Q1: Algorithm goes over each hotel(i) and for ever hotel we consider the possible paths before that hotel(j). Implementation is very similar to what we did in class(OPT pseudo code). We have n subproblems and each of them takes $O(i)$ time, so we get :

$$\sum_{i=1}^{n} O(i) = O(\sum_{i=1}^{n})i = O\frac{n(n-1)}{2} = O(n^2)$$

Q2: In this part there are two key functions that does the heavy work. First function is *isToken (mapOfTokens, value)*. It takes dictionary and a value and checks if the value exists in the dictionary. If it exist returns it return "True". Takes $O(n)$ time n being the size of the provided dictionary. It's a basic linear search algorithm. Second function is *validation(pattern, string, token).* It's a recursive function. Algorithm of this function is:

```
//Base case
-If string is empty AND token is not empty AND token is in the dictionary
     -Return True
-If token is in the dictionary AND string is not empty
     -token = []
     -validation(pattern, string, token)
-If string is not empty
     -token.append(string.pop()) //Pop like in a stack structure.
     -validation(pattern, string, token)
-Else
     -Return False
```

Worse case occurs when there is no match for a given token in the dictionary or the provided string is only a single word. In this case we go through all the dictionary, which takes $O(n)$ time, for each token constructed , size of the string $O(m)$. Therefore resulting in $O(m.n)$.

Example run:

```
String: "tobeornottobe" (13 characters)
Dictionary size: 100 pairs
t -> checkDictionary -> O(n)
to-> checkDictionary -> O(n)
…
tobeornottobe-> checkDictionary -> O(n)
```

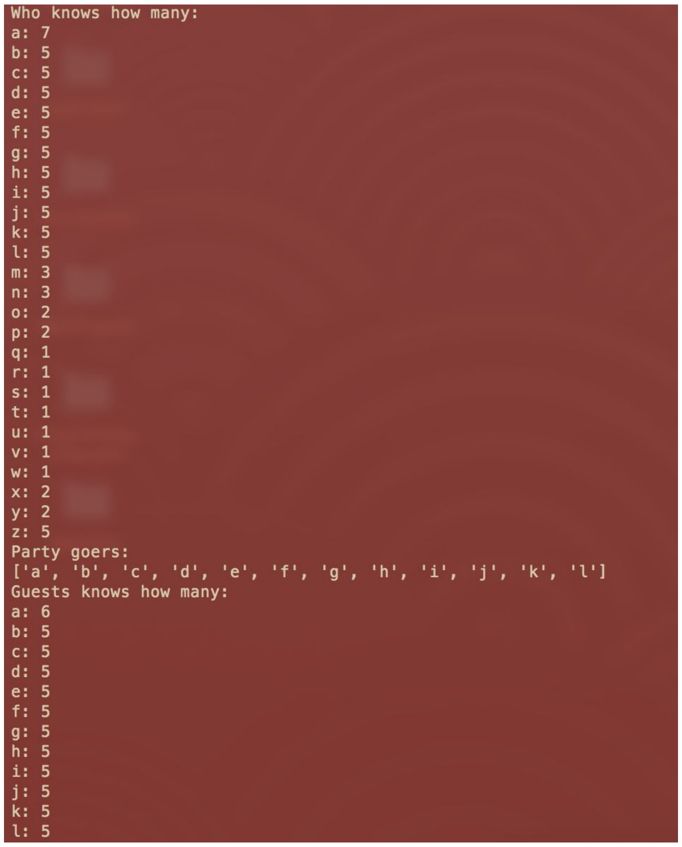
Q3: This problem is very similar to merge part of the mergeSort algorithim. Only difference is subarrays are already sorted and only thing left to do is to merge them. We have k subarrays with each has n elements. We do $O(kn)$ work to merge the k arrays of size n into k/2 arrays of size 2n, and then continue doing $O(kn)$ work $O(logk)$ times until we have a single array of size kn. Thus, the running time of this approach is $O(klognk)$.

Example run:

```
arr =[[1, 3, 5, 7],[-1, 1, 2, 30],[10, 20, 24, 25],[-100,-50,-25,100]]
arr[1] merged into arr[0]
arr =[[-1, 1, 1, 2, 3, 5, 7, 30],[10, 20, 24, 25],[-100,-50,-25,100]]
arr[2] merged into arr[3]
arr =[[-1, 1, 1, 2, 3, 5, 7, 30], [-100,-50,-25, 10, 20, 24, 25, 100]]
arr[1] merged into arr[0]
arr =[-100,-50,-25,-1, 1, 1, 2, 3, 5, 7, 10, 20, 24, 25, 30, 100]
```

Note: Merged parts of the array is removed in this visualization to keep it clear.

Q4: We can approach this problem as a graph problem. People are vertices(V) and relations between them are edges(E). To acquire the invite list we should remove people from the graph that doesn't know at least 5 other people or knows more than n-5 people, n being number of vertices. We keep removing from the V's and E's until all people meets the requirements. Example :



```
Who knows how many:
a: 7
b: 5
c: 5
d: 5
e: 5
f: 5
g: 5
h: 5
i: 5
j: 5
k: 5
l: 5
m: 3
n: 3
o: 2
p: 2
q: 1
r: 1
s: 1
t: 1
u: 1
v: 1
w: 1
x: 2
y: 2
z: 5
Party goers:
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l']
Guests knows how many:
a: 6
b: 5
c: 5
d: 5
e: 5
f: 5
g: 5
h: 5
i: 5
j: 5
k: 5
l: 5
```

All relations can be seen in the q4.py file.

Complexity: $O(n^3)$

Q5: Task is very simple if we follow a greedy methodology. We have n constraints and m variables. For each triplet of a constraint(index1, index2, “constraint”). We check directly access to index1 and index2 of variables and make a comparison that takes constant time. To generate index1 and index 2 we iterate through constraints array. So in total we iterate through constraints array(n) for each constraint, resulting in $O(n^2)$. Number of variables doesn’t matter in this algorithms since access time is constant and we don’t iterate through the variables.

Example :

```
Vars = [1, 1, 5, 5, 4]
Constr =[[0, 1, '=='], [1, 2, '!='], [2, 3, '=='], [2, 4, '!=']]
True
Vars = [1, 1, 4, 5, 4]
Constr =[[0, 1, '=='], [1, 2, '!='], [2, 3, '=='], [2, 4, '!=']]
False
Vars = [1, 1, 4, 5, 4]
Constr =[[0, 1, '=='], [1, 2, '!='], [2, 3, '!='], [2, 4, '==']]
True
```