

Informe Final Proyecto 1

Pablo Alfaro Castro - B20162
Carlos Jimenez Arroyo - B23397

24 de octubre de 2016

Índice

1. Reseña	1
1.1. Estructura	1
2. Funcionamiento	2
3. Experimentos Realizados	5
4. Resultados Obtenidos	6
5. Análisis de tiempos de ejecución y complejidad computacional	10
6. Conclusiones	11
7. Bibliografía	12
8. Anexos	12
8.1. main.cpp	12
8.2. shearsort.h	13
8.3. shearsort.cpp	13

1. Reseña

Shearsort es un algoritmo capaz de ordenar una matriz de dimensiones $N \times N$, con la singularidad de que esta queda ordenada en forma serpenteada siendo $N_{0,0}$ el elemento menor, y $N_{N,0}$ el elemento mas grande.

1.1. Estructura

Teóricamente el algoritmo toma raíz de N veces tiempo para ordenar cada uno de los arreglos, estos ordenamientos suceden periódicamente hasta que un ciclo central se cierre, dentro de este ciclo se ejecutan las instrucciones necesarias para el ordenamiento, este ciclo consta de dos fases:

- Fase 1: Ejecutar N veces:
 - Ordenar las filas pares de derecha a izquierda, o las impares de izquierda a derecha.

- ordenar las columnas de arriba hacia abajo.
- Fase 2: Ordenar filas alternando el método anterior.

Ordenar una fila de longitud n con la transición par-impar toma n pasos. Dado que el algoritmo realiza $\log(n)$ iteraciones, se requiere $n\log(n)$ pasos para la fila de clasificación más n pasos para la fila adicional de clasificación operación. En cada iteración, la altura de la zona sin clasificar disminuye en un factor de 2. Esto significa que las columnas contienen una zona sin clasificar de longitud decreciente. Ordenar una columna que contiene una zona no seleccionada de longitud N toma medidas de transiciones tipo par-impar. Por lo tanto, el algoritmo requiere el siguiente número de pasos:

$$n + \frac{n}{2} + \frac{n}{4} + \dots + 2 = 2n - 2 \quad (1)$$

Donde predomina solamente el n en la ecuación, por lo tanto, si para cada ordenamiento tarda $\log(n)$, la complejidad total vendrá dada por $O(n\log(n))$.

2. Funcionamiento

En la figura 1 se muestra el archivo de encabezados de nuestro programa:

```
#ifndef SHEARSORT_H
#define SHEARSORT_H

#include <cstdlib>
#include <iostream>
#include "string"

using namespace std;

class Shearsort{
public:

    int n;
    double **matriz;

    Shearsort();
    Shearsort(int n, double** matriz);
    ~Shearsort();
    void sort();
    void operator~();

};
#endif /* SHEARSORT_H */
```

Figura 1: Encabezados

Se puede ver la estructura básica del algoritmo que se programó, comenzando por el constructor de la clase, así como los atributos y métodos que se listan continuación:

- public int n: Corresponde al tamaño de la matriz que se desea ordenar.
- public double** matriz: Corresponde a la matriz ya armada y en memoria que se quiere ordenar.
- public void sort(): Este es el método principal que se encarga de realizar todo el ordenamiento y la lógica requerida, es un método vacío ya que todo se trabaja en la matriz que recibe la clase Shearsort().
- public void operator (): Este corresponde al operador de impresión de la matriz, que se usó en la matriz antes y después del ordenamiento.

En las figuras 2 y 3 se presenta el código que se implementó en el método sort, este método tiene dos variables dentro, que son:

- int contador: Esta variable esta encargada de almacenar el limite de iteraciones del ciclo central.
- int temp: Esta variable se encarga de almacenar temporalmente el numero mayor o menor según sea el caso, para poder hacer los intercambios entre las filas y las columnas.

```
void Shearsort::sort(){  
  
    int contador = this->n;  
    int temp;  
    for (int i = 0; i < contador; i++) {  
  
        for (int i = 0; i < this->n; i++) {  
            if (i%2==0) {  
                for (int j = 0; j < this->n; j++) {  
                    for (int k = 0; k < this->n-1; k++) {  
                        if (this->matriz[i][k] > this->matriz[i][k+1]) {  
  
                            temp = this->matriz[i][k];  
                            this->matriz[i][k] = this->matriz[i][k+1];  
                            this->matriz[i][k+1] = temp;  
                        }  
                    }  
                }  
            }  
            else{  
                for (int j = 0; j < this->n; j++) {  
                    for (int k = 0; k < this->n-1; k++) {  
                        if (this->matriz[i][k] < this->matriz[i][k+1]) {
```

