# Quiz 2 Solution

Data Structures, 1999.

July 14, 1999

Total marks: 30

**Name: Theodore Norvell**

**Student #:**

*Answer each question in the space provided or on the back of a page with an indication of where to find the answer.*

Labelled Binary Trees are represented by pointers to nodes, with the NULL pointer (0) representing empty trees. *Any nodes that are removed from the list should be deleted from the heap.* Tree nodes are defined in file "tree.h" as:

```
struct TreeNode { int data ; TreeNode * left; TreeNode* right ;} ;
```

**Q0** [5]

What is the worst case time complexity of the following algorithm for finding a string of length M as a substring of a string of length N:

```
int found = -1 ;
for(int i=0 ; i < N ; ++i ) {
    int k = 0 ;
    for( ; k<M && i+k<N && str[i+k]==pat[k] ; ++k ) { }
    if( k==M ) { found = i ; break ; } }
```

Let N be the problem size (M can be anywhere from 0 to N). Use big-Oh notation:

*Solution:* $O(N^2)$

*Explanation: An class of input that exhibits the worst case behaviour has the following form*

$$\begin{aligned} str &: \overbrace{aa\cdots a}^{N} \\ pat &: \underbrace{aa\cdots a}_{\lfloor N/2 \rfloor} b \end{aligned}$$

*Challenge: Find an $O(1)$ algorithm for pattern matching.*

**Q1** [6]

(a) A set is represented by a labelled binary tree. Suppose that the following strings are added in the following order using the insertion algorithm presented in class. (Consider strings to be ordered alphabetically.)

moe, flo, joe, shmo, ro, lo, zoe, beau

Draw the resulting tree.

*Solution: [2004 Sorry solution diagram seems to be lost! Here is the solution with empty trees omitted]*

(((beau),flo,(joe,(lo))),moe,((ro),shmo,(zoe)))

(b) Now the strings moe and joe are deleted using the deletion algorithm presented in class. Draw the resulting tree.

*Solution: [2004. Sorry solution diagram seems to be lost! Here is the solution with empty trees omitted]*

(((beau),flo),lo,((ro),shmo,(zoe)))

**Q2 [9]**

(a) Consider the following functions of $N$

$$\begin{aligned} f(N) &= N \times \log_{10} N + 1000 \times N \\ g(N) &= 10 \times N + 20 \\ h(N) &= N(N-1)/2 \end{aligned}$$

Which of the sets $O(\log_2 N)$, $O(N)$, $O(N \log_2 N)$, $O(N^3)$ are these functions in. Put an Y (for yes) or an N (for no) in each box:

| | $O(\log_2 N)$ | $O(N)$ | $O(N \log_2 N)$ | $O(N^3)$ |
|---|---|---|---|---|
| $f$ | $N$ | $N$ | $Y$ | $Y$ |
| $g$ | $N$ | $Y$ | $Y$ | $Y$ |
| $h$ | $N$ | $N$ | $N$ | $Y$ |

**Q3 [15]**

Given a possibly empty tree, create another tree that is an exact copy of it, except that the label field of each node of the copy should contain, not the original label, but rather the height of the subtree that starts at that node. E.g., if the input is $((o, 10, (o, 15, o)), 20, o)$ then the "copy" will be $((o, 2, (o, 1, o)), 3, o)$.

Set ok to true if you succeed and to false if you run out of memory. Do not change the input tree.

*Use recursion. Note that out_tree is an output parameter only.*

*Solution:*

---

```
#include <new>
using namespace std;
#include "tree.h"
...
void height_copy( TreeNode* tree, TreeNode* &out_tree, bool &ok ) {
```

```
    int junk ;
    height_copy_aux( tree, out_tree, ok, junk ) ;
}
void height_copy_aux( TreeNode* tree, TreeNode* &out_tree, bool &ok, int
&height ) {
    if( tree == 0 ) {
        ok = true ;
        out_tree = 0 ;
        height = 0 ; }
    else {
        out_tree = new(nothrow) TreeNode ;
        if( out_tree ) {
            int left_height, right_height ;
            bool left_ok, right_ok ;
            height_copy_aux( tree->left, out_tree->left, left_ok, left_height);
            height_copy_aux( tree->right, out_tree->right, right_ok, right_height);
            ok = left_ok && right_ok ;
            height = 1 + (left_height > right_height ? left_height : right_height);
            out_tree->label = height ; }
        else {
            ok = false ;
            height = 0 ; } } }
```

---

*Note. Many overwrote the value of ok returned from the first call without looking at it. This is wrong even if you assume that once the heap is exhausted it stays exhausted. (Find an example.) I've avoided that by using separate ok variables.*

*Note: This is an $O(N)$ algorithm, where $N$ is the tree size. Many student solutions used a separate height calculating function, which yields an $O(N^2)$ algorithm over all.*