CPSC 223 Spring 2025 Exam #2
Tuesday, April 8, 2025

Write your name and NetID on this answer sheet and the cover of this exam booklet, but do not open the exam booklet until told to do so. Turn in both your answer sheet and exam booklet.
Name: _____ NetID: _____

Problem 1 (9 points): Hashtables

a) Given an initially empty hashtable of capacity 4 using open addressing with linear probing and a resize threshold $\alpha > 0.5$, mark an X in the box for the operation(s) below that will trigger a resize:

- ☐ insert 10
- ☐ insert 22
- ☐ insert 31
- ☐ insert 4
- ☐ insert 15

b) Suppose we delete key 22 and then search for key 15. Mark an X for each statement that is guaranteed true:

- ☐ Search finds key 15 without probing past its initial slot.
- ☐ Search may require probing past deleted slots.
- ☐ Search fails if a tombstone is not handled.

c) What assumption about the hash function ensures average-case $O(1)$ performance?
_____

Problem 2 (18 points): Binary Search Trees

For each code fragment below, write the number(s) of the fragments that correctly construct a BST containing the keys [5, 2, 8, 1, 3] when inserted in that order.

1. `Node *root = NULL; insert(&root, 5); insert(&root, 2); insert(&root, 8); insert(&ro`
2. `Node *root = create_node(5); root->left = create_node(2); root->left->left = create`
3. `Node *arr[5]; arr[0] = create_node(5); arr[1] = create_node(2); arr[2] = create_nod`
4. `Node *root = create_node(5); insert(&root, 8); insert(&root, 2); insert(&root, 1);`
5. `Node *root = create_node(5); insert(&root, 2); insert(&root, 8); insert(&root, 3);`

Problem 3 (12 points): Graph Search and Shortest Paths

Consider the undirected graph G with adjacency lists:
A: B, C B: A, D, E C: A, F D: B E: B, F F: C, E

a) BFS traversal starting from A (visit neighbors in alphabetical order): _____

b) DFS preorder starting from A (visit neighbors in alphabetical order): _____

c) DFS postorder starting from A (visit neighbors in alphabetical order): _____

d) Shortest-path distances from A using Dijkstra's algorithm: _____
(list distances for A, B, C, D, E, F)

e) Does G contain a cycle? (Yes/No): _____

f) Is G connected? (Yes/No): _____

Problem 4 (16 points): k-d Tree Nearest Neighbor Search

Complete the following nearest-neighbor search code for a k-d tree.

```
typedef struct kdnode {
    double point[2];
    struct kdnode *left, *right;
} kdnode;

double square(double x) { return x*x; }

void nearest(kdnode *root, double target[2], int depth, kdnode **best, double *bestDist)
    if (root == NULL) return;
    int axis = depth % 2;
    double d = square(root->point[0] - target[0]) + square(root->point[1] - target[1]);
    if (d < *bestDist) {
        *bestDist = d;
        *best = root;
    }
    if (target[axis] < root->point[axis]) {
        /* 1 */
    } else {
        /* 2 */
    }
    double diff = target[axis] - root->point[axis];
    if (square(diff) < *bestDist) {
        /* 3 */
    }
}
```

Problem 5 (6 points): Heap Runtimes

Implementations of a priority queue:

- A: Binary heap
- B: Fibonacci heap
- C: Unsorted array

1. $\Theta(1)$ amortized insert: _____
2. $\Theta(1)$ worst-case find-minimum: _____
3. $\Theta(\log n)$ worst-case extract-minimum: _____

Problem 6 (8 points): Depth-First Search Code Completion

Complete the following recursive DFS traversal:

```
void dfs(Graph *g, int v) {
    /* 1 */               // mark v as visited
    printf("%d ", v);
    for (int u = 0; u < g->n; u++) {
        if (/* 2 */) {    // u is adjacent to v and not yet visited
            /* 3 */       // recursive call on u
        }
    }
}
```

Problem 7 (15 points): AVL Tree Traversals

Starting from an empty AVL tree, insert the following sequences of keys.

a) 30, 20, 10
Preorder: _____ Inorder: _____

b) 10, 20, 30
Preorder: _____ Inorder: _____

c) 20, 10, 30, 25, 27
Preorder: _____ Inorder: _____

Problem 8 (16 points): Dijkstra's Algorithm Code Completion

```
void dijkstra(Graph *g, int src) {
    int dist[MAXV];
    bool known[MAXV] = {false};
    /* 1 */             // initialize dist[v] for all v
    dist[src] = 0;
    for (int i = 0; i < g->n; i++) {
        int v = /* 2 */;    // select unknown v with minimum dist
        known[v] = true;
        for (Edge *e = g->adj[v]; e != NULL; e = e->next) {
            int w = e->to;
            int wgt = e->weight;
            if (/* 3 */) {  // relaxation condition
                /* 4 */     // update dist[w]
            }
        }
    }
}
```