CS2 PA2 Reflection:

1. Consider the data structures you created and chose for your implementation of Dijkstra's Algorithm. Why did you choose them? (Ease of implementation is a perfectly valid reason – don't be afraid to include it.)
    a. I chose to use various arrays for my implementation of Dijkstra's Algorithm. I chose these because of the similarity to how the input file is read and how I would be writing to my output file. This was also partly done to save some time debugging and fixing my own data structures and use that time to work on the actual algorithm implementation.

2. Given the data structures you chose, explain what worst-case and average running time (in asymptotic terms) your program should have.
    a. The worst-case runtime for my implementation is O(n^2). In my implementation I have two *for loops* nested inside of another *for loop*. These loops will make the runtime n+m. The outside *for loop* will multiple its runtime by the internal runtime, resulting in:
        i. n*(n + m) -> n^2 + n*m -> use the highest term = n^2.

    b. The average case runtime for my implementation is still O(n^2). This is unfortunately because the steps where we may save time are not the dominate terms big-o notation uses. Ea represents the average edges a vertex has which is not always the max like m above, but it is still nested inside of a *for loop* with another *for loop*.
        i. n*(n + 2*Ea) -> n^2 + 2*n*Ea -> use the highest term = n^2.

    c. This would be improved if I were to implement an adjacency list. This way I would be able to better traverse the input data and make more *greedy* decisions. In my current implementation, I am repeatedly searching over vertices resulting in unnecessary time spent. An adjacency list would allow me to visit a vertex once and not have to traverse it again, leaving me with only my nested *for loops* and getting rid of the outside n*(…) resulting in an average runtime improvement to O(n*Ea), in theory.

3. Given the data structures you chose, explain what worst-case and average memory space (in asymptotic terms) your program should require.
    a. The worst-case memory space of my implementation is O(n*E). This is because each vertex is contained in an array and each vertex has its own array of edges, which in the worst case, could contain all the edges, like when the input is a complete graph.
    b. The average case memory space is O(n*Ea) because not every vertex will contain every edge in most graphs, i.e., most inputs will not be complete graphs.
    c. This memory space could be improved if I used a different edge data structure. Currently, the edges are stored within vertices, resulting in a multiplication of space. If I were able to reference all edges from outside of the vertex, I can see a possibility of averaging O(n+E) memory space. I can see this working in conjunction with my adjacency list idea for improving time complexity, getting rid of the arrays within each vertex and getting the linear O(n+E) memory space.

Citations:

- What space complexity is – contributed by GeeksforGeeks - available at:
  - https://www.geeksforgeeks.org/g-fact-86/#
- How to measure space complexity – written by Muhammad Ahmad - available at:
  - https://www.educative.io/answers/what-is-space-complexity-of-an-algorithm-and-how-it-is-measured
- Dijkstra's algorithm – numerous Wikipedia contributors – available at:
  - https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
- File writer details in java – written by *goelshubhangi3118* - available at:
  - https://www.geeksforgeeks.org/filewriter-class-in-java/
- Time complexity slides – provided by various UCF professors including Tanvir Ahmed and Neslisah Torosdagli
  - Not publicly available