question 1: Modify the lower-bound argument for insertion sort to handle input sizes that are not necessarily a multiple of 3.

```
     INSERTION-SORT(A, n)
1.       for i = 2 to n
2.              key = A[i]
3.              j=j–1
4.             // Insert A[i] into the sorted subarray A[1 : i – 1]. j=i–1
5.             while j > 0 and A[j] > key
6.                    A[j + 1] = A[j]
7.                      j=j-1
8.                     A[j + 1] = key
```

For some number a that is n(list size) = 0 (mod a) so that the list can be divided evenly. I am assuming not a prime number. If prime i would add a substitute value for the sorting process that could be removed for presentation. the largest values are held within the portion n/a, the 1/a portion of the list. This is similar to the example in the book where they use the first portion such that it has to travel across the whole list to be sorted forcing the time to be n^2. This is the same thing that happens here as it has to travel to the ((a-1)n)/a portion of the array.

Question 2: Selection sort timing: Is done by a nested for loops with the outer loop iterating through the whole list starting the first position while holding a minimum value. This goes until i = the length of the list at which point the loop stops iterating. That minimum value is compared or replaced based on the values seen by the inner for loop that loops through all values of i+1(i from the first loop current position) . For ex: 5,4,3...0. In this list the outer loop has to go through all n of the values n in the inner loop. This is because the inner loop has to go through the whole list in oder to find the minimum value. Technically it only has to from i+1 to n. This leaves us with 0(n^2) for both the average case and the max case as there is no way that algorithm can be made to go faster if portions are already sorted. every value has to be looked at.