University of Illinois at Urbana-Champaign
Department of Computer Science

Second Examination

CS 225 Data Structures and Software Principles
Spring 2014
7-10p, Tuesday, April 8

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 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. The first two questions should be answered on your scantron sheet. Please be sure that your netid is accurately entered on the scantron.
- 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.
- Please put your name at the top of each page.

Problem	Points	Score	Grader
1	35	scantron	
2	20	scantron	
3	10		
4	20		
5	15		
Total	100		

1. **[Miscellaneous – 35 points].**

MC1 (2.5pts)

Suppose you implement a `queue` using a singly linked list with head and tail pointers so that the front of the `queue` is at the tail of the list, and the rear of the `queue` is at the head of the list. What is the best possible worst-case running time for `enqueue` and `dequeue` in this situation? (As a reminder, `enqueue` occurs at the rear of the queue.)

(a) O(1) for both functions.

(b) O(1) for enqueue and O(n) for dequeue.

(c) O(n) for enqueue and O(1) for dequeue.

(d) O(n) for both functions.

(e) None of these is the correct response.

MC2 (2.5pts)

Think of an algorithm that uses a `Stack` to efficiently check for unbalanced brackets. What is the maximum number of characters that will appear on the stack at any time when the algorithm analyzes the string `([]()[()])`?

(a) 3

(b) 4

(c) 5

(d) 6

(e) None of these is correct.

MC3 (2.5pts)

Consider a sequence of push and pop operations used to push the integers `0` through `9` on a stack. The numbers will be pushed in order, however the pop operations can be interleaved with the push operations, and can occur any time there is at least one item on the stack. When an item is popped, it is printed to the terminal.

Which of the following could NOT be the output from such a sequence of operations?

(a) `0 1 2 3 4 5 6 7 8 9`

(b) `4 3 2 1 0 5 6 7 8 9`

(c) `5 6 7 8 9 0 1 2 3 4`

(d) `4 3 2 1 0 9 8 7 6 5`

(e) All of these output sequences are possible.

MC4 (2.5pts)

Consider an array based implementation of a stack, and suppose that it is initially empty. Upon n `push` operations the array will be resized in such a way that the running time per push is $O(1)$ per operation, on average. How many times is the array resized over the n `push`es, using this scheme?

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

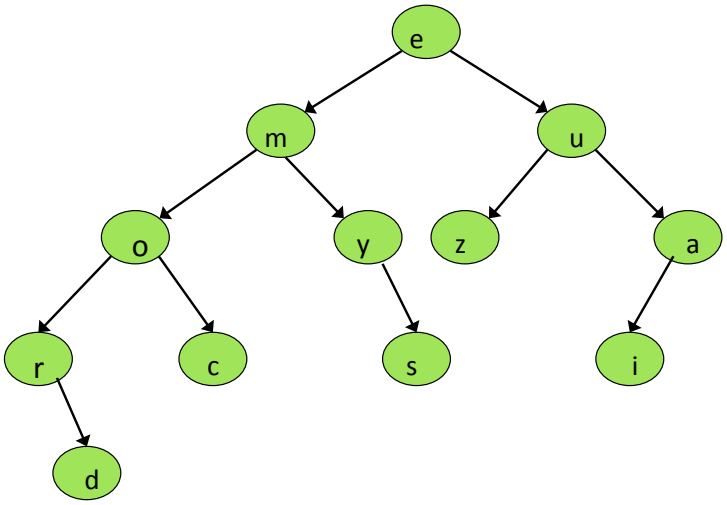
MC5 (2.5pts)

Fill in the blanks so that the following sentence is true: If you have a complete tree with 17 nodes, the height (h) of the tree is _____ and there are _____ nodes on level h.

(a) First blank is 4, second is 1.
(b) First blank is 5, second is 2.
(c) First blank is 8, second is 2.
(d) First blank is 8, second is 9.
(e) None of the above options makes the sentence true.

MC6 (2.5pts)

Consider a level order traversal of the following binary tree. Which node is the last node *enqueued* before the node containing `y` is *dequeued*?



(a) The node containing `c`.

