Family Name:	
Other Names:	
Signature:	
Student Number:	

This PAPER is NOT to be retained by the STUDENT

The University Of New South Wales

C0MP1921/2011/2091 Final Exam

Data Structures and Algorithms/Data Organisation/Computing 2

November 2006

Time allowed: **3 hrs**
Total number of questions: **17**
Total number of marks: **105**

You must hand in this entire exam paper and ALL your answer booklets. Otherwise you will get zero marks for the exam and a possible charge of academic misconduct.

Ensure that you fill in all of the details on the front of this pink paper, and the 5 answer booklets, and then SIGN everything. Mark one booklet PART B, one PART C, one PART D, one PART E, and one WORKING ONLY. The working only booklet will not be marked.

Do not use red pen or pencil in this exam. Unless advised otherwise you may not use any language features or library functions not covered in class. Answers which do not comply with the course style guide or which introduce potential security vulnerabilities will be penalised.

There are two marks for following the examination instructions.

No examination materials permitted.
Calculators may **not** be used.
Questions are **not** worth equal marks.
Answer **all** questions.

Examiner's Use Only:

Inst	Part A	Part B	Part C	Part D	Part E	**Total**

Part A: Short Answer Questions

Answer these questions in the spaces provided on **this pink question paper**. DO NOT answer these questions in an **answer booklet**! Do all your working in the *Working Only* booklet, your working is not marked in Part A.

Write your answers clearly. Keep your answers neat and very brief. Messy or long answers will not be marked. For multiple-choice questions, circle one answer only.

Question 1

(2 mark)
Given the C implementations of sorting algorithms discussed in lectures, which of the following is a *stable* sorting algorithm?

[A] Insertion Sort
[B] Quicksort
[C] Quicksort with Median-of-Three partitioning
[D] Shell Sort
[E] None of the above

Question 2

(2 mark)
The following alternatives list expressions for the worst-case time complexity $T(n)$ of various algorithms. Which alternative has an *asymptotic worst-case time complexity* of $O(n^2)$?

[A] $1000.log(n)$
[B] $0.01n^3 + 2n^2 + 1000000$
[C] $5n + 2n(n + 1) + 100$
[D] $n! + 5n^2$
[E] $2^n + 5n^2 + 3n + 1$

Question 3

(2 marks)
Which of the following methods for traversing a binary tree will output the values of an ordered tree in sorted order?

[A] Breadth first traversal
[B] Infix traversal
[C] Postfix traversal
[D] Prefix traversal
[E] None of the above

Question 4

(2 marks)
Which of the following methods for traversing a binary tree will output the values of a **heap** in sorted order?

[A] Breadth first traversal
[B] Infix traversal
[C] Postfix traversal
[D] Prefix traversal
[E] None of the above

Question 5

(4 marks)
When you implement an ADT which functions should be declared as static?

Briefly justify your answer.

Question 6

(4 marks)
In the space below write a function which reads an integer n using scanf, and then prints all the numbers from 1 to n in order, except not printing any number divisible by 7 or 3. You may assume that n is not negative.
For example if n was 15 the output would be 1 2 4 5 8 10 11 13
The prototype for the function is:

```
void printSome(void);
```

Your function:

Question 7

(10 marks)
In this question definitions are as for the Project.

The following code written in three files was designed to create a graph to represent Europe. To save space only the start of each file is shown and header comments have been removed:

PreGame.h:

```
#include "GameData.h"
#define MAX_LINKS      10
#define NOT_ADJACENT    4
#define ADJACENT        1

typedef int Map [NUM_LOCATIONS] [NUM_LOCATIONS];
void buildMaps (Map seaRoad);
...
```

testPreGame.c:

```
#include <assert.h>
#include "preGame.h"

int main (int argc, char* argv[]) {
   Map roadNsea, railLinks;

   buildMaps (roadNsea, railLinks);

   assert (roadNsea  [1] [66] == ADJACENT);
   assert (roadNsea  [2] [1] == ADJACENT);
...
```

