Daniel Alejandro Fernández Robles
A00354694
Camilo Enríquez Delgado
A00354532

Step 1. Identification of the problem.

The specific needs of the problematic situation, symptoms and conditions for resolution are listed below.

Identification of needs and symptoms:

> Venus needs a plan that helps them win the war against Mars for the territorial sovereignty of the 53 moons of Saturn and its rings.
>
> The Mars spacecraft are strategically located secretly and invisible to the troops of Venus.
>
> The algorithms for multiplying matrices and tracking the Martians must be efficient, so that they allow, very quickly, to make calculations on matrices of very large dimensions and find the right coordinates to attack Mars troops.
>
> To allocate the invisible Mars troops, Venus has some spacecraft locations from past battles and the possibility of knowing where the enemy will be located through a trusted predictor.

Definition of the problem:

> Venus requires a software module that allows them to track the Martians and helps them win the war and territorial sovereignty of the 53 moons of Saturn and its rings.

Requirements:

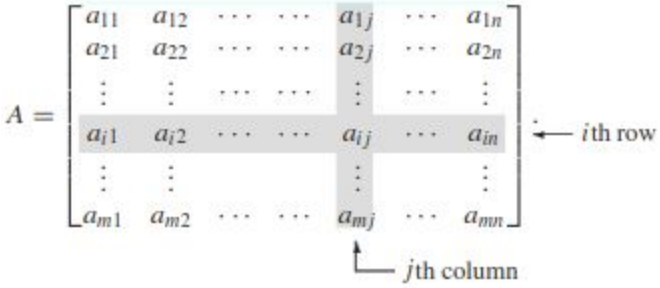
Name	**R.#1. Randomly generate a matrix**
Summarize	The program will allow you to randomly generate a matrix of considerably large dimensions and its values.
Input	● An integer giving the number of rows the matrix will have. ● An integer giving the number of columns the matrix will have. ● A boolean representing whether or not repeated elements are allowed.
Output	The requested randomly generated matrix with the specified restrictions.

Name	**R.#2. Multiply matrices**
Summarize	The program will allow you to compute the matrix product, using the method chosen by the user if the matrices are compatible with It(e.g: If the dimensions of the matrices are not $2^n x 2^n$ where $n \in \mathbb{Z}^+$, then the divide and conquer and Strassen algorithms cannot be executed)
Input	• The matrix with Mars troops locations. • The secret matrix.
Output	The matrix resulting of multiplying the two matrices. This matrix contains the current locations of Mars troops.

Step 2. Gathering needed information.

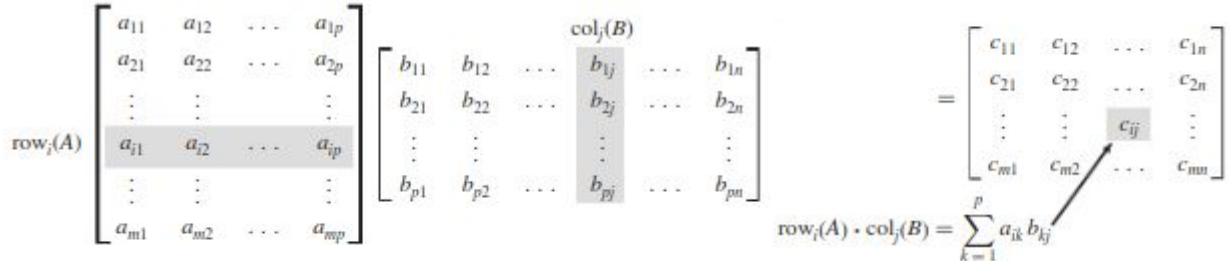
Matrix:

An m×n matrix A is a rectangular array of mn real (or complex) numbers arranged in m horizontal rows and n vertical columns.



Matrix multiplication:

If A = $[a_{i,j}]$ is an m×p matrix and B = $[b_{i,j}]$ is a p×n matrix, then the product of A and B, denotes AB, is the m×n matrix C = $[c_{i,j}]$, in which the i, jth element is the dot product of the ith row, row $_i$(A), and the jth column, col $_j$(B), of B. The number of columns in matrix A must be equal to the number of rows in matrix B in order to multiply them.



Dot product:

The dot product or inner product of the n-vectors a and b is the sum of the products of corresponding entries. Thus, if

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Then

$$\mathbf{a}\cdot\mathbf{b} = a_1b_1 + a_2b_2 + \cdots + a_nb_n = \sum_{i=1}^{n} a_i b_i$$

Step 3. Searching for creative solutions.

