Duration: **150 minutes**
Aids Allowed: **Type C: 8.5x11 inch, handwritten, two-side notes**

Student Number: ___________

Last Name: ___________

First Name: ___________

Examiner: Danny Heap

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

Marking Guide

# 1:	______/ 6
# 2:	______/ 6
# 3:	______/ 12
# 4:	______/ 6
# 5:	______/ 9
# 6:	______/ 8
# 7:	______/ 10
# 8:	______/ 12
# 9:	______/ 15
# 10:	______/ 5
# 11:	______/ 12
# 12:	______/ 12
# 13:	______/ 3
# 14:	______/ 10
TOTAL:	______/126

This examination consists of 14 questions on 12 pages (including this one), printed on one side of the paper. *When you receive the signal to start, please make sure that your copy of the test is complete.* Answer each question directly on the test paper, in the space provided.

Good Luck!

Question 1. [6 MARKS]

Consider the following program:

```
typedef struct{
  int i;
} int_wrap_t;

int inc1(int i){
  i += 1;
  return i;
}

int_wrap_t inc2(int_wrap_t w1){
  w1.i += 1;
  return w1;
}

int_wrap_t inc3(int_wrap_t *w2){
  w2->i += 1;
  return *w2;
}

int main(void){
  int i = 0;
  int_wrap_t iw;

  iw.i = 0;
  inc1(i);
  /* comment A */
  i = inc2(iw).i;
  /* comment B */
  i = inc1(i);
  i += inc3(&iw).i;
  /* comment C */

  return 0;
}
```

What are the values of `i` and `iw.i` when the program reaches each of the following lines?

```
/* comment A */

/* comment B */

/* comment C */
```

Question 2. [6 MARKS]

For each of the following `while` loops, write a single conditional expression between the provided parentheses (...), and a single assignment statement between the curly braces { ... }, to yield the required output.

Part (a) [2 MARKS]

Required output: `14 17 20 23 26`

```
r = 14;
while(                ){


    printf("%d ",r);


}
```

Part (b) [2 MARKS]

Required output: `17 20 23 26`

```
r = 14;
while(                ){


    printf("%d ",r);


}
```

Part (c) [2 MARKS]

Required output: `80 26 8 2`

```
r = 80;
while(                ){


    printf("%d ",r);


}
```

Question 3. [12 MARKS]

Our class notes state that Radix Sort does $O(nk + rk)$ work, where n is the length of the list to be sorted, k is the number of digits in base r of the largest key to be sorted.

Part (a) [5 MARKS]

Use your knowledge of Radix Sort to explain why the amount of work is $O(nk + rk)$.

Part (b) [5 MARKS]

What additional assumption(s) must be made to show that Radix Sort has complexity $(n \log n)$?

Part (c) [2 MARKS]

Under what conditions does Radix Sort have unbounded complexity, that is complexity $> f(n)$ for any function f?

Question 4. [6 MARKS]

`Quick Sort` and `Merge Sort` each have average complexity $O(n \log n)$, while `Exchange Sort` has average complexity $O(n^2)$. Describe an advantage `Quick Sort` has over `Merge Sort`, an advantage `Merge Sort` has over `Quick Sort`, and a situation where `Exchange Sort` is just as efficient as the other two. Explain your answers.

Question 5. [9 MARKS]

Suppose `s1`, `s2` are character stacks that support the following operations:

- `push(s,c)` adds a node with character `c` to the top of stack `s`
- `pop(s)` removes the the node at the top of stack `s` and returns its character value.
- `empty(s)` returns 1 if `s` is empty, 0 otherwise.

Part (a) [5 MARKS]

Describe (in pseudo-code or words) how to use `s1` and `s2` (and no other variables or data structures) to implement a First-In First-Out (FIFO) queue. In other words, show how to implement the following operations in a "virtual" queue `q` composed of some combination of `s1` and `s2`:

- `enqueue(q,c)` adds a node with character `c` to the tail of queue `q`
- `dequeue(q)` removes the the node at the head of queue `q` and returns its character value.
- `empty(q)` returns 1 if `q` is empty, 0 otherwise.

Part (b) [4 MARKS]
If we assume that `push(s,c)` and `pop(s,c)` are $O(1)$ operations, what are the complexities of the `enqueue(q,c)` and `dequeue(q)` operations you implemented in Part (a)? Explain your answers.

`enqueue(q,c)`:

`dequeue(c)`:

Question 6. [8 MARKS]

Suppose we have a group of n people. Let's say there is a **friendship** between persons a and b if a is b's friend and b is a's friend. In addition, let's say there is a **link** between persons a_1 and a_k if there is a sequence of people $a_1, a_2, \ldots, a_k$ where, for each $1 < i \leq k$ there is a **friendship** between a_i and a_{i-1}.

Part (a) [3 MARKS]
What is the **minimum** number of friendships necessary (in terms of n) in order to have a link between every pair of persons in our group of n? Explain your answer.

Part (b) [5 MARKS]
If $n = 10$, what is the **maximum** number of friendships possible if our group is divided into two groups A and B, each group has 5 members, and there are no links from any person in A to any person in B. Explain your answer.

