Introduction:

Using a C program, we are going to randomly generate relations using matrices with random dimensions based on user input ‘n’, where ‘n’ represents the number of relation matrices, and store them in a file. Additionally, we record the generation times in a separate file.Also, verifying the properties of the randomly generated relations, including symmetry, anti-symmetry, transitivity, and equivalence. . Also, it will determine computational time in milliseconds. It will check whether the relation represents any function or not.

Source Code:

```
1.  #include <stdio.h>            // Include standard input/output library
2.  #include <stdlib.h>           // Include standard library
3.  #include <time.h>            // Include time library for randomization
4.  #define MAX_N 100       // Define a constant MAX_N with a value of 100

// Function declarations
5.  void generateRelationMatrix(int matrix[MAX_N][MAX_N], int n);
6.  int isSymmetric(int matrix[MAX_N][MAX_N], int n);
7.  int isAntisymmetric(int matrix[MAX_N][MAX_N], int n);
8.  int isTransitive(int matrix[MAX_N][MAX_N], int n);
9.  int isEquivalence(int matrix[MAX_N][MAX_N], int n);
10. int isFunction(int matrix[MAX_N][MAX_N], int n);
11. void printMatrixToFile(int matrix[MAX_N][MAX_N], int n, FILE* file);

12. int n;                  // An global integer variable 'n' to store the number of relation matrices

13. int main()         // Main function
14. {
15.     clock_t start_time, end_time;                    // Declare variables to measure time
16.     start_time = clock();                            // Record the start time
17.     double generation_time, verification_time; // Declare variables to store time durations
18.     double total_time;                               // Declare a variable to store the total time
19.     printf("Enter the number of relation matrices (n): ");      // Prompt the user for input
20.     scanf("%d", &n);   // Read the user's input and store it in 'n'
21.
```

```
22.    FILE* relationsFile = fopen("Relations.txt", "w");        /* Open a file for writing relation
    matrices*/
23.    FILE* fp_results = fopen("Computational Time.txt", "w"); /* Open a file for writing
    computational time results*/
24.
25.        srand(time(NULL));        // Seed the random number generator with the current time

26.    for (int a = 0; a < n; a++) // Loop to generate 'n' relation matrices
27.    {
28.        int size = rand() % 10 + 1;        // Generate a random size for the matrix
29.        int relationMatrix[MAX_N][MAX_N]; // Declare a matrix to store the relation

30.        generateRelationMatrix(relationMatrix, size); // Generate a relation matrix
31.
32.        fprintf(relationsFile, "Matrix %d (Size: %dx%d):\n", a + 1, size, size); /* Write matrix
    information to the relations file*/
33.        printMatrixToFile(relationMatrix, size, relationsFile);    /*Write the matrix to the
    relations file*/
34.        fprintf(relationsFile, "\n");

35.        generation_time = ((double)clock() / CLOCKS_PER_SEC) * 1000; /* Calculate
    generation time*/


36.        // Check if the matrix properties and write the results to the relations file
37.        if (isSymmetric(relationMatrix, size)) {
38.            fprintf(relationsFile, "Symmetric: Yes\n");
39.        } else {
40.            fprintf(relationsFile, "Symmetric: No\n");
41.        }

42.        if (isAntisymmetric(relationMatrix, size)) {
43.            fprintf(relationsFile, "Antisymmetric: Yes\n");
44.        } else {
45.            fprintf(relationsFile, "Antisymmetric: No\n");
46.        }

47.        if (isTransitive(relationMatrix, size)) {
48.            fprintf(relationsFile, "Transitive: Yes\n");
49.        } else {
50.            fprintf(relationsFile, "Transitive: No\n");
51.        }

52.        if (isEquivalence(relationMatrix, size)) {
```

```
53.             fprintf(relationsFile, "Equivalence Relation: Yes\n");
54.         } else {
55.             fprintf(relationsFile, "Equivalence Relation: No\n");
56.         }

57.         if (isFunction(relationMatrix, size)) {
58.             fprintf(relationsFile, "Function: Yes\n");
59.         } else {
60.             fprintf(relationsFile, "Function: No\n");
61.         }

62.         fprintf(fp_results, "  Matrix %d:\n", a + 1); // Write matrix information to the time
    results file
63.         fprintf(fp_results, "Generation Time: %lf milliseconds\n\n", generation_time); // Write
    generation time to the time results file
64.         fprintf(relationsFile, "\n");
65.     }

66.     end_time = clock();   // Record the end time
67.     total_time = (double)(end_time - start_time) / CLOCKS_PER_SEC; // Calculate total
    time duration
68.     fprintf(fp_results, "\n\t\tVerification Time: %lf milliseconds\n\n", total_time * 1000); //
    Write total verification time to the time results file

69.     fclose(relationsFile);  // Close the relations file
70.     fclose(fp_results);    // Close the time results file
71.     return 0;
72. }

73. // Function to generate a relation matrix
74. void generateRelationMatrix(int matrix[MAX_N][MAX_N], int n)
75. {
76.     for (int a = 0; a < n; a++)
77.     {
78.         for (int b = 0; b < n; b++)
79.         {
80.             matrix[a][b] = rand() % 2; // Fill the matrix with random values (0 or 1)
81.         }
82.     }
83. }
```

