EAST WEST UNIVERSITY

PROJECT REPORT

Course Title: Discrete Mathematics
Course Code: CSE106
Sec: 06

Submitted to:

Dr. Mohammad Salah Uddin

Associate Professor
Department of Computer Science & Engineering

Submitted by:

(1) Md. Saiful Islam
ID: 2022-3-60-045

(2) Erfan Sarker Efte
ID: 2022-3-60-042

(3) Shanghita Naha Sristy
ID: 2022-3-60-311

Submission Date:

22th May, 2023

introduction:

Using C program, we are going to randomly generate a relational matrix (which represent relation) with dimension n where n is the number of distinct elements on a set. Then it is going to verify the properties of the relation; such as symmetric, anti-symmetric, transitive, and equivalence, etc. Also it will determine computational time in milliseconds. It will check whether the relation represents any function or not.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define max 700

void symgen(int n, int m[max][max]) {
    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            if (i == j) {
                m[i][j] = 1;
            } else {
                int value = rand() % 2;
                m[i][j] = value;
                m[j][i] = value;
            }
        }
    }
}

void antisymgen(int n, int m[max][max]) {
    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
                m[i][j] = 0;
            } else {
                int value = rand() % 2;
```

```
                m[i][j] = value;
                m[j][i] = 1 - value;
            }
        }
    }
}

void trangen(int n, int m[max][max]) {
    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
                m[i][j] = 1;
            } else {
                m[i][j] = rand() % 2;
            }
        }
    }

    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (m[i][k] == 1 && m[k][j] == 1) {
                    m[i][j] = 1;
                }
            }
        }
    }
}

void printm(int n, int m[max][max]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", m[i][j]);
        }
        printf("\n");
    }
}
```

```
int symmetricm(int n,int m[max][max]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (m[i][j] != m[j][i]) {
                return 0;
            }
        }
    }
    return 1;
}

int anti_symmetric(int n, int m[max][max]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (m[i][j] && m[j][i] && i != j) {

                return 0;
            }
        }
    }
    return 1;
}

int transitive(int n, int m[max][max]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                if (m[i][j] && m[j][k] && !m[i][k]) {

                    return 0;
                }
            }
        }
    }
    return 1;
}
```

```
int equivalence(int n, int m[][max]) {
    return symmetricm(n,m) && transitive(n,m);
}

int function(int n, int m[max][max]) {
    for (int i = 0; i < n; i++) {
        int count = 0;
        for (int j = 0; j < n; j++) {
            if (m[i][j]) {
                count++;
            }
        }
        if (count != 1) {
            return 0;
        }
    }
    return 1;
}

int main () {
 int n;
  clock_t start,end;
  double cpu_time_m;
  double cpu_time_f;
    start=clock();

printf("Enter the dimension value(n):");
 scanf("%d",&n);
 int m[max][max];
srand(time(NULL));
    int x=rand()% 3;

switch(x) {
    case 0 :
    symgen(n,m);
    break;
    case 1 :
```

```
    antisymgen(n,m);
    break;
    case 2 :
    trangen(n,m);
    break;
    }

    printf("\nRandom Matrix:");
    printf("\n---------------\n\n");

    printm(n,m);

printf("\nVerifying the relation metrix:");
printf("\n------------------------------\n\n");

    if (symmetricm(n, m)) {
        printf("The relation is symmetric.\n");
    } else {
        printf("The relation is not symmetric.\n");
    }
    if (anti_symmetric(n,m)) {
        printf("The relation is anti-symmetric.\n");
    } else {
        printf("The relation is not anti-symmetric.\n");
    }
    if (transitive(n, m)) {
        printf("The relation is transitive.\n");
    } else {
        printf("The relation is not transitive.\n");
    }
    if (equivalence(n,m)) {
        printf("The relation is an equivalence relation.\n");
    } else {
        printf("The relation is not an equivalence relation.\n");
    }

    end = clock();
    cpu_time_m = ((double) (end - start)) / CLOCKS_PER_SEC;
```

```
    start=clock();

    if (function(n,m)) {
        printf("The relation represents a function.\n");
    } else {
        printf("The relation does not represent a function.\n");
    }

    end = clock();
    cpu_time_f = ((double) (end - start)) / CLOCKS_PER_SEC;

    printf("\nComputation Time:");
printf("\n------------------\n\n");

    printf("verification time for relation: %.2f milliseconds\n", cpu_time_m * 1000);
    printf("verification time for function: %.2f milliseconds\n", cpu_time_f * 1000);
return 0;

}
```

Output:

n=1,

```
Verifying the relation metrix:
------------------------------

The dimension of the metrix is : 1

The relation is symmetric.
The relation is anti-symmetric.
The relation is transitive.
The relation is an equivalence relation.
The relation does not represent a function.

