CS106B

Instructor: Cynthia Lee

Autumn 2019

Practice Exam

FINAL EXAM

1. Sorting. Show each step of selection sort on the vector of integers shown below. The pseudocode for selection sort is given as a reminder.

```
print(vector) // Initial state means the values at this point in the code
for (i = 0; i < vector size; i = i + 1) {
    minPosition = i
    for (j = i; j < vector size; j = j + 1) {

        if (vector[j] < vector[minPosition]) minPosition = j

    }
    temp = vector[i]
    vector[i] = vector[minPosition]
    vector[minPosition] = temp
    print(vector) // You write vector's values at this point in the code
}
```

Below is the initial state of the vector of values to sort. Fill in the rest of the steps for each time the "print" executes. You may not need all the provided blank vectors to complete your solution.

5	6	8	4	2	8	3	7	1

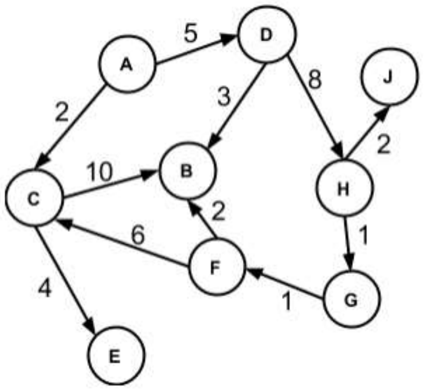
2. **BFS/DFS.** Show the order of nodes visited when running BFS and DFS, starting at the node labeled A on the following graph. If there is more than one correct solution, write any correct solution.

BFS:

A, ____________________

DFS:

A, ____________________



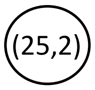
3. **Heaps.** Draw the tree structure that results from interpreting the data in the array below as a binary heap (using 0-based indexing where the 0th index of the array is the leftmost box below). Then answer the question about your drawing.

5	8	6	9	13	16	23	28	10	30

DRAW:

CIRCLE ONE: Is this a valid min-heap? YES NO

4. BSTs . We have implemented the Map ADT using a plain Binary Search Tree (no special balancing measures). Our node structure is defined as "`struct node { int key; int value; node* left; node* right; };`" Draw a diagram of the BST that results from inserting the following (key, value) pairs in the order given. Please label each BST node with both the key and value. (25,2), (5,5), (19,3), (5,12), (40,5), (3,1)

Diagram after inserting (25,2):  *This one is completed for you as a node formatting example.*	Diagram after inserting (5,5):
Diagram after inserting (19,3):	Diagram after inserting (5,12):
Diagram after inserting (40,5):	Diagram after inserting (3,1):

5. **ADTs.** For this problem, you will write part of the game 2048. The game consists of a Grid of integer tiles and blank spaces (we will represent blank spaces with the integer 0), which we try to aggregate to the sum of 2048 by shifting them left, right, up and down.

   The function you are to write is **`moveLeft()`**, which updates the game board when the user hits the left arrow key (or, on a phone, swipes left). The desired functionality is that number tiles "slide" left across blank spaces as far as they can before colliding into another number tile (or the left edge). If a tile collides with another tile that has the same value, the two are merged into a single tile with the sum of the values.

Example 1:

Before:

2	0	**2**	0	0
0	0	**8**	**8**	0
1	**4**	0	**4**	**1**

After:

4	0	0	0	0
16	0	0	0	0
1	**8**	**1**	0	0

Notes:

- Remember that we are only implementing **`moveLeft()`**, so tiles will always slide left.
- In case of more than one possible merge, merges happen with the leftmost pair (see Example 2).
- Each number only merges once per "turn," and doesn't merge again if the new sum matches an adjacent number (see Example 3).
- You must use the function signature provided below.

Example 2:

Before:

2	0	0	**2**	**2**
0	0	0	0	0
0	**8**	**8**	**8**	0

After:

4	**2**	0	0	0
0	0	0	0	0
16	**8**	0	0	0

NOT THIS:

2	**4**	0	0	0
0	0	0	0	0
8	**16**	0	0	0

Example 3:

Before:

2	**2**	0	**2**	**2**
2	0	**2**	**4**	0
4	**0**	**2**	**2**	0

After:

4	**4**	0	0	0
4	**4**	0	0	0
4	**4**	0	0	0

NOT THIS:

8	0	0	0	0
8	0	0	0	0
8	0	0	0	0

You must use the following function signature. It should work for any size/dimensions board.

```
            void moveLeft(Grid<int>& board);
```

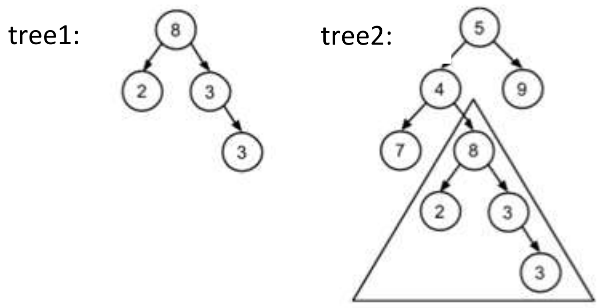
Please write your code on the next page.

```
void moveLeft(Grid<int>& board) {


}
```

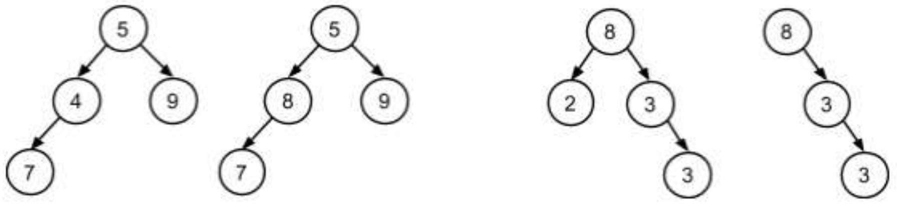
6. **Trees.** In this problem, we'll be determining if a binary tree is a perfect **copy** of a subtree of another binary tree. For this problem, a subtree means a tree made of up a node in the original tree along with all of its descendents (children, plus children of children, and so on) in the original tree.

   - **Example:** The nodes within the triangle in the right tree below are a subtree of the full binary tree, and the binary tree on the left is a copy of this sub-tree.



A sub-tree can start from the root or any other node in the binary tree. Note that for two trees to be a copy, every node in the two trees at the same position must have the same **value** (they will not actually be pointing to the same memory locations). Each node must also have the same children to their left and to their right in the tree.

- **Example:** Here are some examples of trees that are not the same:



- The left two trees are different from one another because the value of the left child of the root nodes are not the same (4 vs. 8). The right two trees are not the same because the farthest right tree is missing the left child node of the root node.

Your function should have the following signature:

```
bool isSubtree(Node *tree1, Node * tree2);
```

- Return true if tree1 is a copy of a subtree of tree2, otherwise return false.
- You may assume that tree1 and tree2 are not NULL when passed into the subTree() function.

Where Node is defined as follows:

```
struct Node {
      int value;
      Node *left;
      Node *right;
};
```

Write your code on the next page.

```
   bool isSubtree(Node *tree1, Node * tree2) {











}
```