First option: Standard multiplication

The usual way to compute the matrix product is the one listed previously. The dot product between each row **i** in matrix **A** and column **j** in matrix **B** is allocated in row **i** and column **j** of the resulting matrix.

Second option: Divide and Conquer

This way for multiplying matrices is only applicable on squared matrices of dimensions $\mathbf{2^n x 2^n}$ where **n** is a positive integer. It's focus is not to multiply the whole matrix but divide it into four submatrices and multiply them as if they were elements of a **4x4** matrix. The process is done recursively until the matrices that have to be multiplied are of dimension **1x1**. Then all the basic results are combined into the whole result.

Third option: Strassen algorithm

As the last one, this algorithm has a divide and conquer approach and also requires the two matrices to be multiplied to be squared matrices of dimensions $\mathbf{2^n x 2^n}$ where **n** is a positive integer.

The Strassen algorithm splits the whole matrix into four submatrices($\mathbf{A_{1,1}}$ $\mathbf{A_{1,2}}$ $\mathbf{A_{2,1}}$ $\mathbf{A_{2,2}}$) and then defines seven new matrices:

- **P** = $(A_{1,1} + A_{2,2}) * (B_{1,1} + B_{2,2})$
- **Q** = $(A_{2,1} + A_{2,2}) * B_{1,1}$
- **R** = $A_{1,1} * (B_{1,2} - B_{2,2})$
- **S** = $A_{2,2} *(B_{2,1} - B_{1,1})$
- **T** = $(A_{1,1} + A_{1,2}) * B_{2,2}$
- **U** = $(A_{2,1} - A_{1,1}) * (B_{1,1} + B_{1,2})$
- **V** = $(A_{1,2} - A_{2,2}) * (B_{2,1} + B_{2,2})$

only using 7 multiplications (one for each submatrix) instead of 8 as the last divide and conquer option. We may now express $\mathbf{C_{i,j,}}$(The resulting matrix) in terms of each one of these submatrices:

- $\mathbf{C_{1,1}}$ = P + S - T + V
- $\mathbf{C_{1,2}}$ = R + T
- $\mathbf{C_{2,1}}$ = Q + S
- $\mathbf{C_{2,2}}$ = P + R - Q + U

We iterate this division process *n* times (recursively) until the submatrices degenerate into numbers (elements of the ring *R*). Then all the basic results are combined into the whole result.

Step 4. Stepping from ideation to preliminary designs (including modeling).

At the beginning of the solutions research, we found one that could work but because of the following reasons we didn't even list it:

Coppersmith-Winograd Algorithm:

-- There was not enough resources in order to get to understand what this solution did specifically.

-- Although It is more efficient than the other options, this algorithm implied a deep linear algebra knowledge which we are not acquainted.

The other alternatives were much easier to understand. Some of its characteristics are listed below:

Standard multiplication:

-- It's by far the easiest way to understand matrix product.

-- It's not just easy to understand but easy to implement.

-- Its time complexity is $O(n^3)$.

Divide and Conquer:

-- A divide and conquer algorithm works by recursively breaking down a problem into two or more subproblems of the same or related type, until these become simple enough to be solved directly. The solutions to the subproblems are then combined to give a solution to the original problem.

-- Divide and conquer algorithms focus is to find more efficient ways to solve a problem.

-- Its time complexity is $O(n^3)$.

Strassen algorithm:

-- It is a divide and conquer algorithm so it works by recursively breaking down the problem into two or more subproblems of the same or related type, until these become simple enough to be solved directly. The solutions to the subproblems are then combined to give a solution to the original problem.

-- It reduces the number of matrix multiplication that should be done in each recursive step from 8 to 7.

-- Its time complexity is $O(n^{\log_2(7)}) \approx O(n^{2.81})$.

Step 5. Evaluation and selection of preferred solution.

Criterion A: Solution Precision.

[2] Accurate

[1] Approximate

Criterion B: Efficiency.

[6] Constant

[5] Logarithmic

[4] Linear

[3] Quadratic

[2] Something in the middle
[1] Cubic

Criterion C: Completeness.

