TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

INGENIERÍA EN SISTEMAS COMPUTACIONALES

AGO-DIC 2018

Estructura de datos AED-1026 SC3B

Proyecto final

Rodriguez Gomez Jesus #16212369

Ray Brunett Parra Galaviz

30/11/18

En este proyecto se creó un programa en java para comparar los siguientes métodos de ordenamiento:

- Burbuja
- Quick
- Merge
- Shell
- Inserción

El programa consta dos clases, la primera llamada proy qué es la clase principal, en la que contiene un método llamado llenar, que se encargará de llenar los vectores con números aleatorios y otro para copiar los elementos y ordenarlo de nuevo; La segunda clase, ordena que es donde se encontrarán los métodos de ordenamiento y un método para mostrar los vectores ya ordenados.

Clase proy (Principal)

```
import java.util.Scanner;

class proy{
        //método para llenar los vectores e imprimir el que se genero
  public static void llenar(int [] vec,int [] vec2, int [] vec3, int [] vec4, int [] vec5,int vec24 []){
        System.out.println("       -Vector generado-          \n");
        for (int i = 0; i < vec.length; i++) {
        //El vector se generara con numeros de 1 a 505
        vec[i]=vec2[i]=vec3[i]=vec4[i]=vec5[i]=vec24[i]=(int)(Math.random() * 505)+1;
        System.out.print(vec[i]+", ");
        }
  }
  public static void copiar(int [] vec,int [] vec2, int [] vec3, int [] vec4, int [] vec5,int vec24 []){
        for (int i = 0; i < vec.length; i++) {
        //copia los datos del vector24 a los otos
        vec[i]=vec2[i]=vec3[i]=vec4[i]=vec5[i]=vec24[i];

        }
  }
  public static void main(String[] args) {

        Scanner owl = new Scanner(System.in);
        //creo el objeto, en donde estan los metodos de ordenamiento
        ordena Ordenar=new ordena();
        //variable o para usarse en el menu y n  para el tamaño del vector
        int o,n=0;
        int[]vector=null;
        int[]vector2=null;
```

```
int[]vector3=null;
int[]vector4=null;
int[]vector5=null;
int [] vector24=null;
//variable booleana para salir
boolean ff=false;
//variables para medir los tiempos de ejecucion
double tBubble,tQ,tMerge,tSH,tInsert;
// do while para el menu
do {
try {
System.out.print("\n\n\n\n          -Métodos de ordenamiento-\n"+
"\n 1. Generar vector."+
"\n 2. Ordenar."+
"\n 0. Salir.\n     Opción: ");
o=owl.nextInt();
switch (o) {
case 0:
ff=true;
owl.close();
break;
// en el caso 1 se crean los vecotres a ordenar
case 1:
System.out.print("          -Ingrese el tamaño del vector: ");
n=owl.nextInt();
vector=new int[n];
vector2=new int[n];
vector3=new int[n];
vector4=new int[n];
vector5=new int[n];
vector24=new int [n];
//se manda a llamar el metodo llenar
llenar(vector,vector2,vector3,vector4,vector5,vector24);
break;
//En el caso 2 se mandan a llamar todos los metodos de ordenamiento y se toman
tiempos
case 2:
for(int prueba=0;prueba<10;prueba++){
//Comprueba si  existe un vetor primero
llenar(vector,vector2,vector3,vector4,vector5,vector24);
double prom=0,prom2=0,prom3=0,prom4=0,prom5=0;
if(vector!=null){
        //variable a1 para el tomar el inicio del tiempo
        long a=System.nanoTime();
        Ordenar.BubbleSort(vector);
        //variable b1 para tomar el tiempo final
```

```
                long b=System.nanoTime();
                //se realiza la diferencia entre los tiempos, se multiplica por 1e-9 para convertir
los
                //nanosegundos a segundos
                tBubble=(double)(b-a)*1.0e-9;
                System.out.println("\n\n ---TIEMPO DE EJECUCION -- Bubble Sort: "+tBubble+"
Segundos");
                System.out.println("       -Vector ordenado-\n");
                Ordenar.Mostrar(vector);