Figura 2: Shearsot

Luego, vienen los ciclos, el ciclo principal, ejecuta los procesos n veces, siendo n el tamaño de la matriz, este límite se determinó gracias a ciertos experimentos realizados a mano, seguidamente, dentro de el ciclo central se encuentran los 2 procesos básicos de algoritmo, el ordenamiento de las filas, y el ordenamiento de las columnas, donde cada uno ejecuta un ordenamiento con el método de intercambio de variables.

```

        temp = this->matriz[i][k];
        this->matriz[i][k] = this->matriz[i][k+1];
        this->matriz[i][k+1] = temp;
    }
}
}
}
}
for (int i = 0; i < this->n; i++) {
    for (int j = 0; j < this->n; j++) {
        for (int k = 0; k < this->n-1; k++) {
            if (this->matriz[k][i] > this->matriz[k+1][i]) {

                temp = this->matriz[k][i];
                this->matriz[k][i] = this->matriz[k+1][i];
                this->matriz[k+1][i] = temp;
            }
        }
    }
}
}
}
}
}

```

Figura 3: Shearsort

En la figura 4 se muestra el código del operador de impresión, el cual es sencillo, ya que este solo utiliza dos ciclos que van recorriendo toda la matriz, y al final simplemente se imprimió el contenido de esta, uno por uno.

```

void Shearsort::operator~() {

    for(int i=0;i<this->n;i++){
        for(int j=0; j<this->n; j++){
            cout<< this->matriz[i][j]<<"\t";
        }
        cout<<endl;
    }
}

```

Figura 4: Operador de impresión

```

int main(int argc, char** argv) {

    int n = atoi(argv[1]);

    double** mat = new double*[n];
    for (int i = 0; i < n; i++) {
        mat[i] = new double[n];
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            mat[i][j] = rand()%100 + 1;
        }
    }

    Shearsort s1(n, mat);
    cout << " " << endl;
    cout << "Matriz sin ordenar" << endl;
    cout << " " << endl;
    ~s1;
    cout << " " << endl;
    s1.sort();
    cout << " " << endl;
    cout << "Matriz ordenada" << endl;
    cout << " " << endl;
    ~s1;
    delete mat;
}

```

Figura 5: Main

Por ultimo, la figura 5 muestra el código del main, en el cual podemos ver como desde ahí se construyo un objeto de la clase matriz, con atributo n establecido por el usuario y una matriz generada aleatoriamente como se muestra en la imagen, de ese modo se asignó memoria para la matriz, ya por ultimo, se muestra la llamada al método sort, y el operador de impresión antes y después de ordenada la matriz.

3. Experimentos Realizados

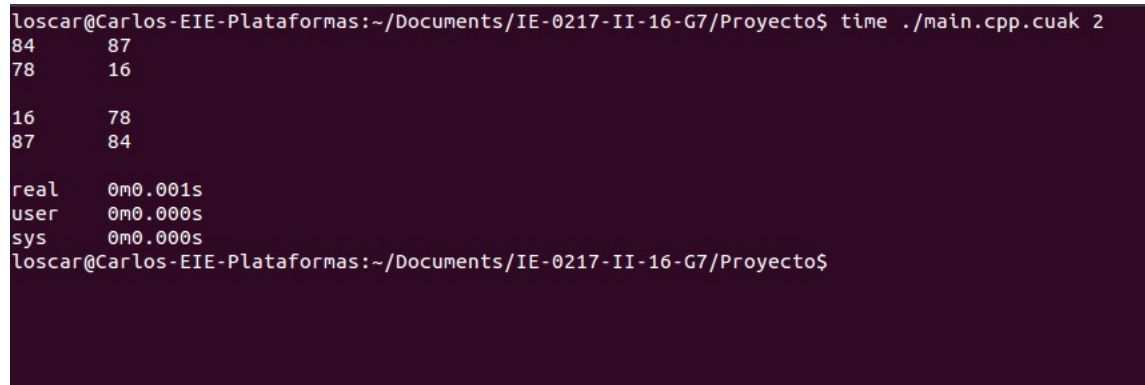
Para determinar la condición de parada del ciclo central del algoritmo, se realizaron pruebas escritas y hechas a mano, probando el algoritmo con matrices de pequeño tamaño, así se pudieron contar las veces que se repetía el ciclo para de esta manera determinar un patrón de repetición, el

