Part 1: Multiple Choice

Enter your answers on the Scantron sheet.
We will not mark answers that have been entered on this sheet.
Each multiple choice question is worth 3.5 marks.

1. Two algorithms, A and B, have time complexities $f_A(n)$ and $f_B(n)$, respectively. Assume that $f_A(n) = O(f_B(n))$ and $f_B(n) \neq O(f_A(n))$; then, which of the following statements is true?

   (A) A is faster than B for all instances of size $n \geq 0$.
   (B) B is faster than A for all instances of size $n \geq 0$.
   $\surd$(C) There is a value $n_0 \geq 1$ such that A is faster than B for every instance of size $n \geq n_0$.
   (D) There is a value $n_0 \geq 1$ such that B is faster than A for every instance of size $n \geq n_0$.
   (E) Depending on the programming language used to implement A and B, it could be that A is faster than B for large size inputs, or it could be that B is faster than A for large size inputs.

2. Let $f(n)$, $g(n)$, and $h(n)$ be three functions with positive values for every $n \geq 0$. Assume that $f(n) < g(n)$, and $h(n) < g(n)$ for all $n \geq 0$. Which of the following statements must be false for **every** set of functions $f(n)$, $g(n)$, $h(n)$ as above?
   (A) $f(n)$ is $O(h(n))$
   $\surd$(B) $f(n)$ is not $O(g(n))$
   (C) $(f(n) + h(n))$ is $O(g(n))$
   (D) $g(n)$ is not $O(f(n))$
   (E) $(f(n) \times h(n))$ is $O(g(n))$

3. Let T be a proper binary tree with root r. Consider the following algorithm.

```
Algorithm traverse(r)
Input: Root r of a proper binary tree.
    if r is a leaf then return 0
    else {
          t ←traverse(left child of r)
          s ←traverse(right child of r)
          if s ≥ t then return 1 + s
         else return 1 + t
    }
```

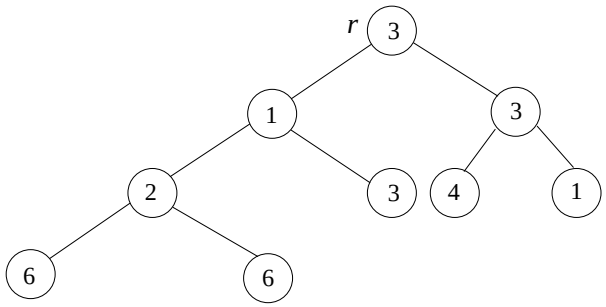
   What does the algorithm do?
   $\surd$(A) It computes the height of the tree.
   (B) It computes the number of internal nodes in the largest subtree of T.
   (C) It always returns the value 1.
   (D) It computes the number of nodes in the largest subtree of T.
   (E) It computes the number of internal nodes in the tree.

4. The following algorithm performs a postorder traversal of a proper binary tree and modifies some of the keys stored in the nodes. In the initial call r is the root of the tree.

```
Algorithm traverse2(r)
Input: Root r of a proper binary tree.
  if r is an internal node then {
     traverse2(left child of r)
     traverse2(right child of r)
     if (key stored in left child of r) = (key stored in right child of r) then
        increase the key stored in r by 1
     else decrease the key stored in r by 1
  }
```

Assume that algorithm `traverse2` is performed over the following tree. After the execution of the algorithm how many nodes will store the key value 2?

(A) 1
√(B) 2
(C) 3
(D) 4
(E) 5



5. How many different proper binary search trees can be built with three internal nodes and keys 5, 2, and 6? (The leaves do not store keys.)
(A) 2
(B) 3
(C) 4
√(D) 5
(E) 6

6. What is the solution of the following recurrence equation?

$f(1) = 3$
$f(n) = f(n-1) + 1$

√(A) $f(n) = n + 2$
(B) $f(n) = 3n$
(C) $f(n) = 2n + 1$
(D) $f(n) = 2^n + 1$
(E) $f(n) = 3$

7. Consider the following algorithm.

```
Algorithm foo(n)
Input: Integer value n
    i ← 0
    j ← 0
    while j < n do {
        if i < j then i ← i + 1
        else {
            j ← j + 1
            i ← 0
    }
```

What is the time complexity of the algorithm?
(A) $O(1)$
(B) $O(n)$
(C) $O(n \log n)$
$\surd$(D) $O(n^2)$
(E) $O(nj)$

8. A *Decrementable Dictionary* is an ADT that can store a set of n data items with integer keys, and it provides (among others) the following four operations: `find(key)`, `smallest()`, `largest()` and `decreaseKey(key)`. Duplicated keys are not allowed in the dictionary. Operations `find`, `smallest`, and `largest` are as in the Ordered Dictionary ADT that we studied in class.