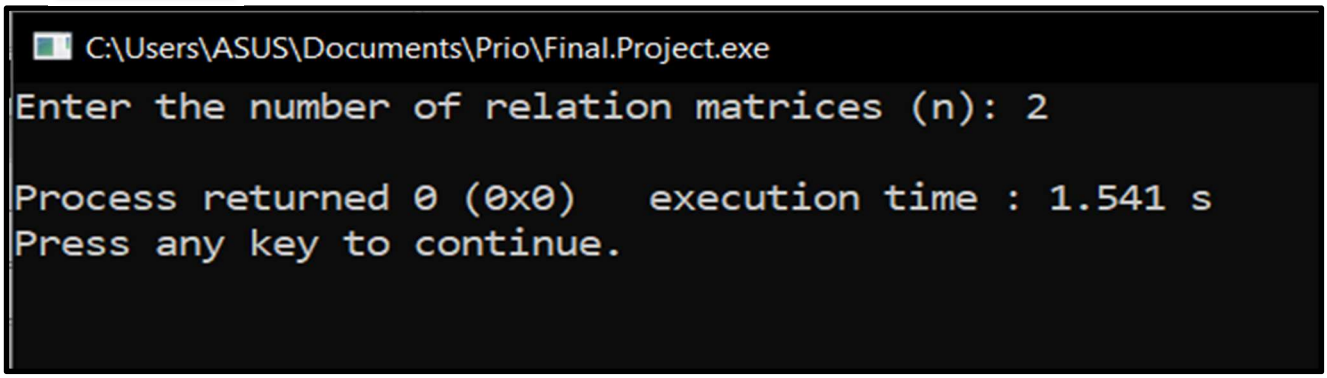
```
    // Functions to check various properties of the relation matrix
    // (isSymmetric, isAntisymmetric, isTransitive, isEquivalence, isFunction)

int isSymmetric(int matrix[MAX_N][MAX_N], int n)
{
    for (int a = 0; a < n; a++)          // Check if it's antisymmetric
    {
        for (int b = a + 1; b < n; b++) // Loop from a+1 to n-1
        {
            if (matrix[a][b] != matrix[b][a]) // Check if the matrix is not symmetric
            {
                return 0; // Return 0 to indicate not symmetric
            }
        }
    }
    return 1; // Return 1 to indicate symmetric
}

int isAntisymmetric(int matrix[MAX_N][MAX_N], int n)
{
            for (int a = 0; a < n; a++) // Loop n times
            {
                for (int b = 0; b < n; b++)
                {
                    if (a != b && matrix[a][b] && matrix[b][a]) // Check if it's antisymmetric
                    {
                        return 0; // Return 0 to indicate not antisymmetric
                    }
                }
            }
            return 1; // Return 1 to indicate antisymmetric
        }

        int isTransitive(int matrix[MAX_N][MAX_N], int n)
        {
            for (int a = 0; a < n; a++)
             {
                for (int b = 0; b < n; b++)
                    {
                    for (int c = 0; c < n; c++)
                         {
                        if (matrix[a][b] && matrix[b][c] && !matrix[a][c]) { // Check if it's transitive
```

