This is a closed book test. Calculators are neither needed nor permitted.

Please write your answers in ink. Pencil answers will be marked, but will not be reconsidered after the test papers have been returned.

The test will be marked out of 50. You have 50 minutes.

<table>
<thead>
<tr>
<th>Question 1</th>
<th>/10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 2</td>
<td>/10</td>
</tr>
<tr>
<td>Question 3</td>
<td>/15</td>
</tr>
<tr>
<td>Question 4</td>
<td>/15</td>
</tr>
</tbody>
</table>

TOTAL

"You miss 100% of the shots you don't take"

Happy Birthday to Wayne Gretzky
General marking philosophy:

A student who gives enough of an answer to show they understood what they were supposed to do, even if they couldn’t do it (or made lots of errors while doing it) should get at least 50% on that question.

Full marks should be given if a solution is sound and not missing anything important.

Feel free to give marks like 9.5/10 to a solution that is correct but contains a minor error.

A student should only get 0/10 on a question if they made no attempt to answer it at all.

Please mark in a contrasting colour of ink.

Please put your initials clearly on the front page of each test you mark.
Question 1 (10 marks):

(a) [5 marks] Suppose \( f(n) \) is in \( \Omega(n^2) \) and \( g(n) \) is in \( \Omega(n^3) \)

What is the \( \Omega \) class of \( f(n) + g(n) \)? Explain your answer.

**Solution:**

\[ f(n) \geq c \cdot n^2 \text{ for all large } n \]

\[ g(n) \geq d \cdot n^3 \text{ for all large } n, \text{ where } c \text{ and } d \text{ are positive constants} \]

Thus \( f(n) + g(n) \geq d \cdot n^3 + c \cdot n^2 \geq d \cdot n^3 \text{ for all large } n \)

Therefore \( f(n) + g(n) \) is in \( \Omega(n^3) \)

(b) [5 marks] Suppose \( f(n) \) is in \( \Theta(n^2) \) and \( g(n) \) is in \( \Theta(n^3) \)

What is the \( \Theta \) class of \( f(n) + g(n) \)? Explain your answer.

**Solution:**

\[ f(n) \leq c \cdot n^2 \text{ for all large } n, \text{ and } g(n) \leq d \cdot n^3 \text{ for all large } (\text{from definition of } \Theta \text{ class}) \]

Thus \( f(n) + g(n) \leq c \cdot n^2 + d \cdot n^3 \leq c \cdot n^3 + d \cdot n^3 = (c+d) \cdot n^3 \)

Thus \( f(n) + g(n) \) is in \( O(n^3) \)

By Part (a), \( f(n) + g(n) \) is in \( \Omega(n^3) \)

Therefore \( f(n) + g(n) \) is in \( \Theta(n^3) \)
Marking guide for Question 1

In each part, if they show that they understand the meaning of Omega and/or Theta they should get at least 2.5/5 for that part
Question 2 (10 marks):

Here is an algorithm that prints the base-2 representation of a base-10 non-negative integer.

convert(n):
    # n is a non-negative integer
    if n <= 1:
        print n
        return
    else:
        r = n % 2  # n % 2 is n mod 2
        convert(n/2)  # integer division rounds down
        print r

Rewrite the algorithm in a non-recursive form that uses a stack.

You may assume that a Stack class has been defined with object methods push(x), pop() and isEmpty().

Here is a start for your revised algorithm. Feel free to replace this if you wish.

convert(n):
    # n is a non-negative integer
    if n <= 1:
        print n
    else:
        myStack = new Stack()

Solution:

convert(n):
    # n is a non-negative integer
    if n <= 1:
        print n
    else:
        myStack = new Stack()
        while n > 0:
            myStack.push(n % 2)
            n = n/2
        while not myStack.isEmpty():
            print myStack.pop()
Marking Guide for Question 2

Students are not required to use my version of pseudo-code – syntax is much less important than semantics here. My solution is quite concise – students may introduce variables without penalty.

If they have the basic concept of using a stack to hold the information that would be “pending” during the recursive calls (ie. the current 0 or 1) they should get at least 6/10.

