15-110 Hw4 - Full

Name:

AndrewID:

Complete the following problems in the fillable PDF, or print out the PDF, write your answers by hand, and scan the results.

When you are finished, upload your hw4.pdf to **Hw4 - Full** on Gradescope.

Don't forget that you can get three bonus points on Hw4 by filling out the midsemester surveys! Check Piazza for links to those surveys and further instructions.

Written Problems

#1 - Best Case and Worst Case - 8pts

Can attempt after Runtime and Big-O Notation lecture

For each of the following functions, describe an input that would result in **best-case efficiency,** then describe an input that would result in **worst-case efficiency**. This generic input must work at **any possible size**; don't answer 1 for `isPrime`, for example.

```
def getEmail(words):
    # words is a list of strings
    for i in range(len(words)):
        if "@" in words[i]:
            return words[i]
    return "No email found"
```

```
def isPrime(num):
    for factor in range(2, num):
        if num % factor == 0:
            return False
    return True
```

What is a **best case input** for `getEmail`?

What is a **worst case input** for `getEmail`?

What is a **best case input** for `isPrime`?

What is a **worst case input** for `isPrime`?

#2 - Calculating Big-O Families - 10pts

Can attempt after Runtime and Big-O Notation lecture

For each of the following functions, check the **Big-O function family** that function belongs to. You should determine the function family by considering how the number of steps the algorithm takes grows as the size of the input grows.

```
def countEven(L): # n = len(L)
  result = 0
  for i in range(len(L)):
    if L[i] % 2 == 0:
      result = result + 1
  return result
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

```
# n = len(L)
def sumFirstTwo(L):
  if len(L) < 2:
    return 0
  return L[0] + L[1]
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

```
# n = len(L1) = len(L2)
def linearSearchAll(L1, L2):
  count = 0
  for item in L1:
    # Hint: linear search complexity..?
    if linearSearch(L2, item) == True:
      count = count + 1
  return count
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

```
# n = len(L1) = len(L2)
def binarySearchAll(L1, L2):
  count = 0
  for item in L1:
    # Hint: binary search complexity..?
    if binarySearch(L2, item) == True:
      count = count + 1
  return count
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

```
# n = len(L); original call has i = 0
def recursiveSum(L, i):
  if i == len(L):
    return 0
  else:
    return L[i] + recursiveSum(L, i+1)
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

#3 - Tree Vocabulary - 5pts

Can attempt after Trees lecture

Consider the following tree, implemented in code with our dictionary implementation:

