CSE100 Midterm 2
Winter 2017

This exam is closed book, closed notes. **Write all your answers on the answer sheet. However, all exam sheets, including scratch paper must be turned in at the end of the exam.**

By signing your name below, you are asserting that all work on this exam is yours alone, and that you will not provide any information to anyone else taking the exam. In addition, you are agreeing that you will not discuss any part of this exam with anyone who is not currently taking the exam in this room until after the exam has been returned to you. This includes posting any information about this exam on Piazza or any other social media. Discussing any aspect of this exam with anyone outside of this room constitutes a violation of the academic integrity agreement for CSE 100.

Please write the following statement in the box below and then sign your name on the line below:

"I excel with integrity"

Signature: _____

Name (please print clearly):_____

PID:_____

You have 50 minutes to complete this exam. Work to maximize points. If you don't know the answer to a problem, move on and come back later. Most importantly, stay calm and don't panic. You can do this.

DO NOT OPEN THIS EXAM UNTIL YOU ARE INSTRUCTED TO DO SO.
GOOD LUCK!

Problem 1: Hashtables and Count-Min Sketches

1.1 What is the *worst-case* time to insert and find an element into a hashtable using the following collision resolution strategies in terms of N, the number of elements in the hash table. Assume you do *not* check for duplicates and you use the most efficient implementation possible. Also assume that the insert is eventually successful without needing to rehash.

	Insert	Find
• Separate chaining	O(1)	O(N)
• Double hashing	O(N)	O(N)
• Cuckoo Hashing (find only)	——	O(1)

Hash table advantages: faster average case time to find, elements do not have to be comparable.

1.2 Give one significant advantage of a Hashtable data structure over a Red-Black Tree data structure. Give one significant advantage of a Red-Black Tree over a Hashtable.

RBT advantages: Can access elements in order, uses less space, better worst-case performance

1.3 Consider a Hashtable of size 40 that uses the hash function H(k) = k mod 40.

1.3.i. Which of the following properties of keys will, on its own, certainly lead to uneven distribution of the keys in the table? *Select ALL that apply.*

A and B only

A. The keys are all even
B. The keys are all odd
C. The keys are multiples of 3
D. The keys are all between 0 and 39
E. The keys are all greater than 40

C **1.3.ii.** Assuming that this Hash Table uses open addressing (e.g. linear probing) and the keys are evenly distributed, at what point would you want to resize the table and rehash? *Choose the SINGLE best answer.*

A. When you insert the 11th element
B. When you insert the 21st element
C. When you insert the 28th element
D. When you insert the 39th element
E. When you insert the 40th element

A **1.4** Which of the following is an advantage of double hashing over linear probing. *Choose the SINGLE best answer.*

A. Double hashing leads to less clustering/clumping of values in the table
B. Double hashing has a faster worst-case time to find an element
C. Double hashing is generally easier to implement than linear probing
D. Double hashing has no limit on the number of elements you can store in a table of a given size

A, D **1.5** Which of the following are required for a Count-Min sketch to be used effectively? *Select ALL that apply.*

A. The hash functions used in the tables must be pairwise independent (i.e. if two values hash to the same position in one table they are no more likely to hash to the same value in another table than random chance).
B. The size of your hash tables must be linearly proportional to the number of elements you will store.
C. The problem you are using it to solve must be one where where it's OK for some counts to be underestimated.
D. The problem you are using it to solve must be one where it's OK for some counts to be overestimated.

Problem 2: Balanced Trees

2.1 In class we proved that a Red-Black Tree has worst-case height O(log N), where N is the number of nodes in the tree. As part of this proof, we created a new tree from an arbitrary Red-Black tree by removing all of the red nodes and connecting the remaining black nodes to their grandparents when necessary. We proved that the leaves of this new tree are all on the same level.

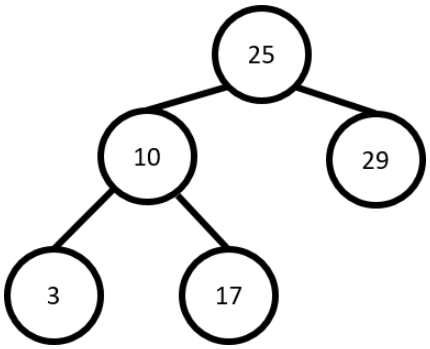
2.1.i. Which property of Red-Black Trees was essential in proving that all leaves in this new tree are on the same level? You must articulate the property (not just its number).

2.1.ii. Now, in one sentence explain how this property leads us to conclude that leaves in the new tree are on the same level.

