Luisa Alvarez Valencia

Taller 2

Diseñar y ejecutar un experimento para comparar los algoritmos (Debe incluir análisis de correctitud y complejidad):

- **ShellSort**
- **QuickSort**
- **MergeSort**

EXPERIMENTO

1. Identificación de un problema:
    a. Objetivo: ordenar una secuencia de números naturales, ver las diapositivas para la definición formal del problema en el cual se ha estado trabajando.
    b. Variables de respuesta: cálculo del tiempo que toma ordenar la secuencia.
2. Diseño del experimento
    a. Condiciones: se llevarán a cabo diferentes algoritmos (ShellSort, QuickSort, MergeSort) en el lenguaje Pascal con el fin de comparar cuál de estos presenta menor tiempo de ejecución para generar una secuencia.
    b. La secuencia será generada aleatoriamente por el computador.
    c. El tamaño de la secuencia se determina en el código de la función.
3. Conclusiones
    a. Determinar cuál algoritmo tiene menor complejidad temporal

A continuación se mostrará un diagrama BPMN con los pasos que lleva a cabo el experimento. Primero inicia, se genera la matriz aleatoria por medio de un "procedure" de pascal en el computador, a cada uno de los algoritmos se les da la misma entrada, posteriormente generan la misma salida de la secuencia ordenada y finalmente se utiliza una función la cual calcula el tiempo de ejecución de cada uno de los algoritmos. Finalmente se debe hacer una comparación de dichos tiempos para su posterior análisis.

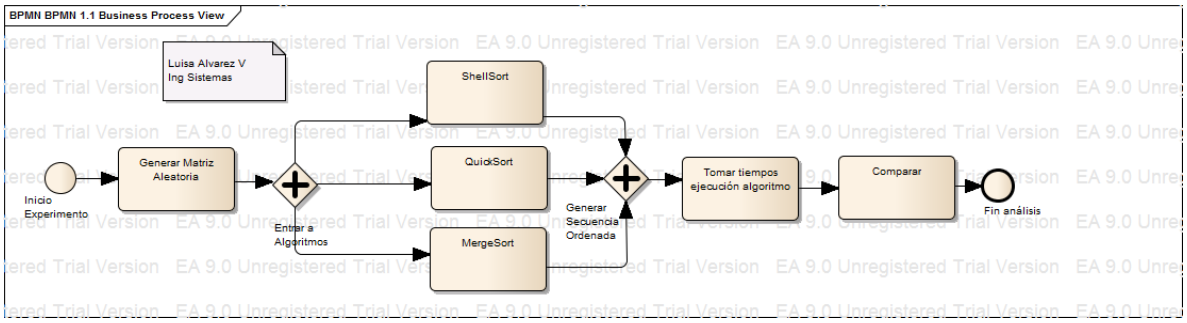


Ilustración 1: BPMN Experimento

EXPLICACIÓN DE CADA UNO DE LOS ALGORITMOS

SHELLSORT

- Problema humano: ordenar una secuencia de números.

- Problema formal: ordenamiento definido previamente en clase.
- Entradas: una secuencia A de n caracteres, todos del mismo tipo. $A = <a_1, a_2 \ldots . , a_n> a_i \in T$
- Salidas: una secuencia A de n caracteres, todos del mismo tipo ordenados.
  $A = <a_1, a_2 \ldots . , a_n> a_i \in T \mid \forall a_1 < a_n$
- Invariante: los elementos menores a j (los que están a la izquierda) y los elementos mayores que el intervalo inicial están ordenados.
  - Iniciación: el intervalo se acomoda en la mitad de los datos
  - Mantenimiento: el intervalo subdivide el intervalo cuantas veces sea necesario
  - Terminación: el intervalo adquiere un valor de cero cuando dicha secuencia ya fue ordenada y por lo tanto lo se puede subdividir más.
- Complejidad Temporal: $O(n^2)$

```
Procedure OrdenarShell(A)
Intv ← |A| %2
   While intv>0 do
     for i ←intv to |A| do
        j ← i-intv
        aux ← A[i]
        while  j ≥0 and aux<A[j] do
           A[j+intv] ← A[j];
           J ← j-intv
         End while
        A[j+intv] ← aux
      End for
     Intv ← intv % 2
   End while
End procedure;
```

