Second Examination
CS 225 Data Structures and Software Principles
Spring 2011
7p-9p, Tuesday, April 5

Name:
NetID:
Lab Section (Day/Time):

- This is a closed book and closed notes exam. No electronic aids are allowed, either.

- You should have 5 problems total on 17 pages. The last sheet is scratch paper; you may detach it while taking the exam, but must turn it in with the exam when you leave. Use scantron forms for Problems 1 and 2.

- Unless otherwise stated in a problem, assume the best possible design of a particular implementation is being used.

- Unless the problem specifically says otherwise, assume the code compiles, and thus any compiler error is an exam typo (though hopefully there are not any typos).

- We will be grading your code by first reading your comments to see if your plan is good, and then reading the code to make sure it does exactly what the comments promise. In general, complete and accurate comments will be worth approximately 30% of the points on any coding problem.

- Please put your name at the top of each page.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Score</th>
<th>Grader</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td></td>
<td>scantron</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td></td>
<td>scantron</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. [Miscellaneous – 25 points].

MC1 (2.5pts)

Suppose a binary tree holds 127 keys. Then our node-based implementation of that tree has how many NULL pointers?

(a) 64  
(b) 128  
(c) 256  
(d) The answer cannot be determined from the information given.  
(e) None of these is the correct response.

MC2 (2.5pts)

Which of the following sequences of keys cannot be the inOrder traversal of an AVL tree?

(a) 1 3 6 8 12 15 17 19  
(b) 50 120 160 172 183 205 200 230  
(c) 12 14 22 24 27 40 45  
(d) More than one of these are not valid inOrder traversals.  
(e) All of these are valid inOrder traversals.

MC3 (2.5pts)

How many data structures in the following list are *dictionaries*?

stack queue binary search tree AVL tree B-Tree

(a) 1  
(b) 2  
(c) 3  
(d) 4  
(e) 5
MC4 (2.5pts)

Consider the following partial C++ code:

```cpp
#include <vector>
#include <iostream>
using namespace std;

template <class Iter, class Formatter>
void mystery(Iter front, Iter end, Formatter doSomething) {
    Iter fast = front; Iter slow = front;

    while (fast != end) {
        fast++; fast++; slow++;
    }

    while (slow != front) {
        fast--; slow--;
        doSomething(*fast, *slow);
    }
}

template <class T>
class dunno {
public:
    void operator()(T & a, T & b) {
        T temp = a;
        a = b;
        b = temp;
    }
};

int main() {
    vector<int> v;
    // insert an even number of elements into the vector

    vector<int>::iterator it1 = v.begin();
    vector<int>::iterator it2 = v.end();

    dunno<int> d;
    mystery<vector<int>::iterator,dunno<int> >(it1, it2, d);

    for (int i = 0; i < v.size(); i++) cout << v[i] << endl;
    return 0;
}
```
If you assume that all iterators are bi-directional and non-constant, which of the following statements is true?

(a) This code does not compile.
(b) If the vector consists of the integers 1, 2, 3, 4, 5, 6 in that order, with \(v[0] == 1\), then the output is 1, 2, 3, 4, 5, 6.
(c) If the vector consists of the integers 1, 2, 3, 4, 5, 6 in that order, with \(v[0] == 1\), then the output is 4, 5, 6, 1, 2, 3.
(d) If the vector consists of the integers 1, 2, 3, 4, 5, 6 in that order, with \(v[0] == 1\), then the output is 6, 5, 4, 3, 2, 1.
(e) None of these options describes the behavior of this code.

MC5 (2.5pts)

Think of an algorithm that uses a \textit{Stack} to efficiently check for unbalanced parentheses. What is the maximum number of parentheses that will appear on the stack at any time when the algorithm analyzes the string (())(())())? 

(a) 1
(b) 2
(c) 3
(d) 4
(e) 5 or more
MC6 (2.5pts)

Suppose we remove the node containing key 10 from the AVL tree below.

What sequence of rotations will restore the balance of the tree? (Assume IOP is used for 2-child removal.)

(a) leftRotate about 9, followed by rightRotate about 8.
(b) rightRotate about 8.
(c) leftRotate about 12, followed by rightRotate about 8.
(d) rightLeftRotate about 9, followed by rightRotate about 8.
(e) None of these choices will rebalance the tree.
MC7 (2.5pts)

Examine the `mysteryFunction` below? (Note that in context, `t->right` will not be NULL.)

```c
void mysteryFunction(treeNode * & t) {
    treeNode * y = t->right;
    t->right = y->left;
    y->left = t;
    y->height = max( height(y->right), height(y->left)) + 1;
    t->height = max( height(t->right), height(t->left)) + 1;
    t = y;
}
```

Which of the following Dictionary functions will certainly not invoke `mysteryFunction`?

(a) `insert(key)`;
(b) `remove(key)`;
(c) `find(key)`;
(d) Two or more of these will certainly not invoke `mysteryFunction`.
(e) All of these could employ `mysteryFunction`.

MC8 (2.5pts)

Consider the AVL Tree built by inserting the following sequence of integers, one at a time: 5, 4, 7, 9, 8, 3, 1. What is 8’s left child?