Computation Time:
------------------

verification time for relation: 319.00 milliseconds
verification time for function: 0.00 milliseconds
```

n=50,

```
Verifying the relation metrix:
------------------------------

The dimension of the metrix is : 50

The relation is not symmetric.
The relation is anti-symmetric.
The relation is not transitive.
The relation is not an equivalence relation.
The relation does not represent a function.

Computation Time:
------------------

verification time for relation: 719.00 milliseconds
verification time for function: 0.00 milliseconds
```

n=100,

```
Verifying the relation metrix:
------------------------------

The dimension of the metrix is : 100

The relation is symmetric.
The relation is not anti-symmetric.
The relation is not transitive.
The relation is not an equivalence relation.
The relation does not represent a function.

Computation Time:
------------------

verification time for relation: 1359.00 milliseconds
verification time for function: 1.00 milliseconds
```

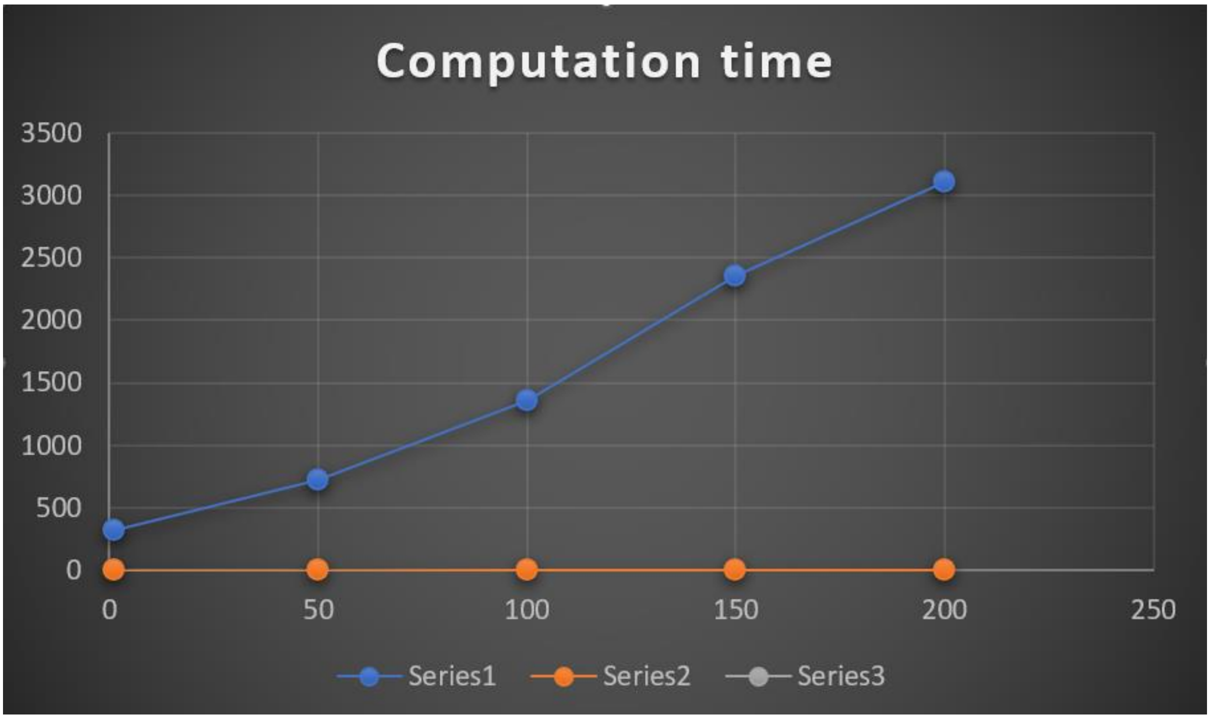
n=150,

```
Verifying the relation metrix:
------------------------------

The dimension of the metrix is : 150

The relation is symmetric.
The relation is not anti-symmetric.
The relation is not transitive.
The relation is not an equivalence relation.
The relation does not represent a function.

