

## COMP225 MID SEMESTER TEST (3 QUESTIONS, 20 MARKS)

1. (5 marks) Consider the definition of a node for binary trees.

```
struct treeNode {
    int item;
    treeNode* LChildPtr;
    treeNode* RChildPtr;
};
```

```
typedef treeNode* ptrType;
```

- (a) (3 marks) Write a program to compute the height of a binary tree  $t$ . You may assume that the height of a NULL tree is 0.

```
int height (treeNode t);
// POST: returns the height of the tree t
```

- (b) (2 marks) What is the worst case complexity of your program for a tree with  $n$  nodes? Sketch a diagram of a tree which has this worst-case behaviour.

2. (5 marks)

- (a) (3 marks) Write a program to compute the sum of all items in a binary tree, having values at least  $a$  and strictly less than  $b$ . You may return 0 for NULL trees.

```
int sumBetween (treeNode t, int a, int b);
//PRE: a <= b
//POST: returns the sum of the items in t with values v, such that a <= v < b
```

- (b) (2 marks) What is the worst-case complexity of your program, for trees with  $n$  nodes. Does the complexity depend on the shape of the tree. State your reasons.

TURN OVER

3. (10 marks) This question is about dynamic programming. Consider the following problem for finding the length of the *longest strictly increasing subsequence*. Given an array  $A$  of length  $n$  having integer values, determine the maximal length of a subsequence (not necessarily contiguous) such that the values in the subsequence form a strictly increasing sequence. (Recall that a subsequence of an array is obtained by “crossing out” elements.) For example if the array is  $[2, 4, 2, 8, 1, 2]$  then the longest strictly increasing subsequence has length 3, and it is  $[2, 4, 8]$ .

The following program partially solves this problem. Given an array  $A$  of length  $n$  and an index  $j$ , it computes the longest strictly increasing subsequence in  $A[0..j]$  which includes the item  $A[j]$ .

```
int subseq (int A[], int n, int j) {
    int m;
    if (j==0) return 1;
    else {
        m= 1;
        for (int i= 0; i < j; i++)
        {
            if (A[i] < A[j]) { int x= subseq(A, n, i);
                if (x + 1 > m) {m= x + 1;}
            }
        }
    }
    return m;
}
```

For example `subseq(A, 6, 2)` returns 1 because the longest increasing subsequence in  $A[0..2]$  which must include  $A[2]$  is  $[2]$ . However `subseq(A, 6, 3)` returns 3, because the longest increasing subsequence in  $A[0..3]$  which must include  $A[3]$  is  $[2, 4, 8]$ .

- (a) (5 marks) The above program does not use a look-up table to store previously computed values. Rewrite the program so that it does. Use the following template.

```
int subseqDP (int A[], int n, int j, int LookUp[]);
```

and use the definition of `subseq` as a basis. In your solution only make the following changes: change one of the `if` statements, add an update to `LookUp` and, in the recursive call, change the name.

- (b) (5 marks) Describe how you would use `subseqDP` to solve the *longest strictly increasing subsequence* problem.