Programming Project 3 Report:

Vinay Shah, vss452

Part 1:

a) Opt(planet, fuel_cost) = $\min_{0<=k<=flights\_from\_planet}$ {flight_time + Opt(flight_dest, current_fuel – fuel_cost)}
This should give the minimum travel time required for the path from source to destination. This solution shows up in the row of the start planet.

b)
```
computeMinimumTime():
   opt[numPlanets][totalFuel+1]
   for each j in totalFuel + 1:
      for each i in numPlanets:
         minTime <- INFINITY
         for each flight in adj(i):
            if flight.fuel <= j
               then if opt[flight.dest][j-flight.fuel] < INFINITY
                  then minTime <- min{flight.time + opt[flight.dest][j-flight.fuel], minTime}
            else if i == end_planet
               then minTime <- 0
         opt[i][j] = minTime

   return min_{0<=k<totalFuel+1}{opt[0][k]}
```

c) Since there is a 2D matrix of our solutions that we have made, and we traverse it as the algorithm goes on, it takes at least $O(n^2)$ time to traverse and fill it. And for each cell in the matrix, we are performing O(1) operations on the number of edges for each planet. Worst case, this will be O(n), where the for loop will run for edges to all the planets. $O(n) * O(n^2) = O(n^3)$ time to run the solution.
It takes $O(n^2)$ space to store the Opt matrix.

Part 2:

a) Opt(i, j) = $\max_{0<=k<=j}\{f_i(k) + Opt(i-1, j-k)\}$
This find the maximum damage dealt for a total time j. It adds the previously computed damage that works with the time constraint to the current damage and adds that to our opt matrix. The solution will show up in the bottom right corner of the matrix.

b) computeDamage():

```
opt[numAttacks][totalTime+1]
for each i in numAttacks
   for each j in totalTime+1
       maxDamage <- 0
        for each k in and including j
          if i==0
            then maxDamage <- damage[i][j]
          else
             maxDamage <- max{damage[i][k] + opt[i-1][j-k], maxDamage}
        opt[i][j] = maxDamage

return opt[numAttacks-1][totalTime]
```

c) In addition to the other values, we also store the value k for which the maximum damage was recorded. We can then use this value to backtrack our solutions starting from the bottom right corner of the matrix. We look at its entry and the value of k we recorded for it, and then look at the previous value which is opt[i-1][j-k] for that particular k and repeat until we have found the list of the optimal solution.

d) There are 3 nested for loops that run for O(n) time each, with the inner most loop having O(1) operations. The total time complexity is $O(n^3)$. The space complexity is $O(n^2)$ as it utilizes an opt matrix to store all the values.