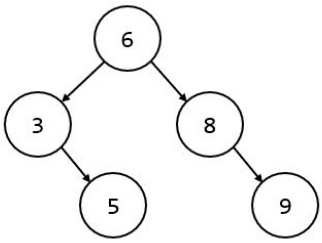
15-110 Hw4 - Written Portion

Name:

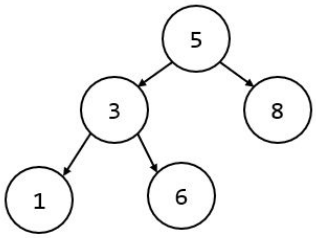
AndrewID:

#1 - Tree Identification - 10pts

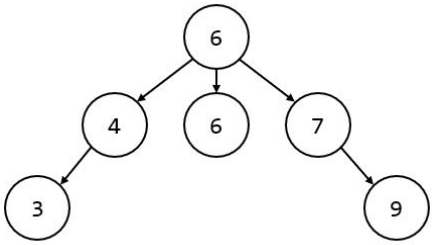
For each of the following trees, determine whether or not it is a Binary Search Tree.



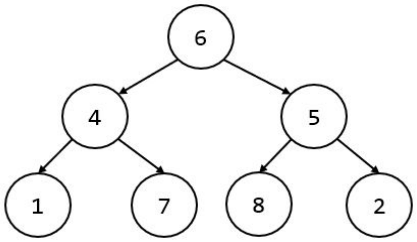
☐ BST
☐ Not BST



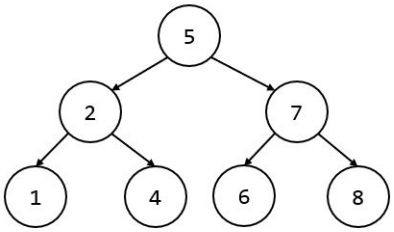
☐ BST
☐ Not BST



☐ BST
☐ Not BST



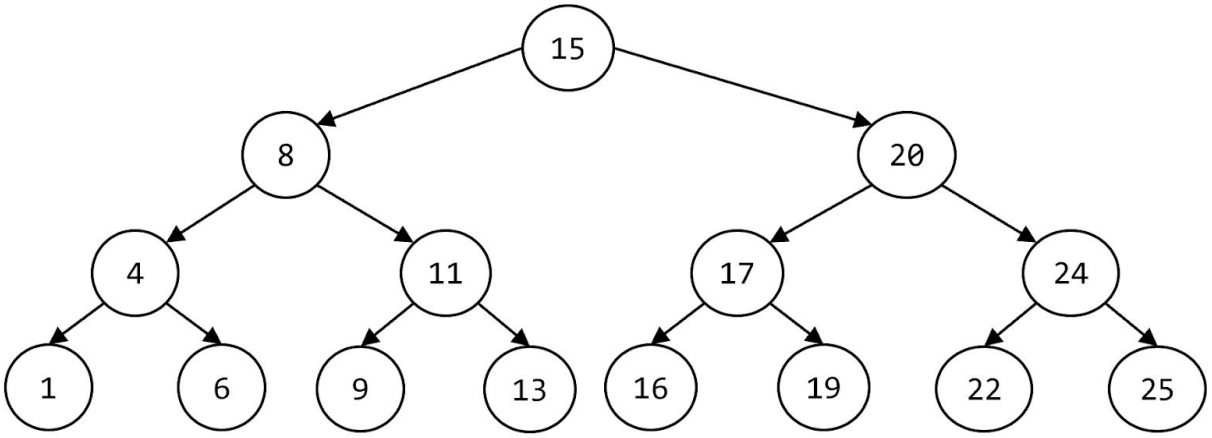
☐ BST
☐ Not BST



☐ BST
☐ Not BST

#2 - Searching a BST - 9pts

Given the Binary Search Tree shown below:



What series of numbers would you visit if you ran a search algorithm that looked for **19**?