(b) The node containing `o`.

(c) The node containing `m`.

(d) The node containing `s`.

(e) None of these is the correct answer.

MC7 (2.5pts)

How many data structures in this list can used to implement a *Dictionary* so that all of its functions have *strictly better than* $O(n)$ running time (worst case)?

linked list stack queue binary search tree AVL tree

(a) 1

(b) 2

(c) 3

(d) 4

(e) 5

MC8 (2.5pts)

Suppose that we have numbers between 1 and 1000 in a binary search tree and we want to search for the number 363. Which of the following sequences can not be the sequence of nodes visited in the search?

(a) 2, 252, 401, 398, 330, 344, 397, 363

(b) 924, 220, 911, 244, 898, 258, 362, 363

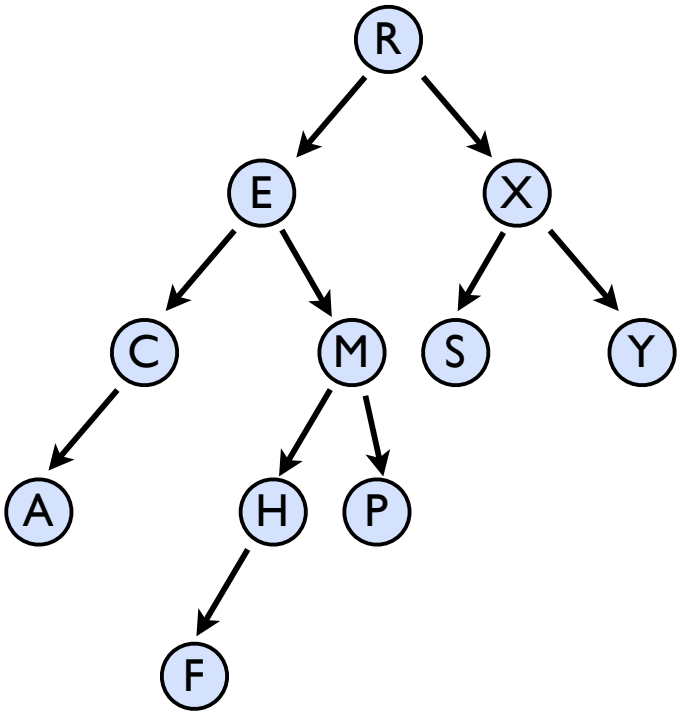
(c) 2, 399, 387, 219, 266, 382, 381, 278, 363

(d) 925, 202, 911, 240, 912, 245, 363

(e) 935, 278, 347, 621, 399, 392, 358, 363

MC9 (2.5pts)

Consider the nearly balanced Binary Search Tree in the figure below.



Perform the appropriate rotation about `R` to restore the height balance of the tree. What is the level order traversal of the tree after it has been balanced?

(a) `R E X C M S Y A H P F`

(b) `R M X E P S Y C H A F`

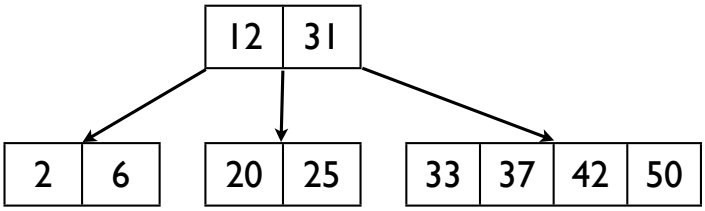
(c) `M E R C H P X A F S Y`

(d) `E C M A H P F R X S Y`

(e) None of these is the correct level order traversal.

MC10 (2.5pts)

Consider the BTree in the figure below.



How many disk seeks are required during the execution of `Find(42)`? Please assume that none of the data exists in memory when the function call is made.

(a) 1
(b) 2
(c) 4
(d) 5
(e) The number of disk seeks cannot be determined because we do not know the order of the tree.

MC11 (2.5pts)

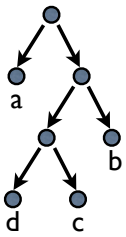
In general, for an order m BTree containing n keys, the number of disk seeks is _____.