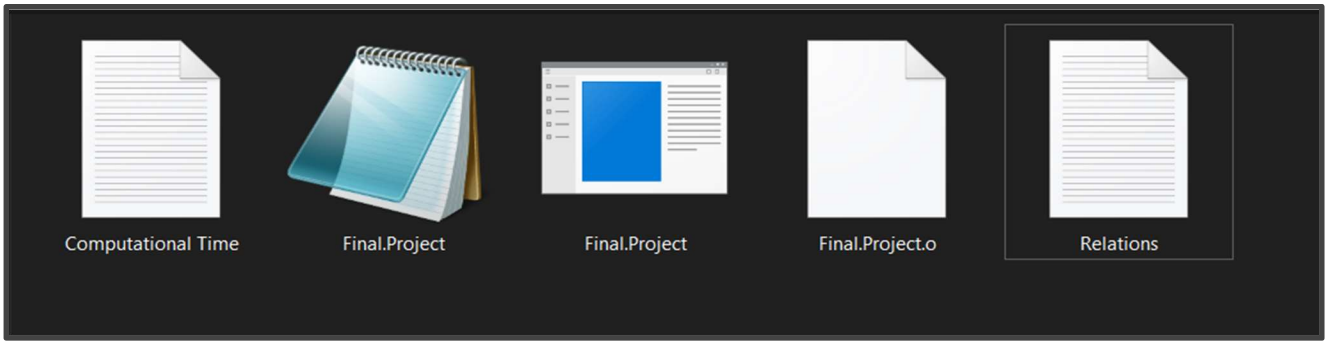
```
121.                         return 0; // Return 0 to indicate not transitive
122.                     }
123.                 }
124.             }
125.         }
126.         return 1; // Return 1 to indicate transitive
127.     }

128.     int isEquivalence(int matrix[MAX_N][MAX_N], int n)
129.     {
130.         if (isSymmetric(matrix, n) && isTransitive(matrix, n)) // Check if it's an
    equivalence relation
131.         {
132.             return 1; // Return 1 to indicate an equivalence relation
133.         }
134.         return 0;   // Return 0 to indicate not an equivalence relation
135.     }
136.
137.     int isFunction(int matrix[MAX_N][MAX_N], int n)
138.     {
139.         int used[n];     /* Declare an array to keep track of how many times each
    element appears in rows*/
140.         for (int a = 0; a < n; a++) // Loop n times
141.         {
142.             used[a] = 0;             // Initialize used array
143.         }

144.     for (int a = 0; a < n; a++)       // Loop n times
145.      {
146.      for (int b = 0; b < n; b++) // Loop n times
147.       {
148.             if (matrix[a][b])         // Check if there is a value in the matrix
149.            {
150.                 used[b]++;         // Increment the count for the element
151.                 if (used[b] > 1)  // Check if an element appears more than once
152.                 {
153.                     return 0;     // Return 0 to indicate not a function
154.                 }
155.             }
156.         }
157.         }
158.         return 1;          // Return 1 to indicate a function
159.     }
```

```
160.    // Function to print a matrix to a file
161.    void printMatrixToFile(int matrix[MAX_N][MAX_N], int n, FILE* file)
162.    {
163.        for (int a = 0; a < n; a++)       // Loop n times
164.        {
165.            for (int b = 0; b < n; b++) // Loop n times
166.            {
167.                fprintf(file, "%d ", matrix[a][b]); // Write the matrix elements to the file
168.            }
169.            fprintf(file, "\n");                    // Write a newline to the file after each row
170.        }
171.    }
```

OUTPUT:

n=2





*Relations - Notepad

File Edit Format View Help

```
Matrix 1 (Size: 1x1):
0

Symmetric: Yes
Antisymmetric: Yes
Transitive: Yes
Equivalence Relation: Yes
Function: Yes

Matrix 2 (Size: 4x4):
1 0 0 1
1 0 0 0
1 1 0 1
1 1 1 0

Symmetric: No
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No
```

Computational Time - Notepad

File Edit Format View Help

```
  Matrix 1:
Generation Time: 1171.000000 milliseconds

  Matrix 2:
Generation Time: 1171.000000 milliseconds


                Verification Time: 1171.000000 milliseconds
```

n=5

Relations - Notepad

File Edit Format View Help

```
Matrix 1 (Size: 10x10):
1 1 1 1 0 1 1 1 1 0
1 0 0 0 1 0 0 0 0 0
1 0 1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0 0 1
0 0 1 1 1 0 0 1 1 1
1 1 1 0 0 0 0 1 0 1
1 1 0 1 1 0 0 1 0 0
1 0 1 1 1 1 1 0 0 1 0
1 0 1 1 0 1 1 0 1 1
0 1 0 0 0 0 0 1 0 0

Symmetric: No
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No

Matrix 2 (Size: 10x10):
1 1 0 1 0 1 0 1 1 0
1 1 0 0 1 0 1 1 0 1
0 0 0 0 0 1 1 1 0 1
0 1 1 1 1 0 0 1 0 0
0 0 0 1 0 1 1 0 1 1
0 1 1 1 0 1 1 0 1 1
1 0 1 0 1 1 1 0 0 1
1 1 0 0 1 1 0 1 0 1
0 1 0 1 0 0 1 0 0 1
0 1 1 1 1 0 1 1 0 1

Symmetric: No
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No
```