cual se encontró por estos medios, de modo que si se contaban los ciclos individualmente (tanto columnas y filas), se requería $2N - 1$ ciclos para poder realizar el ordenamiento, debido a que en la manera que se programó el código los ciclos se ejecutan doblemente (en conjunto fila y columna) el numero de culos se vería reducido a la mitad, dejando el limite como N , ya que hablar de medio ciclo no es una posibilidad en este caso.

Para realizar los experimentos y pruebas se utilizaron 2 computadores portátiles, una computadora personal con sistema operativo Debian, 6GB de RAM, Procesador Core i5, y una máquina con el sistema operativo Ubuntu instalado en una máquina virtual, con 2 GB de RAM y un procesador intel core i7, asimismo se decidió por llenar las matrices de forma aleatoria con números del 1 al 100, y que el usuario sea quien determine las dimensiones de la matriz a ordenar. Se realizaron varias pruebas midiendo el tiempo del sistema en ejecutar lo solicitado aumentando la dimensión de la matriz empezando en 2 elevado a la 1 y aumentando el exponente en una unidad.

4. Resultados Obtenidos

De acuerdo a los experimentos realizados poniendo a prueba el código escrito en el lenguaje de C++ para implementar el algoritmo Shearsort, se realizaron pruebas hasta que el tiempo de respuesta del programa se elevó hasta el punto que el programa no imprime una respuesta y se queda colgado”. Como se mencionó antes, al ir aumentando las dimensiones de la matriz en potencias de 2, se lograron obtener 8 muestras con tiempos constantes, A continuación se muestran capturas de pantalla del programa trabajando en las 8 diferentes pruebas realizadas.



```
loскар@Carlos-EIE-Plataformas:~/Documents/IE-0217-II-16-G7/Proyecto$ time ./main.cpp.cuak 2
84      87
78      16

16      78
87      84

real    0m0.001s
user    0m0.000s
sys     0m0.000s
loскар@Carlos-EIE-Plataformas:~/Documents/IE-0217-II-16-G7/Proyecto$
```

Figura 6: Para una matriz 2x2

```

loscar@Carlos-EIE-Plataformas:~/Documents/IE-0217-II-16-G7/Proyecto$ time ./main.cpp.cuak 4
84      87      78      16
94      36      87      93
50      22      63      28
91      60      64      27

16      22      27      28
63      60      50      36
64      78      84      87
94      93      91      87

real    0m0.001s
user    0m0.000s
sys      0m0.000s
loscar@Carlos-EIE-Plataformas:~/Documents/IE-0217-II-16-G7/Proyecto$ █

```

Figura 7: Para una matriz 4x4

```

loscar@Carlos-EIE-Plataformas:~/Documents/IE-0217-II-16-G7/Proyecto$ time ./main.cpp.cuak 8
84      87      78      16      94      36      87      93
50      22      63      28      91      60      64      27
41      27      73      37      12      69      68      30
83      31      63      24      68      36      30      3
23      59      70      68      94      57      12      43
30      74      22      20      85      38      99      25
16      71      14      27      92      81      57      74
63      71      97      82      6      26      85      28

3       6       12      12      14      16      16      20
27      27      26      25      24      23      22      22
27      28      28      30      30      30      31      36
57      57      50      43      41      38      37      36
59      60      63      63      63      64      68      68
74      74      73      71      71      70      69      68
78      81      82      83      84      85      85      87
99      97      94      94      93      92      91      87

real    0m0.001s
user    0m0.000s
sys      0m0.000s
loscar@Carlos-EIE-Plataformas:~/Documents/IE-0217-II-16-G7/Proyecto$ █

