Name: ________________________________ Userid: __________

CS 2150 (fall 2010) Midterm 2

You MUST write your name and e-mail ID on EACH page and bubble in your userid at the bottom of EACH page, including this page.

If you are still writing when "pens down" is called, your exam will be ripped up and not graded – even if you are still writing to fill in the bubble forms. So please do that first. Sorry to have to be strict on this.

Other than bubbling in your userid at the bottom, please do not write in the footer section of each page.

There are 10 pages to this exam – once the exam starts, please make sure you have all 10 pages.

Questions are worth varying points depending on the question length. The three point questions on this exam should not take more than a line or two to answer – **your answer should not exceed about 20 words**.

This cover page is worth 5 points, and each of the remaining pages are worth 12 points each. Thus, the exam is worth 113 points – which is a prime number, and would thus make an excellent hash table size. There is 1 hour and 35 minutes (95 minutes) to take the exam, which means you should spend about 0.8407079646017699 minutes (50.44247787610619 seconds) per question point. Not coincidentally, those numbers are to 16 digits of accuracy, which is the range of a IEEE 754 double precision floating point value.

If you do not bubble in a page, you will not receive credit for that page!

This exam is CLOSED text book, closed-notes, **closed-calculator**, closed-cell phone, closed-computer, closed-neighbor, etc. Questions are worth different amounts, so be sure to look over all the questions and plan your time accordingly. Please sign the honor pledge below.

____

____

____

____

First snow, then silence.
This thousand dollar screen dies
So beautifully.

C++

1. [3 points] The copy constructor and the `operator=()` method are similar, but they are invoked at different times. When does C++ invoke the copy constructor, and when does C++ invoke the `operator=()` method?

2. [3 points] Give a convincing example of a class where we would *want* to use implicit construction – we aren't looking for C++ code here, but an example of a class type where this would be desired.

3. [3 points] Given a class Foo with a single `ListNode* list` field, what is wrong with this constructor?

```
Foo() {
   ListNode *list = new ListNode();
}
```

4. [3 points] Given a class Foo with a single `ListNode* list` field, what is wrong with this constructor?

```
Foo() {
   ListNode temp;
   list = &temp;
}
```

(the bubble footer is automatically inserted into this space)

Trees

5. [4 points] What are the properties of red-black trees?

6. [4 points] Briefly describe both what a splay operation does, and how it works

7. [4 points] Write the complete algorithm for binary tree insert. This can be in English or pseudo-code.

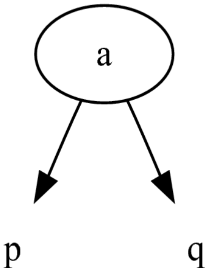
(the bubble footer is automatically inserted into this space)

Trees

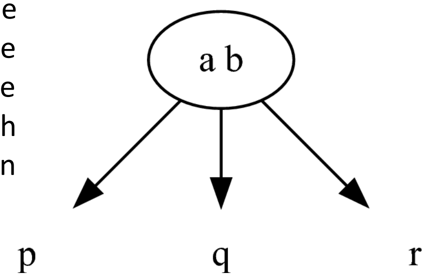
To help you manage your time, you get 12 free points for reading this page of text.

A 2-3 tree is a ordered tree, in a manner similar to binary search trees. However, in a 2-3 tree, a given node can have two or three children; a leaf node has no children. 2-3 tree nodes are not allowed to have only one child. Furthermore, a node with two children holds one data value, and a node with three children holds two data values. We'll go through each of these in turn.

A node with two children holds a single data value. This is shown in the first diagram to the right. Such a node has a ordering property just like a binary search tree – the values in the left sub-tree (*p* in the diagram to the right) are all less than the value in the node (*a*), which is less than the values in the right sub-tree (*q*). This can be expressed mathematically as {*p*} < *a* < {*q*}, where {p} indicates all the values in the given subtree.



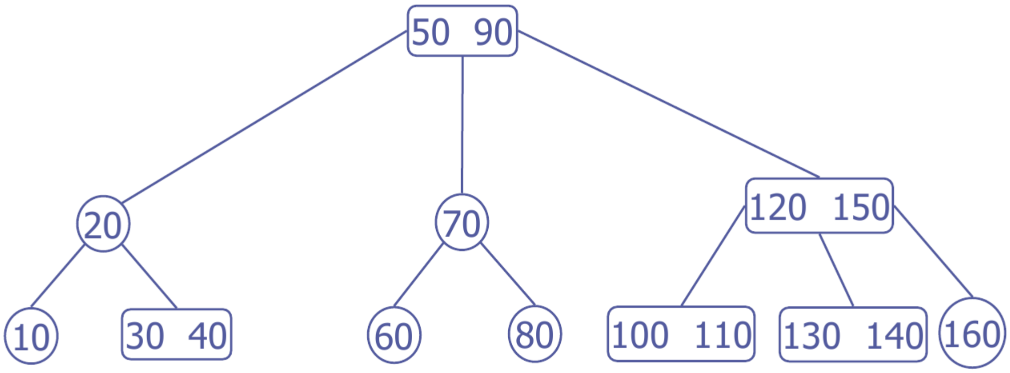
A node with three children has an extra sub-tree – we'll call it the 'middle' sub-tree. This is shown as tree *q* in the second diagram to the right; *p* is still the left sub-tree, and *r* is the right sub-tree. The two values in the node are *a* and *b*. Those values are ordered such that *a* < *b*. Similar to the two children example above, this type of node also has an ordering property: all the values in *p* are less than *a*, which is less than the values in *q*, which is less than *b*, which is less than the values in *r*. This can be expressed mathematically as {*p*} < *a* < {*q*} < *b* < {*r*}.



