm (pseudocode) that sorts these characters in $O(N)$-time and O(N) Space in total.

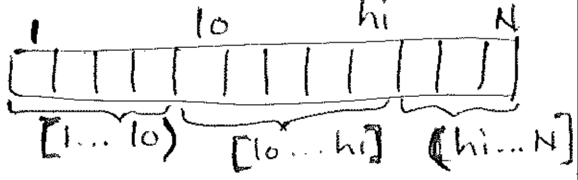
(3 marks)

Concept / Invariant :

arr [1... lo-1] $\longrightarrow$ 0's

arr [lo... hi] $\longrightarrow$ unknown

arr [hi+1... N] $\longrightarrow$ 1's



pseudocode :

```
lo = 1 ; hi = N
while (lo <= hi)
{
   // Invariant : See above↑
   if(arr[lo] == 0)
         lo++;
   else
     {
      swap arr[lo] and arr[hi] ;
      hi-- ;
     }
}
```

3

(E) Provide the dynamic programming recurrence relationships (including boundary conditions) for the problem of finding the edit distance between two strings `s1[1...M]` and `s2[1...N]`. **(2 marks)**

$$D(0,0) = 0$$

$$D(i,0) = i \quad , \quad 1 \leq i \leq M$$

$$D(0,j) = j \quad , \quad 1 \leq j \leq N$$

$$D(i,j) = \min \begin{cases} D(i-1,j-1) + \begin{cases} 0 & \text{if } S_1[i] == S_2[j] \\ 1 & \text{otherwise} \end{cases} \\ D(i-1,j) + 1 \\ D(i,j-1) + 1 \end{cases}$$

2

(F) Assume that you have already implemented correctly a Heap data structure with `upHeap(argument)` and `downHeap(argument)` operations, as discussed in the unit. You are now given an array of distinct random numbers `arr[1...N]`.

Write the <u>**most efficient**</u> code snippet you can to **Heapify** (or convert into a heap) this given array (while treating the above operations as being already a part of your code). Note, your pseudocode should not be more than a loop and its body. Justify your answer.

(3 marks)

```
for (i = N/2 ; i >= 1 , i -- )
        downHeap (i, N)   or   downHeap (i)
```

<u>Justification:</u>

$i = N/2$ because leaves are already a heap and the first node to start downHeap is at $N/2$.

height from bottom:

$h=1$ nodes require at worst $O(1)$ operations to heapify.

No. of such nodes $\approx N/4$ (where N is the no. of leaves)

$\frac{N}{4} ~~\varepsilon~~ O(1) + \frac{N}{8} O(2) + \frac{N}{16} O(3) + \ldots$

$\frac{N}{4} \left[\frac{1}{2^0} + \frac{2}{2^1} + \frac{3}{2^2} + \ldots\ldots \right]$

$\frac{N}{4}$ (converges to a constant)

3

Section 2 (Abstract Data Types and formal proofs) (Section weight = 8 marks)

This section is composed of 2 questions: (A) and (B)

(A) Given below is the list abstract data type:

```
type List e = nil | cons(e, (List e))
```

An operation append and length on the list type is defined as follows:

```
append(nil, L2) = L2                                  ——— (1)
| append(L1,L2) = append( cons(h1,T1), L2 )
                = cons( h1, append(T1,L2) )           ——— (2)

length(nil) = 0                                       ——— (3)
| length(L) = length( cons(h,T) ) = 1 + length(T)     ——— (4)
```

Formally prove:

```
length(append(L1,L2)) = length(L1) + length(L2)
```

(4 marks)

<u>Base Case:</u>

length(append(nil, nil))
= length(nil) . . . . . . using (1)
= 0
= length(nil) + length(nil) //

<u>General Case:</u>

length(append(L1, L2)) = length(cons(h1, append(T1, L2))) . . . . using (2)
= 1 + length(append(T1, L2)) . . . . . . using (4)
= $\underbrace{1 + \text{length}(T1)}$ + length(L2) . . . . Assuming base case is true (Induction step)
= length(cons(h1, T1)) + length(L2)
= length(L1) + length(L2).

4

(Page intentionally left blank for working space)

(B) Given below is the abstract data type definition for a binary tree:

```
type tree e = nilTree | fork( e, (tree e), (tree e) )
```

Further, below are definitions of two algorithms on this data structure:

X which is an algorithm applicable to trees of any kind, is defined as:

```
X(nilTree)   = nilTree
|  X( fork(e,L,R) ) = fork(e, X(R), X(L))
```

Y which is an algorithm applicable only to trees of real numbers:

```
Y(nilTree) = 0
|  Y( fork(e,L,R) ) = e +  Y(L) + Y(R)
```

Prove **formally** that:

(i) X(X(T)) = T, for any tree T.