```

Figura 8: Para una matriz 8x8

```

loscar@Carlos-EIE-Plataformas:~/Documents/IE-0217-II-16-G7/Proyecto5$ time ./main.cpp.cuak 16
84      87      78      16      94      36      87      93      50      22      63      28      91      60      64      27
41      27      73      37      12      69      68      30      83      31      63      24      68      36      30      3
23      59      70      68      94      57      12      43      30      74      22      20      85      38      99      25
16      71      14      27      92      81      57      74      63      71      97      82      6      26      85      28
37      6      47      30      14      58      25      96      83      46      15      68      35      65      44      51
88      9      77      79      89      85      4      52      55      100      33      61      77      69      40      13
27      87      95      40      96      71      35      79      68      2      98      3      18      93      53      57
2      81      87      42      66      90      45      20      41      30      32      18      98      72      82      76
10      28      68      57      98      54      87      66      7      84      20      25      29      72      33      30
4      20      71      69      9      16      41      50      97      24      19      46      47      52      22      56
80      89      65      29      42      51      94      1      35      65      25      15      88      57      44      92
28      66      60      37      33      52      38      29      76      8      75      22      59      96      30      38
36      94      19      29      44      12      29      30      77      5      44      64      14      39      7      41
5      19      29      89      70      18      18      97      25      44      71      84      91      100      73      26
45      91      6      40      55      87      70      83      43      65      98      8      56      5      49      12
23      29      100      44      47      69      41      23      12      11      6      2      62      31      79      6

1      2      2      2      3      3      4      4      5      5      5      6      6      6      6      6
14      14      13      12      12      12      12      12      11      10      9      9      8      8      7      7
14      15      15      16      16      16      18      18      18      18      19      19      19      20      20      20
26      25      25      25      25      25      24      24      23      23      23      22      22      22      22      20
26      27      27      27      27      28      28      28      28      29      29      29      29      29      29      29
35      35      33      33      33      32      31      31      30      30      30      30      30      30      30      30
35      36      36      36      37      37      37      38      38      38      39      40      40      40      41      41
46      45      45      44      44      44      44      44      43      43      42      42      41      41      41      41
46      47      47      47      49      50      50      51      51      52      52      52      53      54      55      55
63      63      62      61      60      60      59      59      58      57      57      57      57      57      56      56
63      64      64      65      65      65      66      66      66      66      68      68      68      68      68      68
73      73      72      72      71      71      71      71      71      70      70      70      69      69      69      69
74      74      75      76      76      77      77      77      78      79      79      79      80      81      81      82
87      87      87      87      87      87      85      85      85      84      84      84      83      83      83      82
88      88      89      89      89      90      91      91      91      92      92      93      93      94      94      94
100      100      100      99      98      98      98      98      97      97      97      96      96      95      94      94

real    0m0.002s
user    0m0.000s
sys     0m0.000s
loscar@Carlos-EIE-Plataformas:~/Documents/IE-0217-II-16-G7/Proyecto5$