Relations - Notepad

File Edit Format View Help

```
Matrix 3 (Size: 10x10):
1 0 1 0 0 1 0 1 0 0
0 1 0 1 1 0 0 0 0 0
0 0 1 1 0 0 0 0 0 1
1 1 1 1 1 0 1 0 0 0
1 1 1 1 1 1 0 0 1 1
0 0 0 0 1 1 1 0 0 1
1 0 1 0 1 1 1 0 1 0
1 0 1 1 1 1 0 0 1 1
0 0 0 1 0 0 1 0 0 1
1 0 0 1 0 1 0 0 1 1

Symmetric: No
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No

Matrix 4 (Size: 2x2):
1 1
1 0

Symmetric: Yes
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No
```

```
Matrix 5 (Size: 9x9):
0 0 1 0 0 0 0 1 1
0 0 0 1 0 1 1 1 1
0 0 0 0 1 1 0 1 0
1 1 1 1 1 0 0 1 1
1 1 1 0 1 0 1 0 1
0 1 1 0 0 0 1 1 0
1 0 0 1 0 0 0 0 1
0 0 0 1 1 1 1 1 1
0 0 0 0 1 0 1 1 1

Symmetric: No
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No
```

Computational Time - Notepad

File Edit Format View Help

```
  Matrix 1:
Generation Time: 1461.000000 milliseconds

  Matrix 2:
Generation Time: 1461.000000 milliseconds

  Matrix 3:
Generation Time: 1461.000000 milliseconds

  Matrix 4:
Generation Time: 1461.000000 milliseconds

  Matrix 5:
Generation Time: 1461.000000 milliseconds


                Verification Time: 1461.000000 milliseconds
```

n=10

Relations - Notepad

File Edit Format View Help

```
Matrix 1 (Size: 3x3):
0 1 0
0 0 1
1 1 0

Symmetric: No
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No

Matrix 2 (Size: 4x4):
0 0 1 1
0 0 0 0
0 0 0 1
1 1 0 1

Symmetric: No
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No

Matrix 3 (Size: 10x10):
0 0 1 1 0 0 0 1 0 0
0 0 0 0 1 1 0 0 0 0
0 1 1 1 0 1 1 1 0 1
1 1 0 1 0 1 0 0 0 1
0 0 0 1 0 0 0 0 1 0
1 0 1 1 1 1 1 0 1 1
0 0 0 1 0 1 0 0 1 1
0 0 1 1 1 0 1 0 0 1
1 1 0 1 0 0 1 0 0 1
1 0 1 1 0 1 1 1 1 0

Symmetric: No
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No
```

Relations - Notepad

File Edit Format View Help

```
Matrix 4 (Size: 3x3):
1 0 1
0 1 0
1 1 0

Symmetric: No
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No

Matrix 5 (Size: 10x10):
0 1 1 0 0 1 1 1 1 1
1 0 0 1 0 1 0 0 1 1
1 0 0 1 0 0 0 1 1 1
0 1 1 0 0 0 0 1 1 1
0 1 1 0 0 1 0 1 0 0
1 0 1 0 1 1 1 0 0 0
0 0 0 0 1 1 0 0 1 1
1 1 0 0 0 1 0 0 0 1
1 1 0 0 1 0 0 1 1 1
1 1 0 0 1 1 1 1 1 0

Symmetric: No
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No

Matrix 6 (Size: 4x4):
0 0 1 1
1 1 1 0
1 1 0 0
0 1 1 0

Symmetric: No
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No
```

