Problem 1 (20 points)

Given an array $\{a_1, a_2, \cdots; a_n\}$, a reverse is a pair (a_i, a_j) such that $i < j$ but $a_i > a_j$. Design a divide-and-conquer algorithm with a run-time of $\mathcal{O}(n\log(n))$ for computing the number of reverses in the array. Your solution to this question needs to include both a written explanation and a Python implementation of your algorithm, including:

(a) *Explain how your algorithm works, including pseudocode.*

The run-time of this algorithm has to be $\mathcal{O}(n\log(n))$, and the algorithm has to be a divide-and-conquer algorithm, so I took a hint and decided to build my algorithm off of the Merge-Sort Algorithm (since it has both the run-time requirement and the algorithm type requirement). Thus, I'm going to have use the same reasoning as the Merge-Sort Algorithm. Here is the pseudocode for the Merge-Sort Algorithm (note that this pseudocode was taken from the book):

```
// The Merge-Sort Algorithm
Merge-Sort(A, p, r)
  1  if p < r
  2     q = floor((p + r)/2)
  3     Merge-Sort(A, p, q)
  4     Merge-Sort(A, q + 1, r)
  5     Merge(A, p, q, r)
```

I have to modify the above pseudocode

(b) *Implement your algorithm in Python.*

Here is my Python code, which is also included in `Letey-John-Final.py`:

(c) *Randomly generate an array of 100 numbers and use it as input to run your code. Report on both the input to your code and on how the output demonstrates the correctness of your algorithm.*

Problem 2 (25 points)

Suppose that you are assigned a task to do a survey about n important issues (such as education policy and health insurance mandate), by asking a group of m persons questions

about these issues. Suppose that a person may not have an opinion about all the issues, and you can ask a person about an issue only if s/he has an opinion about it. We use a bipartite graph $G = \{P \cup I, E\}$ to capture whether a person $p \in P$ has an opinion about an issue $i \in I$ or not: $(p, i) \in E$ means that p has an opinion about i. For each issue i, in order to have a reliable survey you need to ask at least l_i persons about it, but you may have certain budget constraint so that you can only ask at most u_i persons about it. For each person p, you may ask her/him between b_p and t_p issues.

Given G and parameters $(l_i, u_i), i \in I$ and $(b_p, t_p), p \in P$, design an algorithm to determine if these parameters are feasible, by formulating it as a problem of finding a routing with lower bounds as in Problem 1 of homework set #9. You shall solve the problem according to the following steps.

(a) *Show how to formulate the parameter feasibility problem as a problem of finding a routing with lower bounds. The resulting problem should be specified by certain graph $G' = \{V', E'\}$ with capacity $c(e)$ and lower bound $l(e)$ for each edge $e \in E'$ and demand $r(v)$ at each vertex $v \in V'$.*

(b) *Further formulate the problem as a maximum flow problem as in Problem 1 of homework set #9. The resulting problem should be specified by certain graph $\hat{G} = \{\hat{V}, \hat{E}\}$ with source s, sink t and capacity $c(e)$ for each edge $e \in \hat{E}$.*

(c) *Implement $(a) - (b)$ in Python. Your code should take the graph G and parameters $(l_i, u_i), i \in I$ and $(b_p, t_p), p \in P$ as the input, and produce the graph $\hat{G}$ with source s, sink t and capacity $c(e), e \in \hat{E}$ as the output.*

(d) *Further implement the Ford-Fulkerson Algorithm in Python to find the maximum flow from s to t over the graph $\hat{G}$.*

(e) *Generate a test case of parameters according to the following specifications, and run your code to see if the parameters generated are feasible.*

- *The number of issues $n = 10$ and the number of person $m = 1000$;*
- *For any person p and for any issue i, s/he has a probability of 50% to have an opinion about the issue, i.e., there is a 50% probability that there is a link from p to i in the graph G;*

- *For any person p, denote h_p the number of issues that s/he has an opinion about. Let $b_p = \lfloor h_p/2 \rfloor$ and $b_p = h_p$.*
- *For each issue i, l_i is drawn uniformly from the interval [300, 400] and u_i uniformly from [500, 700]*

Problem 3 (10 points)

Suppose you have been sent back in time and have arrived at the scene of an ancient Roman battle. It is your job to assign n spears to n Roman soldiers so that each soldier has a spear. It is best if your assignments minimize the difference in heights between the height of the man and the height of his spear. That is, if the i^{th} man has height m_i, and his spear has height s_i, then you want to minimize: $\sum_i \mid m_i - s_i \mid$

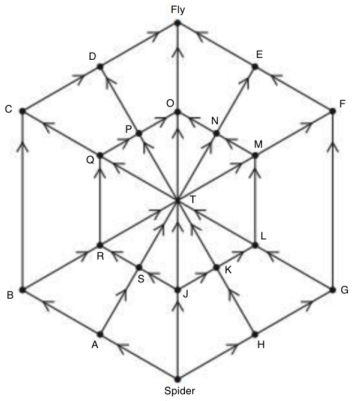
(a) *Design algorithm to find the optimal, or near optimal, solution without evaluating all possible combinations. Include an explanation and pseudocode showing how your algorithm works.*

(b) *Compare the runtime complexity of your algorithm with the complexity of a brute force solution.*

The brute force solution for this algorithm would have a runtime of $\Theta(n!)$.

Problem 4 (20 points)

Consider the following spider-web graph that shows a spider siting at the bottom of its web, and a fly sitting at the top. On moodle, there is a file called graphExample.py that implements the graph using a library called NetworkX.



(a) *Write an algorithm to determine how many different ways can the spider reach the fly by moving along the webs lines in the directions indicated by the arrows?*

(b) *Implement your algorithm in Python using the NetworkX graph provided as your data structure. You may need to install NetworkX if it isnt part of your Python installation. Do not use any of the NetworkX features that would make this problem trivial as part of your solution. However, you can use anything in NetworkX to verify your solution. Your algorithm should return an answer to the question in part (a).*

Problem 5 (25 points)

There are $n \geq 3$ people positioned on a field (Euclidean plane) so that each has a unique nearest neighbor. Each person has a water balloon. At a signal, everybody hurls his or her balloon at the nearest neighbor. Assume that n is odd and that nobody can miss his or her target.

(a) *Write an algorithm to answer the question: Is it true or false that there always remains at least one person not hit by a balloon?*

(b) *Implement your algorithm in Python such that it takes a data structure of people and distances and produces a data structure of who was hit by a balloon.*

(c) *Prove that your algorithm is correct. Your proof needs to include specific features of your algorithm.*

(d) *Analyze the runtime behavior of your algorithm.*