1.

```
maxProfit (n, k, int[] d, double[] P):
 initialize an array A[1,…, n]
 for index in range (1, A.length):
     A[index] = 0
 A[1] = P1
 lastOne = 1
 lastProfit = P1
 For index in range (2, A.length):
      If Math.abs(d[index] – d[lastOne]) < k:     // if two places are within K mile distance
          If lastProfit > P[index]:                    // then try to build one with the highest profit
                A[index] = lastProfit
          Else: A[index] = P[index]
                lastOne = index
                lastProfit = P[index]
     else: A[index] = lastProfit + P[index]
           lastOne = index
           lastProfit = P[index]

return A[A.length]
```

The time complexity of the above algorithm is O(n) and the space complexity is O(n)

Explanation: At first, declare an array A, this array is used to keep track of the cumulative profits of build restaurants. A[1] means build the restaurant in d[1] and the profit is P[1]. The loop starts from the second location in the highway, and it checks to see if the distance between the current location and the previously built restaurant is less than K, Since there is only one restaurant can be built within K range, I check to see if the current location's profit is greater than the last profit. if it is, I decide to build the restaurant in the current location instead of the previous one, because the profit is higher. If it is not, A[index] is assigned the value of lastProfit. In addition, if I found the distance between the current location and the previous chosen location is greater than K, I can decide to build the restaurant in the current location. When the loop finishes, I simply return the last element in the array, because this value is the maximum expected total profit.

2.
I assume that there are no redundant pairs and no persons have the same names.

```
socialParty (n, String[] pairs):
   declare a HashMap called store
   for pair in pairs:
       store[pair[0]] = store.get(pair[0], {}) + {pair[1]}
       store[pair[1]] = store.get(pair[1], {}) + {pair[0]}
   friends = []
   for key in store.keys():
      if len(store[key]) >= 4 and n – len(store[key]) >= 4:
          friends.append(key)

   notReduced = False
   overall = True
   while len(friends) >= 9:
         i = 0
         while i < len(friends):
             friend = friends[i]
             count = 0
             known = store.get(friend)
             notReduced = False
             for j in range(len(friends)):
                if i == j: continue
                if friends[j] in known:  count += 1

             if count >= 4 and len(friends) – count >= 4:
                    notReduced = True
             overall = overall and notReduced

             if ! notReduced:
               exchange the ith element with the last element in list called 'friends' and pop the last
   element.
             Else: i += 1
          If overall: break out of the outer most loop


   If len(friends) < 9:  Return 0
   Else: return len(friends)


   Time complexity is O(n^2) the worst case is everybody is removed from the 'friends' list
   Space complexity is O(n)
```

Explanation: At first declare a HashMap called store, then loop through the array pairs and use store to build an undirected graph. Each entry in the HashMap is the name of a person and the corresponding value is a set. The set contains the name of the persons that key knows about. The loop through the keys in the HashMap and if the key knows at least four persons and he does not know at least 4 persons. This key is appended to a list called 'friends'. Then loop through this 'friends' list, during each iteration, I try to find out does this person knows at least 4 persons and does not know at least 4 persons in this 'friends' list. If he does not, this person is removed from the 'friends' list. I will loop through this 'friends' list multiple times until I found nobody is removed from the list or the size of the list is less than 9. There is a Boolean variable called 'overall', if it is true after I loop through each object in 'friends' list, which means nobody is being removed from the list, no matter how many times I continue to loop through the 'friends' list, there will not be anybody being removed, so I break out from the nested loops and return the size of the 'friends' list. If the size of the list is less than 9, which means nobody knows at least 4 persons and does not know at least 4 persons, so my function returns 0.