CS 2150 Final Exam

Name ______________________

You MUST write your e-mail ID on **EACH** page and bubble in your userid at the bottom of this first page. And put your name on the top of this page, too.

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 form. So please do that first. Sorry to have to be strict on this!

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

There are 12 pages to this exam. Once the exam starts, please make sure you have all the pages. All pages with questions are worth 12 points (really!); pages 1, 9, and 12 are not worth any points. Thus, this entire exam is worth 108 points.

If you do not bubble in this first page properly, you will not receive credit for the exam!

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.

Stay the patient course.
Of little worth is your ire.
The network is down.

(the bubble footer is automatically inserted into this space)

Page 2: Numbers

1. [12 points] Intel has designed a new computer processor! While it has many unique and different features, the important one (for this problem, at least) is how it handles numbers – any integer value is written to (and read from) memory in one Endian format (well say little, but it does not matter), but any floating point number is written to (and read from) memory in a the other Endian format (well say big). An `int` value is 32 bits and is encoded using two's complement; a `float` value is 32 bits and is encoded using the IEEE 754 standard (what we saw in class). Keeping this in mind, consider the following C++ code:

```
union foo {
  int x;
  float f;
};
int main() {
  foo bar;
  bar.x = 14913;
  cout << bar.f << endl;
  return 0;
}
```

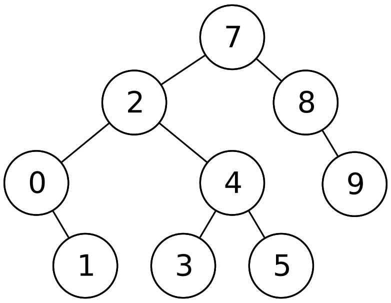
What gets printed? Assume that the code has the necessary namespace and #includes. Be sure to show all your work!

If you don't know how to answer this question, you can get partial credit by encoding 14,913 as a two's complement integer, and decoding the big-Endian value 0x41520000 (which is not what 14,913 encoded is, by the way) as a floating point number.

To save you time on this test, $14913 = 1 + 64 + 512 + 2048 + 4096 + 8192$. And $2^{10} = 1024$.

Page 3: Trees and Hashes

2. [3 points] Insert 6 into the AVL tree below. Show the resulting data structure.



3. [3 points] What is a vector's amortized running time for insert? Briefly, why?

4. [3 points] What is the running time for an insert into a hash table using a probing strategy?

5. [3 points] What is the best collision resolution protocol for hash tables? Briefly, why?

Page 4: x86

AMD, always competing with Intel, has made a new 32-bit x86-compatible processor, but it has four additional registers: eex, efx, egx, and ehx. Rather than have these registers be backed up on the stack like the other six general-purpose registers, they will be used to pass parameters. In particular, the first four parameters passed to a subroutine will be in those registers; any additional parameters are pushed onto the stack. Your task for this question is to design the new calling convention that allows this new type of parameter passing. It is to be based upon the x86 calling convention that we learned in class, but with the modification described above. There are problems – which we are not going to tell you – that will arise with such a parameter passing routine, and you must solve those problems via your calling convention. As a hint to try to figure out what those problems are, think about if your new calling convention can support an indefinite amount of recursion.

To make it easier to write the answer (and to grade it!), we have split this problem into four parts. You must specify the *complete* calling convention, not just the changes.

6. [3 points] What are the new calling convention steps for the caller's prologue?

7. [3 points] What are the new calling convention steps for the caller's epilogue?

8. [3 points] What are the new calling convention steps for the callee's prologue?

9. [3 points] What are the new calling convention steps for the callee's epilogue?

Page 5: C++

10. [3 points] How does shared multiple inheritance differ from replicated multiple inheritance?

11. [3 points] Briefly describe how dynamic dispatch is implemented in C++ (you need to actually explain how it works, not just specify a name).

12. [3 points] What causes the covariant array problem that we discussed in lecture? That was the apples-and-oranges example.

13. [3 points] Other than syntax, what are the three primary differences between references and pointers?

Page 6: Heaps and Huffman coding

14. [3 points] In a *max*-heap, when percolating a node down, you must exchange it with only one of the children. Which one? And why?

15. [3 points] Give an example of when a problem would arise if we exchanged a percolating node in a *max*-heap with the incorrect child (a before-and-after diagram would work well here, although there are multiple ways to answer this question).