QUICKSORT

- Problema humano: ordenar una secuencia de números.
- Problema formal: ordenamiento definido previamente en clase.
- Entradas: una secuencia A de n caracteres, con todos del mismo tipo. Cabe resaltar que adicionalmente contiene dos variables las cuales se ubican en puntos específicos de la

secuencia para evaluar si son mayores o menores que el pivote. A = $<a_1, a_2 \ldots . , a_n> a_i \in T \wedge$ izq, der

- Salidas: una secuencia A de n caracteres, todos del mismo tipo ordenados. $A = <a_1, a_2 \ldots . , a_n> a_i \in T \mid \forall a_1 < a_n$
- Invariante: el pivote siempre a su derecha tendrá los números más grandes y a su izquierda los números más pequeños, hasta donde IZQ y DER avanzado.
    - Iniciación: asignar al pivote |A| dividido 2, asignar IZQ y DER a variables auxiliares para poderlas mover.
    - Mantenimiento: el aumento de los valores en la suma de las variables auxiliares.
    - Terminación: cuando las variables auxiliares ya se cruzan en valor y por lo tanto ya está ordenada la secuencia.
- Complejidad Temporal: T(n) pertenece a $O(n^2)$ (en el peor de los casos) por los ciclos anidados que presenta. Si es el mejor de los casos, en dado caso de que se subdividan dos listas exactamente iguales es O(nlogn).

```
procedure OrdenarQuick( A, Izq,Der);
  auxIzq  ← Izq;
  auxDer ← Der;
  Pivote  ← A[(auxIzq + auxDer) % 2];
  While auxIzq < auxDer
   while A[auxIzq] < Pivote do
       auxIzq ← auxIzq +1
   end while;
   while A[auxDer] > Pivote do
       auxDer← auxDer -1
   end while
   if auxIzq <= auxDer then
    aux  ←  A[auxIzq];
    A[auxIzq] ← A[auxDer];
    A[auxDer] ←aux
    auxIzq ← auxIzq +1
    auxDer← auxDer -1
   end if
 end while
```

```
  if auxDer > Izq then OrdenarQuick(A, Izq,auxDer);

  if auxIzq < Der then OrdenarQuick(A,auxIzq,Der)

end procedure;
```

MERGESORT

- Problema humano: ordenar una secuencia de números.
- Problema formal: ordenamiento definido previamente en clase.
- Entradas: una secuencia A de n caracteres, todos del mismo tipo. $A = <a_1, a_2 \ldots . , a_n> a_i \in T$
- Salidas: una secuencia A de n caracteres, todos del mismo tipo ordenados.
  $A = <a_1, a_2 \ldots . , a_n> a_i \in T \mid \forall a_1 < a_n$
- Invariante: desde el inicio hasta g y desde m hasta d la secuencia se encuentra ordenada.
  - Iniciación: d tiene como valor el tamaño de la secuencia y g tienen el inicio de la secuencia.
  - Mantenimiento: g aumenta de valor y d disminuye.
  - Terminación: cuando g y d se cruzan por lo tanto la secuencia ya terminó de ordenarse.
- Complejidad Temporal: Θ (n) O(nlogn)

```
Procedure OrdenarMerge (LEnteros,  g, d )

   Tam ← |t|

   |S| ←tam

   If d > g Then

      m ←g + d Div 2;

      OrdenarMerge (t, g, m);

      OrdenarMerge (t, m + 1, d);

   For i ← m to g Do

         s[i] ←t[i];

         end for

   For j ←m + 1 To d Do

         s[d + m + 1 - j]  ←t[j];

   end for

   i ← g; j ←d;

      For k ← g To d Do

         If s[i] < s[j] Then
```

```
            t[k] ←s[i];
            i ←i + 1;
        End if
        Else
            t[k] ←s[j]
            j ← j - 1
        End else;
    End for;
End if;
End procedure;
```

