CS 2150 Final Exam

Name

You MUST write your e-mail ID on **EACH** 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 – sorry to have to be strict on this!

There are 10 pages to this exam. Once the exam starts, please make sure you have all the pages. Questions are worth different amounts of points.

Answers for the short-answer questions should not exceed about 20 words; if your answer is too long (say, more than 30 words), you will get a zero for that question!

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.

The Tao that is seen
Is not the true Tao,
until You bring fresh toner.

Page 2: Numbers

1. [3 points] Convert the following 8-bit two's-complement number into decimal: 11110000_2

2. [3 points] Convert the following decimal number into 8-bit two's-complement binary: -5

3. [3 points] Consider an IEEE 754-like floating point notation that uses 1 sign bit, 4 exponent bits, and 5 mantissa bits. Encode 4.5 in this notation.

4. [3 points] Consider the same notation from the previous question. What is the smallest positive non-zero value that can be represented? You may leave this as a base-10 expression.

Page 3: True / False

5. [20 points] For each proposition below, circle true if it is always true, and false otherwise.

True	False	Amortized runtime is the same as expected runtime
True	False	The following x86 instruction is valid: mov [rax], [4*rbx]
True	False	The following x86 instruction is valid: mov rax, [2*rsi+rdx+4]
True	False	Big-Oh always refers to the worst case runtime
True	False	Topological sort can be used to detect a cycle in a graph
True	False	Dijkstra's algorithm can be used on a graph with edges of equal weight
True	False	A high load factor usually speeds up operations by minimizing collisions in a hash table, but is a less efficient use of memory
True	False	x86-64 assembly considers references and pointers to be the same
True	False	All valid c programs are also valid c++ programs
True	False	RBX is the traditional return register for the callee
True	False	When printing out an expression tree in prefix order, parentheses are not necessary to indicate order of operations
True	False	AVL Trees, Red-Black Trees, and Heaps are all just particular types of binary search trees
True	False	Dynamic dispatch determines which member function to call using the compile-time type of the object
True	False	In replicated (non-virtual) multiple inheritance, two parent classes share the same instance of a class as a parent
True	False	A C-style string is a pointer to a char
True	False	Stacks and queues are abstract data types
True	False	5 bits are necessary to uniquely represent all 26 of the lowercase letters of the english alphabet
True	False	The runtime for insertion into a linked list is in the set $O(n^3)$
True	False	Variables of type *int*** stores twice as many bits as one of type *int**
True	False	When a recursive function calls itself recursively too many times, this will result in a stack overflow.

Page 4: IBCM / Assembly

6. [4 points] Here's a cool, recursive C++ function with a partial translation to x86 code. Fill in the missing instructions—you can assume that you have access to the "product" function you wrote in pre-lab 8.

```
int f(int x) {
    if (x <= 1) {
     return 1;
   } else {
     return 2*f(x-1) + f(x-2);
   }
}
```

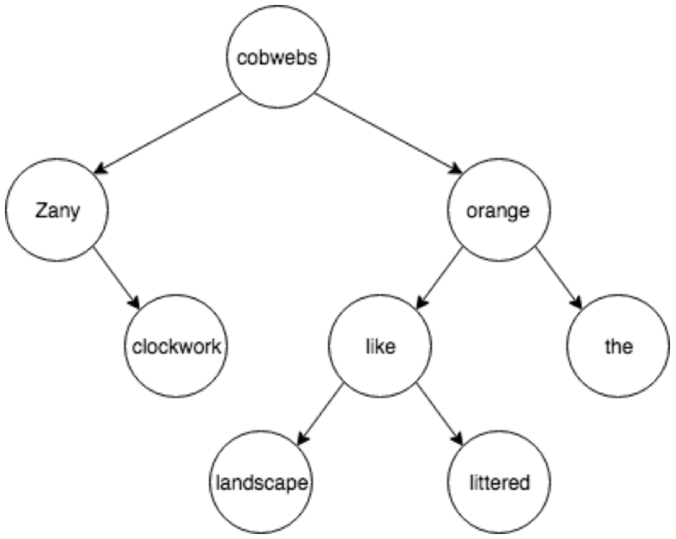
```
f:
  push rbx


  ----------
  cmp   rdi, 1
  jle   done
  dec   rdi


  ----------
  call  f
  mov   rdi, rax


  ----------
  call  product
  mov   rbx, rax
  pop   rdi


  ----------
  call  f
  add   rax, rbx
done:
  pop   rbx
  ret
```