*Relations - Notepad

File Edit Format View Help

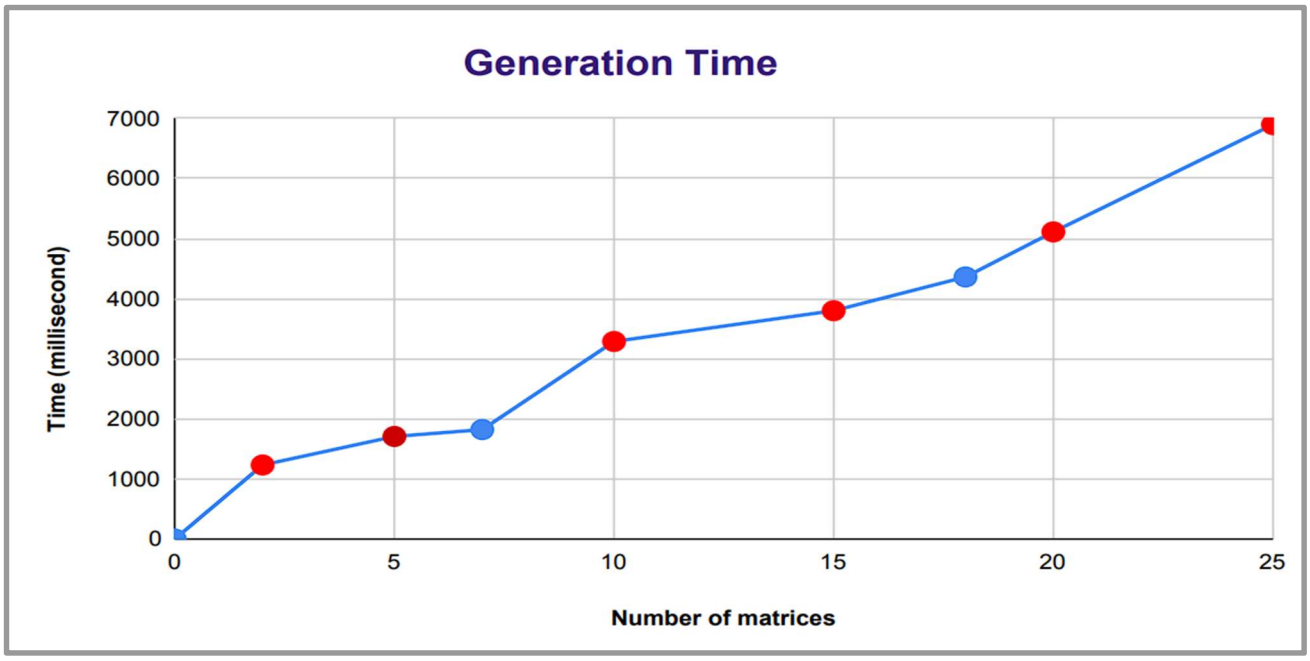
```
Matrix 7 (Size: 3x3):
1 0 1
0 1 0
1 1 0

Symmetric: No
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No

Matrix 8 (Size: 3x3):
0 0 1
0 0 1
0 1 0

Symmetric: No
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No

Matrix 9 (Size: 4x4):
0 0 1 1
1 1 1 0
1 1 0 0
0 1 1 0

Symmetric: No
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No

Matrix 10 (Size: 7x7):
0 0 1 0 0 0 0
1 0 0 1 1 1 0
0 0 1 1 0 0 1
0 0 1 1 0 0 0
1 1 0 0 0 1 0
1 1 0 0 0 0 1
1 0 0 0 1 1 0

Symmetric: No
Antisymmetric: No
Transitive: No
Equivalence Relation: No
Function: No
```

Computational Time - Notepad

File Edit Format View Help

```
  Matrix 1:
Generation Time: 2670.000000 milliseconds

  Matrix 2:
Generation Time: 2670.000000 milliseconds

  Matrix 3:
Generation Time: 2670.000000 milliseconds

  Matrix 4:
Generation Time: 2670.000000 milliseconds

  Matrix 5:
Generation Time: 2670.000000 milliseconds

  Matrix 6:
Generation Time: 2670.000000 milliseconds

  Matrix 7:
Generation Time: 2670.000000 milliseconds

  Matrix 8:
Generation Time: 2670.000000 milliseconds

  Matrix 9:
Generation Time: 2670.000000 milliseconds

  Matrix 10:
Generation Time: 2670.000000 milliseconds


                Verification Time: 2669.000000 milliseconds
```

Ln 1, Col 1



- **Graph Showing Computational time vs the number matrices with dimension**

Number of random matrices *(RxR)*	***Computing time of matrix verification***	***Computing time of function verification****(sec)*
0	0.0 ms	0
2	1232.00 ms	0
5	1707.00 ms	1
7	1821.00 ms	1
10	3288.00 ms	2
15	3800.00 ms	2
20	5112.00 ms	4
25	6898.00 ms	6

