Part 1

In your report, describe briefly how you have represented the graph (adjacency list, adjacency matrix, directed, undirected, etc) and justify the choices you have made. You should consider the remaining parts of this assignment before making decisions on your graph representation.

An adjacency list was used to represent the Graph of houses. In java this was accomplished by creating an Arraylist of Linkedlists. The first element of each Linkedlist represented a node or vertex in the graph and subsequent elements in the Linkedlist represented directed edges from the first node to those nodes. In this case the total memory that was required to represent the graph was O(n+m) where n = # of nodes and m = # of edges. This is less than the memory required to construct an adjacency matrix which would use $O(n^2)$ memory. However checking if a particular edge exists requires iterating through the Linkedlist of a single node which would take O(n) time. In comparison, such an operation would require checking a single element in an adjacency matrix form which would take O(1) time.

In order to implement an adjacency list, an Arraylist of Linkedlists of House objects was used to represent the Graph. Each House object contained information regarding that node including the name of the house, the total number of keys for the house, the keys obtained for the house (used for Part 2), the locked/unlocked state of the house, and whether the house had been visited (used for Part 2).

```
public class House{
    private String Name;
    private int NumKeys;
    private int KeysObtained = 0; //initial value
    private boolean Unlocked;
    private boolean Visited = false; //initial value
}

private ArrayList<LinkedList<House>> Graph; //Construct Graph of Houses
```

Part 2

In your report describe briefly your algorithm, its runtime, and its space complexity. Justify your answers.

Before checking if the robber could rob all of the houses the Graph was sorted based on whether the Houses was unlocked and the number of keys it had (ie first element in Arraylist was unlocked and last element was locked and had greatest number of keys). Sorting was accomplished by using insertion sort which takes $O(n^2)$ time. My algorithm for determining if all the houses could be robbed was as follows:

```
While VisitedAtLeastOneHouse == TRUE:
    For all LinkedLists in Graph:
        If House is Univisited:
            If House is (Unlocked) OR (KeysObtained == NumKeys):
                Visit the House
                Add House to RobbingList
                For each key found in House:
                    For each house in the Graph:
                        If house is same as the house for the key found:
                            Increment house's KeysObtained
```

```
For all LinkedLists in Graph:
    If House is Unvisited:
        Return False

Print out RobbingList
Return True
```

This takes $O(n^3)$. Thus the total time for the algorithm is $O(n^3)$. By visiting all of the unlocked houses first and then visiting locked houses in order of the keys needed to break into that house, and successively finding keys for other houses while robbing the list in which to rob the houses can be determined. If the robber could not visit all of the houses then the algorithm returns false. Otherwise it prints the order in which to rob them.

Part 3

In your report describe your algorithm and its runtime. Will your algorithm work if Fruitcake makes a list of the electronic devices (TVs, laptops, phones, etc) in the house, assuming he cannot steal a fraction of any of the devices? Why or why not?

In order to maximize the value of the loot the robber can implement a greedy algorithm which sorts the items by value and takes the items in order of highest value until he has reached his carrying capacity. Using insertion sort which takes $O(n^2)$ time the list of items was sorted by its value in decreasing order. Then the greedy algorithm was implemented as follows:

```
AmountLeft = Carrying Capacity
For sorted items in Loot List
    If(AmountLeft > AmountofItem):
        Take all of item
        AmountLeft = AmountLeft - AmountofItem
    Else:
        Take AmountLeft of Item
        Break
```

This takes $O(n)$ time. Thus the total time for the algorithm is $O(n^2)$. The algorithm would work with electronic devices if minor changes are made so that the robber can not take a fraction of the device. Assuming the change is made it will still work because it will optimize value of the loot by taking items which are of most value and can be carried.

Part 4

In your report describe your algorithm and its runtime. Prove that your algorithm is correct.

In order to maximize the number of meetings that the robber can attend a greedy algorithm was also implemented. By attending meetings which have the earliest end time the robber would be able to maximize the number of meetings he can attend. My algorithm was implemented by first parsing through the list of meetings and converting each time to a decimal representation (ie 11:15 is 11.25 and 1:45 is 13.75). Then the meetings were sorted by end times using insertion sort which takes $O(n^2)$ time. The algorithm to decide which meetings to attends was implemented as follows:

```
prevEndTime = 0; //intial
For each Meeting sorted by End time:
    If(Meeting’s startTime > prevEndTime):
        Attend the meeting
        prevEndTime = Meeting’s endTime
    Else:
        Don’t attend the meeting

For each Meeting:
    If(Meeting is to be attended):
        Print out buyer’s name
```

This take O(n). Thus the total time for the algorithm is $O(n^2)$. We can prove the greedy algorithm’s correctness by contradiction.

Assume the greedy algorithm is not optimal. Let i1, i2, . . . i_k denote the set of jobs selected by the greedy algorithm. Let j1, j2, . . . j_m denote the set of jobs in the optimal solution, where i1 = j1, i2 = j2, . . . , $i_r = j_r$ for the largest possible value of r. So job i_r+1 in the greedy solution finishes before job j_r+1 in the optimal solution. We can replace job j_r+1 with job i_r+1 without affecting the optimality of the remaining solution. This proof was taken from the course slides.