Operation `decreaseKey(key)` finds the data item with the given `key`, if it exists, and it decreases its key value by one if doing so does not create duplicated keys; otherwise, no keys are modified. Which of the following statements regarding the most efficient way to implement the above operations of a Decrementable Dictionary is true?

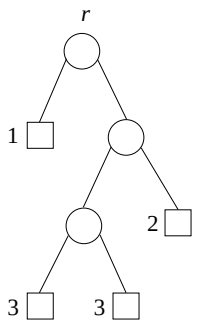
(A) A hash table with a good hash function and separate chaining is the best choice as all operations would have $O(1)$ average running time.
(B) A binary search tree is the best data structure, as all operations can be performed in $O(\log n)$ time in the worst case, and these trees are much simpler than AVL trees as they do not need to be kept balanced.
$\surd$(C) A sorted array is the best data structure, as operations `find`, and `decreaseKey` can be performed in $O(\log n)$ time in the worst case. Operations `smallest` and `largest` only take $O(1)$ time in the worst case.
(D) A hash table using double hashing is the best choice as all operations can be performed in $O(1)$ time in the worst case.
(E) A linked list is the best data structure as `decreaseKey` can be performed in $O(1)$ time in the worst case, while `find`, `smallest`, and `largest` only need $O(\log n)$ time in the worst case.

Part 2: Written Answers

Write your answers directly on these sheets.

9. [15 marks] The *external length* of a tree T is the sum of distances from the root of T to all the leaves of T. For example, the external length of the tree shown below is $1+2+3+3=9$. In the figure the number beside each node is the distance from the node to the root.

   Write an algorithm that receives as input the root r of a **proper binary tree** and an integer value d and it returns the external length of the tree. In the initial call $d=0$ and in subsequent calls the value of d should be equal to the distance from the current node to the root. Each node stores **only** a reference to its left child and one to its right child.



   Algorithm extLen(r, d)
   Input: Root r of a proper binary tree and integer value d
   Output: External length of the tree

   if r is a leaf **then return** d
   else return extLen(left child of r,$d+1$)+ extLen(right child of r,$d+1$)

10. [3 marks] Compute the time complexity of your algorithm as a function of the number n of nodes in the tree. Give the order of the time complexity.

    [4 marks] Explain how you computed the time complexity of the algorithm.

    Ignoring recursive calls, the algorithm performs a constant number c of operations when the node r is a leaf and a constant number of operations c' when node r is internal. The recursive calls make the algorithm traverse the entire tree, so the algorithm performs one recursive call per node of the tree. Therefore, the total number of operations performed by the algorithm is

    $c\times$ number of leaves + $c'\times$ number of internal nodes = $c\frac{n+1}{2} + c'\frac{n-1}{2}$, which is $O(n)$.

11. [15 marks] Given two arrays A, B each storing n different positive integer values, write in pseudocode an algorithm `intersection`(A, B, C, n) that stores in a third array C of size n all those values that are common to A and B. The algorithm must output the number of elements that were stored in C. For example, for the following arrays:

A	8	3	5	1	2	12	11	26	17	7

B	11	9	7	15	8	4	18	1	15	5

The algorithm must output the value 5 and C must store the values 8, 5, 1, 11, and 7. If your `intersection` algorithm invokes other algorithms, you must write those algorithms also.

Algorithm `intersection`(A, B, C, n)
Input: Arrays A. B, and C of size n
Output: Store common values from A and B in C and return the number of common values.

