This practice exam is based on an actual final exam from CS106X (same topics coverage as CS106B, but somewhat higher expectations for mastery). The question types and mix of topics of our CS106B exam will be the same, but this practice may over-prepare you to some degree in terms of the difficulty of a couple of the individual problems. As my basketball coach would always say when we were conditioning—a practice that is harder than the game is a good thing! --Cynthia

CS106B Instructor: Cynthia Lee

Autumn 2017 Practice Exam

PRACTICE FINAL EXAM 2

NAME (LAST, FIRST): ____

SUNET ID:____ @stanford.edu

Problem	Sorting 1	BFS/DFS 2	Heap 3	MST 4	BST 5	ADTs 6	Graphs 7	Trees 8	TOTAL
Score									
Possible									

Instructions:

- The time for this exam is **3 hours**.
- Use of anything other than a pencil, eraser, pen, one 8.5x11 page (one side) of notes, and the official textbook is prohibited. In particular, **no** computers or digital devices of any kind are permitted. Blank scratch paper may be provided by proctors and does not need to be turned in.
- PLEASE rip off the pages of library reference in the back of the exam and do not turn them in.
- **Please do NOT insert new pages** into the exam. Changing the number of pages in the exam confuses our automatic scanning system. You may use the back sides of the exam pages for more space. Thanks.
- SCPD and OAE: Please call or text 760-845-7489 if you have a question.

Please sign *before* you begin:

I agree to abide by the spirit and letter of the Honor Code, and to follow the instructions above.

____ ______________ ______________________

(Signature) (Date) (Start time - HH:MM zone)

[Type here]

---

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

[Type here]

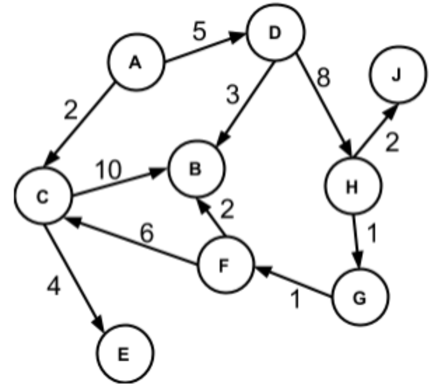
2. BFS/DFS. Show the order of nodes visited when running BFS and DFS, starting at the node labeled A on the following graph. If there is more than one correct solution, write any correct solution.

BFS:

A, ______________________________

DFS:

A, ______________________________



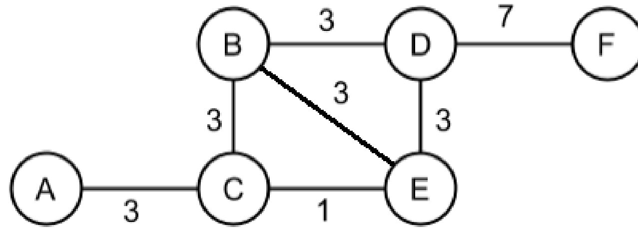
3. Heaps. Draw the tree structure that results from interpreting the data in the array below as a binary heap (using 0-based indexing where the 0th index of the array is the leftmost box below). Then answer the question about your drawing.

5	8	6	9	13	16	23	28	10	30

DRAW:

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

4. Minimum Spanning Trees. Consider the following graph:



How many *distinct* MSTs are found in this graph? ____________

For each distinct MST that you found, circle the edges that participate in that MST. We provide space for you to list 10 MSTs, but you <u>may not need all</u> the provided space, if you found fewer than 10 MSTs; or you may add more lists if you found more than 10 MSTs.

MST #1: AC BC BE CE BD DE DF

MST #2: AC BC BE CE BD DE DF

MST #3: AC BC BE CE BD DE DF

MST #4: AC BC BE CE BD DE DF

MST #5: AC BC BE CE BD DE DF

MST #6: AC BC BE CE BD DE DF

MST #7: AC BC BE CE BD DE DF

MST #8: AC BC BE CE BD DE DF

MST #9: AC BC BE CE BD DE DF

MST #10: AC BC BE CE BD DE DF

5. **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)

