# HW4

1. As mentioned in HW3, the best, worst and average time complexity come down to the for and while loop. There are a lot of constant time operations, but they don't have as large of a degree as the for and while loop so they aren't as important for the best, average and worst time complexity.

Best case:
lower bound: The lower bound would be Big $\Omega$ because even in the best case the algorithm for the for loop goes through all the elements leading to a $\Omega(n)$ notation.
upper bound: In the best case since the values will be sorted, it means that the maximum amount of time in the best case would be $O(n)$ since it goes through all the elements in the array through a for loop
Tight bound: Since $\Omega(n) = O(n)$ we then get the average or tight bound to be $\Theta(n)$

Average case:
lower bound: since we are talking on average, that means that every half of the elements in the list need to be sorted into the correct position so the minimum time would be $n^2$ time and we get $\Omega(n^2)$
upper bound: The most time in an average case would also yield $n^2$ time so we get $O(n^2)$
Tight bound: $\Omega(n^2) = O(n^2)$ so we get the tight bound to be $\Theta(n^2)$

worst case:
lower bound: For the worst case of insertion sort, it means that the list or array is in reverse order. That means that the for loop would run for n time and the while loop inside for n time as well so we get $\Omega(n^2)$ time
Upper bound: Even in the worst case where the array is in reverse order, the maximum time needed to sort it would be $n^2$ time, coming from the for loop and the while loop, so we get $O(n^2)$ time
Tight bound: $\Omega(n^2) = O(n^2)$ so we get $\Theta(n^2)$

Therefore, for insertion sort we get a best case of $\Theta(n)$ time, and the average and worst case are both $\Theta(n^2)$ time.

2. For this problem, the best, worst, and average time complexity that we use for things like sorting algorithms wouldn't apply because in those cases the inputs determine how the algorithm interacts with it and the best, worst, and averge time complexities change as a result.

From the psuedocode, the time complexity comes from the for loops and there is one for loop that goes from 1 to length of rows A, and then 2 nested for loops going through 1 to length of cols B and 1 to length of cols A.

Using this we can conclude that the Best, Average and Worst case are all $\Theta$(row A.length x cols B.length x cols A.length)
If we give the lengths variables like i, j, and k respectively, that gives us $\Theta(j \cdot i \cdot k)$ or really $\Theta(n^3)$ time which is cubic time complexity.