$i_C \leftarrow 0$
for $i_A \leftarrow 0$ **to** n **do** {
  $i_B \leftarrow 0$
  while $(i_B \leq n-1)$ **and** $(A[i_A] \neq B[i_B])$ **do** $i_B \leftarrow i_B + 1$
  if $i_B \leq n-1$ **then** {
    $C[i_C] \leftarrow A[i_A]$
    $i_C \leftarrow i_C + 1$
  }
} **return** i_C

12. [3 marks] Compute the worst case time complexity of your algorithm and give the order of the time complexity.

[3 marks] Explain what the worst case is and how you computed the time complexity.

The worst case for the algorithm is when the two arrays have no common values, as then the **while** loop is repeated the maximum number of times. Observe that the **for** loop always performs n iterations.

In every iteration the **while** loop performs a constant number c of operations. In the worst case the **while** loop performs n iterations as at the beginning $i_B = 0$, the value of i_B increases by one in each iteration, and the loop ends when $i_B = n$. Therefore, the total number of operations performed by the **while** loop in the worst case is cn.

Inside the **for** loop, but outside the **while** loop an additional constant number c' of operations are performed, so each iteration of the **for'** loop performs $c' + cn$ operations. The **for** loop iterates n times, so the total number of operations that it performs is $n(c' + cn) = c'n + cn^2$. Outside the **for** loop a constant number c'' of operations are performed, so the total number of operations performed by the algorithm is $c'' + c'n + cn^2$, which is $O(n^2)$.

13. Consider a hash table of size 7 with hash function $h(k) = k \bmod 7$. Draw the contents of the table after inserting, in the given order, the following values into the table: 35, 7, 42, 31, and 21,

[4 marks] (a) when linear probing is used to resolve collisions

[8 marks] (b) when double hashing with secondary hash function $h'(k) = 5 - (k \bmod 5)$ is used to resolve collisions.

