**CSCI 230 – Data Structures (in JAVA)**

**College of Charleston**

# SAMPLE TEST TWO

**Mar. 11, 2020**

**Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**

**Student ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**(Total is 100 points)**

**1. Brief Answers (5x8 = 40 points)**

The expected big-O performance of hash table.

Expected O(1)

The worst-case big-O performance of balanced BST, such as AVL-tree or Red-Black tree.

Worst-case O(lgN)

Pros and Cons of array, linked-list, hash-table, BST.

Sorted array: quick binary search O(lgN), ordered (approximate queries, range queries), indexing; NOT dynamic (slow insertion and deletion).

Linked-list: dynamic; lost almost all merits of sorted array.

Hash-table: fast search, insertion, deletion (the fastest dynamic ADT), convert large-size data to small-size index; no worst-case guarantee, NO order.

The invariances of AVL-trees, red-black trees, and B-trees

AVL: balance factor (height difference of subtrees)

Red-black: red-black rules

B-tree: lower- and upper-bounds of children

The collision resolve schemes for hash tables.

Chaining, linear probing, quadratic probing, double hashing

**2. Pseudo Code (5x6 = 30 points)**

Traverse a binary tree in in-order, pre-order, or post-order

Resolve collision in a hash table with chaining, linear probing, quadratic probing, or double hashing.

BST search, insertion, or deletion

Rotate in BST at a given node with given direction

**3. Problem Solving (15x2 = 30 points)**

Put a list of keys into a hash table.

Insert and delete a sequence of keys in a BST.