(ii) Y(X(T)) = Y(T), for every tree T of numbers.

(4 marks)

(i) $X(X(T)) = T$

<u>Base case</u>: $X(nil) = nil$ (True!)

<u>General Case</u>:

$X(X(T) = X(fork(e, X(R), X(L)))$

$= fork(e, X(X(L)), X(X(R)))$

$= fork(e, L, R)$ ... Inductive Step.

$= T$ //

(ii) $Y(X(T)) = Y(T)$

<u>Base case</u>: $Y(X(nil)) = Y(Nil) = 0$ (True!)

<u>General Case</u>:

$Y(X(T)) = Y(fork(e, R, L))$

$= e + Y(R) + Y(L)$

$= e + Y(L) + Y(R)$

$= Y(T)$ //

4

(Page intentionally left blank for working space)

Section 3 (Correctness, Verification and Termination) (Section weight = 10 marks)

This section is composed of 3 questions, (A) to (C)

Background: The pseudocode (program) below gives the core functionality of binary search of key in an array of N integers a[1...N]

```
variable lo=1, hi=N+1;
while (lo < hi-1) {
   variable mid = math_floor((lo+hi)/2);

   if (key >= a[mid])
      lo=mid;
   else
      hi=mid;
}
if (N > 0 and a[lo] == key) print key_found at lo;
else print key_not_found;
```

The next three questions in this section are based on this given program.

(A) (i) What is the precondition that has to be met for this program to work?

(ii) What is the loop invariant of this program?

(2 marks)

(i) Precondition: a[1...N] is sorted.

(ii) key is in a[1...N] iff key is in a[lo...hi-1]

2

(B) Explain precisely why the above program is correct. **(5 marks)**

Invariant: Key in a[1...N] iff key is between a[lo ... hi).
MID divides interval [lo...hi) by half.

True Case: if (key >= a[mid])
Invariant holds because key >= a[mid] and lo becomes mid.

False Case:
Invariant holds because key < a[mid] and hi becomes mid

At the end of the loop,
the invariant still holds.

At Termination:
lo = hi - 1
Invariant still holds.
i.e. key is between a[lo...hi-1]
$\Rightarrow$ key == lo (if it exists)

5

(C) Explain precisely why the above program always terminates. **(3 marks)**

$$lo < hi - 1$$
$$\Rightarrow lo + 1 < hi$$
$$\Rightarrow lo + 2 \leq hi$$

The loop condition $lo < hi - 1$ holds until there are <u>AT LEAST</u> 2 elements.

This implies $lo < mid < hi$. (always hold)

Therefore,

$hi - lo$ is always positive

$hi - lo$ always decreases.

This cannot go on as hi & lo are integers.

∴ It terminates.

3

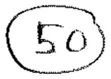
Section 4 (Search and Retrieval Trees) (Section weight = 7 marks)

This section is composed of 2 questions: (A) and (B)

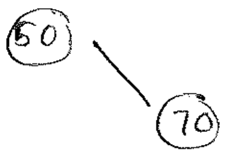
(A) Starting with an initially empty AVL-tree, show what happens as the following elements are inserted in the given order: 50, 70, 30, 10, 20, 15

(4 marks)

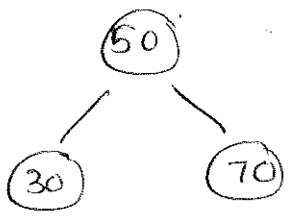
(i)



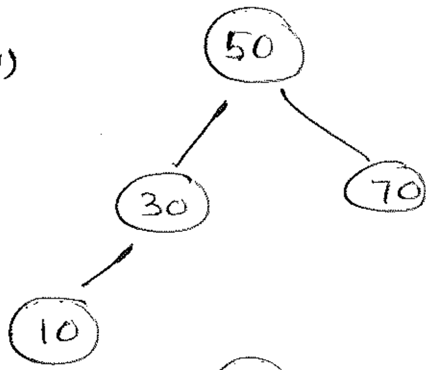
(ii)



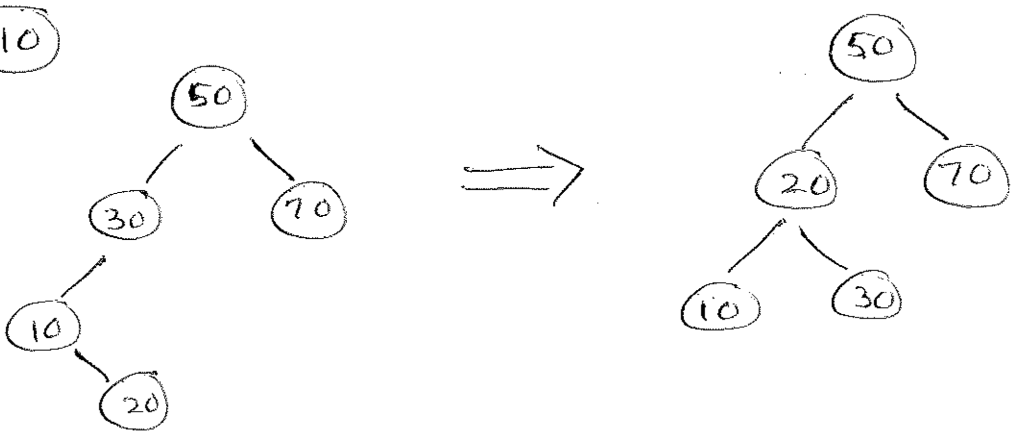
(iii)



