

# CS 14 - Summer 2003 - Quiz 2

July 7, 2003

**Note:** Binary-tree questions are in terms of the simplest implementation of a pointer-based binary tree:

```
struct Node
{
    int val;
    Node* left;
    Node* right;
}
```

## 1 True/False

Circle *T* or *F* as appropriate. Running-time questions are all in terms of input size  $n$

1. **T F** Bubble-sort has a worst case of  $O(n \lg n)$  -  $O(n^2)$
2. **T F** Selection-sort has a worst case of  $O(n^2)$
3. **T F** Intelligently adding a  $O(n)$  check to see if the list is sorted during an  $O(n^2)$  sort makes it  $O(n^3)$  -  $n * (n + O(n))$  is still  $O(n^2)$
4. **T F** Quick-sort has a worst case of  $O(n \lg n)$  - Worst case is  $O(n^2)$
5. **T F** Quick-sort can be done without allocating any significant additional storage ("in place")
6. **T F** A binary tree of  $n$  nodes has a minimum height of  $O(\lg_2 n)$
7. **T F** A file-system (files, directories/folders) is an example of a tree hierarchy
8. **T F** A binary search tree requires at least  $O(n \lg_2 n)$  nodes to be visited to perform the search operation. - Search on a BST is  $O(\lg_2 n)$

## 2 Multiple Choice

Circle the letter of the correct answer. Running-time questions are all in terms of input size  $n$

1. For a binary search tree, which of the following can be implemented in  $O(1)$ 
  - (a) Pre-Order Traversal
  - (b) Insert
  - (c) In-Order Traversal
  - (d) *isEmpty* - The traversals must be at least  $O(n)$  to see all the values, and insert is  $O(\lg n)$ , but *isEmpty* is just a single comparison.
  
2. If Bubble-sort is implemented to move large elements to the end, what would the list 8314 look like after **two** swaps:
  - (a) 3 1 4 8
  - (b) 4 3 1 8
  - (c) 1 8 3 4
  - (d) 3 1 8 4 - Remember, bubble sort “bubbles up” the max vals.
  
3. For a binary-search tree as shown above, which of the following would be easiest to implement without recursion:
  - (a) Size
  - (b) Depth
  - (c) *Max* - This is just a while loop, since you always know which way to go in the tree (Go right).
  - (d) None of the above

### 3 Short Answer

*If you want the option to request a regrade available, your answers to these questions must be made in pen.*

1. Describe in English one of the two simple strategies for removing an element from a binary search tree. Be sure to cover all cases.  
 There are 3 cases
  - (a) The node to delete is a leaf, in which case we simply remove it
  - (b) The node to delete has one child, in which case we promote the child and delete the node
  - (c) The node to delete has two children, in which case we take the next-smallest or next-largest value (the max of the lesser subtree or the min of the greater subtree), swap values with it, and call delete on the same value again.

2. Write the function `int depth(Node* r)` for a binary-tree of *Nodes* whose root is *r*

```
int depth(Node* r)
{
    if (r == NULL) return 0;
    return 1 + max(depth(r->left), depth(r->right));
}
int max(int a, int b) return (a > b ? a : b);
```

3. Describe in English the algorithm for Selection-sort.

Given an unsorted list, find the minimum element, and swap it with the first element. Repeat this process with the 2nd smallest element and the 2nd element, the 3rd smallest and the 3rd, etc etc until the whole list is sorted.

4. What algorithm is this?

*QuickSort* - It is doing comparison and swapping, so it is probably a sort. It is recursive, which means it is probably either QuickSort or MergeSort. It doesn't have any  $\frac{1}{2}$ s in it, which means it is probably not MergeSort. The recursive step comes after a step that swaps  $O(n)$  elements, and is called on two variable-sized parts of the list. This is very much indicative of QuickSort.

```
void mysteryAlg(vector<int>& a, int l, int h)
{
    if (l >= h) return;
    int mid = a[l];
    int lastS1 = l;
    int lUnknown = l + 1;
    while (lUnknown <= h)
    {
        if (a[lUnknown] < mid)
        {
            lastS1++;
            swap(a[lUnknown], a[lastS1]);
        }
        lUnknown++;
    }
    swap(a[l], a[lastS1]);
    mysteryAlg(a, l, lastS1 - 1);
    mysteryAlg(a, lastS1 + 1, h);
}
```