RESULTADOS EXPERIMENTO

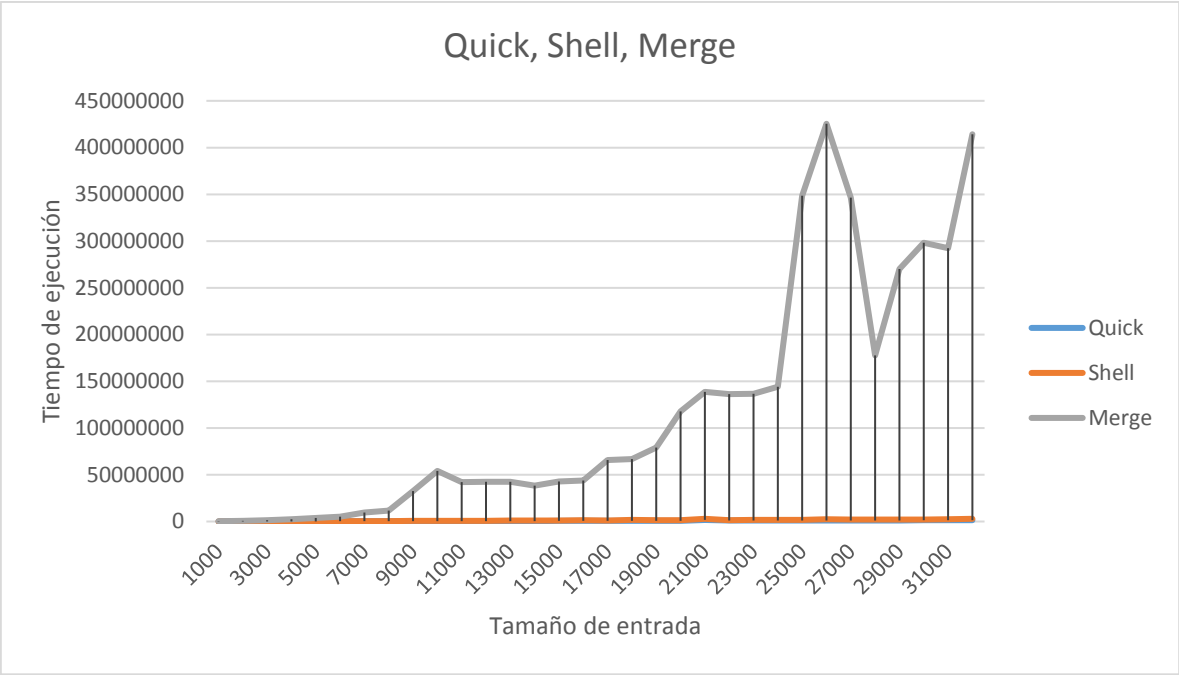
- Tamaño de la secuencia: esta inicia en cero y aumenta de mil en mil hasta que la memoria del computador no alcanza a continuar.
- Rango de valor de los números enteros: -900 a 900
- Formato tiempo: milésimas de segundos. El tiempo se midió, calculando el tiempo antes y después de la realización de cada uno de los algoritmos.

El experimento se puede realizar hasta cierto punto debido a capacidades de memoria del computador, ya que este se realizó en un computador con las siguientes especificaciones.