Question 7. [10 MARKS]

The 8×8 matrix, G, below specifies a weighted, undirected graph. $G[i][j]$ specifies the weight of the edge between i and j. If there is no edge between i and j, $G[i][j]$ is set to ∞. Define the shortest path from node 1 to itself as having length 0. Draw the graph corresponding to G, then find the lengths of the shortest paths from node 1 to each of the 7 remaining nodes. Call these shortest distances $D(1,2)$, $D(1,3)$, ..., $D(1,8)$.

$G =$

	1	2	3	4	5	6	7	8
1	∞	1	2	∞	∞	∞	∞	∞
2	1	∞	3	3	∞	∞	∞	∞
3	2	3	∞	4	4	∞	∞	∞
4	∞	3	4	∞	8	9	∞	∞
5	∞	∞	4	8	∞	16	8	∞
6	∞	∞	∞	9	16	∞	24	27
7	∞	∞	∞	∞	8	24	∞	64
8	∞	∞	∞	∞	∞	27	64	∞

$D(1,2)$

$D(1,3)$

$D(1,4)$

$D(1,5)$

$D(1,6)$

$D(1,7)$

$D(1,8)$

Question 8. [12 MARKS]

Part (a) [4 MARKS]

Describe the practice of **top-down programming**, and explain its benefits.

Part (b) [4 MARKS]

Describe the practice of **bottom-up testing**, and explain its benefits.

Part (c) [4 MARKS]

Describe the practice of **modular programming**, and explain its benefits.

Question 9. [15 MARKS]

In our notes there is the following pseudo-code description of an algorithm for implementing `bubble_up` in a `heap`. The purpose is to move newly-inserted `newKey` from `h[last]` to its correct place in the `heap`, assuming that `h[1..last-1]` is already a `heap`.

```
bubble_up(array h, int last, newKey){
    int c = last
    int p = c/2
    while( p >= 1 and h[p] < newKey){
        h[c] = h[p]
        c = p
        p = p/2
    }
    h[c] = newKey
}
```

Part (a) [10 MARKS]

Fill in pseudo-code for an implementation of `bubble_down`, to move `newKey` from `h[1]` to its correct place in the `heap`, assuming `h[2..last]` has heap-order, and `h[1..last]` has heap-shape.

```
bubble_down(array h, int last, int newKey){



}
```

Part (b) [5 MARKS]
What are the worst-case complexities of: insertion of a new key, finding the maximum key, finding the minimum key, sorting, and listing keys in non-descending order of the heap example covered in class? Explain your answer(s).

Question 10. [5 MARKS]

Suppose $T[0..34]$ is a hash table of size $m = 35$ meant to hold unique keys. You are provided with a ready-made hash function h, and you decide to resolve collisions by probing for an open slot using the formula:

$$\begin{aligned} i_o &= h(k) \\ i_{j+1} &= (i_j + C) \bmod m \end{aligned}$$

Which would be your best choice of step size: $C = 10$, $C = 6$, or $C = 14$? Explain your answer. How would your answer change if your table were expanded to $T[0..36]$, of size $m = 37$?

Question 11. [12 MARKS]

Part (a) [7 MARKS]
A binary tree has upper-case letters for keys, and the following `PreOrder` and `PostOrder` traversals:

PreOrder: `A, C, G, F, B, E, D`

PostOrder: `G, F, C, E, D, B, A`

Give the `InOrder` traversal of the tree. **Hint:** Two nodes' relative position changes between `PreOrder` and `PostOrder` **iff** one node is a descendent of the other.

Part (b) [5 MARKS]

Suppose T is a Binary Search Tree with integer keys. Given two integers, $i \leq j$, describe in pseudo-code (or in words) a modification of the InOrder tree traversal that will print an ordered list of all the keys in T that are between i and j, inclusive.

Question 12. [12 MARKS]

Consider the two recursive functions, mys1 and mys2:

```
int mys1(int n, int i){
  if (n == 1){
    return i;
  }
  else{
    return mys1(n/2, 2*i) + mys1(n/2, 2*i);
  }
}

int mys2(int n, int i){
  if (n == 1){
    return i;
  }
  else{
    return 2*mys2(n/2, 2*i);
  }
}
```

Part (a) [8 MARKS]

Give a short mathematical expression for what each function does when n is a positive integer and i is any integer.

Part (b) [4 MARKS]

What are the complexities of mys1 and of mys2? Explain your answers.

Question 13. [3 MARKS]

Our recursive program solving the Tower of Hanoi problem — how to move a tapered stack of n rings from one peg to another, using a third peg as an intermediate – had $O(2^n)$ complexity. Is it possible to write an iterative program that solves this problem but has lower complexity? Explain why, or why not.

Question 14. [10 MARKS]

Consider the 5 ADTs: `hash table`, `heap`, `AVL tree`, `B+ tree`, and `stack`. State the task(s) and condition(s) for which:

Part (a) [2 MARKS]

A `hash table` is preferable to a `heap`.

Part (b) [2 MARKS]

A `heap` is preferable to an `AVL tree`.

Part (c) [2 MARKS]

An `AVL tree` is preferable to a `B+ tree`.

Part (d) [2 MARKS]

A `B+ tree` is preferable to a `stack`.

Part (e) [2 MARKS]

A `stack` is preferable to a `hash table`.