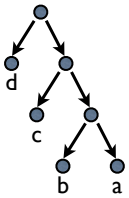
(a) $O(1)$
(b) $O(\log n)$
(c) $O(n)$
(d) $O(n \log n)$
(e) None of these is accurate because they ignore the order of the tree.

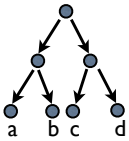
MC12 (2.5pts)

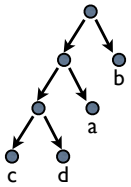
Which of the following trees is a Huffman Tree for the following string of characters?

`b a b a c a d a c a b a b`

(a) 

(b) 

(c) 

(d) 

(e) None of these.

MC13 (2.5pts)

Suppose we would like to build a dictionary that maps a set of student names (short strings) to a study group identifier. Which of the following would work as a key function for our dictionary? Hint: the ordering of the students in the original set should not matter.

(a) Concatenate the names.

(b) Sort the students' names and then sum the values of the characters in their names.

(c) Sort each name by character, then form a concatenation of all the sorted names.

(d) Sort and then concatenate the first letters of the students' names.

(e) None of the above is correct.

MC14 (2.5pts)

Suppose a hash table has size 10, and that the search keys are strings consisting of 3 lower case letters. We want to hash 7 unknown values from this keyspace. In the hash function, when we refer to the alphabet positions of the letters, we mean: "a"= 1, "b"= 2, ..., "z"= 26.

$$h(k) = (\texttt{product of the alphabet positions of } k'\texttt{s letters})^4 \ \ \%10$$

Which of these ideal hash function characteristics are violated by this hash function?

(i) A good hash function distributes the keys uniformly over the array.

(ii) A good hash function is deterministic.

(iii) A good hash function is computed in constant time.

(a) Only (i) is violated.

(b) Only (ii) is violated.

(c) Only (iii) is violated.

(d) At least two of (i), (ii) and (iii) are violated.

(e) None of these characteristics are violated–our hash function is a good one!

2. **[Efficiency – 20 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 (unless otherwise stated).

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

(MC 15) ___ The slower of `Enqueue` or `Dequeue` for a `Queue` implemented with an array.

(MC 16) ___ Find the maximum key in a Binary Tree (not necessarily BST).

(MC 17) ___ Find the *In Order Predecessor* of a given key in a Binary Tree (if it exists).

(MC 18) ___ Find the *In Order Predecessor* of a given key in an AVL Tree (if it exists).

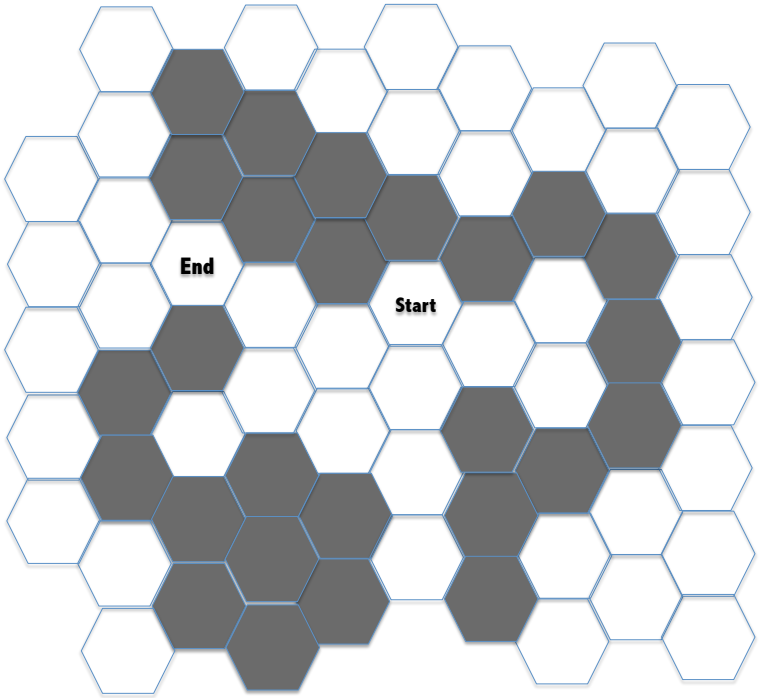
(MC 19) ___ Perform `rightLeftRotate` around a given node in an AVL Tree.

