HOMEWORK 4 REPORT

PART1

Explanation of Algorithm: First we calculate the cost of the current hotel from the beginning and we add it to cost array. Then we check the minimum cost of the current hotel with another loop.If we find the better cost hotel we change it from the cost array.And also we keep the current hotel in a list for Show the sequence.

Time Complexity:

```
for i in range(0, len(A)):
    cost.append((200-A[i])**
    hotels.append(0)
    for j in range(0, i):
```

There are 2 nested for loops. First one is from 0 to n, and the second one is from 0 to i.That means our time complexity will be O((N^2)/2). And if we ignore constants, time complexity will be **O(N^2).**

PART2

Explanation of Algorithm: First i declared a boolean array which length is equal to string length + 1. Then i started from the beginning and check the current Word is in our dictionary or not.If i find it in the dictionary i marked it as True the current index.If i didnt find it in the dictionary i started to check again with Word that i added one more letter.I checked the whole string with it.If the last index of the boolean array is False, that means our string is not constructed with dictionary words.

Time Complexity:

```
def algo2(string):
    boolArr = [];       #Boolean Array for che
    for i in range(0,len(string)+1):    #Fill
        boolArr.append(False)
    boolArr[0] = True       #Mark first index
    for i in range(1,len(string)+1):    #Neste
        for j in range(0,i+1):
            if (boolArr[j] and dict(string[j:i
                boolArr[i] = True  #Mark as Tr
                break
    result=""   #result Word for reconstruct t
    for i in range(0,len(string)):  #Loop for
        result=result + string[i]
        if boolArr[i+1]==True:  #If there is a
            result=result + " "
    print("Reconstructed Document: ", result)
    return boolArr[len(boolArr)-1]
```

First, we assume the dict() calls are unit time.There are 4 loops in this code but only one of them is nested.Nested loop time complexity is O((N^2)/2) which is equal to O(N^2).Other for loops time complexity is O(N).Therefore, the nested loops will determine the time complexity because we ignore the low order terms.So our time complexity is **O(N).**

PART3

Explanation of Algorithm: Merge the array pairs (one from the beginning, one from the end) till the beginning is not less than end.When we do that our total array number is halved.Then i did this until only 1 array left.

Time Complexity:

```
def mergeK(arr):
    lastList=len(arr)-1 #Assign the lastList to a variable
    while(lastList!=0): #Loop until 1 list left
        i=0
        j=lastList
        while(i<j): #Loop until all pairs merged - Merge arr
            arr[i] = mergeTwo(arr[i], arr[j], [])
            i=i+1
            j=j-1
            if(i>=j):
                lastList=j
    return arr[0]  #Return 0th index because only 1 merged a


def mergeTwo(arr1, arr2, result):
    if not arr1 or not arr2:    #If one of the array is empt
        result = result + arr1 + arr2
    elif arr1[0]<=arr2[0]:
        result.append(arr1[0])
        return mergeTwo(arr1[1:], arr2, result)
    elif arr1[0]>arr2[0]:
        result.append(arr2[0])
        return mergeTwo(arr1, arr2[1:], result)
    return result
```

MergeTwo() function takes O(2N) time which is equal to O(N) because on every call we only process one element.The loops on mergeK() function takes O(log(k)) times because every iteration the length is halved and every iteration we call MergeTwo() function which complexity was O(N).Therefore, our time complexity is **O(Nlog(k)).**

PART4

Explanation of Algorithm: First i declare a dictionary with every person with the number of the person they know.Then i delete the persons from the list that not matching the requirements and when we delete them i decrease the person values by 1 who is known by him.I did this job until the everyone match the requirements.

Time Complexity:

```
def party(people, pairs):
    dict = {}   #Dictionary for which person know which of the other
    for person in people:   #Initialize the dictionary
        dict[person]=0
    for i in pairs:     #For every pair increase their dictionary va
        dict[i[0]] += 1
        dict[i[1]] += 1
    while(not(all(i>=5 for i in dict.values()))):   #Loop until find
        temp = []
        for i in dict:
            if dict[i]<5 or len(dict)-1-dict[i]<5:  #If someone know
                for x in pairs:
                    if x[0] in dict.keys() and x[1] in dict.keys():
                        if x[0] == i:
                            dict[x[1]] -= 1
                        elif x[1] == i:
                            dict[x[0]] -= 1
                temp.append(i)  #Add him to temp array for deleting
        for i in temp:
            del dict[i] #Delete from party list
    return " ".join(dict.keys())
```

We have 2 nested for loops which their total complexity is O(N^2) in while loop.If only 1 person deleting from party list from every iteration of the loop the time complexity of the function will be O(N^3).

PART5

Explanation of Algorithm: First, i split the constraints then i check the contraint is True or not. If its false i return False because we dont need the check the rest.I did this job until all constraints is finished.

Time Complexity:

```
def algos(variables, constraints):
    for x in constraints:   #For every constraints
        if "!=" in x:   #If there is inequality
            temp=x.split("!=")  #Split the string and find index
            if variables[int(temp[0][1:])]==variables[int(temp[1][1:])]:
                return False    #No need to check more
        else:   #If there is inequality
            temp=x.split("=") #Split the string and find index
            if variables[int(temp[0][1:])]!=variables[int(temp[1][1:])]:
                return False #No need to check more
    return True
```

The time complexity of the function will be O(N) because we only have a for loop that iterates at most O(N) times.