“AÑO DE LA UNIVERSALIZACIÓN DE LA SALUD”.

UNIVERSIDAD NACIONAL DE TRUJILLO

Informe Académico

Curso: Algoritmos y Complejidad

Profesor: Rodríguez Melquiades, José A.

INTEGRANTE:

- ✓ Quiroz Llicán Carlos Alfonso
- ✓ Montenegro Zelada José

VALLE JEQUETEPEQUE

1.-Objetivos:

1.1- Obtener el mayor y menor número en base a la lista de cada archivo propuesto.

1.2-Determinar el tiempo de procesamiento de cada maquina ejecutada de manera precisa.

1.3-Realizar grafico estadístico en el se va comparar la instancia junto al tiempo de proceso

2.- Fundamento de la práctica:

En este código presentado para la ejecución utilizaremos el algoritmo Minimax, en el cual emplearemos recursividad una solución que involucra volver a utilizar la función original, con parámetros que se acercan más al caso base.

2.1.-Algoritmo Máximo y Mínimo:

Esta técnica está basada en el método divide y vencerás en el cual se encarga de dividir problema y subproblemas que no se resolvieron. Se aplica a veces a algoritmos que reducen cada problema a un único subproblema, como la búsqueda binaria para encontrar un elemento en una lista ordenada.

3.- Instrumentos y materiales:

Lenguaje de programación utilizado Visual Code

Programa de Oficina Excel

Archivo bloc de notas (5 diferentes archivos con contenido de 100,200,400,800,1600 datos)

4.-Procedimiento:

4.1-Lectura de archivo

Implementamos un método para leer el archivo(.txt) que contiene todos datos para luego con el algoritmo implementado proceder a encontrar el mayor y menor número.

El procedimiento consiste en obtener los datos del archivo(.txt) y pasarlo a un arreglo para posterior mente evaluados.

```
57  void leer(){
58      char k[4];
59       char aux;
60       string cadenaNumeros = "";
61       int i = 0, j = 0;
62       ;
63       if ((fd = fopen("200numeros.txt", "rt")) != NULL){
64           while (!feof(fd)){
65
66               aux = fgetc(fd);
67               if (aux == '\n'){
68                   continue;
69               }
70               k[i] = aux;
71
72               i++;
73               if (i == 4){
74
75                   cadenaNumeros = k;
76                   i = 0;
77
78                   arreglo[j] = atoi(cadenaNumeros.c_str());
79                   j++;
80               }
81           }
82       }
83  }
```

4.2 Implemento de temporizador

-Incluimos librería:

Para implementar un temporizador y poder captar con exactitud el tiempo de ejecución del algoritmo se incluyó una librería.

```
4    #include <chrono>
```

Lo cual permitirá utilizar funciones para este objetivo.

-Iniciamos temporizador:

Función que permite inicializar un contador de tiempo

```
43          //EMPESAMOS A CONTROLAR EL TIEMPO
44          auto start = std::chrono::system_clock::now();
```

A continuación, se incluye el código a controlar para luego de la ejecución ser detenido:

```
45          cout << "\n\nel mayor es: ";
46          cout << mayor(arreglo, 0, tamanioArreglo - 1);
47          cout << "\nel minimo es: ";
48          cout << menor(arreglo, 0, tamanioArreglo - 1);
```

-Detenemos el tiempo y convertimos a segundos:

Detenemos el tiempo y guardamos el dato obtenido en la variable duración**.**

```
73      //TERMINAMOS DE CONTROLAR EL TIEMPO
74      auto end = std::chrono::system_clock::now();
75
76      std::chrono::duration<float, std::milli> duration = end - start;
```

Obtenemos el valor en milisegundos y guardamos en la variable tiempo Ejecución para luego ser transformada a segundo.

```
76      std::chrono::duration<float, std::milli> duration = end - start;
77
78      tiempoEjecucion = duration.count();
79      tiempoEjecucion /= 1000;
```