PreGame.c:

```
#include <stdlib.h>
#include <assert.h>
#include "GameData.h"
#include "Game.h"
#include "preGame.h"

const int map [NUM_LOCATIONS] [MAX_LINKS];

void buildMaps (Map seaRoad, Map railLinks) {

   seaRoad = malloc (sizeof (Map));
   assert (seaRoad != NULL);

   // build the sea+road map
   Location from, to;
   int i;



   // clear the map
   for (from=0; from<NUM_LOCATIONS; from++) {
      for (to=0; to<NUM_LOCATIONS; to++) {
         seaRoad [from] [to] = NOT_ADJACENT;
      }
   }

   // record the links
```

```
   for (from=0; from<NUM_LOCATIONS; from++) {
      // rough check that the format is right, starts with location id?
      assert (map[from] [0] == from);

      // skip over the 0 padding at end of the line
      i=MAX_LINKS-1;
      while ((map[from][i]==0) && (i>0)) {
         i--;
      }

      // record each link
      for (; i>1; i--) {
         to = map [from] [i];
         seaRoad [from] [to] = ADJACENT;
      }
   }

   // quick sanity check
   assert (seaRoad  [1] [66] == ADJACENT);
}

// C fills the rest of the each row with 0s when initialising.  We use
// this to detect the end of each row,  so don't put a real location 0
// at the end.
const int map [NUM_LOCATIONS] [MAX_LINKS] = {
   {0,6,49,53,59},
   {1,2,26,34,48,66},
   {2,1,9,16,61},
   {3,55,68},
   {4,48,54,66},
   {5,40,45,69},
   {6,0,10,28,49,50,53},
  ....
```

(i) When executed it produces the following unexpected crash:

```
$ ./testPreGame testPreGame.c:18: failed assertion ‘roadNsea [1] [66] == ADJACENT’
```

Correct the bug (write the correction(s) on the program listing above). State and briefly explain the bug or bugs which caused this crash.
The bug is:

(ii) After fixing the first bug a second bug becomes apparent. Running the program produces *another* crash:

```
$ ./testPreGame testPreGame.c:18: failed assertion ‘roadNsea [2] [1] == ADJACENT’
```

Correct the bug (write the correction(s) on the program listing above). State and briefly explain the bug or bugs which caused this crash.
The bug is:

Part B: Sorting and Analysis of Algorithms

Answer this part in your Part B answer booklet. Start each question on a new page.

Make your answers as clear and easy to understand as possible. Provide type definitions and brief comments where necessary. Confusing or illegible solutions will lose marks.

If you do not wish your answer for a question to be marked, clearly record 1 mark for that question on the front of your Part D answer booklet. If you do this your answer for that question will **not** be marked.

Question 8

(8 marks)

[A] Consider this C function for bubble-sort that takes an array of integers `sequence` and an integer `numElements` giving the number of elements in the array `sequence`. (The numbers on the left-hand side are line numbers.)

```
1   void bubbleSort(int sequence[], int numElements) {
2
3       int i, j, temp;
4
5       for(i = numElements - 1; i >= 0; i--) {
6
7           for(j = 1; j <= i; j++) {
8
9               if (sequence[j - 1] < sequence[j]) { // swap elements
10
11                  temp = sequence[j - 1];
12
13                  sequence[j - 1] = sequence[j];
14
15                  sequence[j] = temp;
16
17              }
18
19          }
20
21      }
22
23  }
```

The purpose of this C code fragment is to sort the elements of the array `sequence` into *increasing* order. This code fragment is valid C and executes without error. However, the code fragment above does NOT perform as expected and has a mistake on one line only. Determine the mistake, state on which line it occurs and give the correct C code that should appear at that line.

[B] Consider this C function for insertion-sort that takes an array of integers `sequence` and an integer `numElements` giving the number of elements in the array `sequence`. (The numbers on the left-hand side are line numbers.)

```
1  void insertionSort(int sequence[], int numElements) {
2
3      int i, j;
4
5      for (i = 1; i < numElements; i++) {
6
7          for (j = i; j > 0; j--) {
8
9              if (sequence[j] < sequence[j - 1]) {
10
11                 temp = sequence[j - 1]; // Swap elements
12
13                 sequence[j - 1] = sequence[j];
14
15                 sequence[j] = temp;
16
17             }
18
19         }
20
21     }
22
23 }
%
```

The purpose of this C code fragment is to sort the elements of the array `sequence` into *increasing* order. This code is valid C and executes without error.

i) One improvement to this code will make it run in near linear time when the data required to be sorted is already close to sorted order. Using the line numbers, indicate which lines of code need to be added, deleted or modified in order to accomplish this. Provide the required C code.

ii) Briefly explain how your code improves on the original algorithm.

iii) Provide one more improvement to the C code that will make it run faster in many cases. Using the line numbers, indicate which lines of code need to be added, deleted or modified in order to accomplish this. Provide the required C code.

iv) Briefly explain how your code improves on the original algorithm.