7. [6 points] Here's another cool, recursive C++ function. Implement the function in x86. You may not use more 13 lines of code for this (It can be done in as few as 9 lines, including labels)

```
int multiply3(int n) {
    if (n == 1) {
     return 3;
   } else {
     return multiply3(n-1) + 3;
   }
}
```

Page 5: AVL Trees

8. [6 points] Draw the resultant balanced AVL tree after inserting the term **misty**. *Note that for this question, any capital letter sorts before any lowercase letter, hence Zany to the left of cobwebs.*



9. [4 points] What is the minimum and maximum number of nodes in an AVL tree of height 5? (assume the height of a tree with a single node is 0).

10. [4 points] Given AVL tree's balance method, fill in the parameters to the rotation calls. The *rotateLeft* and *rotateRight* methods expect a pointer to the AVLNode to be rotated.

```
void AVLTree::balance(AVLNode *&n) {
  //NOTE: balanceFactor returns height of right - height of left
  if (balanceFactor(n) < -1) {
    if (balanceFactor(n->left) > 0) {


      rotateLeft(____________________);
    }


    rotateRight(____________________);
  } else if (balanceFactor(n) > 1) {
    if (balanceFactor(n->right) < 0) {


      rotateRight(____________________);
    }


    rotateLeft(____________________);
  }
}
```

Page 6: Hash Tables

11. [5 points] Insert the following items (here the key and the value for this hash table are the same, so we won't list both seperately) into the hash table below: 3, 7, 2, 1, 9

    Note that the table size is 5 (with array indices 0-4). The hash function is $h(x) = x * 5 + 3$ and you should resolve collisions using *linear probing*.

index	**value**
0	
1	
2	
3	
4	

12. [5 points] Do the exact same thing as in the previous question, except this time resolve collisions using *double hashing*. Your second hash function is $h_2(x) = x\%10$

index	**value**
0	
1	
2	
3	
4	

13. [3 points] Explain in one sentence why $h(x) = x * 5 + 3$ is a poor choice for a hash function. In one more sentence, explain why $h_2(x) = x\%10$ is a poor choice for a secondary hashing function.

Page 7: Heaps are awesome

14. [4 points] For each of the following priority queue operations, circle whichever of *binary min-heaps* or *AVL trees* are **asymptotically** faster, or circle "same" if neither is faster:

insert()	binary min-heap	AVL tree	same
findMin()	binary min-heap	AVL tree	same
deleteMin()	binary min-heap	AVL tree	same
deleteMax()	binary min-heap	AVL tree	same

15. [3 points] A typical binary heap implementation is more space efficient than an AVL tree. **Circle all that apply**:

The amount of space saved is:

a) proportional to the number of items stored

b) proportional to $n \log n$ where n is the number of items stored

c) approximately constant, no matter the number of items

d) usually dependent on the size of each items stored

e) usually dependent on the size of pointers

16. [3 points] Given the string "aabbbbcdddeeeee":

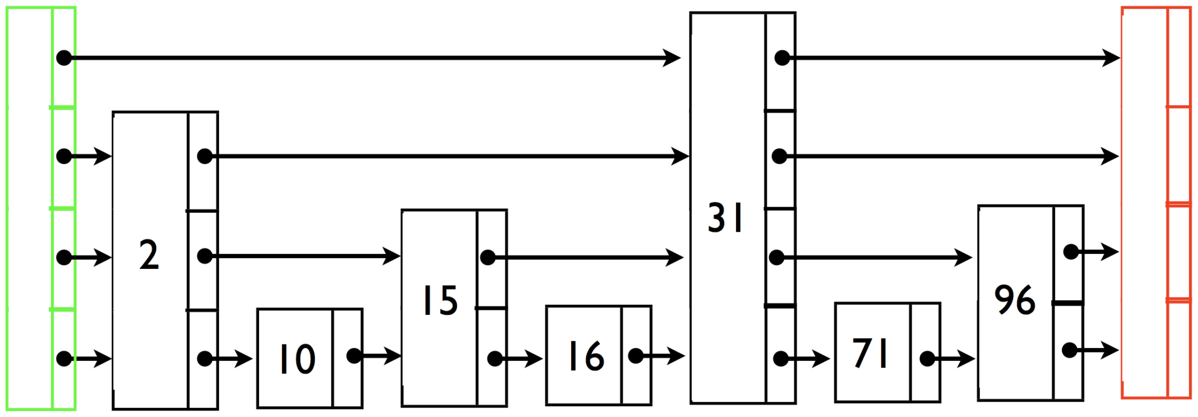
Create and draw a correct Huffman tree. What is the encoding for each character?