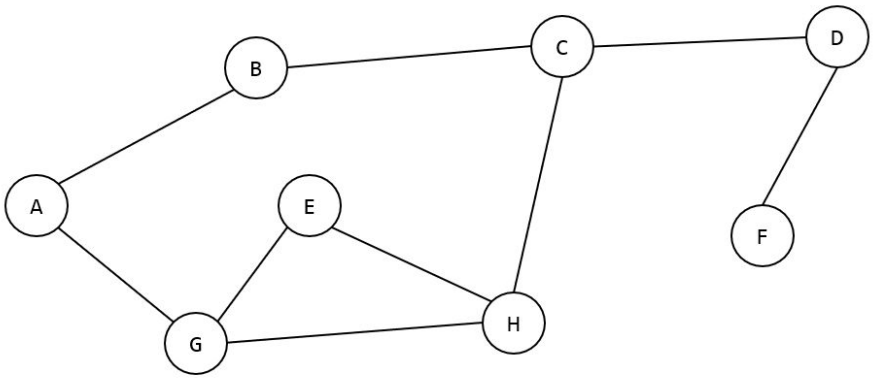
What series of numbers would you visit if you ran a search algorithm that looked for **4**?

What series of numbers would you visit if you ran a search algorithm that looked for **10**?

#3 - Searching a Graph - 12pts

For this problem, note that each prompt has multiple correct answers; you only need to include one. **We recommend that you visit neighbors in alphabetical order.**

Given the undirected graph shown below, where the letter **A** is the start node:



What series of letters could you visit if you ran **Depth-First Search** to find **H**?
(For this and the following problems, there is more than one correct answer.)

What series of letters could you visit if you ran **Breadth-First Search** to find **H**?

What series of letters could you visit if you ran **Depth-First Search** to find **J**?

What series of letters could you visit if you ran **Breadth-First Search** to find **J**?

#4 - P and NP Identification - 10pts

For each of the following problems, identify whether the problem is in the complexity class P, NP, or neither. You should choose just one class (the one that best describes the problem).

Finding the smallest value in a tree	☐ P ☐ NP ☐ Neither
Scheduling final exams for CMU so that there are no conflicts	☐ P ☐ NP ☐ Neither
Determining if an item is in a list	☐ P ☐ NP ☐ Neither
Finding the **best** (fastest) road route through Pittsburgh that takes you over every bridge.	☐ P ☐ NP ☐ Neither
Determining if there is a set of inputs that makes a circuit output 1	☐ P ☐ NP ☐ Neither

#5 - Tractability - 6pts

Select whether each of the following function families is **tractable** or **intractable**.

$O(n)$	☐ Tractable	☐ Intractable
$O(n!)$	☐ Tractable	☐ Intractable
$O(2^n)$	☐ Tractable	☐ Intractable
$O(n^2)$	☐ Tractable	☐ Intractable
$O(\log n)$	☐ Tractable	☐ Intractable
$O(n^{100})$	☐ Tractable	☐ Intractable

#6 - P vs NP - 9pts

Which of the following is the *best* definition of the complexity class **P**?

- ☐ The set of problems that can be solved in polynomial time
- ☐ The set of problems that can be verified in polynomial time
- ☐ The set of problems we discussed in lecture (Travelling Salesperson, Subset Sum, etc)

Which of the following is the *best* definition of the complexity class **NP**?

- ☐ The set of problems that can be solved in polynomial time
- ☐ The set of problems that **cannot** be solved in polynomial time
- ☐ The set of problems that can be verified in polynomial time
- ☐ The set of problems that **cannot** be verified in polynomial time
- ☐ The set of problems we discussed in lecture (Travelling Salesperson, Subset Sum, etc)

Why does it matter whether or not P = NP? Choose the *best* answer.