(MC 20) ___ Determine if a given Binary Search Tree is height balanced.

(MC 21) ___ Build a binary search tree (not AVL) with keys that are the numbers between 0 and n, in that order, by repeated insertions into the tree.

(MC 22) ___ Remove the right subtree from the root of an AVL tree, and restore the height balance of the structure.

3. **[MP4ish – 10 points].**



(a) (4 points) Suppose we execute slight modifications of MP4 functions `BFSfillSolid`, and `DFSfillSolid` on the hexagonal grid above, beginning at the "Start" cell, and changing white pixels to red. If the functions are executed simultaneously, which function changes the cell marked "End" to red, first? Assume that we start the algorithm by adding the "Start" cell to the ordering structure, and that we add the six neighboring cells to the structure clockwise beginning on the top. As a reminder, the fill should change the color when a cell is *removed* from the ordering structure.

`BFSfillSolid` `DFSfillSolid`

(b) (2 points) What ordering structure did you use in your answer to part (a)?

Queue Stack

(c) (4 points) Suppose we want to fill some part of a arbitrary grid containing n cells. What is the worst-case running time of `BFSfillSolid` if we start from an arbitrary location?

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$

4. **[Quadtrees – 20 points].**

For this question, consider the following partial class definition for the Quadtree class, which uses a quadtree to represent a square PNG image as in MP5.

```
class Quadtree
{
public:
   // ctors and dtor and all of the public methods from MP5, including:

   void buildTree(PNG const & source, int resolution);
   RGBApixel getPixel(int x, int y) const;
   PNG decompress() const;
   void prune(int tolerance);
   ...
   // a NEW function for you to implement
   void prunish(int tolerance, double percent);

private:
   class 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;  // pointer to root of quadtree, NULL if tree is empty
  int resolution; // init to be the resolution of the quadtree NEW
  int distance(RGBApixel const & a, RGBApixel const & b); // returns sq dist between color
  void clear(QuadtreeNode * & cRoot); // free memory and set cRoot to null
  // a couple of private helpers are omitted here.
};
```

You may assume that the quadtree is perfect and that it has been built from an image that has size $2^k \times 2^k$. As in MP5, the element field of each leaf of the quadtree stores the color of a square block of the underlying PNG image; for this question, you may assume, if you like, that each non-leaf node contains the component-wise average of the colors of its children. 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 each leaf of the Quadtree is NULL.

(a) (4 points) Write a *private* member function `int Quadtree::tallyNear(RGBApixel const & target, QuadtreeNode const * curNode, int tolerance)`, which calculates the number of leaves in the tree rooted at `curNode` with element less than or equal to `tolerance` distance from `target`. You may assume that you are working on a perfect (unpruned), non-empty Quadtree. Write the method as it would appear in the quadtree.cpp file for the `Quadtree` class. We have included a skeleton for your code below–just fill in the blanks to complete it.

```
int Quadtree::tallyNear(RGBApixel const & target,
           QuadtreeNode const * curNode, int tolerance) __________ {

   // function not called with curNode == NULL;
   if (curNode->__________ == __________) {  // check for leaf

      RGBApixel current = curNode->element;

      if (distance(current, target) __________ tolerance)
         return __________;
      else return 0;

   }

   // otherwise...recurse!
   int devTotal = ______________________


   return __________;
}
```