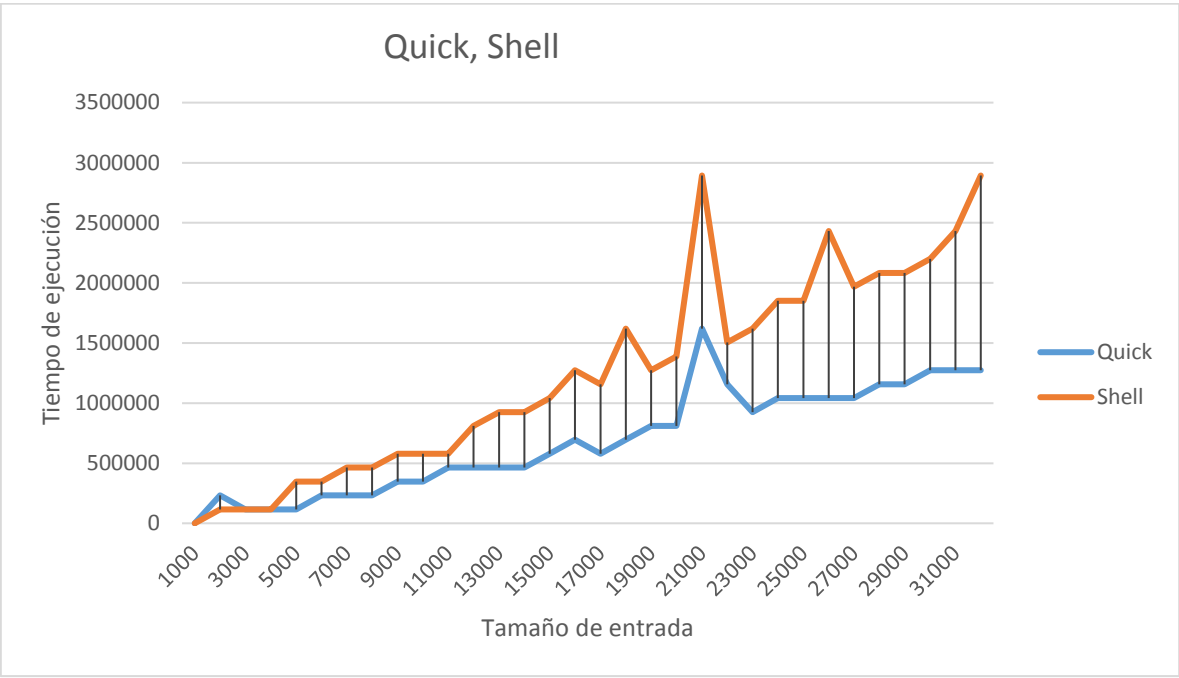
En el experimento se fue alterando el tamaño de las entradas de la secuencia para cada uno de los algoritmos de ordenación y finalmente se obtuvo la tabla a continuación como resultado.

Tamaño	Quick	Shell	Merge
1000	0.0000000000000000E+000	0.0000000000000000E+000	3,47E+05
2000	2,31E+05	1,16E+05	8,10E+05
3000	1,16E+05	1,16E+05	1,50E+06
4000	1,16E+05	1,16E+05	2,43E+06
5000	1,16E+05	3,47E+05	3,82E+06
6000	2,31E+05	3,47E+05	5,32E+06
7000	2,31E+05	4,63E+05	9,49E+06
8000	2,31E+05	4,63E+05	1,18E+07
9000	3,47E+05	5,79E+05	3,26E+07
10000	3,47E+05	5,79E+05	5,41E+07
11000	4,63E+05	5,79E+05	4,22E+07
12000	4,63E+05	8,10E+05	4,24E+07
13000	4,63E+05	9,26E+05	4,25E+07
14000	4,63E+05	9,26E+05	3,85E+07
15000	5,79E+05	1,04E+06	4,29E+07
16000	6,94E+05	1,27E+06	4,40E+07
17000	5,79E+05	1,16E+06	6,59E+07
18000	6,94E+05	1,62E+06	6,68E+07
19000	8,10E+05	1,27E+06	7,91E+07
20000	8,10E+05	1,39E+06	1,18E+08
21000	1,62E+06	2,89E+06	1,39E+08
22000	1,16E+06	1,50E+06	1,36E+08
23000	9,26E+05	1,62E+06	1,37E+08
24000	1,04E+06	1,85E+06	1,44E+08
25000	1,04E+06	1,85E+06	3,49E+08
26000	1,04E+06	2,43E+06	4,25E+08
27000	1,04E+06	1,97E+06	3,46E+08
28000	1,16E+06	2,08E+06	1,78E+08
29000	1,16E+06	2,08E+06	2,70E+08
30000	1,27E+06	2,20E+06	2,98E+08
31000	1,27E+06	2,43E+06	2,92E+08
32000	1,27E+06	2,89E+06	4,14E+08

La grafica a continuación muestra la relación de cada uno de los algoritmos, como se puede ver con claridad MergeSort es el programa que toma más tiempo comparativamente en ejecutarse.

