University of Illinois at Urbana-Champaign
Department of Computer Science

Second Examination

CS 225 Data Structures and Software Principles
Spring 2012
7p-9p, Tuesday, April 3

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

- This is a **closed book** and **closed notes** exam. No electronic aids are allowed, either.
- You should have 6 problems total on 15 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.

Problem	Points	Score	Grader
1	25		scantron
2	25		scantron
3	20		
4	10		
5	10		
6	10		
Total	100		

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 20 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)

Which of the following data structures is used in our implementation of a level order traversal of a binary tree?

(a) linked list

(b) array

(c) stack

(d) queue

(e) hash table

MC4 (2.5pts)

Consider the following recursive C++ function, and assume our standard node-based implementation of the `BinaryTree` class which includes a private definition of a `Node` class.

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

template <typename T>
void BinaryTree<T>::mystery(const Node * cRoot, vector<T> & unk) const
{
     if (cRoot != NULL) {
          unk.push_back(cRoot->elem);
          if (cRoot->left == NULL && cRoot->right == NULL) {
               for (size_t i = 0; i < unk.size(); i++)
                    cout << '  ' << unk[i];
               cout << endl;
          }
          else {
               mystery(cRoot->left,  unk);
               mystery(cRoot->right, unk);
          }
          unk.pop_back();
     }
}
```

Among the following choices, print the best name for the function. (That is, we are asking for a name that describes what the function does.)

(a) `printMirror` - prints the mirror image of the binary tree.

(b) `printTree` - prints each node in the tree exactly once, in preOrder.

(c) `printLevel` - prints each node in the tree exactly once in level order from bottom to top.

(d) `printPaths` - prints every path from the root to a leaf in the tree.

(e) None of these options is a good name for the function.

MC5 (2.5pts)

Think of an algorithm that uses a `Stack` to efficiently check for unbalanced or unpaired delimiters. What is the maximum number of left-delimiters that will appear on the stack at any time when the algorithm analyzes the string `{[({ }[ ])]({ })}`?

(a) 1

(b) 2

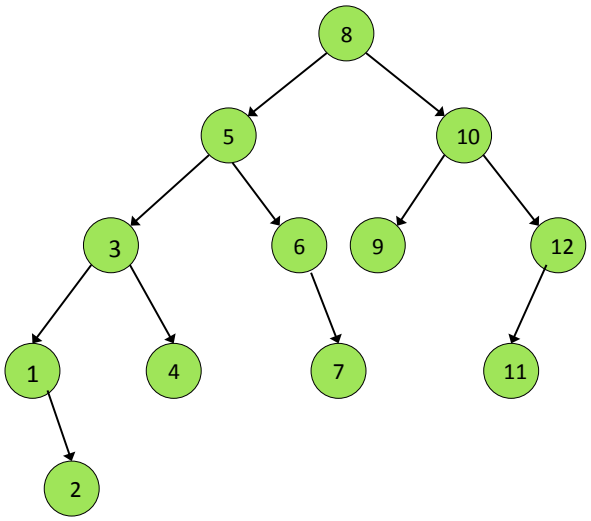
(c) 3

(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`.)

```
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 could invoke `mysteryFunction` more than once?

(a) `insert(key);`

(b) `remove(key);`

(c) `find(key);`

(d) Two or more of these could invoke `mysteryFunction` more than once.

(e) None of these would invoke `mysteryFunction` more than once.

MC8 (2.5pts)

Consider the AVL Tree built by inserting the following sequence of integers, one at a time in the given order: $10, 15, 20, 13, 11$. Which of the following statements is true about the tree?

(a) 11's left child is `NULL` and 15's left child is 11.

(b) 11's left child is `NULL` and 15's left child is 13.

(c) 11's left child is 10 and 15's left child is 11.

(d) 11's left child is 10 and 15's left child is 13.

(e) None of these answers is correct.

MC9 (2.5pts)

What is the maximum number of keys in a binary search tree of height 30?

