John Letey (12/11) — john.letey@colorado.edu

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)
  if p < r
     q = floor((p + r)/2)
     Merge-Sort(A, p, q)
     Merge-Sort(A, q + 1, r)
     Merge(A, p, q, r)
```

I now have to modify the above pseudocode to make the algorithm count how many reverses there are in the array. I also have to modify the pseudocode, so that it returns the count instead of just being a void function. There's not a lot I need to change in the `Merge-Sort` algorithm, but I do need to change a lot in the `Merge` function. In fact, I will completely be redesigning the `Merge` function.

(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)