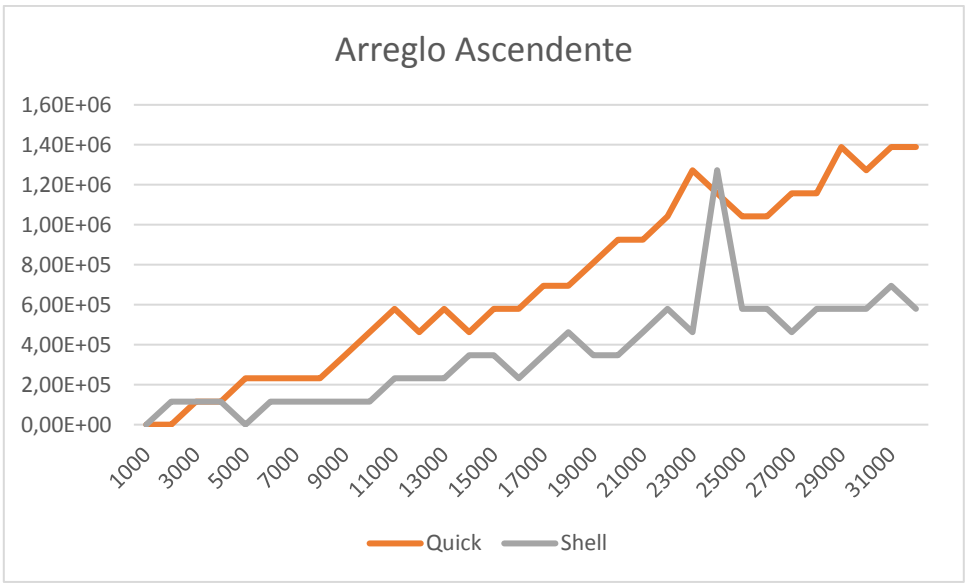


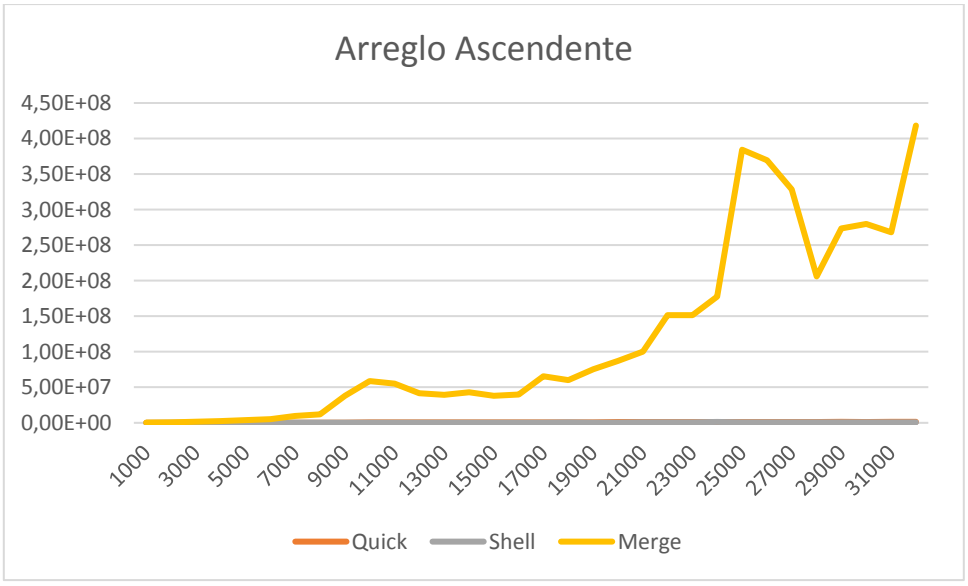
En la grafica de la parte inferior se puede observar que ShellSort toma más tiempo en ejecutarse que QuickSort (Excluyendo los valores que dieron como resultado cero).