Question 9

(6 marks)

Consider the following C code fragment, presented in lectures, which correctly implements Quicksort with Median-of-Three partitioning.

```
void quickSort(int sequence[], int low, int high) {
    int pivot;

    if (high > low) {
        swap(&sequence[(low + high)/2], &sequence[high - 1]);

        // Ensure median value in sequence[high] to act as pivot
        if (sequence[low] > sequence[high - 1]) {
            swap(&sequence[low], &sequence[high - 1]);
        }

        if (sequence[low] > sequence[high]) {
            swap(&sequence[low], &sequence[high]);
        }

        if (sequence[high - 1] < sequence[high]) {
            swap(&sequence[high - 1], &sequence[high]);
        }

        pivot = partition(sequence, low, high);
        quickSort(sequence, low, pivot - 1);
        quickSort(sequence, pivot + 1, high);
    }
}

int partition(int sequence[], int low, int high) {
    int lowerLimit = low - 1;
    int upperLimit = high;
    int key = sequence[high];

    while (lowerLimit < upperLimit) {
        while (sequence[++lowerLimit] < key)
            ;

        while (key < sequence[--upperLimit] && upperLimit != low)
            ;

        if (upperLimit > lowerLimit) {
            swap(&sequence[lowerLimit], &sequence[upperLimit]);
        }
    }

    swap(&sequence[lowerLimit], &sequence[high]);

    return lowerLimit;
}
```

Trace the execution of this algorithm on sorting the array `sequence` initially consisting of the following elements: 5 8 2 2 5 7 8 3 6 7. In your answer draw a tree-like diagram that shows the sequence of recursive calls to the `quickSort` function. Use notation as illustrated in the following example representing the initial call to the function `quicksort`:

quickSort([5, 8, 2, 2, 5, 7, 8, 3, 6, 7], 0, 9)

where [5, 8, 2, 2, 5, 7, 8, 3, 6, 7] represents the contents of array `sequence` between indices 0 and 9 (it is only necessary to show the elements of the array `sequence` between the indices represented by the values of `low` and `high`).

Question 10

(6 marks)

Consider the following C code fragment for Selection Sort (the numbers on the left-hand side are line numbers)..

```
   void selectionSort(int sequence[], int numElements) {
       int i, j, min;

1      for (i = 0; i < numElements; i++) {
2          min = i; // current minimum is first unsorted element

           // find index of minimum element
3          for (j = i + 1; j < numElements; j++) {
4              if (sequence[j] < sequence[min]) {
5                  min = j;
               }
           }

           // swap minimum element into place
6          swap(&sequence[i], &sequence[min]);
       }
   }
```

[A] With each line of the Selection Sort algorithm above that is prefixed with a line number, associate a cost and a formula expressing the number of times that line of C code will be executed when sorting n items (i.e., when the function is called using `selectionSort(sequence, n)`).

[B] Using the cost and the formulas determined in Part A write a formula expressing the time complexity $T(n)$ of this algorithm.

[C] Use big-O notation to express the asymptotic worst case time complexity of this algorithm.

Question 11

(4 marks)

You are given C code for an algorithm capable of sorting data consisting of strings of characters. The code ignores the case of alphabetic characters when comparing strings (i.e., `'a'` and `'A'` are treated as equal). On testing the algorithm's performance on sorting 10 000 randomly generated strings, 100 000 randomly generated strings, 10 000 ordered strings, 100 000 ordered strings, 10 000 reverse ordered strings and 100 000 reverse ordered strings (in all cases strings are a fixed length of 4 characters) you obtain the following timings (in seconds).