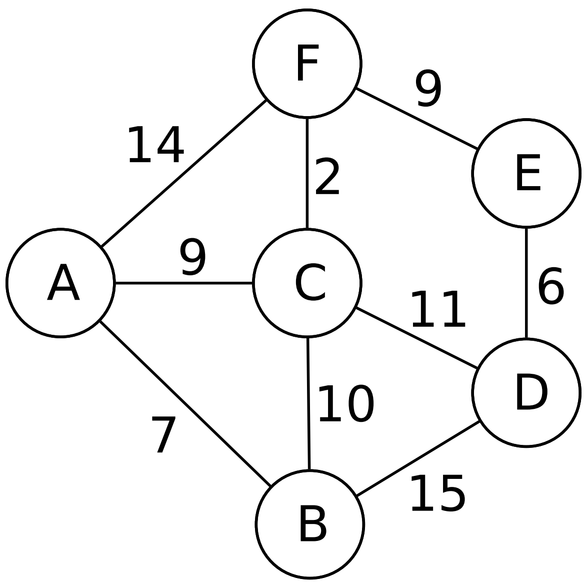
16. [3 points] What is the primary property of a prefix code? We aren't asking what it represents (we know it represents a character), but it's primary property.

17. [3 points] How does the asymptotic complexity analysis (i.e., big-Theta analysis) change in a heap if it is implemented via an array versus using dynamic memory with HeapNodes (or equivalent)?

Page 7: Graphs

18. [3 points] We have seen two ways of representing graphs: adjacency matrices and adjacency graphs. Give one advantage of each, and one disadvantage of each.

19. [9 points] Perform Dijkstra's shortest path algorithm on the following graph. You must start at node A. We have included a table for you to use; the results in this table are what will be graded. The table has enough space for you to make 'corrections' as the algorithm proceeds.



Vertex	Known?	Distance	Path
A			
B			
C			
D			
E			
F			

Page 8: Hopscotch hashing

Hopscotch hashing is a probing collision resolution protocol somewhat similar to linear probing. In hopscotch hashing, each bucket has a *neighborhood* – a small (logarithmic to the size of the hash table) set of nearby consecutive buckets. If a collision occurs, then the other buckets in the neighborhood are searched (via linear probing) for the key; a key cannot be outside of its original bucket's neighborhood. Because the neighborhood size is $\log n$ (where n is the hash table size), any find will run in logarithmic time.

20. [4 points] Insert is easy if there is a spot in the neighborhood. But what if there isn't? How would you handle an insert in this situation (when the entire neighborhood is full)? Doing a re-hash on the entire table is not a valid answer for this question. What is the running time of your insert?

21. [4 points] Briefly describe how the usage of cache for hopscotch hashing compares to both linear probing and quadratic probing (meaning, compare cache hits versus cache misses for the different probing strategies).

22. [4 points] Compare hopscotch hashing to double hashing: give one advantage of each and one disadvantage of each.

Page 9: Skew Heaps

(You don't get any free points for this page, but you also have 3 hours for this final. . .)

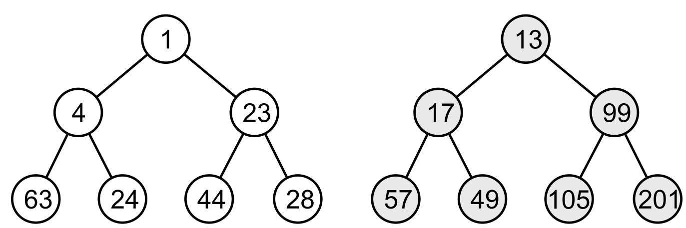
One operation that is missing from a binary heap is `merge()`, which takes two binary heaps and merges them into a single binary heap. A naïve implementation would take all the elements from the smaller heap, and insert them one-by-one into the larger heap. In the worst case, both heaps are the same size (both size n), and this will take $\Theta(n \log n)$ time.

A *skew heap* is a data structure that allows for a more efficient merge. A skew heap heap is similar to a regular heap in many respects: it is based on a binary tree, and has the heap ordering property (we'll assume a min-heap). However, the structure property of binary heaps is *not* enforced in skew heaps.

A skew heap has one main operation, called a *skew heap merge* that is used in every other operation, similar to how the `splay()` operation is used in every other operation in a splay tree. To insert an element e into a heap H, a heap of one node is created for the element e to be inserted, and then that heap is skew heap merged with the heap H. A removal just removes the root node (since it's `deleteMin()`), which causes there to be two resulting heaps (the two former subtrees of the root node); these are then skew heap merged together. The merge operation will use the skew heap merge operation directly.