```
t = { "value" : "A",
      "left"  : { "value" : "B",
                  "left"  : None,
                  "right" : None },
      "right" : { "value" : "C",
                  "left"  : { "value" : "D",
                              "left"  : None,
                              "right" : { "value" : "E",
                                          "left"  : None,
                                          "right" : None } },
                  "right" : { "value" : "F",
                              "left"  : None,
                              "right" : None } } }
```

How many **nodes** does this tree have?	
Which nodes are **children** of the node with value "C"?	
What is the value of the **root** of the tree?	
What are the values of the **leaves** of the tree?	
If we ran the first version of the function `countNodes` from lecture on this tree (with leaf base case), what is the **total** number of function calls that would be made?	

#4 - Graph Vocabulary - 5pts

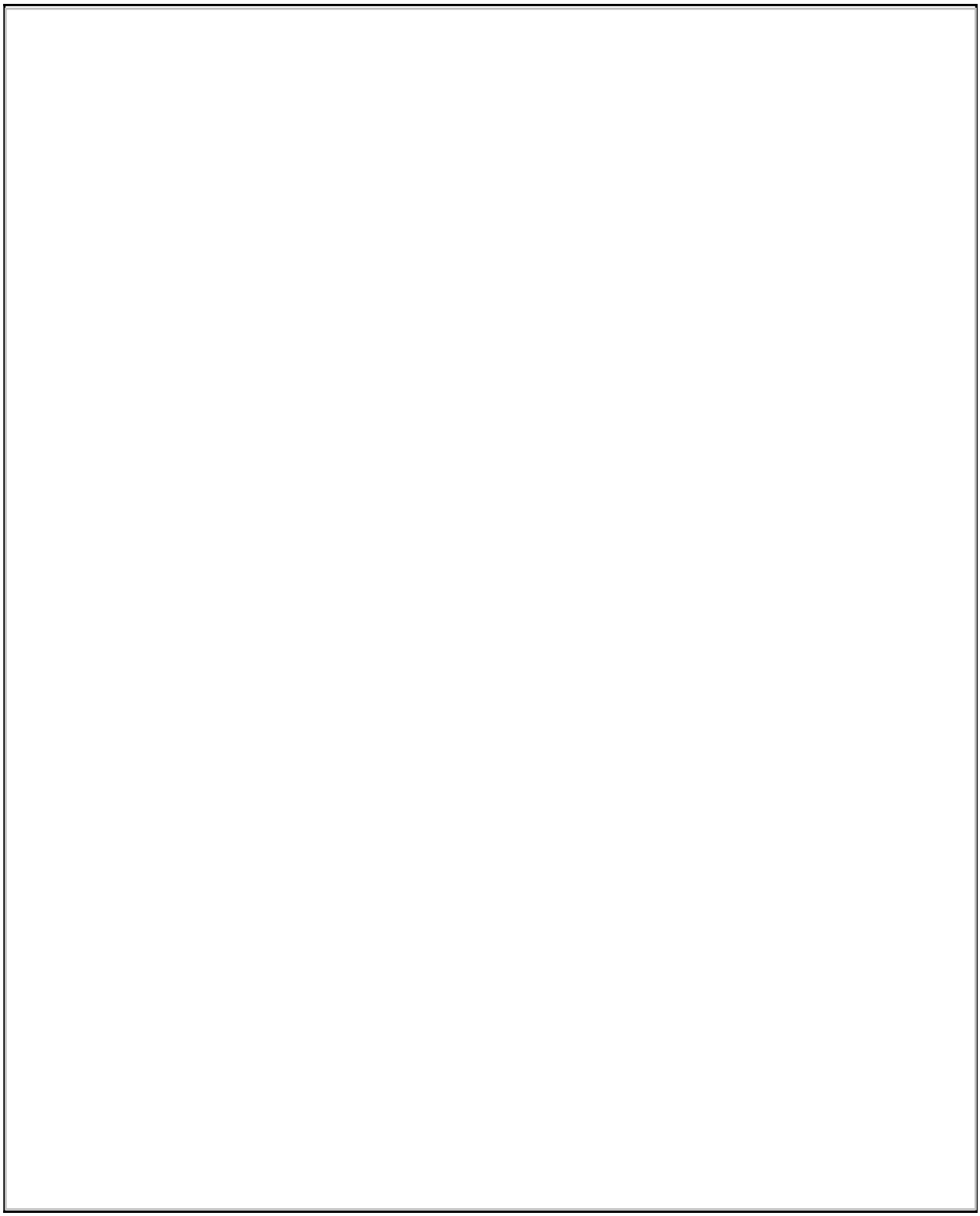
Can attempt after Graphs lecture

In class we discussed how a graph can be used to model a social network. Create a social network graph of your own design with the same notation we used in class (ovals for nodes, lines for edges). You can base the graph on whoever you like- fictional characters, celebrities, people in your own life, etc.- but your graph must meet the following requirements:

- The graph should contain at least five labeled nodes
- The graph should contain at least five edges
- The graph should contain an annotation that points out a pair of neighbors
- The graph should contain an annotation describing whether it is weighted or unweighted (your choice, but the annotation must match the graph)
- The graph should contain an annotation describing whether it is directed or undirected (your choice, but the annotation must match the graph)

You can do this with a picture of a physical drawing or an online image editing tool (like Google Drawings). To upload the image in the next page below, use the same approach you used on Hw2.