(b) (6 points) Our next task is to write a private member function declared as `void Quadtree::prunish(QuadtreeNode * curNode, int tolerance, int res, double percent)` whose functionality is very similar to the `prune` function you wrote for MP5. Rather than prune a subtree if ALL leaves fall within a tolerance of the current node's pixel value, `prunish` will prune if at least `percent` of them do. Parameter `res` is intended to represent the number of pixels on one side of the square represented by the subtree rooted at `curNode`. All the constraints on pruning from the `prune` function apply here, as well. That is, you should prune as high up in the tree as you can, and once a subtree is pruned, its ancestors should not be re-evaluated for pruning. As before, we've given you most of the code below. Just fill in the blanks on the next page.

```
void Quadtree::prunish(QuadtreeNode * curNode, int tolerance,
                       int res, double percent) {

   if (curNode == NULL)
      return;

   // count the number of leaves more than tolerance distance from curNode

   int nearNodes = ________________; //(1 points)

   double percentNear =________________; //(1 points)

   // prune conditions
   if ( ___________________________) { //(2 points)

      clear(curNode->neChild);
      clear(curNode->nwChild);
      clear(curNode->seChild);
      clear(curNode->swChild);

      return;
   }

   // can't prune here :( so recurse!

   ________________________   //(2 points)

   ________________________

   ________________________

   ________________________

   return;
}
```

(c) (2 points) Next, write the public member function `void Quadtree::prunish(int tolerance, double percent)` that prunes from the `Quadtree` any subtree with more than `percent` leaves within `tolerance` color distance of the subtree's root.

```
void Quadtree::prunish(int tolerance, double percent) {

   ____________________________

   return;
}
```

(d) In this part of the problem we will derive an expression for the *maximum* number of nodes in a `Quadtree` of height h, and prove that our solution is correct. Let $N(h)$ denote the maximum number of nodes in a `Quadtree` of height h.

i. (3 points) Give a recurrence for $N(h)$. (Don't forget appropriate base case(s).)

We solved the recurrence and found a closed form solution for $N(h)$ to be:

$$N(h) = \frac{4^{h+1} - 1}{3}, h \geq -1$$

ii. (3 points) Prove that our solution to your recurrence from part (i) is correct by induction:
Consider a maximally sized `Quadtree` of arbitrary height h.

- If $h = -1$ then the expression above gives: _____ which is the maximum number of nodes in a tree of height -1 (briefly explain).

- otherwise, if $h > -1$ then by an inductive hypothesis that says:

---

we have $N($_____$)$ = _____ nodes.

so that $N(h)$ = _____ = _____,
which was what we wanted to prove.

iii. (2 points) Use your result from part (d) to give a lower bound for the height of a quad tree containing n nodes.

5. **[Stacks and Queues – 15 points].**

In this problem you will write a function `reverseOdd` that takes a queue of integers as a parameter, and that modifies that queue, reversing the order of the odd integers in the queue while leaving the even integers in place.

For example given this queue (back to front):

```
< 14 13 17 8 4 10 11 4 15 18 19 >
```

calling the function would change it to:

```
< 14 19 15 8 4 10 11 4 17 18 13 >
```

We have given you the `Stack` and `Queue` interfaces below. You may also assume the existence of a helper function `isOdd()` that returns true for odd integers and false for even integers.

```
template <T>
class Stack
{
public:
   // ctors and dtor and all of the public methods, including:

   T pop();
   void push(T data);
   bool isEmpty();
private:
...
}

template <T>
class Queue
{
public:
   // ctors and dtor and all of the public methods, including:

   T dequeue();
   void enqueue(T data);
   bool isEmpty();
private:
...
}
```

(a) (3 points) Write a function that returns the size of a queue.

```
int Qsize(queue q)  {




}
```

(b) (8 points) Write the function `reverseOdd` as described above. We have provided you with a skeleton. Just fill in the blanks!

```
void reverseOdd(queue & input) {

   stack s;

   int n = Qsize(input);

   int counter = 0;

   while (__________) {
      int temp = input.dequeue();

      if ____________

            _________________;

      ___________;
      counter++;
   }

   counter = 0;

   while (__________) {
      int temp = input.dequeue();

      if ____________

            _________________;

      ___________;
      counter++;
   }
}
```

(c) (2 points) Suppose the queue contains $O(n)$ even integers, and $O(\log n)$ odd integers. What is the worst case total running time of the algorithm? Give the tightest bound you can.

(d) (2 points) Suppose the queue contains $O(n)$ even integers, and $O(\log n)$ odd integers. How much memory does the algorithm use? Give the tightest bound you can.

scratch paper