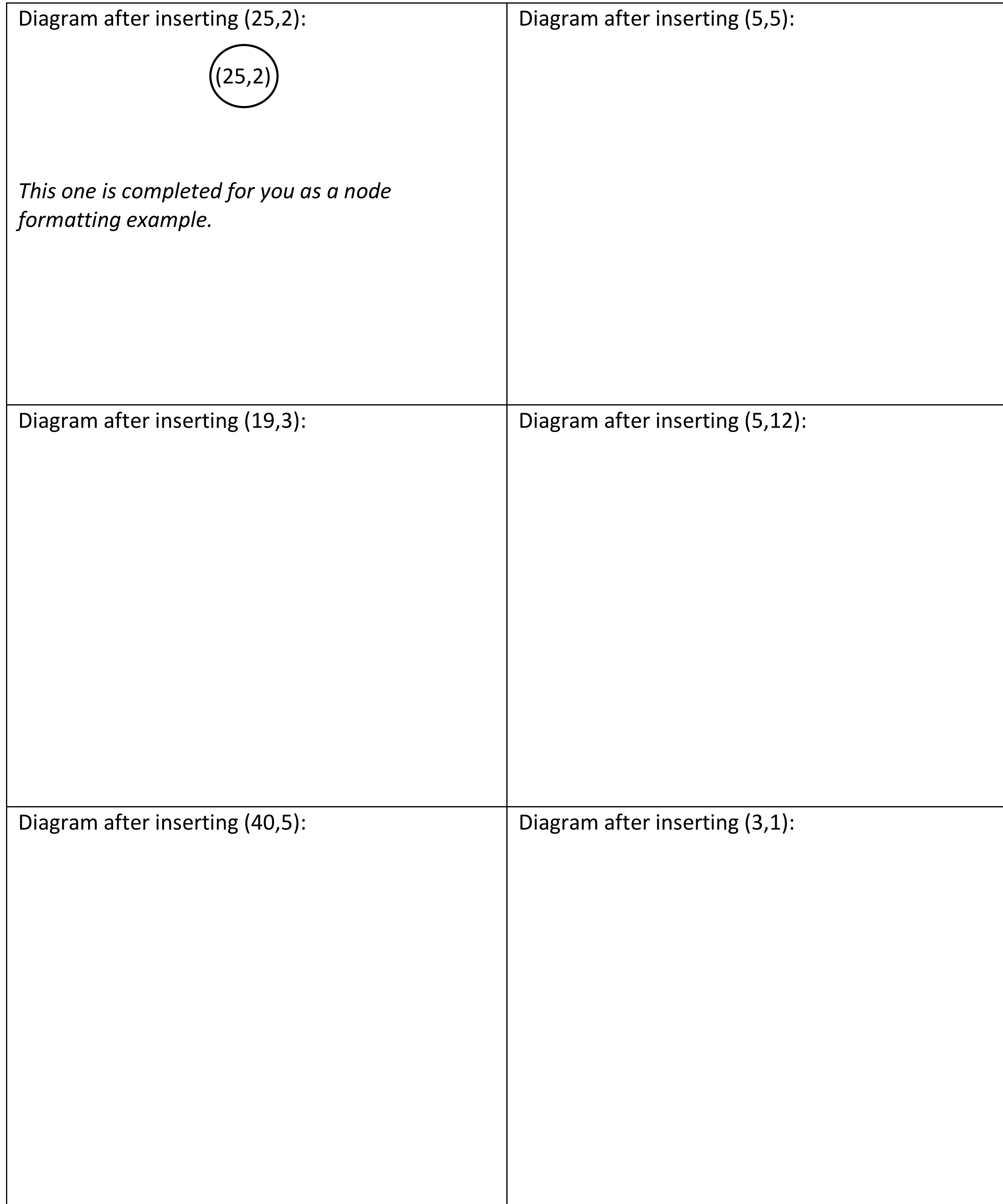


Diagram after inserting (25,2): (25,2) *This one is completed for you as a node formatting example.*	Diagram after inserting (5,5):
Diagram after inserting (19,3):	Diagram after inserting (5,12):
Diagram after inserting (40,5):	Diagram after inserting (3,1):

6. **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.

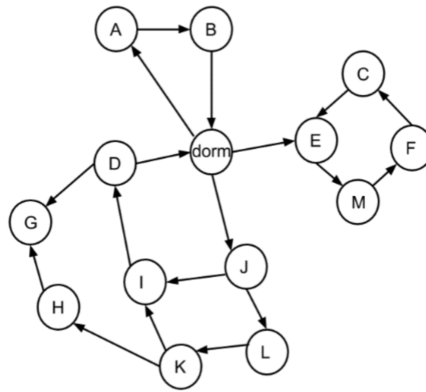
```
void moveLeft(Grid<int>& board) {

}
```

7. Graphs. You are traveling around the United States this summer, using your Stanford dorm as your "base camp," from which you will launch several trips. Each trip must start and end at your Stanford dorm, and each trip will consist of visits with friends in cities throughout the country. You don't want to overtax the generosity of your friends with whom you will be staying, so you only want to visit each city at most once per trip (but you could stay with that friend again on subsequent trips).

These are driving trips, and the road map will be given to you as an unweighted, directed graph.

(A) How many different trips can you take this summer? Write a function with the following signature:

- `int countItineraries(BasicGraph& map, Vertex* dorm);`
- **Example:** For the `map` and `dorm` below, your function should return 3. The itineraries are [dorm, A, B, dorm], [dorm, J, I, D, dorm], and [dorm, J, L, K, I, D, dorm].