A 2-3 tree node is not allowed to have only one child.

A leaf node has zero children, and can have one or two data values.

The diagram below illustrates a 2-3 tree. The questions on the next page are all about 2-3 trees.



We learned about binary search trees (BSTs), which can be expanded to include balanced trees, such as AVL or red-black. However, BSTs are not inherently balanced by themselves. Similarly, 2-3 trees can be expanded to include balanced trees, but 2-3 trees are not necessarily inherently balanced by themselves. The questions below will consider both *regular* 2-3 trees and *balanced* 2-3 trees.

Lastly, we will assume for this question that duplicate values are not allowed in a 2-3 tree, similar to BSTs.

Trees

8. [3 points] Consider a regular 2-3 tree containing *n* elements. What is the maximum height of such a tree?

9. [3 points] Consider a 2-3 tree that consists of a single leaf node with two elements. How would you handle an insert of a value into this tree? What does the resulting tree look like?

10. [3 points] What would the running time be for both regular 2-3 trees and balanced 2-3 trees? Briefly, why?

11. [3 points] Briefly, what are the pros and cons of balanced 2-3 trees compared to balanced binary trees?

(the bubble footer is automatically inserted into this space)

Hashing

To help you manage your time, you get 6 free points for reading this half-page of text.

We have seen four collision resolution strategies: separate chaining, and three probing strategies (linear, quadratic, and double hashing). We will now consider a new strategy: dynamic perfect hashing.

Dynamic perfect hashing is similar to separate chaining with a secondary hash table as each bucket data structure. If there are *k* elements in a given bucket, then the secondary hash table will be of size k^2. Furthermore, the hash function of the secondary hash table will be chosen to ensure that the secondary hash table is collision-free. One can create a hash function with certain variables, and when the new hash function is chosen, the values of the variables are set to ensure a collision-free secondary hash table. The method for creating such a hash function, or for modifying those variables, is beyond the scope of this question – we'll just assume that this can be reliably done in a reasonable (i.e. constant) time.

Upon an insert of key *x* with a empty bucket, the secondary hash table is created, and the key inserted. When a bucket already has a secondary hash table with one or more values in it, then the secondary hash table is expanded to the appropriate size (k^2, where *k* is the number of elements that will be in the secondary hash table upon the completion of this insert), the new hash function chosen, all the elements all rehashed, and finally the key *x* inserted into the secondary hash table.

12. [3 points] Without dynamic hashing, and using another collision resolution method (a probing strategy or separate chaining with linked lists), what is the big-theta running time of a hash table? Briefly explain why.

13. [3 points] Using dynamic hashing as the collision resolution strategy, what is the big-theta running time of a hash table? Briefly explain why.

(the bubble footer is automatically inserted into this space)

Hashing

14. [3 points] If we can assume that the hash keys are evenly distributed, does this change your answer to the previous question? If so, give the new running time, and briefly explain why.

15. [3 points] What is the worst-case amount of space (memory) needed for such a hash table? Similar to the big-theta analyses of running time, you can leave this as a function of *n* while ignoring constants. Briefly explain why.

16. [3 points] If we can assume that the hash keys are evenly distributed, does this change your answer to the previous question? If so, give the new amount of space needed, and briefly explain why.

17. [3 points] Briefly, what are the pros and cons of this type of collision resolution strategy?

(the bubble footer is automatically inserted into this space)

IBCM

18. [12 points] Write a complete IBCM program that will do the following: read in an integer, and print out it's additive inverse (i.e. given *x*, the program will print out -*x*). The trick is that you can **NOT** use the sub opcode, but you can use the neg (negate) opcode. Your result should be left as IBCM opcodes – we aren't interested in the hexadecimal encoding of your program.

IBCM

You will get full credit for the questions on this page, as long as you take the time to answer them honestly. For each question, please indicate how much you agree with the statement by circling one of the responses.

19. [2 points] IBCM increased my understanding of the basics of machine language (please circle one)

    Strongly agree Agree Neutral Disagree Strongly disagree

20. [2 points] IBCM increased my understanding of how computers work at a low level (please circle one)

    Strongly agree Agree Neutral Disagree Strongly disagree

21. [2 points] IBCM was easy to use, once I got the hang of programming in it (please circle one)

    Strongly agree Agree Neutral Disagree Strongly disagree

22. [2 points] I enjoyed learning IBCM (please circle one)

    Strongly agree Agree Neutral Disagree Strongly disagree

23. [2 points] Considering what was taught, IBCM was a worthwhile module to have in this course (please circle one)

    Strongly agree Agree Neutral Disagree Strongly disagree

24. [2 points] IBCM should be used in future iterations of this course (please circle one)

    Strongly agree Agree Neutral Disagree Strongly disagree

(the bubble footer is automatically inserted into this space)

x86

25. [3 points] What are the steps necessary in the x86 C calling convention caller's prologue?

26. [3 points] What are the steps necessary in the x86 C calling convention caller's epilogue?

27. [3 points] What are the steps necessary in the x86 C calling convention callee's prologue?

28. [3 points] What are the steps necessary in the x86 C calling convention callee's epilogue?

(the bubble footer is automatically inserted into this space)