HW3

1.
```
def insertion_sort(A,n):
    for i in range(1,n):
        key = A[i]
        j = i-1
        while j ≥ 0 and key < A[j]:
            A[j+1] = A[j]
            j -= 1
        A[j+1] = key
    return A
```

There are 2 important lines for the time complexity analysis. The first is the outer for loop which will take n+1 iterations b/c it goes for the range 0 to n. The other line is the while loop which gives us 3 possible cases that determines the time complexity

Best case of while loop: if the array is already sorted, then the while loop will not run, which is why the best case of insertion sort is $O(n)$ time.

Average case: on average when sorting it makes sense for the while loop to execute exactly $\frac{n}{2}$ times which is half. Since this while loop is inside the for loop, we get $O((n+1)(\frac{n}{2}))$ which simplifies to $O(n^2)$ since that is the biggest degree.

Worst case: If array is sorted in reverse order, the while loop will run n times, so we get $O((n+1)(n))$ which is just $O(n^2)$ for the worst case as well.

HW 3

1. Insertion sort is known for being an in-place sorting algorithm meaning that there is no extra space needed to sort it. The only space used it to create some variables, so the space complexity will be O(1)

Matrix_Multiply (A,B)	1 time
if cols(A) ≠ rows (B)	1 time
raise ValueError	1 time
rows_A	1 time
cols_A	1 time
cols_B	1 time
result	1 time
for i from 1 to rows_A do:	(n+1) time
for j From 1 to cols_B do:	n(n+1) time
Sum	(n)(n) time
for K from 1 to cols_A do:	n(n)(n+1) time
Sum	n(n)(n) time
result[i][j]	n(n) time
return result	n(n) time

$7 + (n+1) + (n^2+n) + (n^2) + (n^3+n^2) + (n^3) + (n^2) + n^2)$

$2n^3 + 5n^2 + n + 8$ time

$ax^3 + bx^2 + cx + d$

This means that the time complexity is cubic. In this case what n represents is rows_A, cols_B, cols_A runtime meaning that we get O(rows_A · cols_B · cols_A) run time. The space complexity should be constant b/c we do have variables but all are constant time.