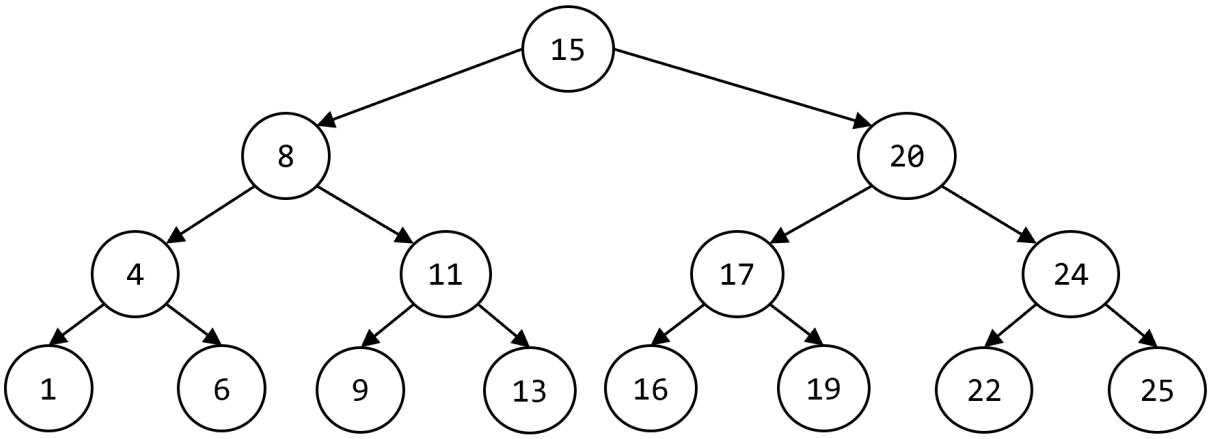
(continued on next page….)



#5 - Searching a BST - 6pts

Can attempt after Search Algorithms II lecture

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**?

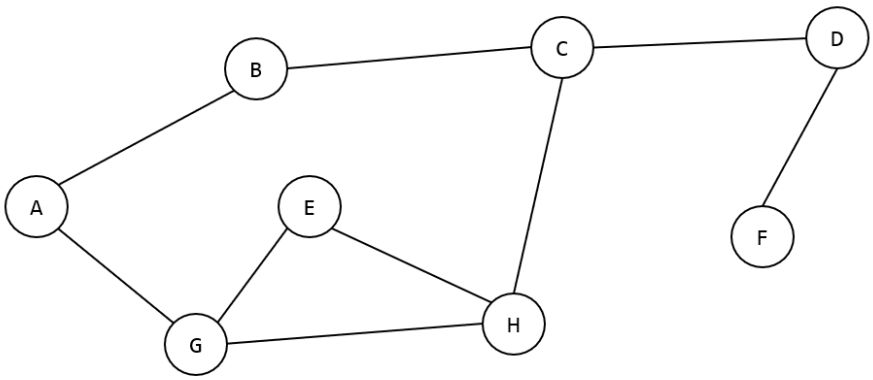
What series of numbers would you visit if you ran a search algorithm that looked for **10**?

#6 - Searching a Graph - 8pts

Can attempt after Search Algorithms II lecture

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**?

#7 - P and NP Identification - 5pts

Can attempt after Tractability lecture

For each of the following problems, identify whether the problem is in the complexity class P, NP, or neither. Please choose just one class (the one that **best** describes the problem), even if multiple classes are technically correct.

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

#8 - P vs NP - 8pts

Can attempt after Tractability lecture

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 (Puzzle solving, 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 (Puzzle solving, Subset Sum, etc)

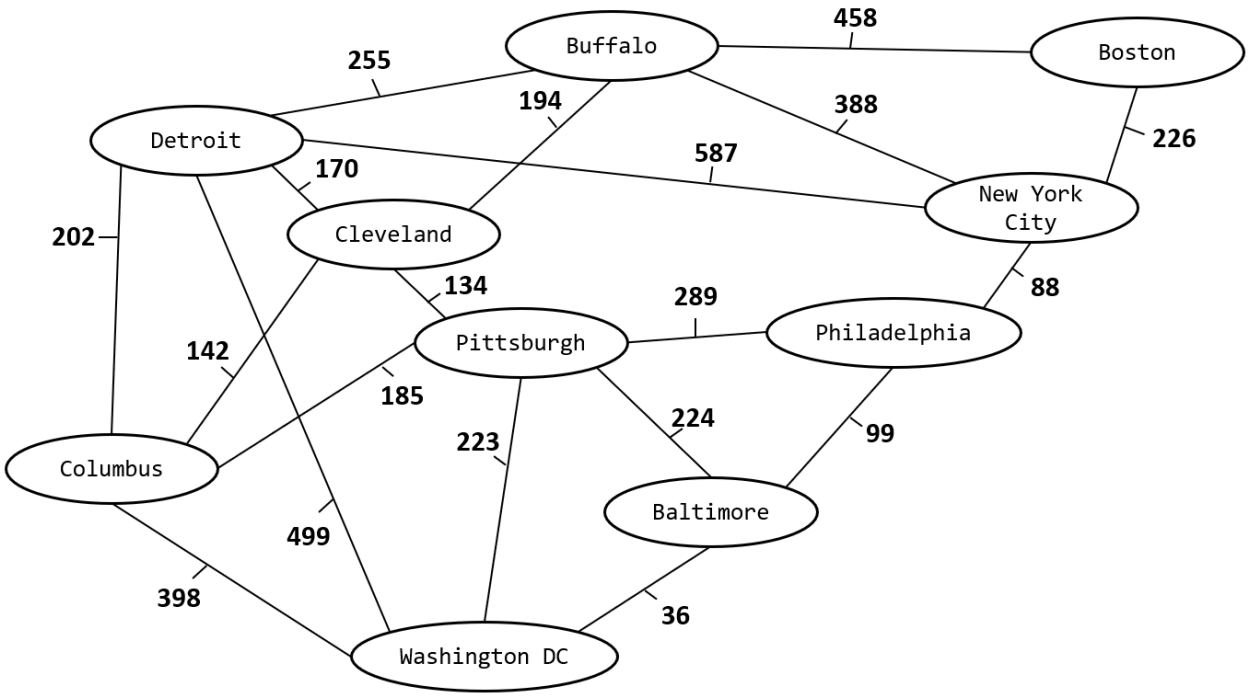
Why does it matter whether or not P = NP? Choose the *best* answer.

- ☐ 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

#9 - Heuristics - 4pts

Can attempt after Tractability lecture

We want to apply the Travelling Salesperson algorithm to the graph shown here to find a short route that visits each city once, but we want to use a **heuristic** to get the answer quickly. Our heuristic is this: rank the neighbor cities that have not yet been visited based on the weight of the edge leading to them, and choose the lowest-weight edge (the shortest distance).



Say we want to start in New York City and visit each city once (returning to New York City at the end). What path would this heuristic generate?

#10 - Recognizing Data Structures - 5pts

Can attempt after Graphs lecture

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

#11 - Optimizing for Search - 6pts

Can attempt after Search Algorithms II lecture

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