(a) `NULL`
(b) The node containing 4.
(c) The node containing 5.
(d) The node containing 7.
(e) None of these answers is correct.
MC9 (2.5pts)

What is the minimum number of keys that can be stored in a B-Tree of order 64 and height 5?

(a) $2^{25} - 1$
(b) $2^{25} + 1$
(c) $2^{26} - 1$
(d) $2^{30} - 1$
(e) $2^{30} + 1$

MC 10 (2.5pts)

Which of the following statements is true for a B-tree of order $m$ containing $n$ items?

(i) The height of the B-tree is $O(m \log_m n)$ and this bounds the total number of disk seeks.
(ii) A node contains a maximum of $m - 1$ keys, and this an upper bound on the number of
    key comparisons at each level of the tree during a search.
(iii) Every AVL Tree is also an order 2 B-tree.

Make one of the following choices.

(a) Only item (i) is true.
(b) Only item (ii) is true.
(c) Only item (iii) is true.
(d) Two of the above statements are true.
(e) All statements (i), (ii), and (iii) are false.
2. [Efficiency – 25 points].

Each item below is a description of a data structure, its implementation, and an operation on the structure. In each case, choose the appropriate running time from the list below. The variable \( n \) represents the number of items (keys, data, or key/data pairs) in the structure. In answering this question you should assume the best possible implementation given the constraints, and also assume that every array is sufficiently large to handle all items. Please use the scantron sheets for your answers.

\[
\begin{align*}
(a) & \quad O(1) \\
(b) & \quad O(\log n) \\
(c) & \quad O(n) \\
(d) & \quad O(n \log n) \\
(e) & \quad O(n^2)
\end{align*}
\]

(MC 11) _____ **Enqueue** for a *Queue* implemented with an array.

(MC 12) _____ **Dequeue** for a *Queue* implemented with an array.

(MC 13) _____ Find a key in a Binary Tree (not necessarily BST).

(MC 14) _____ Remove a single key from a height balanced Binary Search Tree.

(MC 15) _____ Find the largest key in a Binary Search Tree.

(MC 16) _____ Find the largest key in an AVL Tree.

(MC 17) _____ Compute the height of every subtree in a Binary Search Tree.

(MC 18) _____ Given a Binary Search Tree whose subtree heights have been computed, determine if the Binary Search Tree is an AVL Tree.

(MC 19) _____ Determine if a binary tree is a Binary Search Tree.

(MC 20) _____ Remove all the nodes from a BST (as is done by the destructor).
3. [Quadtrees – 25 points].

For this question, consider the following partial class definition for the Quadtree class, which uses a quadtree to represent a $d$-by-$d$ square bitmap image as in MP5. As a simplifying assumption for this problem, you may assume that all non-leaf nodes' RGBApixel elements already hold the component-wise average of their children's RGBApixel elements, and that the Quadtree resolution is set to $d$. Just as in MP5, you may assume that $d$ is a power of 2.

class Quadtree
{
    public:
    // constructors and destructor, big three, other public methods

    void buildTree(BMP const & source, int newResolution);
    RGBApixel getPixel(int x, int y) const;
    BMP decompress() const;
    void clockwiseRotate(); // 90 degree turn to the right
    void prune(int tolerance);
    int pruneSize(int tolerance) const;
    int idealPrune(int numLeaves) const;

    private:
    struct QuadtreeNode {
        QuadtreeNode * nwChild; // pointer to northwest child
        QuadtreeNode * neChild; // pointer to northeast child
        QuadtreeNode * swChild; // pointer to southwest child
        QuadtreeNode * seChild; // pointer to southeast child

        RGBApixel element; // the pixel stored as this node's "data"
    };

    QuadtreeNode * root; // ptr to root of quadtree, NULL if tree is empty.
    int resolution; // number of pixels on a side of the image (assume $2^k$)

    void pruneHelper(QuadtreeNode * croot, tolerance);
    void clear(QuadtreeNode **& croot);
    int squaredDist(const RGBApixel & a, const RGBApixel & b);
    int max(int a, int b, int c, int d)

    // other helper functions
};
(a) In this part you will implement a helper to the `prune` function above (`pruneHelper`).

A node \( n \) can be pruned if there is no leaf in the subtree rooted at node \( n \) which has a color whose squared distance from the color of \( n \) in RGB-space is greater than tolerance. If a node \( n \) can be pruned, the subtrees rooted at the children of \( n \) are cleared from the Quadtree.

Background information: Here we have given you an implementation of the `prune` function in the public portion of the Quadtree definition above. You will be writing the function `pruneHelper` called here.

```c++
void Quadtree::prune(int tolerance)
{
    if (root == NULL) return;
    pruneHelper( root, tolerance );
}
```

If you have an implementation that differs significantly from the code provided here, you may change the above function and parameters to `pruneHelper`.