                // se repite el proceso para los otros metodos
                a=System.nanoTime();
                Ordenar.Qsort(vector2, 0, vector2.length-1);
                b=System.nanoTime();
                tQ=(double)(b-a)*1.0e-9;
                System.out.println("\n\n ---TIEMPO DE EJECUCION -- Quick Sort: "+tQ+"
Segundos");
                System.out.println("       -Vector ordenado-\n");
                Ordenar.Mostrar(vector2);

                a=System.nanoTime();
                vector3=Ordenar.MergeSort(vector3);
                b=System.nanoTime();
                tMerge=(double)(b-a)*1.0e-9;
                System.out.println("\n\n ---TIEMPO DE EJECUCION -- Merge Sort: "+tMerge+"
Segundos");
                System.out.println("       -Vector ordenado-\n");
                Ordenar.Mostrar(vector3);



                a=System.nanoTime();
                Ordenar.SHell(vector4);
                b=System.nanoTime();
                tSH=(double)(b-a)*1.0e-9;
                System.out.println("\n\n ---TIEMPO DE EJECUCION -- SHell Sort: "+tSH+"
Segundos");
                System.out.println("       -Vector ordenado-\n");
                Ordenar.Mostrar(vector4);

                a=System.nanoTime();
                Ordenar.insercion(vector5);
                b=System.nanoTime();
                tInsert=(double)(b-a)*1.0e-9;
                System.out.println("\n\n ---TIEMPO DE EJECUCION -- Inserción Sort: "+tInsert+"
Segundos");
                System.out.println("       -Vector ordenado-\n");
                Ordenar.Mostrar(vector5);
```

```
            // se crea el vector reine para almacenar los tiempos
            double  reine[]={tBubble,tInsert,tMerge,tQ,tSH};
            //se ordenan los tiempos de menor a myor
            Ordenar.b(reine);
            System.out.println("\n\n\n----Tiempos----");
            //ciclo for para mostrar los tiempos en orden acendente
            for (int i = 0; i < reine.length; i++) {
            System.out.print("  "+(i+1));
            if(reine[i]==tBubble){System.out.print(". Burbuja");}
            if(reine[i]==tInsert){System.out.print(". Inserción");}
            if(reine[i]==tMerge){System.out.print(". Merge");}
            if(reine[i]==tQ){System.out.print(". Quick");}
            if(reine[i]==tSH){System.out.print(". Shell");}
            System.out.print(" con "+reine[i]+"s.\n ");
            prom=(prom+tBubble)/2;
            prom2=(prom2+tInsert)/2;
            prom3=(prom3+tMerge)/2;
            prom4=(prom4+tQ)/2;
            prom5=(prom5+tSH)/2;

            }
            //se calculan los promedios del tiempo de cada vector
            double promedios[]={prom,prom2,prom3,prom4,prom5};
            Ordenar.b(promedios);
            System.out.println("\n\n***Promedios***\n");
            for (int i = 0; i < promedios.length; i++) {
            System.out.print("  "+(i+1));
            if(promedios[i]==prom){System.out.print(". Burbuja");}
            if(promedios[i]==prom2){System.out.print(". Inserción");}
            if(promedios[i]==prom3){System.out.print(". Merge");}
            if(promedios[i]==prom4){System.out.print(". Quick");}
            if(promedios[i]==prom5){System.out.print(". Shell");}
            System.out.print(" con "+promedios[i]+"s.\n ");


            }
}else{
            System.out.println("\n----Cree un vector primero----\n");
}

}
break;


default:
System.out.println(" \n-Ingese  una opcion valida...");
```

```
        break;

        }
        } catch (Exception e) {
        System.out.println("\n -Intente de nuevo. Caracter no valido...\n\n") ;
        owl.next();
        }
        } while (!ff);
  }
}
```

