Homework 3

1)

To solve this problem, we have to create a bipartitite, directed acyclical graph.
- It must be bipartite because the graph has two kinds of nodes, birth and death nodes, and one must come after the other.
- The direction implies that the node the arrow is coming from happened before the node the arrow is point towards
- It must be acyclical because if there were a cycle, it would mean that someone died before they were born.

If the graph is a DAG, then we can say that the graph has a toplogical ordering. This topological ordering would provide us with a series of birth/death events that would make sense. If we can show that one of these exists, then we can show that the data we have is possible.

Since the only thing we've determined could cause the topological ordering to fail is if the graph has a cycle, then if we can prove that the graph is cyclic, we can prove that the data must be false.

```
// Make all of the nodes for each person, and create an edge for the birth to the death
for person in people {
    make_node(person.birth);
    make_node(person.death);

    make_edge(person.birth, person.death);
}

// Make an edge for each person j who died before person k
for (person_j, person_k) in died_before_birth_data {
    make_edge(person_j.death, person_k.birth);
}

// Make an edge for each time two people coexisted
for (person_j, person_k) in coexisted_data {
    make_edge(person_j.birth, person_k.death);
    make_edge(person_k.birth, person_j.death);
}

order = 0
average_time_between_node = 200/len(make_node_list);
start_time = 0;

for person in people {
    if person.birth.jumps == 0 {
        make_node_list.push(person);
    }
}

while !empty(make_node_list) {
    node = make_node_list.pop();
    order++;
    date[node] = start_time + order*average_time_between_node;
    for next_node in adjacency_list[node] {
        next_node.jumps--;
        if next_node.jumps == 0 {
            make_node_list.push(next_node);
        }
    }
}

if (order < abs(len(people)) {
    possible = false;
} else {
    possible = true;
}

return possible;
```

This algorithm would provide a basic idea of whether the data was possible.

2)

There are two ways you could approach this solution.
- If you move all the jobs over a set amount, there there is a period that no two jobs intersect, you could set that as time 0, and run your usual greedy algorithm with earliest end time first. This would have a runtime of O(nlogn + n + n), since you have to go through all the jobs and determine if there is a time that no jobs are in the middle of running. This does not work if there are no such times like this.
- If you take all jobs that run over midnight, and remove all but one, testing how many jobs can be run with that (through the same greedy algorithm above), and repeat until we've tested all possiblities, then select the one that was most productive, we can have a runtime of O(nlogn + n).