TIME COMPLEXITY:

The time complexity of an algorithm approximates just how much time it would take to solve a task of a specific size. Also, the time complexity of an algorithm may be represented as the number of operations performed by the algorithm when the input is of a certain size. According to the directions for our project, we created a graph of processing time vs n-vertices and compared it to the BigO notation graph. As an outcome, we determined Big O's estimated time complexity ( ). In the theory, we implemented three nested loops and a couple of extra functions to correctly build the entire program.

"THEORETICAL TIME COMPLEXITY"

Statement	Big O notation

Void generateRelationMatrix(int matrix[MAX_N][MAX_N], int n)
{
for (int a = 0; a < n; a++)
{
for (int b = 0; b < n; b++)
{
matrix[a][b] = rand() % 2;
}
}
}
	'generateRelationMatrix' function generates a matrix of size 'n' x 'n' where each element is set to a random value (0 or 1).
	
	The nested loops in this function run for n rows and n columns,
	
	resulting in a time complexity of O( ).

int isSymmetric(int matrix[MAX_N][MAX_N], int n)
{
for (int a = 0; a < n; a++)
{
for (int b = a + 1; b < n; b++)
{
if (matrix[a][b] != matrix[b][a])
{
return 0;
}
}
}
return 1;
}
	'isSymmetric' Function:
	
	This function checks if a matrix is symmetric.
	It has two nested loops, but it only iterates over the upper triangular part of the matrix (triangle without diagonal).
	
	Time Complexity: O(n^2)

int isAntisymmetric(int matrix[MAX_N][MAX_N], int n)
{
for (int a = 0; a < n; a++)
{
for (int b = 0; b < n; b++)
{
if (a != b && matrix[a][b] && matrix[b][a])
{
return 0;
}
}
}
return 1;
}
	'isAntisymmetric' Function:
	
	This function checks if a matrix is antisymmetric.
	It has two nested loops iterating over n rows and n columns.
	
	Time Complexity: O(n^2)

int isTransitive(int matrix[MAX_N][MAX_N], int n)
{
for (int a = 0; a < n; a++)
{
for (int b = 0; b < n; b++)
{
for (int c = 0; c < n; c++)
{
if (matrix[a][b] && matrix[b][c] && !matrix[a][c])
{
return 0;
}
}
}
}
return 1;
}
	The 'isTransitive function' checks if a matrix is transitive. It has three nested loops, each running for 'n' rows and 'n' columns.
	
	Therefore, the time complexity is O(n^3).

int isEquivalence(int matrix[MAX_N][MAX_N], int n)
{
if (isSymmetric(matrix, n) && isTransitive(matrix, n))
{
return 1;
}
return 0;
}
	The isEquivalence function checks if a matrix is an equivalence relation, which it calls isSymmetric and isTransitive. As mentioned earlier, both of these functions have a time complexity of O(n^2).
	
	Therefore, the time complexity of isEquivalence is also O(n^2).

int isFunction(int matrix[MAX_N][MAX_N], int n)
{
int used[n];
for (int a = 0; a < n; a++)
{
used[a] = 0;
}
for (int a = 0; a < n; a++)
{
for (int b = 0; b < n; b++)
{
if (matrix[a][b])
{
used[b]++;
if (used[b] > 1)
{
return 0;
}
}
}
}
return 1;
}
	The 'isFunction' function checks if a matrix represents a function. It has nested loops that run for 'n' rows and 'n' columns,
	resulting in a time complexity of O(n^2).

The big O notation	*The **main loop*** runs n times and generates, analyzes, and writes results for 'n' random matrices. Within each iteration, it calls the functions described above. Therefore, the total time complexity is approximately O(n * (n^2 + n^2 + n^3)) = O(n^4 + n^3). In Big O notation, this can be simplified to *O(n^4)*

The dominant factor in the time complexity of the program is the main loop, which has a time complexity of O(n^4). In summary, the program has a time complexity of O(n^4) due to the main loop, where n is the number of relation matrices to be processed.

Hence, the time complexity of our program is: $O(n)=n^4$
So, we can see that the graph's time complexity and the program's time complexity which we determined have been matched.

~~~~~~ The End ~~~~~~
~~~~~~