CSE 250 – Final Exam

Fall 2013
Time: 3 hours.

Dec 11, 2013

Total points: 100	14 pages

Please use the space provided for each question, and the back of the page if you need to. Please do not use any extra paper. The space given per question is **a lot** more than sufficient to answer the question. Please be brief. Longer answers do not get more points!

- **No electronic devices of any kind. You can open your textbook and notes**
- **Please leave your UB ID card on the table**
- **This booklet must not be torn or mutilated in any way and must not be taken from the exam room**
- **Please stop writing when you are told to do so. We will not accept your submission otherwise.**
- **If you wanted to, you can answer the extra credit question without answering all of the other questions**

Your name:	______________________________
Your UB ID:	______________________________

The rest of this page is for official use only. Do not write on the page beyond this point.

Problem	Score	Problem	Score	Problem	Score
name and id (2 max)		Problem 1 (10 max)		Problem 2 (4 max)	
Problem 3 (4 max)		Problem 4 (8 max)		Problem 5 (8 max)	
Problem 6 (8 max)		Problem 7 (8 max)		Problem 8 (4 max)	
Problem 9 (4 max)		Problem 10 (8 max)		Problem 11 (8 max)	
Problem 12 (8 max)		Problem 13 (8 max)		Problem 14 (8 max)	
Total Score: (100 max)					

Problem 1 (10 points). Mark the correct choice(s) or give a brief answer. Each question is worth 1 point. All codes are in C++.

1. Of the following C++ statements, which ones are both declarations *and* definitions? (Mark all that apply.)

   ☐ `int foo(int b) { return b*b;}` ☐ `long i;` ☐ `char a='h';`
   ☐ `bool easy;` ☐ `typedef mytype string;` ☐ `struct BSTNode;`

2. Consider the following fragment

   ```
   char name[] = {'D','a','v','i','d','\0','B','l','a','i','n','e','\0' };
   cout << name;
   ```

   What does `cout` print?

   ☐ David ☐ Blaine ☐ David Blaine ☐ None of the above

3. Consider the following fragment

   ```
   int a[] = {1, 2, 3, 4}; *(a+2) += 3;
   ```

   What is the value of `a[2]` after the fragment is executed?

   ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6

4. Consider the following fragment

   ```
   int a[] = {1, 2, 3, 4}; int* p = a+1; p++; (*p)++;
   ```

   What is the value of `a[2]` after the fragment is executed?

   ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6

5. Will the following program compile without error? ☐ YES ☐ NO

   ```
   #include <iostream>
   using namespace std;
   int foo(int i) { return i; }
   int main() {
       int  a = 2;
       int *b = &a;
       cout << foo(*b) << endl;
       return 0;
   }
   ```

6. Suppose you wanted to make use of terminal control routines and the Lexer class I gave and all your codes are put in `yourprog.cpp`. Write down one `g++` compilation line that will compile `term_control.cpp` and produce the object file `term_control.o`

7. Write one line of C++ that defines a new type, named `mytype`, which represents a pointer to a function that takes two `char`s and returns nothing.

8. Write *one* C++ statement that defines a variable `mymap`, where `mymap` is a map from `string` to pointer to function that has two `int` arguments and returns an `int`.

9. To print all keys of a BST in decreasing order, we use the following traversal order

   ☐ In- ☐ Post- ☐ Pre- ☐ Reverse In- ☐ Reverse Post- ☐ Reverse Pre-

10. In the B-Tree data structure we discussed in class, the last pointer of each leaf node points to its sibling node on the right (except for the rightmost leaf node, whose last pointer points to `NULL`). Explain why those pointers point that way?

Problem 2 (4 points)**.** Order the following functions in increasing order of asymptotic growth rate. You don't have to explain how you get the order.

$$\frac{n}{(\log n)^3}, \quad \sqrt{n}, \quad 2^{n+10}, \quad n^2(\log n)^3, \quad n^{3/4}, \quad n \cdot 2^n, \quad n\sqrt{n}.$$

Problem 3 (4 points)**.** Use the recurrence tree method to solve the recurrence $T(n) = 6T(n/2) + n$. As usual, you can assume $T(k) = O(1)$ for $k \leq 10$. Draw the tree. Show your work.

Problem 4 (8 points). You can assume that `using namespace std;` is at the top of the file.

1. (**4**) Write a C++ function `iterative_range_count()` that takes a stack `st` of `int` and an integer `a` as arguments, and returns the number of integers in `st` that are at least `a`.

```
int iterative_range_count(stack<int> st, int a)
{

}
```

2. (**4**) Write a C++ function `recursive_range_count()` that solves the same problem recursively.

```
int recursive_range_count(stack<int> st, int a)
{

}
```

Problems 5-7 make use of the following linked list `Node` structure:

```
struct Node {
    int key;
    Node* next;
    Node(int k=0, Node* n=NULL) : key(k), next(n) {};
};
```

Problem 5 (8 points). Write a function `median_element` that takes a head pointer to a `NULL`-terminated singly linked list whose keys are sorted in increasing order. All keys are distinct. The function returns a pointer to the node which stores the median key of all keys. If there are n keys then the median key is the $\lceil n/2 \rceil$-smallest key. (If the list is empty then NULL is returned.) For example, if the input list is

$$a.1 \to b.5 \to c.7 \to d.8 \to \text{NULL}$$

then a pointer to b is returned ($n = 4$ in this case). As another sample, if the input list is

$$a.1 \to b.5 \to c.7 \to d.8 \to e.10 \to \text{NULL}$$

then a pointer to c is returned ($n = 5$ in this case). **Most importantly**, you can only use **one** `while` loop in your function, and that's the only looping structure you can use. In particular, the easy solution of looping through to find n, and traverse the second time to report the median is not valid. (**Hint:** recall the linked-list cycle detection problem I discussed in class.)

```
Node* median_element(Node* head) {




}
```

Problem 6 (8 points). Write a function `fold_list()` that takes a head pointer to a `NULL`-terminated singly linked list consisting of `Node`s. The function modifies the list in the following way: it cuts the list in half, moves the first half to the end, and returns the head pointer to the new list. **Only pointer manipulation is allowed.** If the list has an odd number of elements, then the middle element belongs to the first half. If the list is empty, `NULL` is returned. Feel free to call the `median_element()` function from Problem 5.