[3] All solutions
[2] Some of them
[1] Just one or none

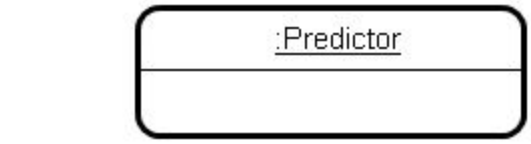
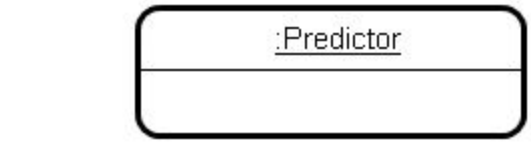
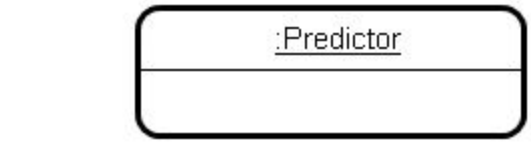
	Criterion A	Criterion B	Criterion C	Total
First Option	2	1	3	6
Second Option	2	1	2	5
Third Option	2	2	2	6

Selection:

In accordance with the previous table, the best options are the Standard multiplication and the Strassen Algorithm. Strassen should be applied whenever possible because of efficiency, but if It can't be applied then Standard multiplication is necessary.

Step 6. Preparation of reports, plans, and specifications.

SCENARIO SETUP

Name	**Class under test**	**Stage**
setupStage1	Predictor	 $A\begin{pmatrix} -18 & 15 & 7 \\ 8 & 3 & 4 \\ 11 & 2 & 13 \end{pmatrix}$ $B\begin{pmatrix} 4 & 37 & 73 \\ 5 & -13 & 4 \\ 11 & 2 & 1 \end{pmatrix}$
setupStage2	Predictor	 $A\begin{pmatrix} 4 & 2 & 1 & 0 \\ 9 & 20 & 2 & 8 \\ 3 & 4 & -5 & 3 \end{pmatrix}$ $B\begin{pmatrix} 8 & 11 & 13 & 0 \\ 4 & 9 & 5 & 7 \\ 1 & 3 & 2 & 0 \end{pmatrix}$
setupStage3	Predictor	

		$A\begin{pmatrix} 16 & 9 & 7 & 6 \\ -6 & 0 & 8 & 11 \\ 2 & 3 & 5 & 4 \\ 0 & 58 & 1 & 6 \end{pmatrix}$ $B\begin{pmatrix} 5 & 3 & 7 & 9 \\ 5 & 6 & 4 & 6 \\ 13 & 8 & 2 & 6 \\ 0 & 8 & 0 & 1 \end{pmatrix}$

TEST CASES

Test Objective: Verificate that the method generateRandomMatrix generates a matrix specifying the number of rows and columns and if there may be some repeated elements or not at all inside the matrix.

Class	Method	Stage	Input	Output
Predictor	generateRandomMatrix	setupStage 1	rows= 60 columns= 60 repeatedElements= false	A matrix 60x60 without any repeated element at all.

Test Objective: Verificate that the method standardMultiply multiplies two matrices mxp and pxn that arrives as parameter correctly.

Class	Method	Stage	Input	Output
Predictor	standard Multiply	setupStage 1	A and B	A matrix 3x3 with the Mars troops locations. $\begin{pmatrix} 80 & -739 & -1247 \\ 91 & 217 & 600 \\ 197 & 341 & 824 \end{pmatrix}$

Test Objective: Verificate that the method standardMultiply does not multiply two matrices where the number of columns from the first one is different to the number of rows from the second one.

Class	Method	Stage	Input	Output
Predictor	standard Multiply	setupStage 2	A and B	It throws an IllegalArgumentException: Incorrect Dimensions.

Test Objective: Verificate that the method divideAndConquerMultiply does not multiply two matrices where the number of columns from the first one is different to the number of rows from the second one.

Class	Method	Stage	Input	Output

Predictor	divideAnd Conquer Multiply	setupStage 2	A and B	It throws an IllegalArgumentException: Incorrect Dimensions.

Test Objective: Verificate that the method strassenMultiply does not multiply two matrices where the number of columns from the first one is different to the number of rows from the second one.				
Class	Method	Stage	Input	Output
Predictor	strassen Multiply	setupStage2	A and B	It throws an IllegalArgumentException: IncorrectDimensions.

Test Objective: Verificate that the method strassenMultiply does not multiply two matrices different from $2^n x 2^n$ type.				
Class	Method	Stage	Input	Output
Predictor	strassen Multiply	setupStage1	A and B	It throws an IllegalArgumentException: Incorrect Dimensions.

Test Objective: Verificate that the method divideAndConquerMultiply does not multiply two matrices different from $2^n x 2^n$ type.				
Class	Method	Stage	Input	Output
Predictor	divideAnd Conquer Multiply	setupStage 1	A and B	It throws an IllegalArgumentException: Incorrect Dimensions.