```

Figura 9: Para una matriz 16x16

```

Carlos-EIE-Plataformas: ~/Documents/IE-0217-II-16-G7/Proyecto5
9      29      29      29      29      29      32      32      32      32      32      32      32      33      33      33      33      33      33      33      33      34      34      3
31      31      31      31      32      32      32      32      32      32      32      32      33      33      33      33      33      33      33      33      34      34      3
4      34      34      34      34      34      34      34      34      34      34      34      34      34      34      34      34      34      34      34      34      34      3
37      37      37      37      37      37      37      37      37      37      36      36      36      36      36      36      36      36      35      35      35      35      3
5      35      35      35      35      35      34      34      34      34      34      34      34      34      34      34      34      34      35      35      35      35      3
37      37      37      37      38      38      38      38      38      38      38      38      39      39      39      39      39      39      39      40      40      40      40      4
0      40      40      40      40      40      40      40      40      40      40      40      41      41      41      41      41      41      41      41      41      41      41      4
44      44      43      43      43      43      43      43      43      42      42      42      42      42      42      41      41      41      41      41      41      41      41      4
1      41      41      41      40      40      40      40      40      40      40      40      40      40      40      40      40      40      40      40      40      40      4
44      44      44      44      44      44      44      44      44      45      45      45      45      45      45      45      45      46      46      46      46      47      47      4
7      47      47      47      47      48      48      48      48      48      48      48      48      48      48      48      48      48      48      48      48      48      48      4
51      51      51      51      51      51      51      51      51      50      50      50      50      50      50      50      50      50      49      49      49      49      49      48      4
8      48      48      48      48      48      48      48      48      48      48      48      48      48      48      48      48      48      48      48      48      48      48      4
51      51      51      51      52      52      52      52      52      52      52      53      53      53      53      53      53      53      53      54      54      54      54      5
4      54      54      55      55      55      55      55      55      55      55      56      56      56      56      56      56      56      56      56      56      55      55      5
5      55      55      55      55      55      55      55      55      55      55      55      55      55      55      55      55      55      55      55      55      55      55      5
58      58      58      58      58      58      58      58      58      58      58      59      59      59      59      59      59      59      59      59      59      60      60      6
0      60      60      60      60      60      60      60      60      60      60      60      60      60      60      60      60      60      60      60      60      60      60      6
63      63      63      63      63      63      63      63      63      63      63      62      62      62      62      62      62      62      61      61      61      61      61      6
1      61      61      60      60      60      60      60      60      60      60      60      60      60      60      60      60      60      60      60      60      60      60      6
64      64      64      64      64      64      64      64      64      64      65      65      65      65      65      65      66      66      66      66      66      67      67      6
68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      6
71      71      71      71      71      71      71      71      71      70      70      70      70      70      70      69      69      69      69      69      69      69      69      69      6
68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      68      6
71      71      71      71      71      71      71      71      71      71      71      72      72      72      72      72      72      72      72      73      73      73      73      7
73      73      73      73      74      74      74      74      74      74      74      74      74      74      74      74      74      74      74      74      74      74      74      7
77      77      77      77      77      77      77      77      77      77      77      76      76      76      76      76      76      76      75      75      75      75      75      7
74      74      74      74      74      74      74      74      74      74      74      74      74      74      74      74      74      74      74      74      74      74      74      7
77      77      77      78      78      78      78      78      78      78      78      78      78      79      79      79      79      79      79      79      80      80      80      80      8
0      80      80      80      81      81      81      81      81      81      81      81      81      81      81      81      81      81      81      81      81      81      81      81      8
83      83      83      83      83      83      83      83      83      83      83      83      83      82      82      82      82      82      82      82      82      82      82      82      8
2      82      82      82      81      81      81      81      81      81      81      81      81      81      81      81      81      81      81      81      81      81      81      81      8
83      83      84      84      84      84      84      84      84      84      84      84      84      84      84      84      84      85      85      85      85      85      85      8
5      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      8
89      89      89      89      89      89      89      89      89      89      88      88      88      88      88      88      88      88      87      87      87      87      87      87      8
7      87      87      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      86      8
89      89      89      90      90      90      90      90      90      90      90      90      90      91      91      91      91      91      91      91      91      91      91      91      9
1      91      91      91      91      91      91      91      91      91      91      91      91      91      91      91      91      91      91      91      91      91      91      91      9
95      95      94      94      94      94      94      94      94      94      94      94      94      94      94      93      93      93      93      93      93      93      93      92      9
2      92      92      92      92      92      92      92      92      92      92      92      92      92      92      92      92      92      92      92      92      92      92      92      9
95      95      95      95      95      95      96      96      96      96      96      96      96      96      97      97      97      97      97      97      97      97      97      97      98
98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98
100      100      100      100      100      100      100      100      100      100      100      100      100      100      99      99      99      99      99      99      99      99      99      99
8      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98      98

real    0m0.019s
user    0m0.000s
sys     0m0.000s
loscar@Carlos-EIE-Plataformas:~/Documents/IE-0217-II-16-G7/Proyecto5$