For example, if the input list is

$$a.4 \to b.1 \to c.3 \to d.6 \to e.19 \to f.14 \to \texttt{NULL}$$

then the output list is

$$d.6 \to e.19 \to f.14 \to a.4 \to b.1 \to c.3 \to \texttt{NULL},$$

and a pointer to d is returned. And, if the input list is

$$a.4 \to b.1 \to c.3 \to d.6 \to e.19 \to \texttt{NULL}$$

then the output list is

$$d.6 \to e.19 \to a.4 \to b.1 \to c.3 \to \texttt{NULL},$$

and a pointer to d is returned.

```
Node* fold_list(Node* head) {









}
```

Problem 7 (8 points)**.** Write a function `difference()` that takes *two* head pointers to `Node` as arguments. Each head pointer points to the head element of a `NULL`-terminated singly linked list which contains *distinct* keys sorted in increasing order. (The keys within each list are distinct, but keys in different lists aren't necessarily different.) The function `difference()` returns a *new* **sorted** linked list of `Node`s that contain all keys of the two input linked lists that are **not** in common between the two lists. You have to keep the input linked lists intact. For example, if the two input lists are

$$a.1 \to b.4 \to c.7 \to d.16 \to \text{NULL}$$

$$e.2 \to f.4 \to g.9 \to h.16 \to i.17 \to \text{NULL}$$

Then, the output list is

$$a'.1 \to e'.2 \to c'.7 \to g'.9 \to i'.17 \to \text{NULL}$$

and a pointer to a' is returned. (I used `a', e', c', g', i'` because they are not the same nodes as the ones from the input lists.) Write the function so that its asymptotic running time linear in the total input list size.

```
Node* difference(Node* head1, Node* head2) {











}
```

Problem 8 (4 points). Draw the binary tree that has the following inorder and postorder sequences

```
Inorder  : 4 15  6 10  8  7  5
Postorder: 4  6 15  7  8  5 10
```

Problem 9 (4 points). Suppose we insert node 3 into the following AVL tree; Show where 3 is inserted at first, then circle the node that becomes imbalanced and explain how the re-balance step works to keep the AVL property. Draw the resulting AVL tree.

```
    2
   / \
  1   8
     / \
    4   9
```

Problem 10 (8 points). (a) Suppose we splay node 6 in the following tree, what does the resulting tree look like? Draw all the intermediate trees after each zig-zig, zig-zag, or zig baby step and label the step with the name zig-zig, zig-zag, or zig.

```
      4
     / \
    2   10
       / \
      8   11
     / \
    7   9
   /
  6
```

(b) Consider the following Binary Search Tree. Is it an AVL Tree? Why or why not? Put a color red (R) or black (B) next to each node so that it is a red-black tree. (The x are NULL nodes.)

```
           _____40______
          /             \
       __30__            50
      /      \          /  \
     10       35       x    60
    /  \     /  \          /  \
   8    15  x    x        x    x
  / \  /  \
 x   x x   x
```

Problems 10 to 14 make use of the following binary search tree node structure:

```
struct BSTNode {
    int key;
    BSTNode* left;
    BSTNode* right;
    BSTNode* parent;
};
```

Problem 11 (8 points)**.** Write a function that takes a pointer to the root of a binary search tree with the above `BSTNode` structure, and that removes the node with the *minimum* key from the tree. Note that the root pointer is passed by reference. If the minimum node is the root node then the root pointer has to be modified properly. All your code has to be contained within the function, do not write any auxiliary function.

```
void delete_min(BSTNode*& root) {

 }
```

Problem 12 (8 points). Write a function `common_ancestors()` that takes two (not necessarily distinct) pointers to `BSTNode`s in a BST and returns a vector of pointers to nodes in the tree which are the common ancestors of the two given nodes. (Note that by definition a node is an ancestor of itself.)

```
vector<BSTNode*> common_ancestors(BSTNode* node1, BSTNode* node2)
{

}
```

Problem 13 (8 points). (a) Write a function `size()` that takes the root pointer of a binary tree with the `BSTNode` structure and returns the size of the tree, which is the number of non-NULL nodes in the tree.

```
size_t size(BSTNode* root) {









}
```

(b) Write a function `rank()` that takes a pointer `node` to a node (not necessarily the root) of a binary search tree with the `BSTNode` structure and returns the *rank* of that node. The rank of a node is k if the node's key is the kth smallest key among all keys. You can assume that all keys in the tree are distinct. You should make use of the `size()` function from part (a). If `node` is `NULL` then 0 is returned.

```
size_t rank(BSTNode* node) {









}
```

Problem 14 (8 points). Write a function `last_node_depth_k()` that takes the root pointer of a binary tree with the `BSTNode` structure, a non-negative integer `k` and returns a pointer to the last node at depth k of the tree. (Last as we go from left to right.) The root node is at depth 0, the root's children are at depth 1, and so on. If k is greater than the largest depth, then `NULL` is returned.

For example, consider the tree drawn in Problem 10b. The last node at depth 1 is node 50, at depth 2 is node 60, at depth 3 is 15, at any depth > 3 is `NULL`.

```
BSTNode* last_node_depth_k(BSTNode* root, size_t k) {

}
```