10 000 rand	**100 000 rand**	**10 000 ord**	**100 000 ord**	**10 000 rev**	**100 000 rev**
0.01	0.18	1.64	151.40	1.63	147.86

Furthermore, given as input the three strings "aa zz ZZ", the algorithm outputs "aa ZZ zz".

[A] In point form briefly state the conclusions that you can draw from the test results above.

[B] Name a sorting algorithm which is consistent with these conclusions.

Part C: Abstract Data Types

Answer this part in your Part C answer booklet. Start each question on a new page.

Make your answers as clear and easy to understand as possible. Provide type definitions and brief comments where necessary. Confusing or illegible solutions will lose marks.

If you do not wish your answer for a question to be marked, clearly record 1 mark for that question on the front of your Part C answer booklet. If you do this your answer for that question will **not** be marked.

.

For this Part suppose you are a software developer who has been asked to write an ADT to represent a priority queue of up to 1000 integers. Larger integers will have higher priority than smaller integers. You are to implement the queue using a heap implemented as an array of integers. You will write the ADT in the files PriorityQueue.c and PriorityQueue.h.

Question 12

(8 marks)

(i) Write the file **PriorityQueue.h**.
It should include the following prototype:

```
void deleteMax(PriorityQueue pq);
```

(and it will need to include other things too!) Don't forget to include brief comments.

(ii) Give a complete definition and implementation of the type `PriorityQueue` in C, stating in which file, or files, the code goes and how you have made it abstract.

Question 13

(10 marks)
(i) Draw a simple picture explaining how the `deleteMax` function (declared above) works on the heap.
(ii) Write a C implementation of the `deleteMax` function and state in which file, or files, the code goes.

Part D: Graphs and Trees

Answer this part in your Part D answer booklet. Start each question on a new page.

Make your answers as clear and easy to understand as possible. Provide type definitions and brief comments where necessary. Confusing or illegible solutions will lose marks.

If you do not wish your answer for a question to be marked, clearly record 1 mark for that question on the front of your Part D answer booklet. If you do this your answer for that question will **not** be marked.

Question 14

(10 marks)

(i) What are the two most common ways of representing a graph in C?
(ii) State two advantages of each representation.
(iii) Show how a complete graph of 4 nodes (with nodes named 0..3) might be stored in memory for each of the above ways of representing graphs.
(For your answer you should draw a picture, showing the contents of memory locations. You may write in decimal, no need to use binary or hex.)

Question 15

(5 marks)

Show the effect of inserting the following items into an initially empty treap:

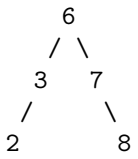
(5, 0.75), (10, 0.4), (8, 0.5), (4, 0.6), (13, 0.35), (3, 0.1), (9, 0.7), (1, 0.5), (2, 0.9), (11, 0.65), (7, 0.2), (14, 0.05), (6, 0.55), (12, 0.3), (15, 0.8).

In each case the key is represented by the first value and the priority by the second. Suppose we want larger priorities at the top. Draw the treap produced after all the items have been inserted.

Question 16

(8 marks)

Consider the following AVL tree.



[A] Show the effect of inserting the following items into this AVL tree one at a time: 1, 5, 9, 4. Draw the AVL tree produced after each insertion.

[B] How could you delete the node with key value 6 from the final tree produced in Part A? Draw the AVL tree at each stage of the deletion process.

Part E: Challenge Question

Answer this part in your Part E answer booklet.

Partial solutions for this question (i.e. attempts worth less than 50%) will score no marks in this part. If you do not wish your answer for the question to be marked, record 1 mark for the question on the front of the answer booklet. If you do this your answers for the question will **not** be marked.

Your solutions must be clear, elegant and easy to understand. In this part confusing or difficult to understand solutions will score no marks.

Question 17

(12 marks)

In this question everything is defined as for the project except that there are no rail moves, no teleporting, the mucking about at sea rule does not apply, and `isLegalHunterMove` has the prototype:

```
int isLegalHunterMove(Game game, Location to, Location from);
```

(It is invalid to call `isLegalHunterMove` in round 0.)

Write a function `int distance(Game game, Location from, Location to)` to go in `hunter.c`, which computes the shortest route between *to* and *from* using Dijkstra's algorithm.
Make your function clear and brief. (Note: you **must** use Dijkstra's algorithm to score marks for this question.)