CSCE 310 Assignment 1 Summer 2020

Name(s) ______________________ CSE Login ______________________

Programming Language(s) Used ____

Question	Points	Score
1	5	
2	10	
3	10	
4	10	
5	25	
6	70	
7	70	
Total:	200	

Grader Notes:

Instructions Follow instructions *carefully*, failure to do so may result in points being deducted.

- Clearly label each problem and submit the answers *in order*.
- Be sure to show sufficient work to justify your answer(s). If you are asked to prove something, you must give a formal, rigorous, and complete proof
- You must adhere to the CSE academic integrity policy: `http://cse.unl.edu/academic-integrity-policy`
- You are encouraged to typeset your homework using LaTeX; if your answers are not legible, you may be required to use LaTeX in future assignments. We recommend that you use the following packages:
  - Algorithm2e package for pseudocode
  - Minted for typesetting code
  - Tikz/PGF for graphs and figures
- **Hand in your written answers as a PDF file through Canvas**
- Hand in all program source files *softcopy* using the CSE webhandin and be sure that they properly execute in the webgrader. Failure to do so will mean that your program is not graded.

Partner Policy You may work in groups of at most two students. This is optional and you may work alone if you wish. If you opt to work in pairs, you must follow these guidelines:

1. You must work on *all* problems *together*. You may not simply partition the work between you.
2. You should not discuss problems with other groups or individuals beyond general questions.
3. Hand in only one hard copy (and possibly soft copy) under the first author's name/cse login.

Programming Instructions: you are *highly encouraged* to solve the programming problems using a high level programming language that you are not already familiar with (C++, PHP, Python, C# etc.); however, you **may not use Java**. Your program will be required to compile and run on the CSE server from the command line. Because we are allowing a variety of languages, you are *required* to also provide a script file named `run.sh` that contains the commands to compile and execute your program passing any command line arguments on to your program. Several examples have been provided.

You are **required** to hand in all source file, test cases, and other artifacts necessary to compile and run your program through webhandin. Moreover, you are **required** to rigorously test your programs locally and ensure that they work. Your program is required to work through the course webgrader as well. If your program fails to run, you may receive a zero.

Your programs will be evaluated based on the following:

- 10 pts – You must provide at least 2 *non-trivial* test cases to show that you properly tested your program (test cases each include a plaintext input file and output file with the formatting and naming specified for each problem)
- The remainder of the points will be awarded based on the correctness of your program. We have provided numerous test cases (if you find an error with any, please notify us and you will be awarded bonus points). You should use these test cases to debug and test your programs. The correctness of your program, will be evaluated using these test cases as well as your classmates' test cases. Points will be awarded based on the proportion of test cases you successfully pass. If *your* test case(s) are found to be in error, you will lose points for your test cases.

1. $\boxed{\text{5 points}}$ (Levitin 2.2.6)
   (a) Prove that every polynomial of degree k,$p(n) = a_k n^k + a_{k-1}n^{k-1} + \cdots + a_0$ with $a_k > 0$ belongs to $\Theta(n^k)$.
   (b) Prove that exponential function α^n have different orders of growth for different bases $\alpha > \beta > 0$.

2. $\boxed{\text{10 points}}$ In computability theory, it is well-known that a queue provides provably more computational power than a stack (which also provides provably more power than a finite-state machine without a stack). You will demonstrate this at least partially by showing the following.
   (a) Demonstrate that a queue (FIFO) can simulate a stack (LIFO) by giving pseudocode for the basic *push* and *pop* operations in terms of a queue's *enqueue* and *dequeue* operations.
   (b) You (provably) cannot demonstrate the converse (that a single stack can simulate a queue). Instead, demonstrate that *two* stacks can simulate a queue by giving pseudocode for the basic *enqueue* and *dequeue* operations in terms of the two stacks' *push* and *pop* operations.

