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

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

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

1. if columns(A) ≠ rows(B): This is a conditional check which takes constant time, O(1).

2. rows_A = number of rows in A, cols_A = number of columns in A, cols_B = number of columns in B: These operations involve reading the size details of matrices and hence, take constant time which is O(1).

3. result = matrix of size rows_A x cols_B filled with zeros: Creating a matrix of certain size can be seen as an operation that operates on every cell once, hence it can be seen as a O(rows_A * cols_B) operation.

4. Three nested loops each of which ranges from the number of elements:

```
for i from 1 to rows_A do:
  for j from 1 to cols_B do:
    sum = 0
    for k from 1 to cols_A do:
```

These loops indicate that for every element in the resulting matrix, we do a constant amount of work (multiplication operation). Hence, this takes O(rows_A * cols_B * cols_A) time.

5. sum = sum + A[i][k] * B[k][j]: Constant time, but this operation runs once for each loop iteration described above, hence its time complexity is included in the triple nested loop's complexity.

6. result[i][j] = sum: Constant, but this runs once for each double-loop iteration described above, hence its time complexity is included in the double loop's complexity.

Adding all these, the total time complexity would be O(1) + O(1) + O(rows_A * cols_B) + O(rows_A * cols_B * cols_A) = O(rows_A * cols_B * cols_A) as this term will dominate the overall time complexity for large matrices.