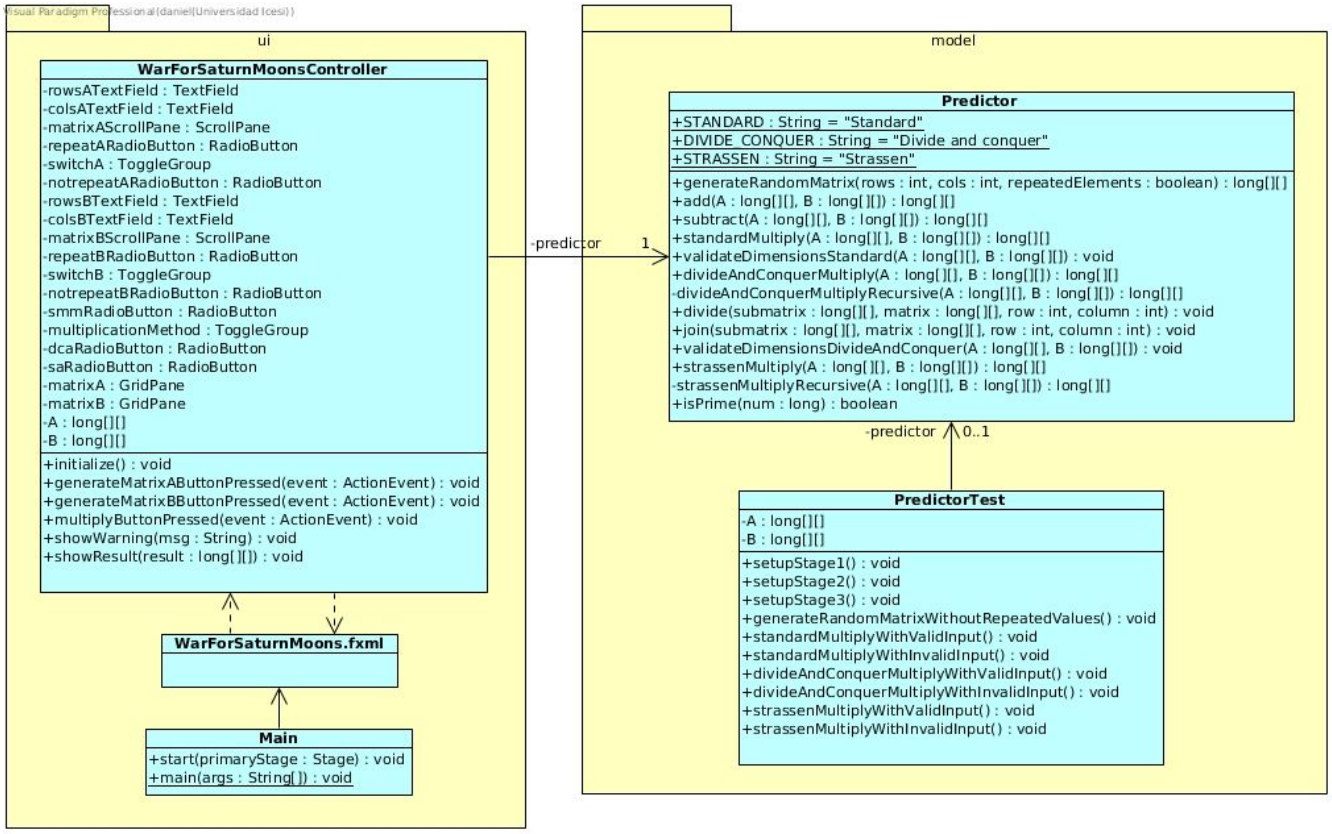
Test Objective: Verificate that the method divideAndConquerMultiply multiplies two matrices of $2^n x 2^n$ type.				
Class	Method	Stage	Input	Output
Predictor	divideAnd Conquer Multiply	setupStage 3	A and B	A matrix 4x4 with the Mars troops locations.

				$\begin{pmatrix} 216 & 206 & 162 & 246 \\ 74 & 134 & -26 & 5 \\ 90 & 96 & 36 & 70 \\ 303 & 404 & 234 & 360 \end{pmatrix}$

Test Objective: Verificate that the method strassenMultiply multiplies two matrices of $2^n x 2^n$ type.				
Class	Method	Stage	Input	Output
Predictor	strassen Multiply	setupStage 3	A and B	A matrix 4x4 with the Mars troops locations. $\begin{pmatrix} 216 & 206 & 162 & 246 \\ 74 & 134 & -26 & 5 \\ 90 & 96 & 36 & 70 \\ 303 & 404 & 234 & 360 \end{pmatrix}$

Step 7. Implementation of the design.