`pruneHelper` may call the provided functions in the Quadtree class above. Specifically, these may be useful:

```c++
void clear(QuadtreeNode * & croot):
This function clears a node and all descendent nodes in this node’s subtree. It also deletes `croot` itself and sets `croot` to point to NULL.

int squaredDist(const RGBAPixel & a, const RGBAPixel & b):
This function returns the squared distance in RGB-space between RGBApixels `a` and `b`.

int max(int a, int b, int c, int d):
This function returns maximum of 4 integers.
```
i. (10 points) Write the main private helper function that actually does the work for the `prune` function in part (a). Your solution will be graded on comments and efficiency, in addition to correctness. Note that there is room for other helper functions on the next page. Note: This is a long question. You may want to move on to other questions and come back to this part.

```cpp
void Quadtree::pruneHelper(QuadtreeNode * croot, int tolerance) {
}
```
You may implement any other helper functions here. As a suggestion, we have a single stub function below, but you may implement more helper functions if you need to. If you run out of space, you may put a note here and continue your solution on the back of this page.

```cpp
Quadtree::_____________(____________________)
```

```cpp
```
(b) In this part we will examine the relationship between \( n \), the number of nodes in an unpruned quadtree, \( h \), the height of the quadtree.

i. (1 point) Let \( N(h) \) denote the number of nodes in an unpruned *Quadtree* of height \( h \). Give a recurrence for \( N(h) \)

ii. (2 points) Solve this recurrence (i.e. by unrolling) to get a tight Big-\( O \) bound on the number of nodes in a *Quadtree* of height \( h \)

iii. (2 points) Give a tight Big-\( O \) bound on the height of a *Quadtree* with \( n \) nodes
(c) Give the runtimes of the following functions from MP 5, in terms of \( n \), the number of nodes in the Quadtree. Assume all functions are called on an unpruned Quadtree. Be sure to give brief justifications of your responses.

i. (2 points) `buildTree`: This function takes two arguments, a BMP object by reference and an integer \( d \), and creates a Quadtree representing the upper-left \( d \) by \( d \) block of the BMP.

ii. (2 points) `getPixel`: This function takes two arguments, \( x \) and \( y \), and returns the RGBA pixel corresponding to the pixel at coordinates \((x, y)\) in the bitmap image which the quadtree represents.

iii. (2 points) `decompress`: This function returns the underlying BMP object represented by the quadtree.

iv. (2 points) `clockwiseRotate`: This function rotates the Quadtree object’s underlying image clockwise by 90 degrees.

v. (2 points) `prune`: This function is described in part (a) of the problem.
4. [Flood Fill – 15 points].

(a) (4 points) Suppose we execute function `BFSfillSolid`, one of two main fill routines from MP4, on the following 8x8 pixel image, beginning at the circled node, (3,5), and changing white pixels to solid red. Place the numbers 1 through 12, in order, in the first 12 pixels whose colors are changed by the function. Assume that we start the algorithm by adding the circled cell to the ordering structure, and that we add the four neighboring pixels to the structure clockwise beginning on the right, followed by down, left, and up.

![Image of 8x8 pixel grid with a circled node at (3,5)](image)

(b) (4 points) Suppose we want to fill some part of an arbitrary n-by-n image. What is the worst-case running time of `BFSfillSolid` if we start from an arbitrary location? Be sure to give your running time in terms of `n`, the length of one side of the square image.

(c) (4 points) Suppose we execute function `DFSfillSolid`, one of two main fill routines from MP4, on the following 8x8 pixel image, beginning at the circled node, (3,5), and changing white pixels to solid red. Place the numbers 1 through 12, in order, in the first 12 pixels whose colors are changed by the function. Assume that we start the algorithm by adding the circled cell to the ordering structure, and that we add the four neighboring pixels to the structure clockwise beginning on the right, followed by down, left, and up.

![Image of 8x8 pixel grid with a circled node at (3,5)](image)

(d) (3 points) What data structure was used to order the points for filling in function `DFSfillSolid`?
5. [AVL Tree Height – 10 points].
In this problem you will complete the proof of the following theorem:

The height of an \( n \)-node AVL Tree is \( O(\log n) \).

Fill in the blanks to complete the proof.

**Preliminaries:** Let \( H(n) \) denote the maximum height of an \( n \)-node AVL Tree, and let \( N(h) \) denote the minimum number of nodes in an AVL tree of height \( h \). To prove that \( H(n) = O(\log n) \), we argue that

\[
H(n) \leq 2 \log_2 n, \text{ for all } n.
\]

Rather than prove this inequality directly, we’ll show equivalently that

\[
N(h) \geq \underline{\text{__________}}.
\]

**Proof:** For an arbitrary value of \( h \), the following recurrence holds for all AVL Trees:

\[
N(h) = \underline{\text{__________}} + \underline{\text{__________}} + \underline{\text{__________}}
\]

and \( N(0) = \underline{\text{__________}}, N(1) = \underline{\text{__________}} \)

This expression for \( N(h) \) simplifies to the following inequality which is a function of \( N(h-2) \):

\[
N(h) \geq \underline{\text{__________}} \times \underline{\text{__________}}
\]

By an inductive hypothesis which states:

\[
\underline{\text{__________________________________________________________________________}}
\]

we now have:

\[
N(h) \geq 2 \times \underline{\text{__________}} = \underline{\text{__________}},
\]

which was what we wanted to prove.'
scratch paper