Practice Midterm Exam: Solutions

Name: _____

1. ADTs (read)

- {4, 7, 3, 2, 6, 5}
- {1, 5, 9, 13, 11, 7, 3}

Name: _____

2. Big-Oh (read)

a. $O(N)$

b. $O(N^3)$

c. $O(NlogN)$

Name: _____

3. Collections and File IO

```
// solution 1 - works for > 2 people in a conversation (not required)
void chatBuddies(string filename) {
    ifstream file;
    file.open(filename);
    if (file.fail()) {
        throw "File cannot be read!";
    }

    Map<Vector<string>, int> convoCounts;
    Set<string> peopleInConvo;
    string line;
    int largest = 0;
    while (getline(file, line)) {
        if (line != "---") {
            int nameDelim = line.find(":");
            string name = line.substr(0, nameDelim);
            peopleInConvo.add(name);
        } else {
            Set<string> others = peopleInConvo;
            for (string person : peopleInConvo) {
                for (string partner : others) {
                    if (person != partner) {
                        Vector<string> pair { person, partner };
                        convoCounts[pair]++;
                        largest = max(largest, convoCounts[pair]);
                    }
                }
                others.remove(person);
            }
            peopleInConvo.clear();
        }
    }

    for (Vector<string> pair : convoCounts.keys()) {
        if (convoCounts[pair] == largest) {
            cout << pair[0] << " and " << pair[1] << endl;
        }
    }
}
```

Name: _____

```
// solution 2 - assumes <= 2 people in every conversation

// read a single conversation (until ---) and extract a string
// representing who was in the conversation (e.g. "Bert and Ernie")
string readConversation(ifstream& file);

void chatBuddies(string filename) {
    // open file
    ifstream file;
    if (!openFile(file, filename)) {
        throw "File cannot be read!";
    }

    // read file into map of pair => count (e.g. "Bert and Ernie" => 2)
    Map<string, int> counts;
    while (!file.eof()) {
        string people = readConversation(file);   // e.g. "Bert and Ernie"
        if (people != "") {
            counts[people]++;
        }
    }

    // figure out which pair has the highest count
    int largest = 0;
    for (string people : counts.keys()) {
        if (counts[people] > largest) {
            largest = counts[people];
        }
    }

    // print all pairs that have the highest count
    for (string people : counts.keys()) {
        if (counts[people] == largest) {
            cout << people << endl;
        }
    }
}

string readConversation(ifstream& file) {
    Vector<string> people;
    string line;
    while (getline(file, line)) {
        if (line == "---") break;
        string name = line.substr(0, line.find(":"));
        if (!people.contains(name)) {
            people.add(name);
```

Name: _____

```
            if (people.size() == 2) {
                break;
            }
        }
    }

    if (people.size() == 2) {
        people.sort();
        return people[0] + " and " + people[1];
    } else {
        return "";
    }
}
```

Name: _____

4. Recursion (read)

a. `17 ( [8] , [1] )`

b. `31 ( 15 ( [7] , [1] ) , [1] )`

c. `62 ( 31 ( 15 ( [7] , [1] ) , [1] ) , [0] )`

Name: _____

5. Recursion (write)

```
int countDuplicates(Stack<int>& stack, int prev = -1) {
    if (stack.isEmpty()) {
        return 0;
    } else {
        int dupes = 0;
        if (prev >= 0 && stack.peek() == prev) {
            dupes++;
        }
        prev = stack.pop();
        dupes += countDuplicates(stack, prev);
        stack.push(prev);
        return dupes;
    }
}
```

Name: _____

6. Backtracking (write)

```
bool knightsTourHelper(Grid<int>& board, const Location& loc, int knights) {
    if (knights > board.size()) {
        // base case: stop if we fill the whole board with knights
        printBoard(board);
        return true;
    } else if (!board.inBounds(loc.row, loc.col) || board[loc.row][loc.col] != 0) {
        // base case: don't go out of bounds or revisit the same square twice
        return false;
    } else {
        // recursive case: mark this square and explore future moves
        board[loc.row][loc.col] = knights;           // choose
        for (Location neighbor : getMoves(loc)) {    // explore
            if (knightsTourHelper(board, neighbor, knights + 1)) {
                return true;
            }
        }
        board[loc.row][loc.col] = 0;                 // unchoose
        return false;
    }
}

void knightsTour(Grid<int>& board, Location startLoc) {
    knightsTourHelper(board, startLoc, 1);
}
```