Clase ordena

```
class ordena{

        //Metodo para mostrar los arreglos
        public void Mostrar(int [] vec){
        for (int i = 0; i < vec.length; i++) {
        System.out.print(vec[i]+", ");
        }
        }

        //Primer metodo de ordenamiento- burbuja
        public void BubbleSort(int[]vec){
        //variable auxiliar para hacer el cambio
        int aux;
        for (int i = 0; i < vec.length; i++) {
        for (int j = i + 1; j < vec.length; j++) {
        /*Condicion para hacer el cambio, si el numero que se encuentra en la posicion i
        es mayor al que esta en la posicion j se hace el cambio*/
        if(vec[i]>vec[j]){
                aux=vec[i];
                vec[i]=vec[j];
                vec[j]=aux;
        }
        }
        }

        }

        /*Segundo metod burbuja, este de tipo double, que recibira los tiempos de los metodos
        para ordenarlos */
        public void b(double[]vec){

        double aux;
        for (int i = 0; i < vec.length; i++) {
        for (int j = i + 1; j < vec.length; j++) {
```

```
        if(vec[i]>vec[j]){
                aux=vec[i];
                vec[i]=vec[j];
                vec[j]=aux;
        }
        }
        }

        }


        /*Segundo metodo de ordenamiento, Quick, recibira el vector y los indices a y b, que
seran
        primero y ultimo respectivamente */
        public void Qsort(int [] vec,int a, int b){

        /*Se crea la variable auxiliar para hacer los cambios, el pivote que estara lo mas cerca a
la mitad
        y las variables i y j que se les asignaran los valores de a y b respectivamente */
        int aux,i=a,j=b,pivote=vec[(a+b)/2];
        //inicio del ciclo do-while que se epetira mientras i sea menor o igual a j
        do {
        //Ciclo while para aumentar el indie i, siempre y cuando el numero que este en la
posicion
        //i sea menor al numero que este en el pivote.
        while(vec[i]<pivote){
        i++;
        }
        /*en este ciclo se disminuira j siempre y cuando el numero que este en la posicion j
        sea mayor al que este en la posicion pivote. */
        while(vec[j]>pivote){
        j--;
        }
        /* Condicion para realizar el cambio, cuando el numero en i sea menor o igual
        al que esta en j, de esta manera se enviaran los numeros mayores a la derecha y los
        menores a la izquierda*/
        if(i<=j){
        aux=vec[i];
        vec[i]=vec[j];
        vec[j]=aux;
        //una vez realizado el cambio los indices avanzan 1
        i++;
        j--;
        }

        } while (i<=j);
```

```
        /*se manda a llamar este mismo metodo para ordenar el lado izquierdo y derecho,
creando nuevos pivotes
        e indices para ser ordenados*/
        if(a<j){
        Qsort(vec, a, j);
        }
        if(i<b){
        Qsort(vec, i, b);
        }
        }
        //Metodo de ordenamiento merge recibira un vector y lo retornara ordenado
        public int [] MergeSort(int [] vec){
        //se crean los indices i, j y k, para el vector principa y izquierdo y el derecho
respectivamente
        int i,j,k;
        //se realizara una comprobacion, ya se se estara subdividiendo en diferentes vectores
        //se realizara el ordenamiento siempre y cuando el vector sea mayoy a 1, de lo contrario
        //ya estara ordenado
        if(vec.length>1){
        // se toman la medidas, para crear los vectores izquierdo y derecho
        int nIzq=vec.length/2,nDer=vec.length-nIzq;

        int vIzq[]=new int[nIzq];
        int vDer[]=new int [nDer];

        //ciclo para copiar los valores del  original al izquierdo
        for ( i = 0; i < nIzq; i++) {
        vIzq[i]=vec[i];
        }
        //se copian los valores del original al derecho
        for ( i = nIzq; i < nIzq+nDer; i++) {
        vDer[i-nIzq]=vec[i];
        }
        //se manda a llamar este mismo metodo hasta que los elementos queden individuales.
        vIzq=MergeSort(vIzq);
        vDer=MergeSort(vDer);
        //se igualan los indicies a 0, con i para el original, j para el izquierdo, k para el derecho
        i=j=k=0;
        //ciclo while para realizar la mezcla, con un ciclo while que se recorrera
        /*siempre y cuando la longitud de vector izquierdo sea diferente a j y
        la longitud del vector derecho sea diferente de k*/
        while (vIzq.length!=j&&vDer.length!=k) {
        /*se asignara lo que este en la posicion i al vector izquierdo en j siempre y cuando
        lo que este en la posicion  j del izq sea menor alo que este en la posicion k del derecho */
        if(vIzq[j]<vDer[k]){
```

```
            vec[i]=vIzq[j];
            i++;
            j++;
//si ees menor se realiza lo contrario
}else{
            vec[i]=vDer[k];
            i++;
            k++;
}
}
//se empieza a llenar el arreglo final pasando la parte izquierda
while(vIzq.length!=j){
vec[i]=vIzq[j];
i++;
j++;
}
//se  pasa la parte derecha al arreglo final
while (vDer.length!=k) {
vec[i]=vDer[k];
i++;
k++;
}
}
return vec;
}
//método shell
public void SHell(int [] vec){
//variable dist que sera la distancia con la que se compararan los elementos
//auxiliar para realizar el cambio y los indices i, j y k que son los que se compararan
int dist,aux,i,j,k;
dist=vec.length/2;
//ciclo while que se repetira hasta que la distancia sea 0
while (dist>0) {
//ciclo for para recorrer la distancia con que se comparan los elementos
for(i =dist; i<vec.length; i++){
//se asigna a la distancia variable j con la que se comparara i
j=i-dist;
//ciclo while, que se recorrera cuando j sea mayor o igual a 0
while (j>=0) {
            k=j+dist;
            //condicion para realizar el ordenamiento
            //si el elemento de la posicion j es menor al de k se asigna un valor menor
            //a 0 para que ya no se repita el ciclo while.
            if (vec[j]<=vec[k]){
            j=-24;
            }else{
```

```
                //
                aux=vec[j];
                vec[j]=vec[k];
                vec[k]=aux;
                j=j-dist;

                }
        }
        }
        //se acorta la distancia
        dist=dist/2;
        }
        }
        //metodo de insercion
        public void insercion(int [] vec){
        //indices i y j ara realizar comparaciones y variale aux para realizar los cambios
        int aux,i,j;
        //ciclo que recorrera el vector desde la posicion 1
        for(i=1;i<vec.length;i++){
        //se guardara el elemento de la posicion i en la variable auxiliar
        aux=vec[i];
        //se asigna al indice j una posicion antes del indice i
        j=i-1;
        // Se recorrera este ciclo para realizar comprobaciones con los elementos anteriores*/
        while (j>=0&&vec[j]>aux) {
        //se asigna el elemento en nla posicion j+1 en la posicion j
        vec[j+1]=vec[j];
        j--;
        }
        //se asignara elemento alamcenado en la posicion j+1
        vec[j+1]=aux;
        }
        }
}
```