(a) $2^{29} - 1$

(b) $2^{30} - 1$

(c) $2^{31} - 1$

(d) The answer cannot be determined by the given information.

(e) None of these answers is correct.

MC 10 (2.5pts)

Use the following 3 code examples to answer the question below. Please assume that all arrays and images have been properly initialized to hold valid data.

(i)
```
RGBAPixel colorArray[100];
loadColors(colorArray); // initialize array values
#pragma omp parallel for
for (int i = 1; i < 100; i++) {
     colorArray[i] = colorArray[i-1];
}
```

(ii)
```
PNG image("dogPic.PNG");  // load from file
#pragma omp parallel for
for (int i = 0; i < image.width(); i++) {
     for (int j = 0; j < image.height(); j++) {
            RGBAPixel temp = *image(i, j);
            *image(i, j) = *image(i, image.height() - 1 - j);
            *image(i, image.height() - 1 - j) = temp;
     }
}
```

(iii)
```
int table [10][10];
#pragma omp parallel for
for(int i = 0; i < 10; i++)
     for(int j = 0; j < 10; j++)
          table[i][j] = (i+1)*(j+1);
```

Which of the code examples above is/are NOT correctly parallelized?

(a) Only item (i) is incorrect.

(b) Only item (ii) is incorrect.

(c) Only item (iii) is incorrect.

(d) Two of the above examples are incorrect.

(e) All statements (i), (ii), and (iii) are correct.

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 worst case 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.

(a) $O(1)$
(b) $O(\log n)$
(c) $O(n)$
(d) $O(n \log n)$
(e) $O(n^2)$

(MC 11) A `Enqueue` for a `Queue` implemented with an array.

(MC 12) A `Pop` for a `Stack` implemented with an array.

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

(MC 14) B Remove the root of a balanced Binary Search Tree.

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

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

(MC 17) C For each node in a Binary Search Tree, compute the length of the longest path from the node to a leaf.

(MC 18) C Make a copy of an AVL tree.

(MC 19) C Determine if two given Binary Search Trees are copies of one another.

(MC 20) C Remove all the nodes in the right subtree of a non-empty AVL tree.

3. **[Quadtrees – 20 points].**

For this question, consider the following partial class definition for the Quadtree class, which uses a quadtree to represent a square bitmap image as in MP5. As a simplifying assumption for this problem, you may assume that only leaf nodes contain valid `RGBApixel` elements, and that the `element` field in all non-leaf nodes is not initialized to any particular value (we will not be doing any pruning of this tree).

```
class Quadtree
{
  public:
     // constructors and destructor; all of the public methods from MP5, including:

    void buildTree(PNG const & source, int d);
    RGBApixel getPixel(int x, int y) const;
    PNG 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(RGBApixel const & elem);
        QuadtreeNode();
     };

     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 copy(QuadtreeNode *& firstNode, QuadtreeNode * secondNode);
     void clear(QuadtreeNode *& curNode);
};
```

You may not use any methods or member data of the Quadtree or QuadtreeNode classes which are not explicitly listed in the partial class declaration above. You may assume that each child pointer in a new `QuadtreeNode` is NULL.

(a) (3 points) Write a *public* member function `void buildTree(PNG const & source, int d)`, which creates a Quadtree rooted at private member `root` representing the d-by-d block in the upper left corner of the image in `source`. Your function can call the *private* helper function that you will be writing in the next part of the problem, and it can use any of the other private member functions listed above. You may assume that the image is at least d-by-d, that $d \geq 1$, and that d is a power of 2. Write the function as it would appear in the quadtree.cpp file for the `Quadtree` class.

```
void buildTree(PNG const & source, int d) {

    clear(root); // 1 point
    resolution = d; // 1 point
    buildTree(root,source,0,0,resolution);
    return;

}
```

