1. The worst case performance of insertion sort (assuming you want to sort in ascending order) is where each number at i is less than all the numbers before it in a[0… i-1]. In this case the sort is order (1+2+…+n-1+n) or O((n^2-n)/2) or O(n^2) asymptotically. The best case is an already sorted array in which for each number at i you have to do 0 swaps and this is order (1+1…+1+1) n times or O(n). The best and worst case performance of the algorithm above are equal (assuming swap is of order O(1)) because in all cases the inner loop runs till the end of the array. The complexity is O((n^2-n)/2) because for each index i of the array the inner loop runs i times. Asymptotically this is O(n^2). In practice insertion sort is better because the average case of insertion sort is better than the worst case of the above sort, so on average its asymptotic complexity is of order less than O(n^2) whereas the above algorithm's average case is still O(n^2). Since the worst case of insertion sort is still as good as the average case of the above algorithm, in practice insertion sort is likely to be better because in practice it is rare to be asked to sort an array sorted in reverse order, which is the only case for which insertion sort is not better than the above algorithm.

2. The best and worst case performances of selection sort are equal and they are of order O(n^2). The same is true of the given algorithm. Therefore, the performance of both algorithms will generally be very similar with a slight difference caused by the number of swaps. In selection sort, there will only be a maximum n swaps to account for each number swapping to its sorted position. In the given algorithm, there may be multiple swaps before a number is finally in its correct position. Thus, although minor, the selection sort will require fewer computations than the given algorithm, causing the given algorithm to be slower in practice.

3. The outer loop invariant is that after every iteration a[0…i-1] is sorted in ascending order. Part of the invariant is also that all the numbers in a[0…i-1] are lesser than any of the numbers in a[i…n-1].
    a. Establishment: At the very beginning, i=0 so a[0…i-1] is an empty range. Thus this empty range can be said to be sorted in ascending order. Further all the numbers in an empty range (none) are naturally smaller than any of the numbers in a[i…n-1].
    b. Preservation: We can see that at the end of the loop a[i] is the smallest element in the range a[i…n-1]. If we assume the invariant is true, i.e. a[0…i-1] is sorted at the beginning of the loop and that any of the numbers in a[i…n-1] are greater than all the numbers in a[0…i-1], then when i's value increments by 1, a[0…i-1] will still be sorted in ascending order, and any number in a[i…n-1] is greater than a[i-1] which is the greatest number in a[0…i-1].
    c. The loop guard says that i < n-1, which means the loop terminates at i = n-1. At i = n-1, a[0…n-2] is sorted in ascending order and a[n-1] is greater than any number in a[0…n-2] which means the entire array is indeed sorted in ascending order, thus the loop only terminates when it has achieved its objective of sorting in ascending order. The loop invariant ensures that.