Pruebas

Vector con 100 elementos

282, 477, 305, 241, 285, 397, 329, 487, 453, 258, 377, 470, 197, 303, 109, 16, 254, 344, 299, 388, 21, 77, 2, 224, 6, 45, 278, 177, 395, 175, 126, 44, 500, 434, 131, 266, 395, 297, 372, 150, 162, 89, 361, 458, 350, 193, 124, 285, 383, 281, 39, 257, 382, 486, 17, 95, 471, 268, 222, 500, 413, 372, 397, 105, 137, 306, 75, 483, 433, 287, 379, 128, 317, 352, 234, 26, 481, 105, 482, 65, 298, 218, 332, 409, 389, 193, 379, 119, 384, 490, 422, 245, 495, 168, 36, 93, 234, 143, 42, 180

1	1. Quick con 4.8302000000000006E-5s. 2. Shell con 5.2663E-5s. 3. Inserción con 7.902E-5s. 4. Merge con 9.016200000000001E-5s. 5. Burbuja con 6.660650000000001E-4s.
2	1. Quick con 3.0857E-5s. 2. Shell con 4.0584E-5s. 3. Inserción con 6.247800000000001E-5s. 4. Merge con 7.058700000000001E-5s. 5. Burbuja con 1.34551E-4s.
3	1. Quick con 3.5197E-5s. 2. Shell con 5.0361E-5s. 3. Inserción con 8.254200000000001E-5s. 4. Merge con 1.25563E-4s. 5. Burbuja con 1.8625100000000002E-4s.
4	1. Merge con 2.3539000000000003E-5s. 2. Quick con 3.118E-5s. 3. Shell con 5.2203E-5s. 4. Inserción con 8.0857E-5s. 5. Burbuja con 1.3156300000000001E-4s.
5	1. Merge con 1.4588000000000002E-5s. 2. Quick con 1.6853E-5s. 3. Shell con 3.2712E-5s. 4. Inserción con 5.2107E-5s. 5. Burbuja con 1.25956E-4s.
6	1. Quick con 1.249E-5s. 2. Merge con 2.3544E-5s. 3. Shell con 5.1030000000000004E-5s. 4. Inserción con 7.3077E-5s. 5. Burbuja con 1.80852E-4s.
7	1. Quick con 1.2415E-5s. 2. Merge con 2.0443E-5s. 3. Shell con 5.0877E-5s. 4. Inserción con 7.582200000000001E-5s. 5. Burbuja con 1.8350200000000002E-4s.
8	1. Quick con 1.2744E-5s. 2. Merge con 1.9832000000000003E-5s. 3. Shell con 5.6769000000000005E-5s. 4. Inserción con 7.826E-5s. 5. Burbuja con 2.05712E-4s.

9	1. Quick con 5.9281000000000004E-5s. 2. Merge con 7.0543E-5s. 3. Shell con 1.4372E-4s. 4. Inserción con 2.0533700000000002E-4s. 5. Burbuja con 8.1189400000000001E-4s.
10	1. Quick con 2.0907E-5s. 2. Merge con 3.3296E-5s. 3. Shell con 9.0572E-5s. 4. Inserción con 1.35214E-4s. 5. Burbuja con 3.93073E-4s.
Prom	1. Quick con 2.025365625E-5s. 2. Merge con 3.22555E-5s. 3. Shell con 8.7741625E-5s. 4. Inserción con 1.309885625E-4s. 5. Burbuja con 3.8078946875000005E-4s.