- ☐ If they are the same, we'll be able to solve hard and useful problems a lot faster
- ☐ If they are the same, we'll need to change how we implement some adversarial algorithms, like encryption, to keep them from being broken easily
- ☐ If they are not the same, we can spend less time trying to invent super-fast solutions to hard but useful problems
- ☐ All of the above

#7 - Recognizing Data Structures - 5pts

For each of the following types of data, choose the data structure that would be the **best/most natural choice** to represent the data

Carnegie Mellon's organizational structure: ie, departments within each college, and majors within each department	☐ 1D List ☐ 2D List ☐ Dictionary ☐ Tree ☐ Graph
A chess board that has pieces located at specific row-column positions	☐ 1D List ☐ 2D List ☐ Dictionary ☐ Tree ☐ Graph
A set of chores you need to do over the weekend	☐ 1D List ☐ 2D List ☐ Dictionary ☐ Tree ☐ Graph
The subway map for London	☐ 1D List ☐ 2D List ☐ Dictionary ☐ Tree ☐ Graph
A deck of flashcards with words on one side and definitions on the other	☐ 1D List ☐ 2D List ☐ Dictionary ☐ Tree ☐ Graph

#8 - Search Efficiencies - 8pts

For each of the following prompts, consider the search algorithms we've discussed in class to determine the optimal Big-O runtime for the given situation.

We store a music collection in a dictionary mapping artists to lists of their songs. What is the runtime to find whether a specific artist is in a collection if n is the size of the dictionary?

- ☐ O(1)
- ☐ O(log n)
- ☐ O(n)
- ☐ O(n log n)
- ☐ $O(n^2)$
- ☐ $O(2^n)$

We store a game of 20 questions in a binary tree where each node is a question whose left branch answers 'yes' and whose right branch answers 'no'. The leaves hold all eventual solutions. What is the runtime to determine whether *any* of the answer paths lead to the solution 'octopus' if n is the number of nodes in the tree?

- ☐ O(1)
- ☐ O(log n)
- ☐ O(n)
- ☐ O(n log n)
- ☐ $O(n^2)$
- ☐ $O(2^n)$

We store all current players in the Basketball Hall of Fame in a list, organized alphabetically by name. What is the runtime to find whether a specific player is in the Hall of Fame if n is the length of the list?

- ☐ O(1)
- ☐ O(log n)
- ☐ O(n)
- ☐ O(n log n)
- ☐ $O(n^2)$
- ☐ $O(2^n)$

We make a graph of everyone in the world where two people are connected if they have ever met. What is the runtime to find if there is a chain of introductions that links you to Oprah Winfrey if n is the number of nodes in the graph?

- ☐ O(1)
- ☐ O(log n)
- ☐ O(n)
- ☐ O(n log n)
- ☐ $O(n^2)$
- ☐ $O(2^n)$

#9 - Optimizing for Search - 6pts

You have been given a very large dataset of temperatures (represented as floats), and your task is to find the most extreme temperatures that fall into a given temperature range (such as 40 degrees to 50 degrees, or 75.7 degrees to 78.2 degrees). To do this, you want to store the data in a data structure so that, given any range, you'll be able to:

- find the smallest value in the structure that falls in that range
- find the largest value in the structure that falls in that range

You want to optimize how quickly you can run the algorithm shown above, assuming the data structure has already been created. In other words, **you don't know what range you'll need to check when you create the structure.**

Choose the best search algorithm + data structure combination for the task. There might be multiple correct answers; you only need to choose one per question.

Search Algorithm:

- ☐ Linear Search
- ☐ Binary Search
- ☐ Hashed Search
- ☐ Breadth-First Search

Data Structure:

- ☐ Sorted List of degrees
- ☐ Dictionary mapping degree->count
- ☐ Binary Search Tree of degrees
- ☐ Graph connecting close degrees

15-110 Hw4 - Programming Portion

Each of these problems should be solved in the starter file available on the course website. Submit your code to the Gradescope assignment Hw4 - Programming for autograding.

All programming problems may also be checked by running the starter file, which calls the function `testAll()` to run test cases on all programs.