```

Figura 10: Para una matriz 32x32

[illegible]

Figura 11: Para una matriz 64x64

[illegible]

Figura 12: Para una matriz 128x128

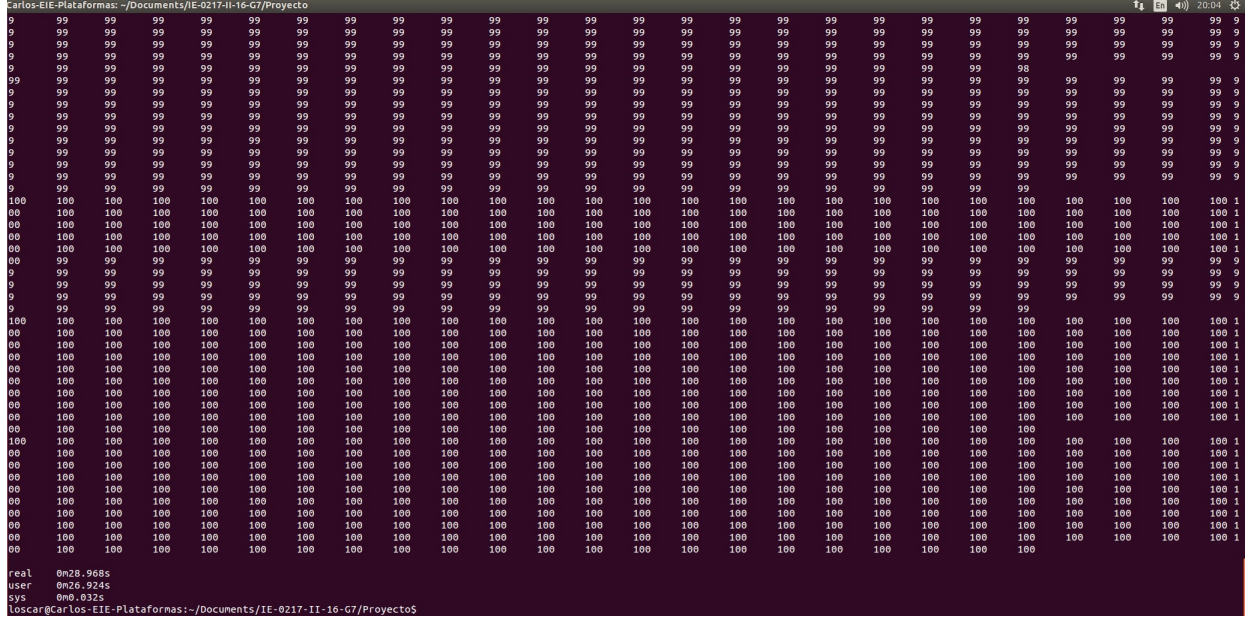


Figura 13: Para una matriz 256x256

Así como se puede apreciar en las capturas de pantalla del programa mostradas, el tiempo que se toma para verificar el funcionamiento correcto del programa corresponde al tiempo mostrado como real.^{en} las 8 figuras anteriores. Se decidió realizar la prueba varias veces para cada una de las 8 dimensiones que el programa logró manejar, a continuación se presenta una tabla que muestra el tiempo promedio en segundos que el programa duró en ordenar las matrices, estos promedios se sacaron luego de realizar 8 pruebas por cada dimensión de matriz.

Dimensión de la Matriz	Duración registrada del algoritmo	
Matriz de tamaño 2x2.	0.001 segundos	
Matriz de tamaño 4x4.	0.001 segundos	
Matriz de tamaño 8x8.	0.001 segundos	
Matriz de tamaño 16x16.	0.002 segundos	
Matriz de tamaño 32x32.	0.017 segundos	
Matriz de tamaño 64x64.	0.168 segundos	
Matriz de tamaño 128x128.	1.917 segundos	
Matriz de tamaño 256x256.	28.023 segundos	+

De esta forma vemos como el tiempo que requiere el programa para poder trabajar se aumenta considerablemente a partir de la dimensión 256x256, hasta el punto que que no se pudo obtener una muestra de tiempo para un arreglo de tamaño 512x512.