(B) What is the longest trip you will take this summer? Write a function with the following signature:

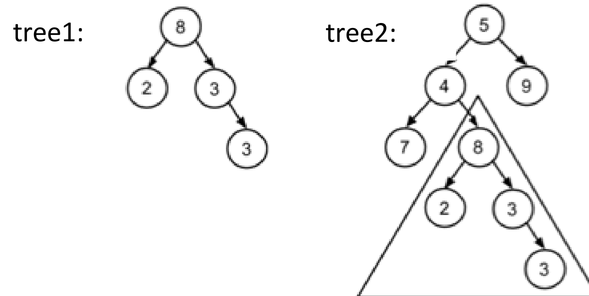
- `int longestItinerary(BasicGraph& map, Vertex* dorm);`
- **Example:** For the `map` and `dorm` above, your function should return 7, because the [dorm, J, L, K, I, D, dorm] itinerary is 7 vertices long.

```
int countItineraries(BasicGraph& map, Vertex* dorm){

}
```

```
int longestItinerary(BasicGraph& map, Vertex* dorm){

}
```

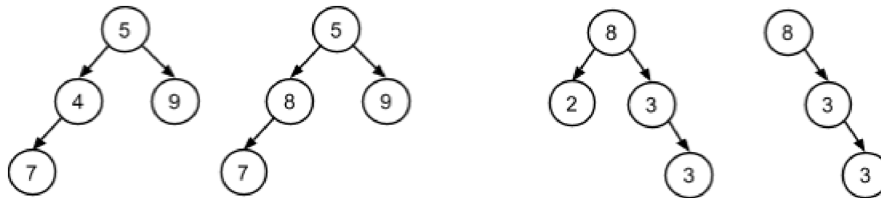
8. Trees. In this problem, we'll be determining if a binary tree is a perfect **copy** of a subtree of another binary tree. For this problem, a subtree means a tree made of up a node in the original tree along with all of its descendents (children, plus children of children, and so on) in the original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

}
```

Summary of Relevant Data Types

We tried to include the most relevant member functions for the exam, but not all member functions are listed. You are free to use ones not listed here that you know exist. **You do <u>not</u> need `#include`.**

```
class string {
 bool empty() const;
 int size() const;
 int find(char ch) const;
 int find(char ch, int start) const;
 string substr(int start) const;
 string substr(int start, int length) const;
 char& operator[](int index);
 const char& operator[](int index) const;
};

class Vector {
 bool isEmpty() const;
 int size() const;
 void add(const Type& elem); // operator+= used similarly
 void insert(int pos, const Type& elem);
 void remove(int pos);
 Type& operator[](int pos);
};

class Grid {
 int numRows() const;
 int numCols() const;
 bool inBounds(int row, int col) const;
 Type get(int row, int col) const; // cascade of operator[] also works
 void set(int row, int col, const Type& elem);
};

class Stack {

 bool isEmpty() const;
 void push(const Type& elem);
 Type pop();
};

class Queue {
 bool isEmpty() const;
 void enqueue(const Type& elem);
 Type dequeue();
};
```

```
class Map {
 bool isEmpty() const;
 int size() const;
 void put(const Key& key, const Value& value);
 bool containsKey(const Key& key) const;
 Value get(const Key& key) const;
 Value& operator[](const Key& key);
};
```

Example range-based for: `for (Key key : mymap){…}`

```
class Set {
 bool isEmpty() const;
 int size() const;
 void add(const Type& elem);
 bool contains(const Type& elem) const;
};
```

Operators:

```
set + value // Returns the union of set set1 and individual value value
set += value // Adds the individual value value to the set set
set1 += set2 // Adds all the elements from set2 to set1
```

Example range-based for: `for (Type elem : mymap){…}`

```
class Lexicon {
  int size() const;
  bool isEmpty() const;
  void clear();
  void add(std::string word);
  bool contains(std::string word) const;
  bool containsPrefix(std::string prefix) const;
};
```

Example range-based for: `for (string str : english){…}`

```
struct Edge {
    Vertex* start;
    Vertex* finish;
    double cost;
    bool visited;
};
struct Vertex  {
    std::string name;
    Set<Edge*> arcs;
    Set<Edge*>& edges;

    bool visited;
    Vertex* previous;
};
```

```
class BasicGraph : public Graph<Vertex, Edge> {
public:
BasicGraph();
    Vertex* addVertex(Vertex* v);
    const Set<Edge*>& getEdgeSet() const;
    const Set<Edge*>& getEdgeSet(Vertex* v) const;
    const Set<Edge*>& getEdgeSet(std::string v) const;
    Vertex* getVertex(std::string name) const;
    const Set<Vertex*>& getVertexSet() const;
    void removeEdge(std::string v1, std::string v2, bool directed = true);
    void removeEdge(Vertex* v1, Vertex* v2, bool directed = true);
    void removeEdge(Edge* e, bool directed = true);
    void removeVertex(std::string name);
    void removeVertex(Vertex* v);
}
```