#1 - `findParents(t, name)` - 15pts

Write the function `findParents(t, name)` which takes a genealogical family tree in the form of a binary tree (as discussed in class) and a string, the name of a person. If the person exists in the tree, the function returns a list containing their parents.

You'll need to use if statements to separate the four possible arrangements of parents. Return an empty list if the parents are unknown, a singleton list in the two cases where only one parent is known, and a two-element list if both parents are known. If the person does not exist in the tree, the function returns `None`. You are guaranteed that every name in the tree only shows up once (to avoid conflicting information).

For example, consider the following family tree (as shown in the course slides):

```
t = { "contents" : "Arya",
      "left"  : { "contents" : "Ned",
                  "left"  : { "contents" : "Rickard",
                              "left"  : None, "right" : None },
                  "right" : { "contents" : "Lyarra",
                              "left"  : None, "right" : None } },
      "right" : { "contents" : "Catelyn",
                  "left"  : { "contents" : "Hoster",
                              "left"  : None, "right" : None },
                  "right" : { "contents" : "Minisa",
                              "left"  : None, "right" : None } } }
```

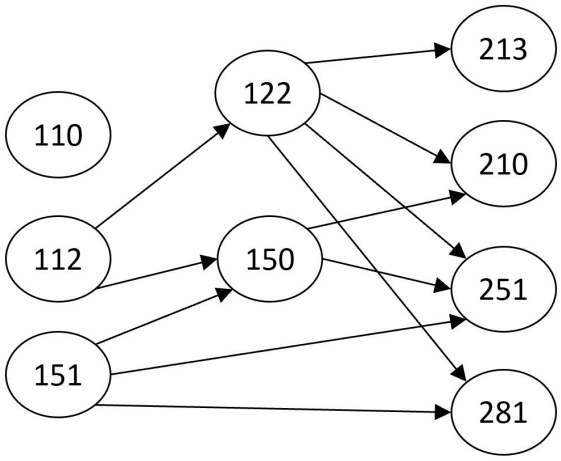
If we called `findParents(t, "Ned")`, the function would return the list `["Rickard", "Lyarra"]`. If we called `findParents(t, "Rickard")`, the function would return `[ ]`. If we called `findParents(t, "Jon")`, the function would return `None` because `"Jon"` is not in the tree.

Hint 1: treat this like a recursive search problem (in fact, you might want to reference tree linear search and binary search if you get stuck). You'll need to make **two** base cases - one for when you find the person, one for when you reach a leaf or an empty tree. For this problem, it may be easier to set up the base case as an empty tree.

Hint 2: unlike recursive binary search, you need to check **both** branches of the tree to see if the person occurs in either branch. How can you combine their results? If one of the branches gives you a list, you've found the parents - return them immediately. If you get `None`, try again on the other branch. If *both* branches return `None`, then the name was not found and you should return `None`.

#2 - `getPrereqs` - 10pts

College course prerequisites are notoriously complicated. However, we can make them a little easier to understand by representing the course dependency system as a **directed graph**, where the nodes are courses and an edge leads from course A to course B if A is a prerequisite of B. For example, the core Computer Science courses (almost) produce the following prereq graph:



Which would be represented in code as:

```
g = { "110" : [],
      "112" : ["122", "150"],
      "122" : ["213", "210", "251", "281"],
      "151" : ["150", "251", "281"],
      "150" : ["210", "251"],
      "213" : [],
      "210" : [],
      "251" : [],
      "281" : [] }
```

Write the function `getPrereqs(g, course)` that takes a directed graph (in our adjacency list dictionary format, without weights) and a string (a course name) and returns a list of all the immediate prerequisites of the given course. If we called `getPrereqs` on our graph above and `"210"`, for example, the function should return `["122", "150"]`.

Hint: you can't just return the neighbors of the course, because the edges are going in the opposite direction! Instead, iterate over all the nodes to find those that have the course as a neighbor. Construct a new list out of these nodes as the result.