5. Análisis de tiempos de ejecución y complejidad computacional

Teóricamente se sabe que el algoritmo Shearsort puede llegar a poseer complejidad de:

$$\sqrt{n} * \log(n) \quad (2)$$

Esto en el peor de los casos, sin embargo, en la implementación hecha en el experimento realizado se tiene un comportamiento más parecido a una complejidad de:

$$n * \log(n) \quad (3)$$

Dicha complejidad también puede presentarse en algunos casos de este algoritmo.

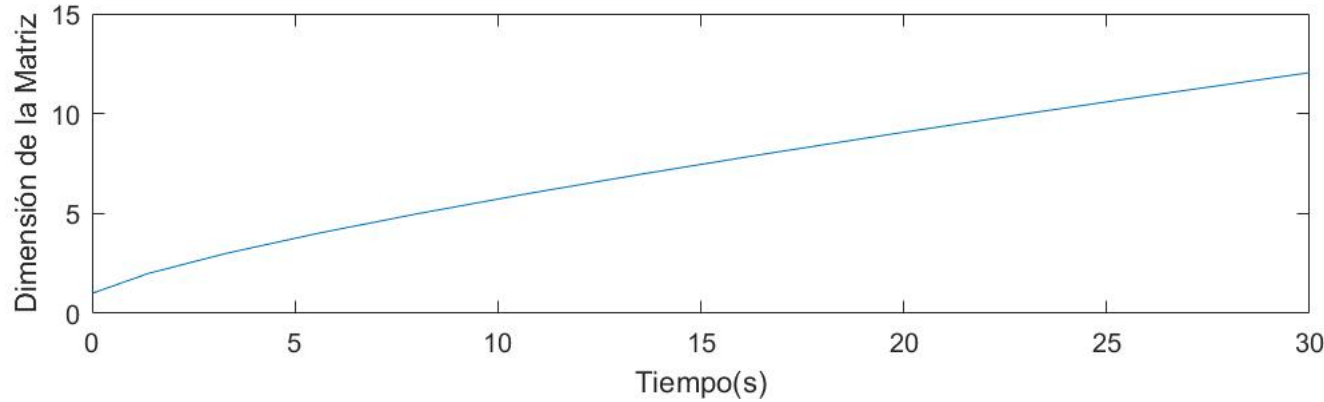


Figura 14: Gráfica para la complejidad $n \log(n)$.

Como se puede ver en la figura 14, la gráfica corresponde a la representación gráfica de la complejidad de la $n \log(n)$.

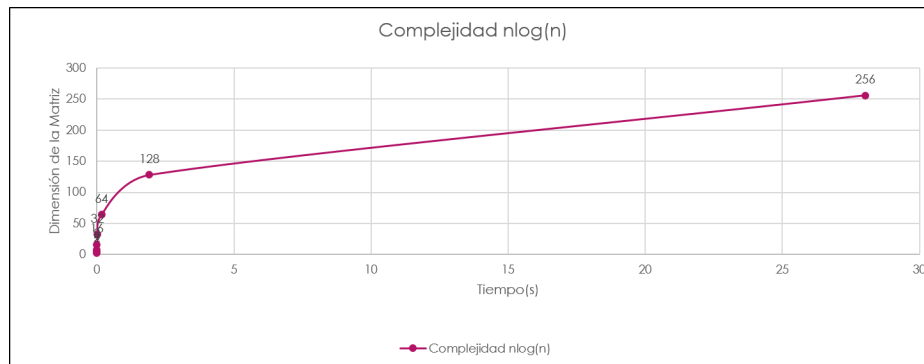


Figura 15: Gráfica obtenida en durante la experimentación.

En la figura 15 tenemos la gráfica que se obtuvo con los tiempos promediados de trabajo del programa, como se puede ver, existe un comportamiento similar con la gráfica presentada en la figura 14. El hecho de que el comportamiento no sea exactamente igual al teórico puede venir de la mano de ya sea por elementos de hardware o software, por ejemplo, podría afectar el hecho de que las pruebas se realizaran en una máquina virtual, lo cual pudo limitar un poco el funcionamiento del programa de alguna u otra forma, no obstante el comportamiento obtenido es bastante aceptable cuando comparamos la reacción obtenida con la teórica.