If they understand how the stack is used but they get mixed up on the arithmetic part (possibly by not understanding what % does) they should get most of the marks.
For Question 3 and Question 4:

Assume that we have a **Vertex** class and a **Tree** class.

**Vertex** objects have the following attributes:
- **value**, which we will assume is an integer
- **left**, which is either **None** or a link to another **Vertex**
- **right**, which is either **None** or a link to another **Vertex**

**Tree** objects have just one attribute:
- **root**, which is either **None** or a link to a **Vertex**
**Question 3 (15 marks):**

Create an algorithm which will take a binary search tree T and a value x and return the largest value in T which is \( \leq x \). If there is no such value, your algorithm should return “None”. For example, if your tree contains the values \{3,4,7,19,20\} and \( x = 18 \), the algorithm should return 7. If \( x = 2 \), the algorithm should return “None”.

Your solution should take advantage of the lexical ordering in the tree, and should be as efficient as you can make it. Your algorithm may be recursive or non-recursive. You should add comments to explain what your algorithm is doing.

Here is a header for your algorithm. You can substitute your own if you wish:

```python
find(T, x):
    # T is a tree, x is a value

Solution:

find(T, x):
    return rec_find(T.root, x)

rec_find(v, x):
    if v == None:
        return None
    else:
        if v.value == x:
            return x
        elif v.value > x:
            return rec_find(v.left, x)
        else:
            sub = rec_find(v.right, x)
            if sub == None:
                return v.value
            else:
                return sub
```
Marking guide for Question 3

There are certainly other solutions, or different ways of expressing this solution.

Students who search the entire tree (ie. they don't use the lexical ordering) and get the correct answer should get about 10/15

As noted in the statement of the question, their solutions are not required to be recursive.

Any small error (such as finding the largest value < x instead of the largest value <= x) should cost one mark.

As outlined in the general marking instructions, if the student shows that they understand the question and show an understanding of what needed to be done to answer it, they should be given at least 50%, regardless of the instruction to subtract one mark for each small error.

Some students may try to compare None to an integer (as in “if x > y” where x is an integer and y is None). It’s not really plausible (nor should it be necessary!) but if it forms an essential part of their solution, we can accept it without penalty.

If a student’s solution is nowhere near correct but they show they do know how to search a tree, they should get at least 6/15
Question 4 (15 marks):

What is the Big-O (worst case) complexity of the total time required to build a binary search tree containing \( n \) values using the recursive insert algorithm given in class? The algorithm is repeated here for your convenience. Explain your answer.

```python
insert(T, x):
    T.root = rec_insert(T.root, x)

rec_insert(current, x):
    if current == None:
        return new Vertex(x)
    else if current.value > x:
        current.right = rec_insert(current.right, x)
    else:
        current.left = rec_insert(current.left, x)
    return current
```

Hint: Suppose the values to be inserted are in ascending order. What does the binary tree look like? How long does it take to insert each value?

Solution: If the values being added to the tree are in ascending order, each one will be added as the right child of the previous one. When searching for the insertion point for each value, we will have to work down from the root to the lowest vertex, then add the new vertex below that.

To insert the first value, we add a new vertex. (1 operation)
To insert the second value, we examine one vertex and add a new vertex. (2 operations)
To insert the third value, we examine two vertices and add a new vertex. (3 operations)
...
To insert the \( n \)th value, we examine \( n-1 \) vertices and add a new vertex. (\( n \) operations)

Thus the total number of operations is \( 1 + 2 + 3 + ... + n \), which we know is about \( (n^2)/2 \)

Thus the algorithm is in \( O(n^2) \)
Marking Guide for Question 4

If the student recognizes that the number of operations can be \( 1 + 2 + \ldots + n \) (or \( 1+2+\ldots+n-1 \), if they just count the number of existing vertices examined during each insertion) they should get at least 10/15.

If they stop there, 10/15 is appropriate.

If they get to that sum but incorrectly reduce that sum (for example, to \( O(n \log n) \) or \( O(n) \)), they should get 11/15.

As always, if they can't answer the question but wrote down enough to show that they understood it, they should get at least 50%.