Page 8: Skip Lists

Skip Lists are a data structure that allow for fast search in an **ordered** sequence of elements. Skip Lists are the primary indexing structure for many databases and are also often used for parallel applications.

A *Skip List* is a hierarchy of probabilistically generated linked-lists where the lowest list is the full original list and the highest list is the sparsest. Some randomly chosen nodes in the lower lists are copied and replicated in an upper list, but not all of them.

For example, a *Skip List* filled with sorted integers might look like:



In the image above, the left-most node is the dummy head node and the right-most the dummy tail node. Note that each node contains multiple next pointers.

One way to generate a Skip-List is by starting with a sorted linked-list and then: proceeding from the first node to the last, roll a die and replicate that node in the upper list with 50 percent probability. Then, repeat that process on the newly created list; until there are only 1 or 2 items in the upper-most list.

On the next page, you will answer a few questions about *Skip Lists*.

Page 9: Questions about Skip Lists

17. [6 points] Write a function in c++ that performs *find()* on a Skip List of integers. Your method will take in the value to search for (int x) and the node you are currently searching (ListNode* curNode). Notice that this second parameter is available in case you'd like to implement your method recursively, but feel free to implement this recursively or iteratively. Your expected runtime must be strictly better than searching a regular linked-list.

```
class ListNode{
  public:
    //value stored (e.g., 31)
    int value;

    //list of next pointers.
    //Node at index 0 at same level this
    //Node at index 1 at lower level, etc.
    vector<ListNode*> nextNodes;
};
```

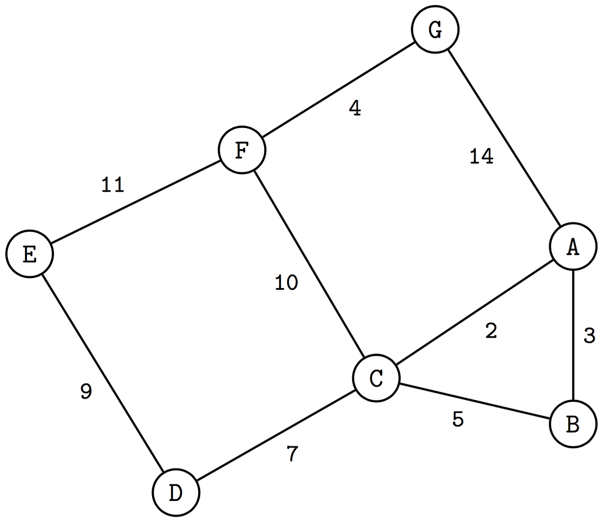
```
class SkipList {
  public:
    /* some methods omitted */
    bool find(int x) {return find(x, head);}
    bool find(int x, ListNode *curNode);
  private:
    ListNode *head;
    ListNode *tail;
};
```

```
//Returns true iff x is in this skip list. TODO: IMPLEMENT THIS METHOD
//this method initially called with dummy head as param
bool SkipList::find(int x, ListNode* curNode) {











}
```

18. [3 points] What is the **worst-case** Big-Theta time complexity of the find() function in terms of n where n is the number of elements in the lowest list? What is the **expected** Big-Theta time complexity of find()?

Page 10: Graphs

Consider the following weighted, undirected graph:



19. [3 points] What edges of the above graph would be included in a **minimum spanning tree**?

20. [3 points] When performing Dijkstra's algorithm starting with vertex F, how many times will the distance to vertex A be updated? For each of these updates, to what value will it be updated and due to what edge?

21. [3 points] Draw an example of a three-node, undirected, weighted graph which has **exactly two** distinct **minimum spanning trees**.