Class diagram



Pseudocode and complexity analysis

Function	Pseudocode	Java implementation

Add two matrices	`def add(matrix A, matrix B)`
	`  validateDimensionsStandard(A, B)`
	`  rows = A.length`
	`  cols = A[0].length`
	`  C = matrix[rows][cols]`
	`  for(i = 0; i < rows; i++)`
	`  for(j = 0; j < cols; j++)`
	`  C[i][j] = A[i][j] + B[i][j]`
	`  return C`
		https://github.com/7yrionLannister/matrix-multiplication-and-position-tracking/blob/141426d28c939e53de43e6eb4dba28ded1390359/src/model/Predictor.java#L47

Complexity analysis for add function

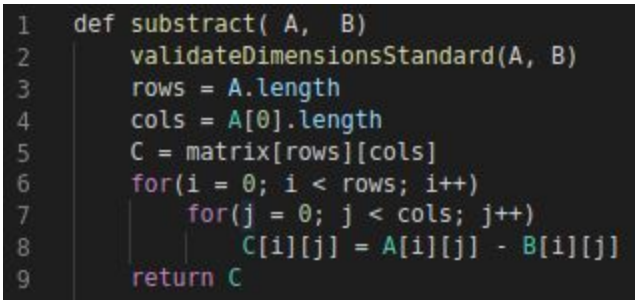
Temporal complexity:

Line	Times executed	DIstributed
	Let n = max(rows,cols)	
2.	1	1
3.	1	1

4.	1	1
5.	1	1
6.	n+1	n+1
7.	n*(n+1)	n^2+n
8.	n*n	n^2
X	T(n) =	$2n^2+2n+5$

Space complexity:

Input	• A • B	• n*n • n*n
Output	• C	• n*n
Auxiliary	• rows • cols • i • j	• 1 • 1 • 1 • 1
X	S(n) =	$3n^2+4$

Function	**Pseudocode**	**Java implementation**
Subtract two matrices		https://github.com/7yrionLannister/matrix-multiplication-and-position-tracking/blob/141426d28c939e53de43e6eb4dba28ded1390359/src/model/Predictor.java#L64

```
def substract( A,  B)
    validateDimensionsStandard(A, B)
    rows = A.length
    cols = A[0].length
    C = matrix[rows][cols]
    for(i = 0; i < rows; i++)
        for(j = 0; j < cols; j++)
            C[i][j] = A[i][j] - B[i][j]
    return C
```

Complexity analysis for substract function

Temporal complexity:

Line	Times executed	DIstributed
	Let n = max(rows,cols)	

2.	1	1
3.	1	1
4.	1	1
5.	1	1
6.	n+1	n+1
7.	n*(n+1)	n^2+n
8.	n*n	n^2
X	T(n) =	$2n^2+2n+5$

Space complexity:

Input	• A • B	• m*n • m*n
Output	• C	• m*n
Auxiliary	• rows • cols • i • j	• 1 • 1 • 1 • 1
X	S(n) =	$3n^2+4$

Function	Pseudocode	Java implementation

Take a quarter of a matrix and put it in a submatrix	`def divide(submatrix, matrix, row, column)`
	`  n = submatrix.length`
	`  for( i1 = 0, i2 = row i1 < n i1++, i2++)`
	`  for( j1 = 0, j2 = column j1 < n j1++, j2++)`
	`  submatrix[i1][j1] = matrix[i2][j2]`
		https://github.com/7yrionLannister/matrix-multiplication-and-position-tracking/blob/141426d28c939e53de43e6eb4dba28ded1390359/src/model/Predictor.java#L162

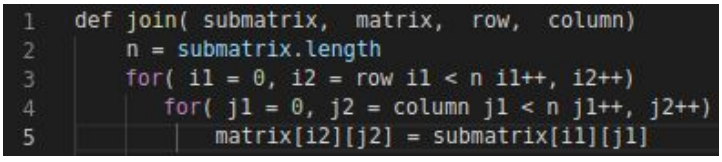
Complexity analysis for divide function

Temporal complexity:

Line	Times executed	DIstributed
2.	1	1
3.	n+1	n+1
4.	n*(n+1)	n^2+n
5.	n*n	n^2
X	T(n) =	$2n^2+2n+2$

Space complexity:

Input	• submatrix • matrix • row • column	(n / 2)*(n / 2) n*n 1 1
Output		
Auxiliary	• n • i1 • i2 • j1 • j2	1 1 1 1 1
X	S(n) =	$1,25n^2+7$