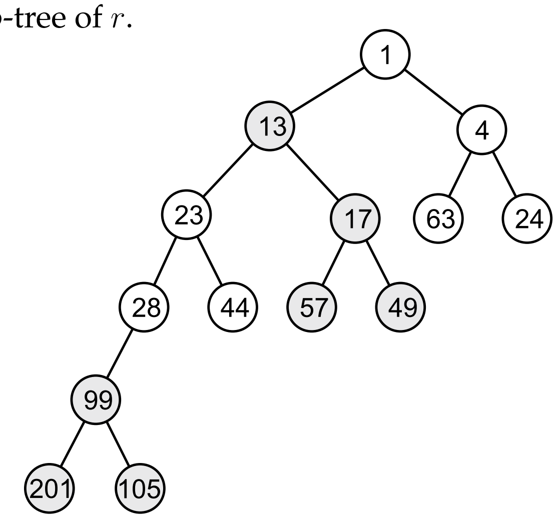
A skew heap merge operates as follows:

- Compare the roots of the two heaps; let p be the heap with the smaller root, and q be the other heap. Let r be the name of the resulting new heap that is being created by this merge.
- Let the root of r be the root of p (the smaller root), and let r's right sub-tree be p's left sub-tree.
- Now, compute r's left sub-tree by recursively merging p's right sub-tree with q.



Consider the merging of the two skew heaps shown to the right. The resultant heap – after the entire merge – is shown below.

On the first iteration, the left heap is p (since it has the smaller root), and the right heap is q (since it has the larger root). We can start to construct r, our resultant heap, at this point. We take the root of p (the node 1), and make p's left sub-tree (the sub-tree rooted at 4) be the right sub-tree of r.



So far we have only created *part* of our final heap – the root node of 1, and it's right sub-tree (containing the elements 4, 63, an 24). This can be seen in the figure to the right. We then take the remaining elements to merge – the right sub-tree of p (containing elements 23, 44, and 28) and all of heap q, and skew heap merge them together to form the left sub-tree of our resultant heap.

On the second iteration, p is the right-hand heap shown above (rooted at 13) – since it has the smaller root – and q is the right sub-tree of the left-hand heap shown above (rooted at 23) – since it has the larger root. Thus, node 13 becomes the left sub-tree of r (shown to the right), and the algorithm continues as above.

Page 10: Skew Heaps, page 2

23. [4 points] As presented, what is the running time of the skew heap merge operation? Briefly, why?

24. [4 points] In any min-heap, a lower key will have a greater chance of having more nodes beneath it (since it's lower in value, it has more values greater than it, and those values would be below it in the min-heap). The skew merge operation puts lower values at a higher level in the tree during the skew merge. How does this fact affect the performance?

25. [4 points] An advantage of the skew heap is, of course, the ability to perform a merge operation. Give two *disadvantages* of skew heaps compared to regular binary heaps (i.e., explain two reasons why binary heaps are better).

Page 11: Demographics

We meant to ask these in an end-of-the-semester survey, but we did not get to it in time. So we'll put it here for some extra points on the exam!

26. [0 points] Did you put your name and userid at the top of this page? You need to in order to get the points on this page.

27. [2 points] What is your major or minor (if you have not declared, then your intended major or minor)? Please circle one.

    - BS CS
    - BA CS
    - BS CpE
    - CS minor
    - Neither majoring nor minoring in computing
    - Other (please explain): ____________________

28. [2 points] What CS 1 class did you take? Please circle one.

    - CS 1110 (a.k.a. CS 101)
    - CS 1111 (a.k.a. CS 101-E)
    - CS 1112 (a.k.a. CS 101-X)
    - CS 1120 (a.k.a. CS 150)
    - AP credit
    - Transfer credit
    - Placed out of it via the CS 101/1110 placement exam
    - Other (please explain): ____________________

29. [2 points] If you took your CS 1 class in college (i.e. CS 101/1110, 101-E/1111, 101-X/1112, 150/1120, or a transfer class), in what semester did you take it?

30. [2 points] What CS 2 class did you take? Please circle one.

    - CS 2110 (a.k.a. 201)
    - CS 2220 (a.k.a. 205)
    - AP credit
    - Transfer credit
    - Other (please explain): ____________________

31. [2 points] If you took your CS 2 class in college (i.e. CS 201/2110, 205/2220, or a transfer class), in what semester did you take it?

32. [2 points] Did you attend the final exam review session? You'll get full credit for this question, as long as you answer it honestly (we know most of the people that were there, but not all).

Page 12: When Dijkstra's shortest path algorithm fails...

(from `http://xkcd.com/461/`)