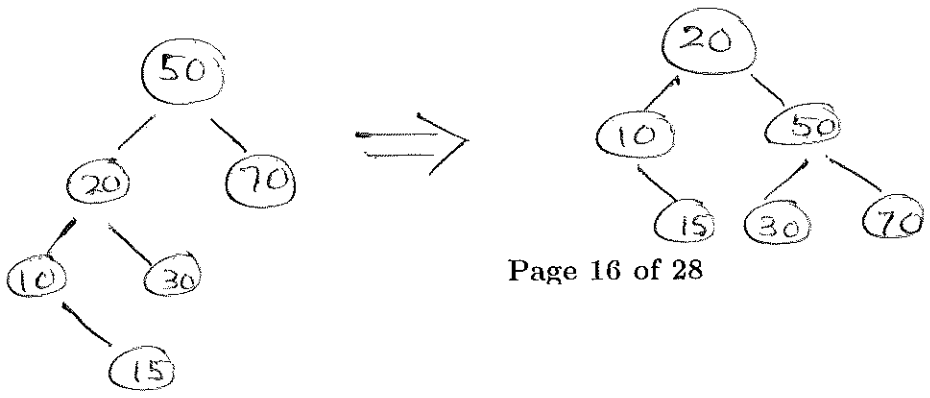
(iv)



(v)



(vi)



4

(Page intentionally left blank for working space)

(B) Given a string of size N, explain in detail why the space-complexity to store a suffix tree (that is, a compact suffix trie) is bounded by $O(N)$. **(3 marks)**

- No. of leaves in a compact suffix trie is N.
- Each internal node has at least 2 children because all single child paths in a suffix trie are compressed into one edge.
- This implies that the 'arity' of each internal node is ≥ 2.
- In the worst case, all internal nodes are <u>binary</u>.
- A full binary tree with N leaves has (N-1) internal nodes.
- Total nodes = O(N)
- Each node is associated with a substring.
- Prefix of a suffix is a substring.
- Requires only 2 numbers to be stored.
  (1) Suffix id
  (2) length of substring.
- ∴ Space required = O(N)

3

Section 5 (Graphs) (Section weight = 12 marks)

This section is composed of 3 questions, (A) to (C)

(A) (i) What is the time-complexity of Dijkstra's algorithm when the set of 'Remaining' vertices (as described in this unit) is implemented using a min-heap. Explain your reasoning clearly.

(ii) Explain why Dijkstra's algorithm does not work for graphs with negative weights?

$G(V, E)$ (4 marks)

```
While (Remaining ! empty)
{
  x = extract-min (Remaining)
  . . . . .
  for (each y in Remaining adjacent to x)
  {
    est = estimate-dist (x → y)
    if (est < dist(y))
        update dist(y);
  }
}
```

Generic Time Complexity of Dijkstra's algorithm:

$O(|V| \times T_{extract\text{-}min} + |E| \times T_{update\text{-}dist})$

When Remaining is a min heap/priority Queue

$T_{extract\text{-}min} = O(\log |V|)$

$T_{update\text{-}dist} = O(\log |V|)$

$\therefore$ Time complexity of Dijkstra's using min-Heap for Remaining is

$O[(|V| + |E|) * \log |V|]$

4

(B) Draw the Directed Graph that corresponds to the following information:

```
Gough is older than Mel, Paul and Kylie.
Claudia is older than Gough and Kylie.
Kylie is older than Natalie.
Natalie is older than Paul.
Mel is older than Natalie and Paul.
```

In addition, suggest if the graph you have drawn is acyclic or not. **(3 marks)**

$$\begin{tikzcd}
 & \text{Mel} \arrow[dr] & \\
\text{Natalie} \arrow[ur] \arrow[dr] & \text{Paul} \arrow[l] \arrow[u] \arrow[r] & \text{Gough} \arrow[d] \\
 & \text{Kylie} \arrow[ur] \arrow[r] & \text{Claudia}
\end{tikzcd}$$

The graph is Directed Acyclic.