Function	**Pseudocode**	**Java implementation**
Take the values in a submatrix and fill a quarter of a bigger matrix with its values		https://github.com/7yrionLannister/matrix-multiplication-and-position-tracking/blob/141426d28c939e53de43e6eb4dba28ded1390359/src/model/Predictor.java#L176

```
1  def join( submatrix,  matrix,  row,  column)
2      n = submatrix.length
3      for( i1 = 0, i2 = row i1 < n i1++, i2++)
4          for( j1 = 0, j2 = column j1 < n j1++, j2++)
5              matrix[i2][j2] = submatrix[i1][j1]
```

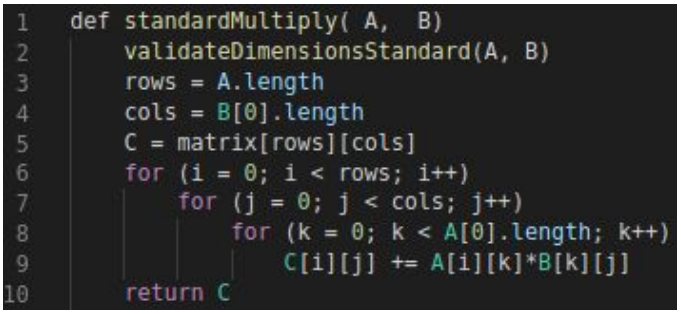
Complexity analysis for divide function
Temporal complexity:

Line	Times executed	DIstributed

2.	1	1
3.	n+1	n+1
4.	n*(n+1)	n^2+n
5.	n*n	n^2
X	T(n) =	$2n^2+2n+2$

Space complexity:

Input	• matrix • submatrix • row • column	n*n (n / 2)*(n / 2) 1 1
Output		
Auxiliary	• n • i1 • i2 • j1 • j2	1 1 1 1 1
X	S(n) =	$1{,}25n^2+7$

Function	**Pseudocode**	**Java implementation**
Multiply by standard method		https://github.com/7yrionLannister/matrix-multiplication-and-position-tracking/blob/141426d28c939e53de43e6eb4dba28ded1390359/src/model/Predictor.java#L81

```
def standardMultiply( A,  B)
    validateDimensionsStandard(A, B)
    rows = A.length
    cols = B[0].length
    C = matrix[rows][cols]
    for (i = 0; i < rows; i++)
        for (j = 0; j < cols; j++)
            for (k = 0; k < A[0].length; k++)
                C[i][j] += A[i][k]*B[k][j]
    return C
```

Complexity analysis for standard multiplication
Temporal complexity:

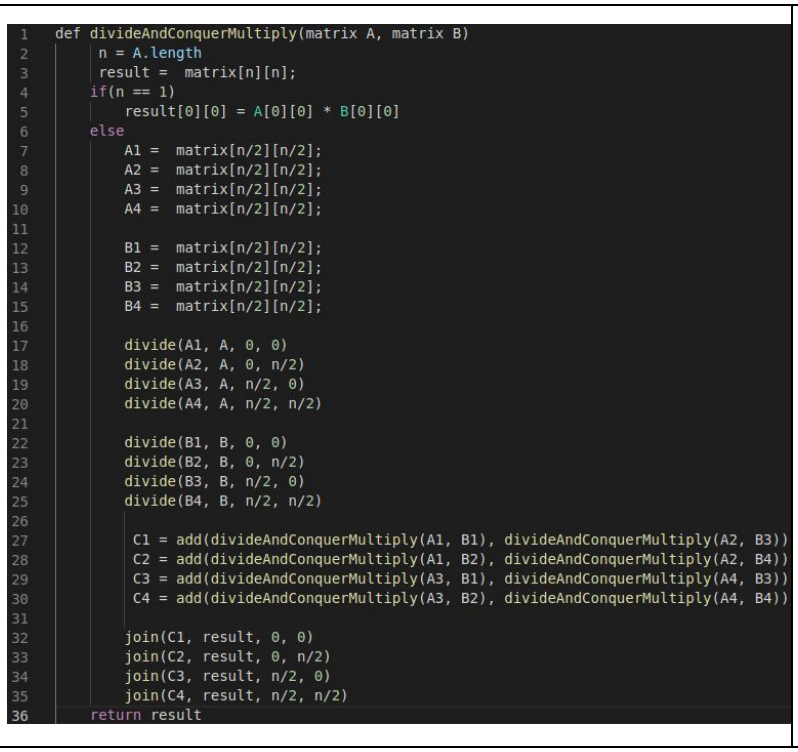
Line	Times executed	DIstributed

	Let n = max(A[rows], B[rows], A[cols], B[cols])	
2.	1	1
3.	1	1
4.	1	1
5.	1	1
6.	n+1	n+1
7.	n*(n+1)	n^2+n
8.	n*n*(n+1)	n^3+n^2
9.	n*n*n	n^3
10.	1	1
X	T(n) =	$2n^3+2n^2+2n+6$