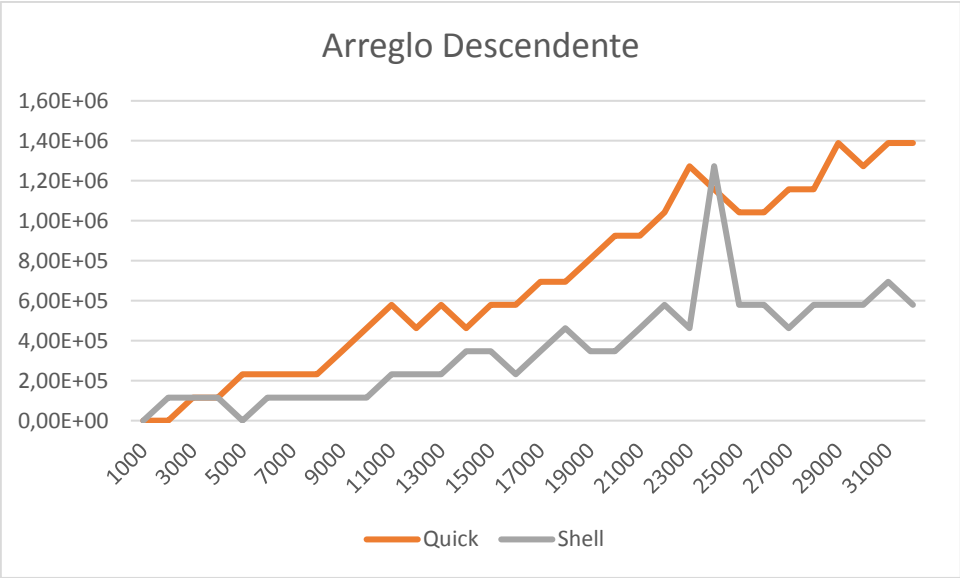


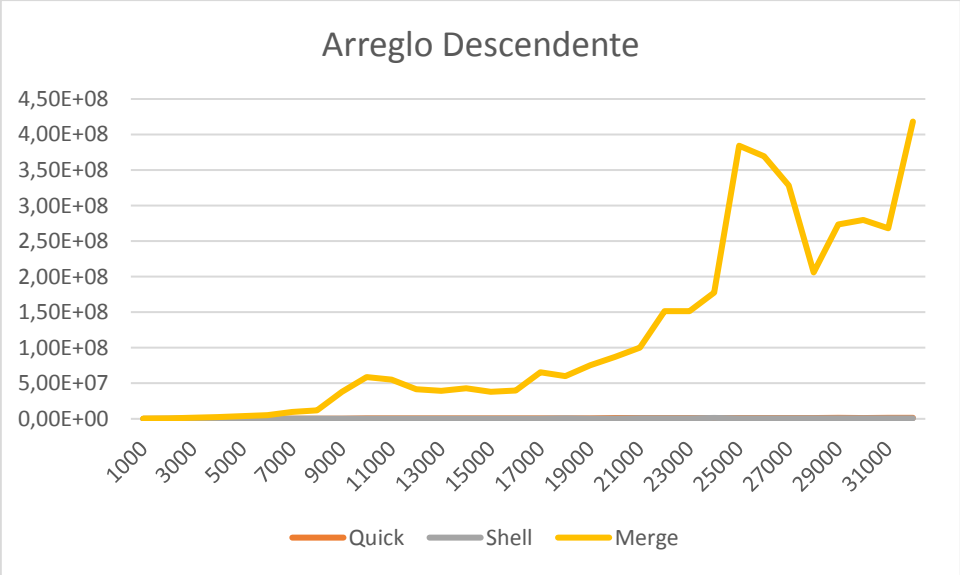
Adicionalmente se llevó a cabo una prueba de extremos con las secuencias ordenadas ascendentemente y descendentemente con el fin de observar cómo se comportaban.

En las gráficas de secuencias ordenadas ascendentemente y descendentemente se puede observar que el ordenamiento que más toma tiempo es el MergeSort, QuickSort y finalmente ShellSort.


Arreglo Ascendente
1,60E+06
1,40E+06
1,20E+06
1,00E+06
8,00E+05
6,00E+05
4,00E+05
2,00E+05
0,00E+00
1000
3000
5000
7000
9000
11000
13000
15000
17000
19000
21000
23000
25000
27000
29000
31000
Quick
Shell


Arreglo Ascendente
4,50E+08
4,00E+08
3,50E+08
3,00E+08
2,50E+08
2,00E+08
1,50E+08
1,00E+08
5,00E+07
0,00E+00
1000
3000
5000
7000
9000
11000
13000
15000
17000
19000
21000
23000
25000
27000
29000
31000
Quick
Shell
Merge





Conclusiones:

- El algoritmo de ordenamiento que más tiempo toma es MergeSort.
- Si las variables de las secuencias son aleatorias QuickSort es el más rápido, si por el contrario la secuencia esta ordenada ascendente o descendentemene es más rápido ShellSort.