Computation Time:
------------------

verification time for relation: 2351.00 milliseconds
verification time for function: 1.00 milliseconds
```

n=200,

```
Verifying the relation metrix:
------------------------------

The dimension of the metrix is : 200

The relation is symmetric.
The relation is not anti-symmetric.
The relation is transitive.
The relation is an equivalence relation.
The relation does not represent a function.

Computation Time:
------------------

verification time for relation: 3104.00 milliseconds
verification time for function: 1.00 milliseconds
```

Graph:



Graph Showing Computational time vs the number of dimension

Dimension	Computing time of matrix verification	Computing time of function verification
1	319	0
50	719	0
100	1359	1
150	2351	1
200	3104	1

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 Big O notation graph. As an outcome, we determined Big O's estimated time complexity (n3). In the theory, we implemented three nested loops and a couple of extra functions to correctly build the entire program.

THEORETICAL TIME COMPLEXITY

<table>
<tr><th>Statement</th><th>Big O notation</th></tr>
<tr><td><pre>
  for (int i = 0; i < n; i++) {
      for (int j = i; j < n; j++) {
          if (i == j) {
              m[i][j] = 1;
          } else {
              int value = rand() % 2;
              m[i][j] = value;
              m[j][i] = value;
          }
      }
  }
}
</pre></td><td>f1=n*(3n+1)+1

$=O(n^2)$</td></tr>
</table>

<table>
<tr>
<td>
<pre><code>   for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
                m[i][j] = 0;
            } else {
                int value = rand() % 2;
                m[i][j] = value;
                m[j][i] = 1 - value;
            }
        }
    }
}</code></pre>
</td>
<td>

$f2=n*(3n+1)+1$

$=O(n^2)$

</td>
</tr>
<tr>
<td>
<pre><code>   for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
                m[i][j] = 1;
            } else {
                m[i][j] = rand() % 2;
            }
        }
    }

    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (m[i][k] == 1 && m[k][j] == 1) {
                    m[i][j] = 1;
                }
            }
        }
    }
}</code></pre>
</td>
<td>

$f3=(n*(3n+1)+1)+(n*(n*(2n+1)+1)+1)$

$=O(n^3)$

</td>
</tr>
</table>

`for (int i = 0; i < n; i++) {`
`    for (int j = 0; j < n; j++) {`
`        printf("%d ", m[i][j]);`
`    }`
`    printf("\n");`
`}`
	$f4=n*(n+1)+1$
	$=O(n^2)$

`for (int i = 0; i < n; i++) {`
`    for (int j = 0; j < n; j++) {`
`        if (m[i][j] != m[j][i]) {`
`            return 0;`
`        }`
`    }`
`}`
`return 1;`
	$f5=n*(2n+1)+1$
	$=O(n^2)$

`for (int i = 0; i < n; i++) {`
`    for (int j = 0; j < n; j++) {`
`        if (i == j) {`
`            m[i][j] = 0;`
`        } else {`
`            int value = rand() % 2;`
`            m[i][j] = value;`
`            m[j][i] = 1 - value;`
`        }`
`    }`
`}`
	$f6=n*(3n+1)+1$
	$=O(n^2)$

`for (int i = 0; i < n; i++) {`
`    for (int j = 0; j < n; j++) {`
`        if (m[i][j] && m[j][i] && i != j) {`
 	
`            return 0;`
`        }`
`    }`
`}`
`return 1;`
	$f7=n*(2n+1)+1$
	$=O(n^2)$

<table>
<tr><td><pre>for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        for (int k = 0; k < n; k++) {
            if (m[i][j] && m[j][k] && !m[i][k]) {
                return 0;
            }
        }
    }
}
return 1;</pre></td><td>$f8=n*(2n+1)+1$
$=O(n^2)$</td></tr>
<tr><td><pre>for (int i = 0; i < n; i++) {
    int count = 0;
    for (int j = 0; j < n; j++) {
        if (m[i][j]) {
            count++;
        }
    }
    if (count != 1) {
        return 0;
    }
}
return 1;</pre></td><td>$f9=n*(3n+1)+1$
$=O(n^2)$</td></tr>
<tr><td>The big O notation</td><td>$O(\max(n^2, n^3)) = O(n^3)$</td></tr>
</table>

Hence, the time complexity of our program is: $O(n)=n^3$
So, we can see that the graph's time complexity and the program's time complexity which we determined have been matched.

{ THE END }