a. A list/array of size N has integer data. There may be duplicate values in the list. You have at your disposal, a genie, who can return the index or all indices (in case of duplicates) of a given data in 2 nanoseconds. What is the Big-O order of an algorithm that uses such a genie to determine the locations in the list of a given data? Explain your answer in detail. Note: you do not need to write any code for this question.

I think that an algorithm that uses a "genie" to determine the indices of a given data would be some type of searching algorithm. Assuming the list is unsorted (our data was not specified to be numerical values that can be sorted), this "genie" would have to iterate through the list to find the values and their indices using a linear searching algorithm. Also, since we have to find duplicate values as well it would have to continue searching even if it found the value it is searching for already. Since it would have to iterate through N values, the big O would be O(N). Meaning the amount of time taken grows linearly.

b. How would the Big-O order change in problem a., if the genie were to take time proportional to the number of duplicates? Explain your answer in detail.

If the genie were to take time proportional to the number of duplicates it would be big O of O(KN). It might grow faster though. Let K be the number of the data in the list (ex- 1 for no duplicate, 2 for if there is a duplicate value, and so on) and N be the number of items in the list. So K*N would give us the time taken proportional to the duplicates. Thus big O would be (KN). The function would still grow linearly, but K would increase the time taken proportional to the number of duplicates.

c. Propose one or more modification(s) to the below bubble sort code to improve the runtime. Submit your source code with the modifications. Explain why the modification works using comments in your code.

Code is submitted in a separate .py file along with this document.

d. Suppose the array/list of size N gets completely sorted in iteration k (of the outer for-loop). What is the worst-case Big-O order of the bubble sort code?

The worst-case for regular bubble sort is O(N^2) complexity. Meaning the list was reverse sorted and took the maximum number of swaps possible for each iteration. Thus the outer loop iterated N-1 times in the worst scenario. Then k=N-1. To calculate the big O, you would take $1+2+3+4+\cdots+(N-2)+(N-1)=\frac{(N-1)(N-1+1)}{2}=\frac{N(N-1)}{2}=\frac{N^2}{2}-\frac{N}{2}$ time which can be simplified to O(N^2).

e. What is the average-case Big-O order for problem d?

The average-case for regular bubble sort is also O(N^2). The average case is expected to take half as many swaps as the worst case. Therefore K should be half as many as N-1. So to calculate this you would take half of the worst time. So,

$\frac{1}{2}\left(\frac{N(N-1)}{2}\right) = \frac{N(N-1)}{4} = \frac{N^2}{4} - \frac{N}{4}$time which can be simplified to Big O of O(N^2).