4.3 Algoritmo Máximo y Mínimo

4.3.1 Máximo

```
96   int mayor(int arreglo[], int inicio, int fin)
97   {
98       int maxIzquierda, maxderecha, medio;
99
100      if (inicio == fin)
101      {
102          return arreglo[inicio];
103      }
104      else
105      {
106          medio = ((inicio + fin) / 2);
107          //busca el maximo valo de lado izquierdo
108          maxIzquierda = mayor(arreglo, inicio, medio);
109          //buca el maximo valor de lado derecho
110          maxderecha = mayor(arreglo, medio + 1, fin);
111
112          //compara y devuelve el maximo valor
113          if (maxIzquierda > maxderecha)
114          {
115              return maxIzquierda;
116          }
117          else
118          {
119              return maxderecha;
120          }
121      }
122  }
```

4.3.2 Mínimo

```
124  int menor(int arreglo[], int inicio, int fin)
125  {
126      int minIzquierda, minderecha, medio;
127      if (inicio == fin)
128      {
129          return arreglo[inicio];
130      }
131      else
132      {
133          medio = (int)((inicio + fin) / 2);
134          //busca el minimo valo de lado izquierdo
135          minIzquierda = menor(arreglo, inicio, medio);
136          //busca el minimo valo de lado izquierdo
137          minderecha = menor(arreglo, medio + 1, fin);
138
139          //compara y devuelve el menor valor
140          if (minIzquierda < minderecha)
141          {
142              return minIzquierda;
143          }
144          else
145          {
146              return minderecha;
147          }
148      }
149  }
```

RESULTADOS:

Con 100 datos:

```
1.ver arreglo
2.ver Maximo y Minimo
3.salir
opc: 1
ARREGLO
1542
9869
6627
9951
1286
1617
3690
5340
6542
7105
4425
2610
5032
6507
7552
1165
4453
5358
```

```
el mayor es: 9977
el minimo es: 1014
tiempo de ejecucion: 0 nanosegundo
```

Con 200 datos:

```
1.ver arreglo
2.ver Maximo y Minimo
3.salir
opc: 1
ARREGLO
1542
9869
6627
9951
1286
1617
3690
5340
6542
7105
4425
2610
5032
6507
7552
1165
4453
5358
```

```
el mayor es: 9977
el minimo es: 1014
tiempo de ejecucion: 0 nanosegundo
```

Con 400 datos:

```
1.ver arreglo
2.ver Maximo y Minimo
3.salir
opc: 1
ARREGLO
1542
9869
6627
9951
1286
1617
3690
5340
6542
7105
4425
2610
5032
6507
7552
1165
4453
5358
```

```
el mayor es: 9925
el minimo es: 1153
tiempo de ejecucion: 0.001e+6 nanosegundo
```

Con 800 datos:

```
1.ver arreglo
2.ver Maximo y Minimo
3.salir
opc: 1
ARREGLO
1542
9869
6627
9951
1286
1617
3690
5340
6542
7105
4425
2610
5032
6507
7552
1165
4453
5358
```

```
el mayor es: 9925
el minimo es: 1153
tiempo de ejecucion: 0.001e+6 nanosegundo
```

Con 1600 datos:

```
1.ver arreglo
2.ver Maximo y Minimo
3.salir
opc: 1
ARREGLO
1542
9869
6627
9951
1286
1617
3690
5340
6542
7105
4425
2610
5032
6507
7552
1165
4453
5358
```

```
el mayor es: 9995
el minimo es: 1010
tiempo de ejecucion: 1.001e+06 nanosegundo
```

-Si desea constatar el tiempo de cada archivo especificar en esta parte del código, el número de datos y archivo a leer.

```
13    int arreglo[1600];
```

```
68        char text[16] = "1600numeros.txt";
```

5.-Diagrama de análisis de tiempo:

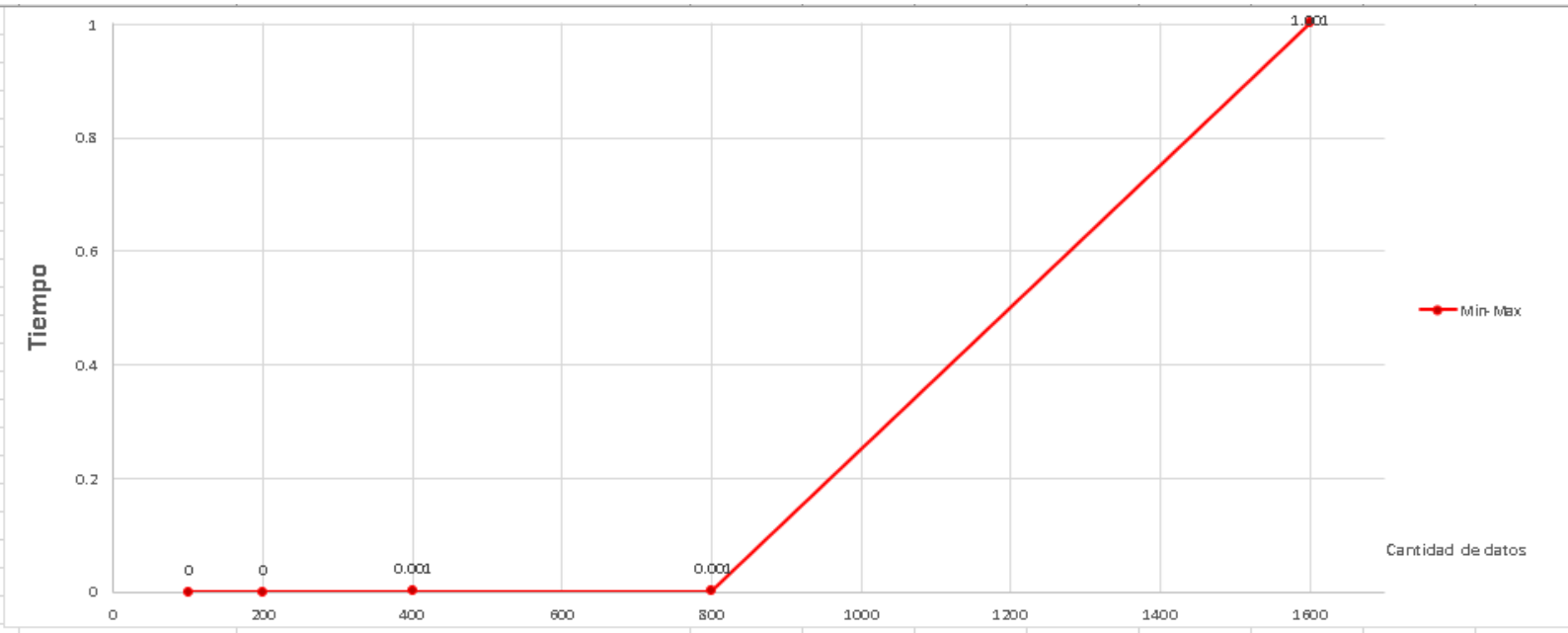
5.1-Variaciones de tiempo

N° de Datos	Tiempo en Nanosegundos
100	0ns
200	0ns
400	0.001e+6
800	0.001e+6
1600	1.001e+06

Conversión de datos		
N° de datos	Tiempo en nanosegundos	Tiempo en milisegundos
100	0ns	0ms
200	0ns	0ms
400	0.001e+6ns	0.001ms
800	0.001e+6ns	0.001ms
1600	1.001e+06ns	1.001ms

5.2-Grafico de análisis estadístico

En el siguiente grafico podemos apreciar de una manera más detallada como se va comportando y cambiando los diferentes puntos que constituyen los puntos de tiempo en la compilación



6.- Conclusiones:

-Al observar detenidamente los diversos cambios que existen con respecto al algoritmo empleado y su variación de tiempo en base a los diferentes archivos con contenido reducido, podemos concluir que el máximo tiempo que se puede visualizar es de 1.001 ms con el archivo de 1600 datos.

-El mínimo tiempo que se puede apreciar es de 0 ms, tanto en el archivo de 100 como de 200 datos.

-Con respecto al tiempo de procesamiento se podría concluir que depende mucho de la maquina en la cual ejecutemos el código ya que su procesador puede variar rotundamente.

7.- Bibliografía:

- https://es.stackoverflow.com/questions/90496/c%C3%A1lculo-m%C3%A1ximo-vector

- http://biolab.uspceu.com/aotero/recursos/docencia/TEMA%207.pdf