Projects:

* Assumptions:
  + There are two cases; one with one elevator, and another with two elevators.
  + There are exactly six floors in each building.
  + Any given elevator can hold between zero to six passengers.
  + One new rider is added after every one tic, while the elevator is moving.
  + Initial direction is up, and we’re beginning from the lobby, which is floor zero.
  + The elevator begins in lobby and work its way up. When it reaches the top, it will work its way down, picking up any passengers along the way that are created.
* References:
  + Elevator Simulator Design: <http://www.angelfire.com/trek/software/elevator.html>
    - The team relied on this algorithm to help point us in the right direction. We by no means copied it, but we did take some ideas from it. In the original algorithm there are three scenarios:
      * Passengers moving from the lobby to the higher levels.
      * Passengers moving from the upper floors to the lobby.
      * Passengers moving in between the floors that are not the lobby and top floor.
    - We followed the general example above to help us narrow down what should go where in regards to our logic for moving the elevators from floor to floor. Another concept that we borrowed from the example algorithm was how to set up the elevator class in relation to the rider/passenger class.
      * The example algorithm we stated above uses a passenger class that only holds the destination floor. We went a step further and added to the class the starting floor as well as a wait time. The purpose of adding the weight time was so that we could calculate the average weight time after the simulation ends.

**Efficiency of algorithm**

1. increment\_wait\_time(): The big(O) of this function is O(1) because it will always execute in the same time regardless of the size of the input data. And there are no loops nor comparisons involved in this function.
2. Rider():this is the constructor of the Rider class, The big(O) of this function is O(1) because it will always execute in the same time regardless of the size of the input data. And there are no loops nor comparisons involved in this function.
3. Elevator (): This is the constructor of the Elevator class, the big(O) of this function is O (1) because it will always execute in the same time regardless of the size of the input data. And there are no loops nor comparisons involved in this function.
4. Incerement\_list\_wait\_time(): The big(O) of this function is O(N), this function is just iterating through the Rider list and getting the wait time for each rider so the big(O) will be O(N), it will be the size of the list.
5. Go\_up():this function does not return anything and it is only incrementing the location to move the location of the elevator up. So, the big(O) is O (1) because it will always execute in the same time regardless of the size of the input data.
6. Go\_down():this function does not return anything and it is only decrementing the location to move the location of the elevator down. So, the big(O) is O (1) because it will always execute in the same time regardless of the size of the input data.