2.1.i. The number of black nodes on every path from a node X through a null pointer must be the same

2.1.ii. The new tree has only black nodes, so all paths from the root to the leaves (which have null pointers) will have the same number of total nodes, meaning every leaf is on the same level.

2.2 Consider the AVL tree shown below.



Insert the value 5 into this tree and then answer the questions below:

2.2.i. Did this insertion cause a single rotation, a double rotation, or no rotation?

2.2.ii. What value is at the root of the tree after the insertion?

2.2.iii. What value is the left child of 25 after the insertion? If 25 has no left child write "null".

(i) Single; (ii) 10; (iii) 17

Problem 3: Huffman Coding, Bitwise operations and C++ programming

3.1 Consider an alphabet of 8 symbols: A, B, C, D, E, F, G, H.

3.1.i. Give a probability distribution over these symbols that has the highest possible entropy. There might be more than one correct answer. Any correct answer will receive full credit.

0.125 or 1/8 for all entries. It has to be distribution, not frequency (sum of all entries = 1)

3.1.ii. For the probability distribution you gave above, consider the following two encoding schemes for these 8 symbols:

Code I: The optimal variable length lossless encoding

Code II: The shortest possible fixed-length encoding.

How does the expected length of a coded symbol in Code I compare to the length of a coded symbol in Code II?

B

A. The expected length of a coded symbol in Code I is **less than** the length of a coded symbol in Code II
B. The expected length of a coded symbol in Code I is **equal to** the length of a coded symbol in Code II
C. The expected length of a coded symbol in Code I is **greater than** the length of a coded symbol in Code II
D. You cannot tell with the information given.

3.2 Complete the function `copyBits` which takes four arguments: `source, dest, pos` and `n`. `copyBits` should copy `n` bits of the *signed* char `source` starting from position `pos` from the right (using 0-based indexing) and moving to the left, into the same positions in the *signed* char `dest`, leaving the other bits of `dest` unchanged. It should modify the argument `dest` directly and it should not return anything.

For example consider the case where the binary value of `source` is `111**110**01`, and that of `dest` is `10101011`. If `pos` is 2 and `n` is 3 it will copy the bits in positions 2 through 4 from the right (the bits shown in bold in source) into the same location in dest, resulting in the following value for dest: `101**110**11`. You may assume that the value of `pos` is always in the range 0 and 7 and that `n` is always between 0 and `8-pos`. (i.e. pos and n are never out of bounds). If `n` is 0, no bits in `dest` are modified.

Complete the code below, following the comments. We have provided substeps using the above example in comments in the code to help make it more clear what each step should do.

```
// Example: source starts as  11111001, dest starts as  10101011
//          pos is 2 and n is 3
void copyBits(char source, char & dest, int pos, int n ) {
    // Create a bitmask to extract the relevant bits from source
    char mask = __[BLANK A]___;  ((1<<n)-1) << pos
    // Extract the relevant bits from source
    char bits = source & mask;     // example: bits will be 00011000
    // zero out the relevant bits in dest.
    dest = ___[BLANK B]____;  // example: dest  will be 10100011
                              dest & ~mask
    // Finish the copy
    dest = ___[BLANK C]____;  // example: dest  will be 10111011
                              dest | bits
}
```

Problem 4: Tree Search and Auto-Complete

Consider the following predict completions algorithm for a multiway trie. As in PA2, the function returns a vector containing up to `num_completions` words that start with the string `stem`, sorted from most frequent to least frequent:

```
predict_completions (stem, num_completions ):
    Initialize a Linked List, L, to store the words found
    Find the node stem_root containing the stem. If not found, return an empty vector.
    Starting at stem_root, BFS through its subtree. For each node encountered during BFS:
        If the node contains a word:
            Add the word to L, maintaining L in order of decreasing frequency
    Copy the first num_completions words from L into a new vector and return this vector
```

4.1.i. If there are N nodes in the trie and `num_completions` is considered a constant, how long does this algorithm take in terms of N, in the worst case (tightest Big-O bound)? Briefly justify your answer. *Hint*: In the worst case, the length of `stem` is very short.

4.1.ii. Explain one way to make this algorithm faster and give the new Big-O running time. To receive any credit, your improvement must achieve an asymptotic improvement (i.e. smaller tightest Big-O) and your Big-O bound must be correct based on the improvement you describe. Larger improvements might receive marginally more credit that smaller improvements.

4.1.i.
running time: O(N^2)
explanation: BFS will traverse O(N) nodes that represent words and each insertion into the sorted linked list takes O(N) -> O(N^2)

4.2.ii.
Limit the number of elements in the linked list to num_completions, running time becomes O(N).

Scratch paper

WRITE YOUR ANSWERS ON THE ANSWER SHEET. This exam is closed book, closed notes.