6. Conclusiones

- Se logró construir un programa en C++ que generara un ordenamiento de un arreglo de tamaño $n \times n$ de la forma que lo hace el algoritmo Shearsort.

- Se comprobó experimentalmente que la función de tiempo generada de acuerdo a las pruebas realizadas al programa cumplen con la forma de la función de complejidad teórica del algoritmo.

7. Bibliografía

- Parallel Sorting Algorithms (s. f.). Recuperado de <http://www.cpp.edu/~gsyoung/CS370/14Sp/Parallel>
- Tomey, M. Á. (13 de Octubre de 2016). Algoritmos de Ordenación Paralelos
- Shearsort (s. f.). Recuperado el 23 de Octubre del 2016, de <https://www.cse.iitb.ac.in/~rana-de/606/shearsort.pdf>

8. Anexos

8.1. main.cpp

```
#include "shearsort.h"
#include "cstdlib"

int main(int argc, char** argv) {

    int n = atoi(argv[1]);

    double** mat = new double*[n];
        for (int i = 0; i < n; i++) {
            mat[i] = new double[n];
        }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            mat[i][j] = rand()%100 +1;
        }
    }

    Shearsort s1(n, mat);

    cout << " " << endl;
    cout << "Matriz sin ordenar" << endl;
    cout << " " << endl;
    ~s1;
    cout << " " << endl;
    s1.sort();
    cout << " " << endl;
```

```
cout <<"Matriz ordenada"<<endl;
cout <<" "<<endl;
~s1;
```

```
delete mat;
```

```
}
```

8.2. shearsort.h

```
#ifndef SHEARSORT_H
#define SHEARSORT_H

#include <cstdlib>
#include <iostream>
#include "string"

using namespace std;

class Shearsort{

public:

    int n;
    double **matriz;

    Shearsort();
    Shearsort(int n, double** matriz);
    ~Shearsort();
    void sort();
    void operator~();

};
#endif /* SHEARSORT_H */
```

8.3. shearsort.cpp

```
#include <cstdlib>
#include "iostream"
#include "shearsort.h"

Shearsort::Shearsort()
{
    ///@brief Constructor simple
}
Shearsort::Shearsort(int n, double** matriz)
{
    this->n =n;
```

```

        this->matriz =matriz;
    }
    Shearsort::~Shearsort()
    {
        ///@brief Destructor
    }

void Shearsort::sort(){

int contador = this->n;
int temp;
for (int i = 0; i < contador; i++) {

        for (int i = 0; i < this->n; i++) {
            if (i%2==0) {
                for (int j = 0; j < this->n; j++) {
                    for (int k = 0; k < this->n-1; k++) {
                        if (this->matriz[i][k]
> this->matriz[i][k+1]) {

                                temp = this->matriz[i][k];
                                this->matriz[i][k] = this->matriz[i][k+1];
                                this->matriz[i][k+1] = temp;
                                }
                            }
                        }
                    }
                else{
                    for (int j = 0; j < this->n; j++) {
                        for (int k = 0; k < this->n-1; k++) {
                            if (this->matriz[i][k]
< this->matriz[i][k+1]) {

                                    temp = this->matriz
this->matriz[i][k] = this->matriz[i][k+1];
                                    this->matriz[i][k+1] = temp;
                                    }
                                }
                            }
                        }
                    }
                }
            for (int i = 0; i < this->n; i++) {
                for (int j = 0; j < this->n; j++) {
                    for (int k = 0; k < this->n-1; k++) {

```

```

        if (this->matriz[k][i] > this->matriz[k+1][i])
        {
            temp = this->matriz[k][i];
            this->matriz[k][i] = this->matriz[k+1][i];
            this->matriz[k+1][i] = temp;
        }
    }
}

```

```

void Shearsort::operator~() {
    for(int i=0;i<this->n;i++){
        for(int j=0; j<this->n; j++){
            cout<< this->matriz[i][j]<<"\t";
        }
        cout<<endl;
    }
}

```