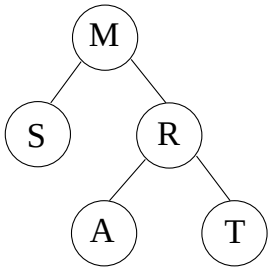
Linear probing

0	35
1	7
2	42
3	31
4	21
5	
6	

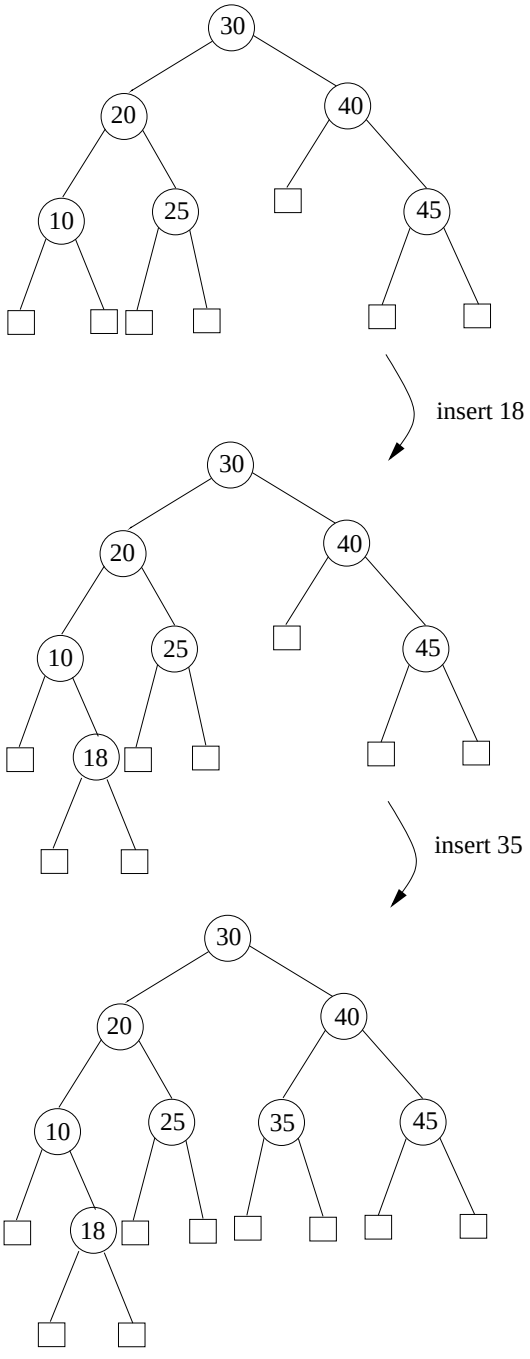
Double hashing

0	35
1	21
2	
3	7
4	31
5	
6	42

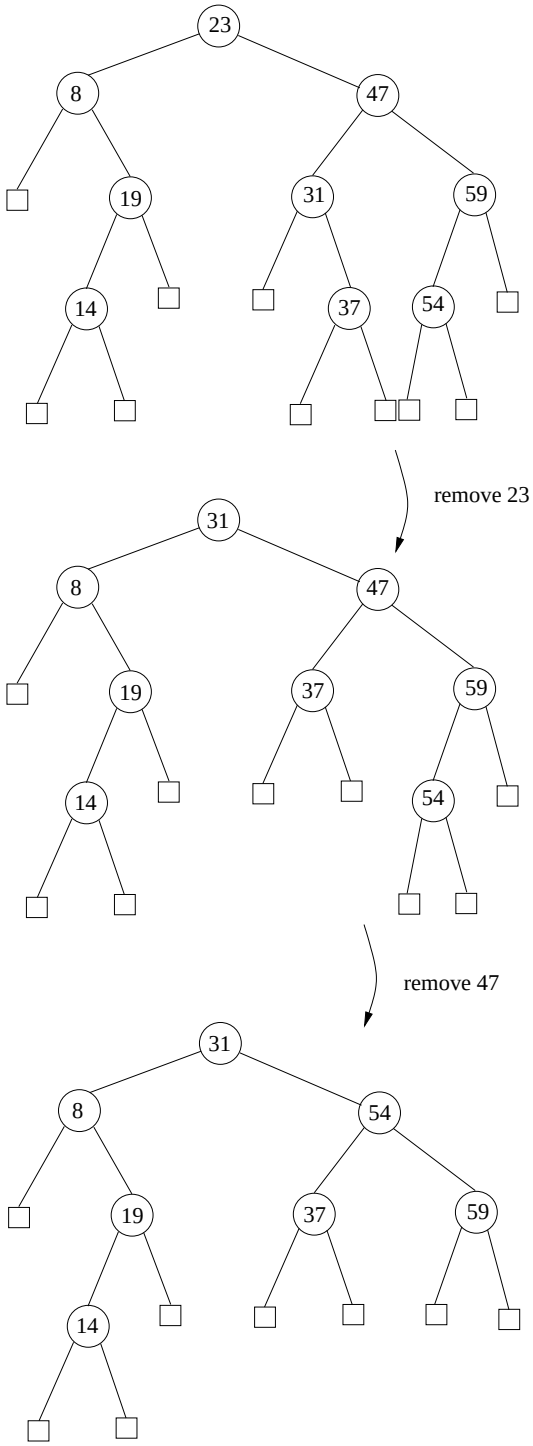
14. [8 marks] Draw a binary tree containing the keys A, M, R, S, and T such that an inorder traversal of the tree visits the nodes in this order: $S\ M\ A\ R\ T$ and a preorder traversal visits the nodes in this order: $M\ S\quad R\ A\ T$.



15. [4 marks] Consider the following binary search tree. Insert the key 18 and then insert the key 35 into the resulting binary search tree (so in the final tree both keys, 18 and 35, have been inserted). Draw the final tree and **all** intermediate trees that you need. You **must** use the algorithms described in class for inserting information in a binary search tree.



16. [5 marks] Consider the following binary search tree. Remove the key 23 from the tree and draw the resulting tree. Then remove the key 47 from this new tree and show the final tree (so in the final tree both keys, 23 and 47, have been removed). You **must** use the algorithms described in class for removing data from a binary search tree.



The University of Western Ontario
Department of Computer Science

PART I	
PART II	
9	
10	
11	
12	
13	
14	
15	
16	
Total	

CS2210A Midterm Examination
November 5, 2013
9 pages, 16 questions
110 minutes

Name: _____

Student Number: _____

Instructions

- Write your name and student number on the space provided.
- Please check that your exam is complete. It should have 9 pages and 16 questions.
- The examination has a total of 100 marks.
- When you are done, call one of the TA's and he will pick your exam up.