(b) (6 points) Write the *private* helper method you invoked in the previous part. For this one, you may choose the return value and the number and types of parameters. Note that our skeleton below should have sufficiently many lines for your solution, but you are welcome to add more if you need to do so. Please comment your code.

```
void Quadtree::buidTree
    (QuadtreeNode*& curNode, PNG const& souce, int x, int y, int res) {
    // 0.5 point

    if (res == 1) {
        curNode = new QuadtreeNode(*source(x,y)); // 1 point
        return;
    }

    curNode = new QuadtreeNode(); // 0.5 point

    // create child subtrees -- order of recursion doesn't matter
    // 3 points
    buildTree(curNode->nwChild, source, x,         y        , res/2);
    buildTree(curNode->neChild, source, x + res/2, y        , res/2);
    buildTree(curNode->seChild, source, x + res/2, y + res/2, res/2);
    buildTree(curNode->swChild, source, x,         y + res/2, res/2);

    return;

}

    1 point: For commenting
```

(c) (2 points) In our implementation of a Quadtree we have said that if a node has one child,

then all four children must exist–even in a pruned tree. Which one of the following terms is an appropriate description of this characteristic of our Quadtree? (Hint: we discussed this term in the context of binary trees, but it applies to Quadtrees, as well.)

perfect complete **full** balanced binary

(3 points, or 1 point for incorrect answer with explanation)

(d) (3 points) Suppose a pruned Quadtree has height $h \geq 0$. What is the least number of nodes it contains in terms of h?

$$3d - 4 * h + 1$$

(3 points, or 1 point for incorrect answer with explanation)

(e) Analyze and give tight asymptotic bounds on the running time of each of the following functions from MP5. Your bound should be stated in terms of N, the number of nodes in the `Quadtree`, and you can assume that the functions are called on an unpruned tree. Briefly justify your answers.

i. (2 points) `buildTree`: This function takes two arguments, a PNG object by reference and an integer d, and creates a Quadtree representing the upper-left d by d block of the PNG. (N is the number of nodes after the tree is built.)

$$O(n)$$

(2 points, or 1 point for incorrect answer with explanation)

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

$$O(\log(n))$$

(2 points, or 1 point for incorrect answer with explanation)

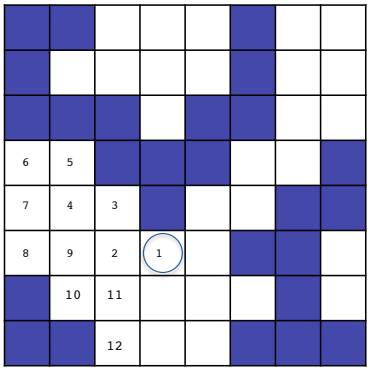
iii. (2 points) `clockwiseRotate`: This function rotates the Quadtree object's underlying image clockwise by 90 degrees.

$$O(n)$$

(2 points, or 1 point for incorrect answer with explanation)

4. **[Flood Fill – 10 points].**

(a) (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.



- 1point: for avoiding the black squares (irrespective of traversal).
- 1point: for a DFS.
- 1point: in the correct direction (pixels should be to the left of the starting pixel).
- 1point: all pixels are labeled in the correct order.

(b) (3 points) Suppose we want to fill some part of a arbitrary n-by-n image. What is the worst-case running time of `DFSfillSolid` 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.

$$O(n^2)$$

(c) (3 points) What data structure was used to order the points for filling in function `DFSfillSolid`?

Stack

- 3 points: 'Stack'
- 3 points: 'Queue' and BFS in part (a)
- 1 point: 'Stack' but performed a BFS in part (a)

5. **[AVL Tree Height – 10 points].**

In this problem you will complete the proof of the following theorem:

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

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

$$\underline{N(h) \geq 2^{h/2}}. \quad (2 \text{ points})$$

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

$$N(h) = \underline{N(h-1) + N(h-2) + 1} \quad (2 \text{ points})$$

$$\text{and } N(0) = \underline{1}, N(1) = \underline{2 \text{ or } 3} \quad (1 \text{ point each})$$

