1. For this algorithm, I want to use a divide and conquer technique with recursion to maximize the sum of happiness score from various dishes at the Olin Dining Hall. Constants that are given in the problem are as follows:
   a. Number of foods available = n
   b. Happiness value for item (i) = h_i

   To start with, I will create an array (Foods) containing each food available in the dining hall. I would then find the midpoint of this array. Now, I have two subarrays – one that starts on index 0 of Foods and ends on the midpoint of Foods, called LeftFoods + one that starts on the element next in index with respect to the midpoint of Foods till the last element of Foods, called RightFoods.

   The idea here is to recursively find the maximum happiness value subarray out of subarrays of LeftFood and RightFood as well as the subarrays of Foods that compulsorily include the midpoint of Foods as an element. The latter is a special case for this problem and can be dealt with by appending the maximum happiness value subarray from RightFood that includes its first element to the maximum happiness value subarray from LeftFood that includes its last element to find the maximum happiness value subarray that contains the midpoint of Foods. This can then be used for comparison with the RightFood and LeftFood maximum happiness value subarrays for finding the optimized solution. To clarify, the maximum happiness value for the subarray is calculated by adding the individual happiness value for all the foods in it.

   Hence, this algorithm needs a total of two functions – One function (Func1) to find the maximum happiness value subarrays for LeftFood and RightFood, and another (Func2) for the special case. This algorithm can find the optimal solution, because it goes through each possibility of creating subarrays comprising the foods, including those that produce negative returns.

   Runtime analysis for Func2:

   Since we find the happiness values for subarrays that always include the midpoint – the total number of iterations will essentially be equal to the length of Foods (the total number of food items in LeftFood and RightFood), hence the runtime will be linear, O(n).

   Runtime analysis for recursive Func1:

   Using the master method,

   T(n) = a*T(n/b) + f(n)

   Here, T(n) = O(n(log(n))), b = 2, since the sub of a subarray is half the size of the array, a = 2, since T(n) is called initially on 2 subarrays of Foods, and f(n) is the runtime for Func2, which is O(n).

   Resultant equation: T(n) = 2*T(n(log(n))/2) + O(n)

2. To develop this algorithm, I would like to use lists.
   The assumptions stated in the question are:
    1. The bug can only be passed during class time
    2. The patient 0 has been identified (I have called them as 0th student)
    3. A list of student's classmates is available

   To start with, the first batch of suspected cases of the plague would be the classmates of the 0th student. I will make a list called Schedule, which will contain all the classes that the 0th student goes to, starting from when the 0th student got the plague until we found out about this case. I will make a second list called Suspected, to collect the names of all the students who might have received the plague from another student in the period, and a third list called Check with the 0th student as its index 0. Starting with the first class on the Schedule of the 0th student, I would add all the classmates of the 0th students to the Suspected, as well as to Check (Only the students who are not already a part of these lists, otherwise the recursion that I have described later would never come to an end). Similarly, I will iterate through the whole Schedule of the 0th student and collect the names of all such suspected students. I would recursively call this function for each of the students in Check. The function would exit from execution as soon as it reaches the end of the Check list. The return value for the function would be the Suspected list.

3. Please see sbansal.py
4. The average value of the max interval is about 65, while the average length of the max interval is about 35. My conclusion for this observation is that each time, about a third of foods at the dining hall in a contiguous array bring the maximum level of happiness – which includes a lot of food items that don't produce any happiness or produce sadness.
   The maximum possible happiness level is 1000 (all foods give the maximum amount of happiness) while the minimum possible happiness level is about -1000 (all foods give the maximum amount of sadness). This means that a lot of the foods in the max subarray with positive happiness get cancelled out by those with negative happiness, since 65 is a very low score relative to the maximum.
5. Due to how the spacing of the happiness levels is being controlled in this section, using probabilities – with a higher probability linked to a higher mean value (but with a larger standard deviation, making it more susceptible to tend toward relatively smaller happiness values, compared to the mean) and the lower probability linked to a negative mean value (but with comparatively half the standard deviation, making it less susceptible to tend toward relatively higher happiness values, compared to the mean). Apart from some outliers to the averages, the general average sum was in the 200s of happiness units, while the average length was in the 80s – 90s. This is a direct observation from the fact that the normal distribution that is being used for creating

happiness values is biased towards more positive happiness values. The outlier mentioned above is when the smaller probability case comes up, and suddenly the sum goes down by a smaller proportion, compared to the length of the max subarray.