3. $\boxed{\text{10 points}}$ Suppose that you are given a Binary Search Tree (though there is no guarantee on the tree's depth, d). Design an algorithm that sorts a collection $A = \{a_1, \ldots, a_n\}$ using the BST. Give full pseudocode and rigorously analyze your algorithm with respect to the depth d and/or the size of the collection, n in the *worst case.*

4. $\boxed{\text{10 points}}$ Let $G = (V, E)$ be an undirected graph. Design an algorithm that, given two vertices $u, v \in V$, determines *how many* distinct simple paths exist between u and v (simple paths may not traverse a vertex more than once). Give a complete worst-case analysis of your algorithm.

5. $\boxed{\text{25 points}}$ **Program A** Implement an algorithm to solve the a variation on the weather station problem mentioned before. Note that it will be necessary to implement a better (non-brute force) algorithm or use a smart data structure as a brute force solution will not be feasible. The variation will be as follows. First, your program will accept two file path/names from command line arguments representing data from each station. The format of the files will be as follows. On the first line, will be the total number of data points contained in the file. On each subsequent line will be a formatted date-time stamp followed by a space with a temperature value. For example:

```
5
2017-01-01T00:00 32.78
2017-01-01T01:00 18.97
2017-01-01T02:00 -24.92
2017-01-01T03:00 -25.40
2017-01-01T04:00 -26.70
```

You will process both files (which may not necessarily be in chronological order) and instead of aggregating and reporting all data points, you will only report data points in which either a) one of the stations has missing data or b) the two stations have data, but it disagrees. Your output should look something like the following.

```
Inconsistent Data (2017-01-01T00:00): A: 32.78 B: 32.9
Missing Data (2017-01-01T08:00) in data set A but not in B
Missing Data (2017-01-02T07:00) in data set B but not in A
```

Name your test case files:

- `input001.a.txt`
- `input001.b.txt`
- `output001.txt`
- `input002.a.txt`
- `input002.b.txt`
- `output002.txt`

respectively.

6. $\boxed{\text{70 points}}$ **Program B** Let $G = (V, E)$ be an undirected graph. A *Hamiltonian path* is a path that visits every vertex $v \in V$ exactly once. You will write a program that will read in a representation of a graph from a plain text file with the following format: the first line is n, the number of vertices in G which are numbered 0 through $n-1$. Each line after that contains a sequence of space-delimited numbers. The first number is a vertex v and each following number is a vertex that v is connected to. This is an undirected graph so the symmetric edges may or may not be present in the representation. An example is provided below.

```
9
0 3 1
1 4 0 2
2 5 1
3 0 6 4
4 1 7 3 5
5 2 8 4
6 3 7
7 4 6 8
8 5 7
```

Your program will accept a file name from the command line arguments, parse the graph representation and determine whether or not there is a Hamiltonian path in the graph. If not, your program will output `No Hamiltonian Path` otherwise it will output a sequence of vertex numbers that represent a Hamiltonian path. Note that there could be multiple Hamiltonian paths so your output might find a different Hamiltonian path. For the example above, one such solution could be:

```
Hamiltonian Path: 0 3 4 1 2 5 8 7 6
```

Name your test case files:

- `input001.txt`
- `output001.txt`
- `input002.txt`
- `output002.txt`

respectively.

The graph from the input can be drawn as depicted in Figure 1 with the discovered Hamiltonian path highlighted (one of many).

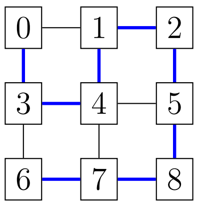

Figure 1: Graph with Hamiltonian Path

7. $\boxed{\text{70 points}}$ **Program C** Let $G = (V, E)$ be an undirected graph. Recall that a *clique* is a subset of vertices, $C \subseteq V$ that are all connected (a complete subgraph). That is,

$$\forall x, y \in C[(x, y) \in E]$$

The maximal clique problem is: given a graph $G = (V, E)$, find a clique of maximal size (there may be many, you need to report at least one).

Write a program that will read in a representation of a graph from a plain text file with the same format as before and outputs a list of vertices (in order) that form a maximal clique (as well as the size of the clique). For the same graph as before, the output should look something like the following.

```
Maximal Clique Size: 2
Clique: {0, 3}
```

Name your test case files:

- `input001.txt`
- `output001.txt`
- `input002.txt`
- `output002.txt`

respectively.