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

$$N(h) \geq \underline{2 * N(h-2)} \quad (1 \text{ point})$$

By an inductive hypothesis which states:

$$\underline{\forall x < h, N(x) >= 2^{x/2}} \quad (2 \text{ points})$$

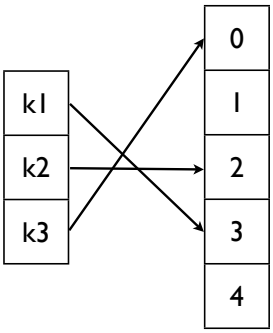
we now have:

$$N(h) \geq 2 * \underline{2^{(h-2)/2}} = \underline{2^{h/2}}, \quad (2 \text{ points})$$

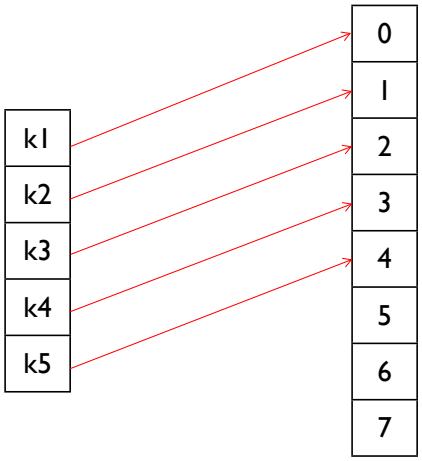
which was what we wanted to prove.

6. **[Hashing – 10 points].**

The following diagram is intended to illustrate the behavior of a given hash function. Specifically, our diagram illustrates the hash function defined by $h(k1) = 3$, $h(k2) = 2$, and $h(k3) = 0$, when mapped to a table of size 5. In the first 3 parts of this problem we will ask you to complete diagrams for hash functions with particular characteristics.

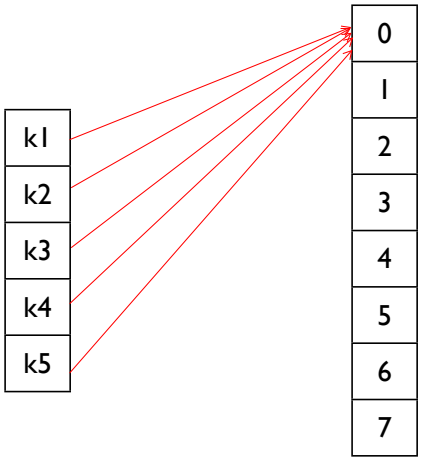


(a) (2 points) Complete the diagram below to illustrate a perfect hash function.



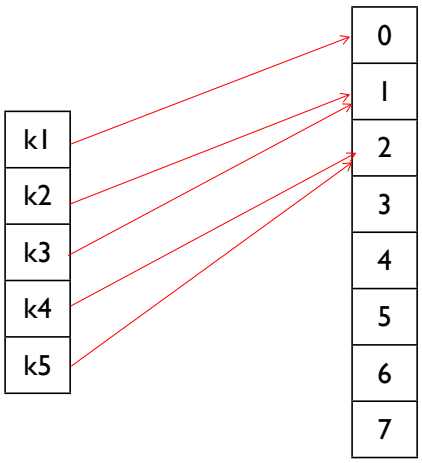
Full credit if there are no collisions.

(b) (2 points) Complete the diagram below to illustrate a hash function that clearly and extremely violates the Simple Uniform Hashing Assumption.



- Full credit if all keys map to the same value
- 1 point if the keys map to a few values

(c) (2 points) Complete the diagram below to illustrate a hash function with exactly two collisions on the given set of keys.



- Full credit if there are 2 collisions
- 1 point if there is 1 collision

(d) (2 points) What is the load factor for each of the hash tables in parts (a) through (c)?

$$\frac{5}{8}$$

- Full credit for the right answer
- 1 point for 8/5

(e) (2 points) We argue that the dictionary function `find` runs in constant time using a hash table. This argument depends on two things. List them here:

- Hash function is O(1)
- SUHA/few collisions
- Resizing to keep alpha constant

(Any two)

scratch paper