Due on March 18, 2019
61 points total

General directions: We will exclusively use Python 3 for our programming assignments, and allow only the use of modules in the Python 3 standard library unless explicitly specified otherwise on an individual assignment basis. This forbids the use of common third-party libraries such as Numpy, Sympy, etc., but not the use of math or io.

Unless specified otherwise, for the X-th homework, download the single "hwX_skeleton.py" file from the course website, and rename it to "hwX.py" on your machine. When you are done and ready to submit, upload your file named **exactly** "hwX.py" on Gradescope for assignment "HW X (Programming)." When you upload your file, the autograder will run a simple test for each function so that you can confirm it was properly uploaded and executed. Generally, if an assignment involves printing or writing a file in a specific format, there will be at least one simple test that checks your output is formatted as we expect. These tests are not worth any credit — once the due date is over, your submission will be graded by a collection of additional test cases.

All answers to non-programming questions must be typed, preferably using LaTeX. If you are unfamiliar with LaTeX, you are strongly encouraged to learn it. However, answers typed in other text processing software and properly converted to a PDF file will also be accepted. To submit your file, upload your PDF on Gradescope for assignment "HW X (PDF)." Handwritten answers or PDF files that cannot be opened will not be graded and will not receive any credit.

Finally, please read the detailed collaboration policy given on the course website. You are **not** allowed to discuss homework problems in groups of more than 3 students. **Failure to adhere to these guidelines will be promptly reported to the relevant authority without exception.**

Point values: Every problem has a specified amount of points which are awarded for the correctness of your solutions. In addition, each proof-oriented problem has an additional **style point**. In the homework handout, this is signified by a "+1" in the point value. To earn this point, your solutions should be clear, well organized, and easy to follow. This is to encourage not only perfectly correct solutions, but well presented ones.

Problem 1 (10+1 points)
Prove the following by induction: the number of ways to order n people is the product of the first n positive integers.

Problem 2 (10+1 points)
For any node u in a binary tree, let $A[u]$ denote the value stored at u. A *binary min-heap* is a rooted binary tree with the following property: if u is the parent of v, then $A[u] \leq A[v]$.

Prove that the minimum value in a binary min-heap is stored at the root.

Problem 3 (20+1 points)
A *walk* in an undirected graph is a sequence of vertices such that there exists an edge between every two consecutive vertices in the sequence. A walk *traverses* an edge if its endpoints appear consecutively in the walk. An *Eulerian circuit* is a walk that traverses each edge exactly once, and starts and ends at the same vertex.

Let $G = (V, E)$ be an undirected connected graph with n vertices and m edges. Prove the following by induction: G has an Eulerian circuit if and only if the degree of every vertex in G is even.

Problem 4 (18 points)
Programming: In Lecture 13, we defined a graph as an ordered pair (V, E) where V is a non-empty finite set and E is a set of two-element subsets of V. Thus, one way to represent a graph is by explicitly listing its vertices and edges. However, in practice, the two most common representations are the *adjacency list* and *adjacency matrix* representations, which we define below.

Let $G = (V, E)$ be a graph with n vertices. For this problem, we assume $V = \{0, 1, \ldots, n-1\}$. Recall that for all $u \in V$, we say vertex v is a *neighbor* of u if $(u, v) \in E$. The *adjacency list* representation of G is a list L of lists of vertices such that $L[u]$ contains the neighbors of vertex u. The *adjacency matrix* representation of G is an $n \times n$ matrix M where $M[u, v] = 1$ if v is a neighbor of u, and 0 otherwise.

For this problem, you will implement three functions as documented in "hw6_skeleton.py" which is on the course website as usual: `getAdjacencyList(V,E)`, `getAdjacencyMatrix(V,E)`, and `getDegreeList(V,E)`. Note that we have also included a handful of helper functions and examples that you can use to test your code. For more details, see the skeleton file.