# CS 211 Midterm

Notes:

* You will be asked to solve the coding challenges below.
* During that lab, you will not be allowed to use internet resources or have notes of any kind.
* **FOR ALL QUESTIONS**: state the runtime complexity of your algorithm

## Challenges

1. Given the following definition of a Linked List

|  |
| --- |
| class LinkedList{  public:  int value;  LinkedList\* next;  }; |

1A. Write a function that reverses the order of the linked list.

LinkedList\* reverseLinkedList(LinkedList\* root)

1B. Write a function that merges two sorted linked lists.

LinkedList\* mergedSortedLinkedLists(LinkedList\* first, LinkedList\* second)

2. Given the following definition of a Binary Node:

|  |
| --- |
| class BinaryNode{  public:  int value = -1;  BinaryNode \*left = nullptr;  BinaryNode \*right = nullptr;  } |

2A. Write an ***iterative*** function that determines whether or not a given value exists within the supplied BST.

bool exists\_iter(BinaryNode \*root, int value)

2B. Write a ***recursive*** function that determines whether or not a given value exists within the supplied BST.

bool exists\_rec(BinaryNode \*root, int value)

2C. Write a function called bstToVector that converts and returns the supplied binary search tree into a sorted STL vector (HINT #1: a certain traversal makes this much easier. HINT #2: using a recursive helper function may be necessary).

vector<int> bstToVector(BinaryNode \*root)

2D. An AVL tree is a tree whose height of left and right subtrees differs by no more than one thereby ensuring balance. Write a function called isAvl that returns true when the supplied tree is AVL compliant.

bool isAVL(BinaryNode \*root)

2E. Write a function called getHeight that returns the height of the supplied tree.

int getHeight(BinaryNode \*root)

3. Use a hash table to write a function called isBalanced that determines whether or not the following braces are balanced: (); []; {}. E.g. "(ab[cd])" is balanced, "(ab" is not balanced, "(ab[cd)e]" is not balanced.

bool isBalanced(string text, unordered\_map<string, string> parens)

4. Determine whether or not all characters in a string are unique (HINT: use an unordered\_map)

bool has\_unique\_chars(string text)

5. Determine the most commonly occurring word in a string. (HINT: use an unordered\_map)

string most\_common\_word(string text)