3

(C) Write a pseudocode to find and print a topological sort, given a Directed Acyclic Graph (DAG) represented by an adjacency matrix. (Note: Do not write any pseudocode for reading the DAG from an input file; consider the adjacency matrix as given and just write the pseudocode for the algorithm that finds the topological sort and prints it.)

(5 marks)

```
Main (G)
 { For v=1 to N in G do
       if v is not yet visited
            do DFS(v)
 }

DFS (v)
 {
   Mark v as visited
   For each unmarked vertex w
        which is a neighbour of v
            do DFS(w)
   return
 }
```

* This is the Depth-First Search Algorithm you've learnt in week 10 lectures.

5

Section 6 (Pattern Matching) **(Section weight = 8 marks)**

This section is composed of 1 (two-part) question

(A) The Burrows-Wheeler Transform (BWT) of a reference string is `AGTTTTCC$GGC`.

(i) Without reconstructing the first column, use LF-mapping to reconstruct only the **last 3 letters** of the original string, excluding the terminal character '$'.

(ii) If a given pattern is `ATC`, using **backwards search** described in the unit, how many times does each suffix of this pattern occur in the original text?

(The progression of your worked out steps of LF-mapping (in the first part) and backwards search (in the second part) should be clearly written down for your answer to be marked.)

(8 marks)

(i)

A	G	T	T	T	T	C	C	$	G	G	C
0	1	2	3	4	5	6	7	8	9	10	11

Rank

$	A	C	G	T
0	1	2	5	8

pos = Rank($) + nOcc[$, L[0 .... i)]

Start with A$, i = 0

i = pos = Rank(A) + nOcc[A, L[0 ... 0)]

= 1 + 0 = 1

⇒ GA$

i = pos = Rank(G) + nOcc[G, L[0 .... 1)]

= 5 + 0 = 5

⇒ TGA$

8

(Page intentionally left blank for working space)

(ii) sp = rank (pat[i]) + nOcc (pat[i], L[0...sp))
ep = rank (pat[i]) + nOcc (pat[i], L[0...ep]) - 1

pat = ATC

i = 2
pat[i] = C
sp = 2+0 ; ep = 2+2 = 4

i = 1
pat[i] = T
sp = 8+0 ; ep = 8+2 = 10

i = 0
pat[i] = A
sp = 1+1 = 2 , ep = 1+1-1 = 1

∴ No. of occurances of ATC = 0 //

Section 7 (Miscellaneous) **(Section weight = 9 marks)**

This section is composed of 3 questions: (A) to (C)

(A) Write the pseudocode for a linear **recursive** program to print the Nth Fibonacci number, where the Fibonacci sequence is $1, 1, 2, 3, 5, 8, 13, \cdots$.

(4 marks)

```
fib (x, y, n) {
    if (n ≤ 1)
        return y;
    else
        return fib (y, x+y, n-1);
}

Driver call:
    fib (0, 1, n)
```

4

(B) Given is a function $f(x) = x^2 - 2$. With the $x_0 = 1$ as the initial guess what are the values of x_1 and x_2 using Newton's numerical algorithm for finding the root of this function. (Write down clearly the steps involved in arriving at the values x_1 and x_2.)

(2 marks)

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

$$f(x) = x^2 - 2$$

$$f'(x) = 2x$$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 1 - \frac{(1-2)}{2} = \frac{3}{2} \;//$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = \frac{3}{2} - \frac{(\frac{9}{4} - 2)}{3}$$

$$= \frac{3}{2} - \frac{1}{12}$$

$$= \frac{17}{12} \;//$$

2

(C) The *longest common subsequence* (LCS) problem requires one to find the *longest* possible *subsequence* shared between two different strings $s1$ and $s2$. Note that a **subsequence** is *different* from a **substring**, in that the letters in a subsequence need not appear consecutively in the original string, whereas in a substring they do appear consecutively.

For example, if a string s = `abcdefghijklm`, a **subsequence** is some (or all) of its elements, in the same order, for instance, `cfijm` is the subsequence of s. If $s1$ = `arun` and $s2$ = `arvind`, the LCS between these two string is `arn` – there is no other subsequence common between $s1$ and $s2$ that is longer than the common subsequence `arn`.

The LCS problem between any two given strings $s1$ and $s2$ can be solved using a dynamic programming approach, which is similar to the approach used to solving the edit distance problem.

Write the dynamic programming recurrence relationship for this problem, including the boundary conditions.

(3 marks)

Boundary Condition:

$LCS(i,j) = 0$ if $i=0$ or $j=0$

General recurrence

$$LCS(i,j) = \begin{cases} LCS(i-1, j-1) & \text{if } x_i = y_j \\ \max\{LCS(i, j-1), LCS(i-1, j)\} & \text{otherwise.} \end{cases}$$

3