Problem 1 Gerald Susanteo

Best case = fully sorted array since our last HW, 3N^2+4N+1 is what we would expect
But since the while loop only runs N times as the array is fully sorted meaning we can disregard it.

```
Insertion-Sort(A,n)  1
   for(i-1) to n                  N+1
      key=A[i]                    N
      j=i-1                       N
      while j>=0 and A[j]>key        N
         A[j+1]=A[j]                 N
         j=j-1                    N
      A[j+1]= key                    N
```

=> N+1+N+N+N+N+N+N+1
=> 7N+2
Since the while loop may not loop itself, the best case scenario will be Because the while loop O(g(n))=7*N Hence, **O(n)** is the best case

Average Case = Considering the array is scrambled but the elements are only one off from sorted

```
Insertion-Sort(A,n)                  1
                           - //nothing
   for(i-1) to n                  N+1
      key=A[i]                    N
      j=i-1                       N
      while j>=0 and A[j]>key        N(N+1)
         A[j+1]=A[j]                 N*N
         j=j-1                    N*N
      A[j+1]= key                    N
```

=> N+1+N+N+N^2+N+N^2+N^2+N
=> 3N^2+4N+1
Since to run the while loop which in this case is inside the for loop so the costs is Ω(g(n)) is Ω(n^2)

Hence the worse case is similar to the average case = Θ(g(n)) would also be Θ(n^2)

Problem 2

The best case = when the number of rows and columns in both a and b are 1

MATRIX_MULTIPLY(A, B): 1

```
 if columns(A) ≠ rows(B): 1
  raise ValueError("Matrix multiplication is not defined.") 1

 rows_A ← number of rows in A  1
 cols_A ← number of columns in A 1
 cols_B ← number of columns in B 1
 result ← matrix of size rows_A x cols_B filled with zeros 1

 for i from 1 to rows_A do: 1
  for j from 1 to cols_B do: 1
  sum ← 0 1
  for k from 1 to cols_A do: 1
   sum ← sum + A[i][k] * B[k][j] 1
  result[i][j] ← sum return result 1
```

=> 1+1+1+1+1+1+1+1+1+1+1+1+1

=> 13

Hence the base case is O(1) since the code only run once

Average case = Since the highest order is n^2 the time constraint is also Ω(n^2)

```
MATRIX_MULTIPLY(A, B): 1
 if columns(A) ≠ rows(B): 1
  raise ValueError("Matrix multiplication is not defined.") 1
 rows_A ← number of rows in A 1
 cols_A ← number of columns in A 1
 cols_B ← number of columns in B 1
 result ← matrix of size rows_A x cols_B filled with zeros 1
 for i from 1 to rows_A do: N+1
  for j from 1 to cols_B do: N(N+1)
  sum ← 0 N*N
  for k from 1 to cols_A do: N(N+1)
   sum ← sum + A[i][k] * B[k][j] N*N
  result[i][j] ← sum return result N
```

=> 7+N+1+N^2+N+N^2+N^2+N+N^2+N

=> 3n^2+4N+8

Since the highest order is n^2 the time constraint is also Ω(n^2)

For the worst case we will get the same as average case where Θ(g(n)) would also be Θ(n^2)