Space complexity:

Input	• A • B	n*n n*n
Output	• C	n*n
Auxiliary	• rows • cols • i • j • k	1 1 1 1 1
X	S(n) =	$3n^2+5$

Function	**Pseudocode**	**Java implementation**

Multiply by divide and conquer method		https://github.com/7yrionLannister/matrix-multiplication-and-position-tracking/blob/141426d28c939e53de43e6eb4dba28ded1390359/src/model/Predictor.java#L118

Complexity analysis for divide and conquer multiplication

Temporal complexity:

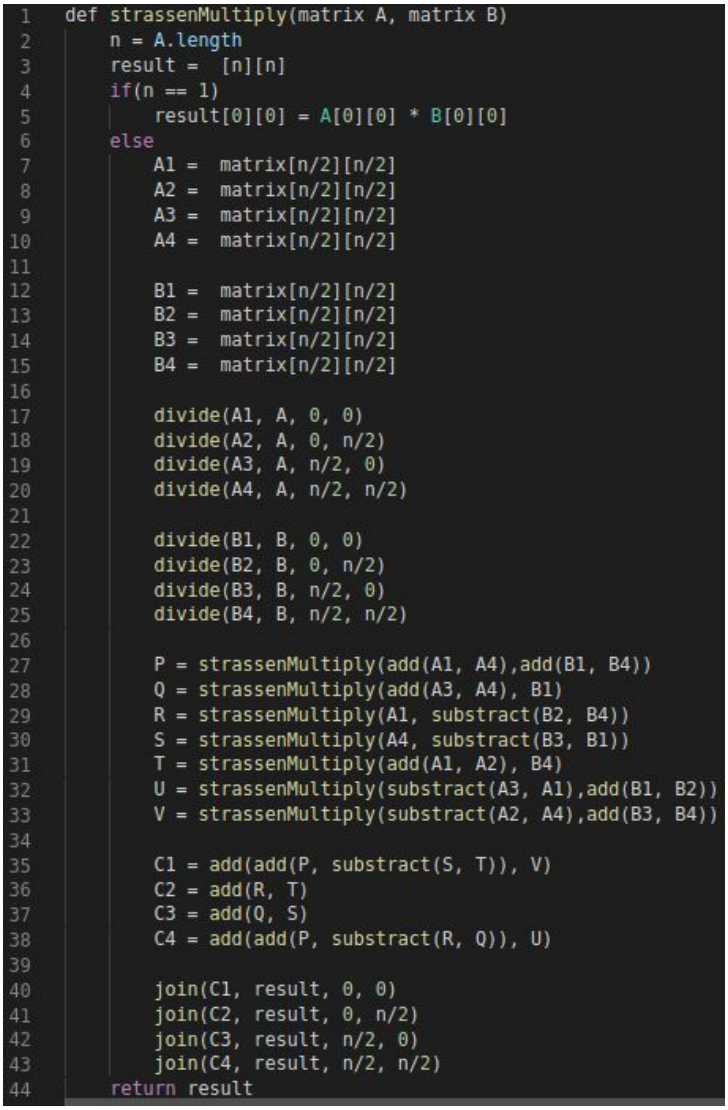
Line	Times executed	DIstributed
2.	1	1
3.	1	1
4.	1	1
5.	1	1
6.		
7.	1	1
8.	1	1
9.	1	1
10.	1	1
11.		
12.	1	1
13.	1	1

14.	1	1
15.	1	1
16.		
17.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
18.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
19.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
20.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
21.		
22.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
23.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
24.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
25.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
26.		
27.	$(2(n/2)^2+2(n/2)+5) + (T(n/2) + T(n/2))$	$2T(n/2) + (n^2+2n+10)/2$
28.	$(2(n/2)^2+2(n/2)+5) + (T(n/2) + T(n/2))$	$2T(n/2) + (n^2+2n+10)/2$
29.	$(2(n/2)^2+2(n/2)+5) + (T(n/2) + T(n/2))$	$2T(n/2) + (n^2+2n+10)/2$
30.	$(2(n/2)^2+2(n/2)+5) + (T(n/2) + T(n/2))$	$2T(n/2) + (n^2+2n+10)/2$
31.		
32.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
33.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
34.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
35.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
36.	1	1
X	T(n) =	$8*T(n/2) + 8n^2 + 16n + 57$

Space complexity:

Input	• A • B	n*n n*n
Output	• result	n*n
Auxiliary	• n • A1 • A2 • A3 • A4 • B1 • B2 • B3 • B4 • C1 • C2 • C3 • C4	1 (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2)
	Additionally add the space complexity of the 8 divides, 4 additions and 4 joins	$27n^2+100$
X	S(n) =	$8S(n/2) + 33n^2 + 101$

Function	**Pseudocode**	**Java implementation**

Multiply by Strassen method		https://github.com/7yrionLannister/matrix-multiplication-and-position-tracking/blob/141426d28c939e53de43e6eb4dba28ded1390359/src/model/Predictor.java#L212

Complexity analysis for Strassen multiplication

Temporal complexity:

Line	Times executed	DIstributed
2.	1	1
3.	1	1
4.	1	1
5.	1	1
6.		
7.	1	1

8.	1	1
9.	1	1
10.	1	1
11.		
12.	1	1
13.	1	1
14.	1	1
15.	1	1
16.		
17.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
18.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
19.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
20.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
21.		
22.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
23.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
24.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
25.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
26.		
27.	$T(n/2) + 2(2n^2+2n+5)$	$T(n/2)+4n^2+4n+10$
28.	$T(n/2) + 2n^2+2n+5$	$T(n/2)+2n^2+2n+5$
29.	$T(n/2) + 2n^2+2n+5$	$T(n/2)+2n^2+2n+5$
30.	$T(n/2) + 2n^2+2n+5$	$T(n/2)+2n^2+2n+5$
31.	$T(n/2) + 2n^2+2n+5$	$T(n/2)+2n^2+2n+5$
32.	$T(n/2) + 2(2n^2+2n+5)$	$T(n/2)+4n^2+4n+10$
33.	$T(n/2) + 2(2n^2+2n+5)$	$T(n/2)+4n^2+4n+10$

34.		
35.	$3(2n^2+2n+5)$	$6n^2+6n+15$
36.	$2n^2+2n+5$	$2n^2+2n+5$
37.	$2n^2+2n+5$	$2n^2+2n+5$
38.	$3(2n^2+2n+5)$	$6n^2+6n+15$
39.		
40.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
41.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
42.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
43.	$2(n/2)^2+2(n/2)+2$	$(n^2+2n+4) / 2$
44.	1	1
X	T(n) =	$7T(n/2)+42n^2+48n+127$

Space complexity:

Input	• A • B	n*n n*n
Output	• result	n*n
Auxiliary	• n • A1 • A2 • A3 • A4 • B1 • B2 • B3 • B4 • C1 • C2 • C3 • C4 • P • Q • R • S	1 (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2)

	• T • U • V	(n/2)*(n/2) (n/2)*(n/2) (n/2)*(n/2)
	Additionally add the space complexity of the 8 divides, 17 additions and 4 joins	$66n^2+152$
X	S(n) =	$7S(n/2) + 73,25n^2 + 153$

Summary of complexities of the algorithms in Big-O notation

Because we do not have the necessary knowledge to perform the analysis of temporal and space complexity of recursive algorithms, we will only leave them expressed and place the result that we find on the web.

Function	**Complexity equations**	**Big-O notation**
Add	• $T(n) = 2n^2+2n+5$ • $S(n) = 3n^2+4$	$O(n^2)$ $O(n^2)$
Subtract	• $T(n) = 2n^2+2n+5$ • $S(n) = 3n^2+4$	$O(n^2)$ $O(n^2)$
Divide	• $T(n) = 2n^2+2n+2$ • $S(n) = 1,25n^2+7$	$O(n^2)$ $O(n^2)$
Join	• $T(n) = 2n^2+2n+2$ • $S(n) = 1,25n^2+7$	$O(n^2)$ $O(n^2)$
Standard multiplication	• $T(n) = 2n^3+2n^2+2n+6$ • $S(n) = 3n^2 + 5$	$O(n^3)$ $O(n^2)$
Divide and conquer multiplication	• $T(n) = 8*T(n/2) + 8n^2 + 16n + 57$ • $S(n) = 8S(n/2) + 33n^2 + 101$	$O(n^3)$ $O(n^2)$
Strassen multiplication	• $T(n) = 7T(n/2)+42n^2+48n+127$ • $S(n) = 7S(n/2) + 73,25n^2 + 153$	$O(n^{2.8074})$ $O(n^2)$

Sources.

https://www.amazon.com/Introductory-Linear-Algebra-Applied-Course/dp/0131437402

https://en.wikipedia.org/

https://youtu.be/0oJyNmEbS4w