Vector con 150

413, 174, 316, 381, 227, 112, 222, 34, 495, 497, 247, 298, 229, 467, 133, 272, 299, 138, 242, 174, 122, 474, 157, 451, 19, 275, 170, 210, 373, 505, 57, 451, 16, 488, 28, 180, 440, 429, 158, 362, 301, 165, 428, 313, 43, 36, 425, 173, 122, 485, 409, 477, 473, 29, 275, 364, 174, 305, 61, 253, 158, 259, 350, 241, 481, 104, 397, 57, 104, 62, 26, 26, 266, 471, 133, 79, 52, 144, 218, 274, 170, 462, 247, 467, 189, 32, 463, 329, 257, 248, 70, 304, 147, 371, 75, 244, 479, 497, 70, 64, 255, 299, 202, 68, 339, 128, 386, 278, 58, 451, 164, 144, 104, 287, 423, 365, 354, 46, 288, 118, 71, 281, 433, 188, 459, 384, 346, 62, 496, 126, 52, 477, 237, 420, 496, 133, 432, 254, 2, 369, 388, 38, 154, 406, 389, 155, 166, 89, 152, 467,

1. Shell con 1.56343E-4s. 2. Quick con 2.6906200000000003E-4s. 3. Inserción con 2.7431E-4s. 4. Merge con 4.5917900000000004E-4s. 5. Burbuja con 0.001772511s.	1. Quick con 6.133E-5s. 2. Shell con 9.7775E-5s. 3. Merge con 1.96393E-4s. 4. Inserción con 1.9845800000000002E-4s. 5. Burbuja con 5.74182E-4s.
1. Merge con 4.9225000000000004E-5s. 2. Shell con 6.0072000000000004E-5s. 3. Quick con 6.5201E-5s. 4. Inserción con 1.12836E-4s. 5. Burbuja con 4.13279E-4s.	1. Quick con 3.8445E-5s. 2. Merge con 4.2886E-5s. 3. Shell con 8.8026E-5s. 4. Inserción con 1.6538700000000002E-4s. 5. Burbuja con 3.54612E-4s.
1. Quick con 2.1377000000000002E-5s. 2. Merge con 3.9602000000000005E-5s. 3. Shell con 8.601400000000001E-5s. 4. Inserción con 2.22076E-4s. 5. Burbuja con 4.19601E-4s.	1. Quick con 2.0169000000000003E-5s. 2. Merge con 3.4669E-5s. 3. Shell con 8.7643E-5s. 4. Inserción con 1.69592E-4s. 5. Burbuja con 4.05636E-4s.
1. Quick con 1.8974E-5s. 2. Merge con 3.4663E-5s. 3. Shell con 8.5516E-5s. 4. Burbuja con 1.42134E-4s. 5. Inserción con 1.77657E-4s.	1. Quick con 1.8844E-5s. 2. Merge con 3.4519E-5s. 3. Shell con 8.5062E-5s. 4. Burbuja con 8.8978E-5s. 5. Inserción con 1.6931700000000002E-4s.
1. Quick con 1.9253E-5s. 2. Merge con 3.5089E-5s. 3. Shell con 8.5666E-5s. 4. Burbuja con 1.00507E-4s. 5. Inserción con 1.7177500000000001E-4s.	1. Quick con 1.9506000000000003E-5s. 2. Merge con 3.4939000000000005E-5s. 3. Shell con 7.3762E-5s. 4. Burbuja con 1.00753E-4s. 5. Inserción con 1.91296E-4s.
Promedio	1. Quick con 1.8896437500000004E-5s. 2. Merge con 3.384715625E-5s.

	3. Shell con 7.145693750000001E-5s. 4. Burbuja con 9.760446875E-5s. 5. Inserción con 1.8531800000000002E-4s.

Vector con 200

44, 260, 140, 40, 152, 110, 295, 148, 66, 37, 448, 282, 59, 143, 439, 250, 403, 64, 1, 433, 199, 481, 145, 50, 437, 320, 458, 460, 49, 384, 89, 368, 12, 199, 476, 27, 11, 211, 504, 111, 303, 135, 165, 50, 248, 495, 36, 60, 218, 199, 440, 446, 331, 260, 440, 327, 420, 502, 13, 427, 211, 495, 259, 453, 404, 273, 4, 289, 132, 115, 132, 119, 437, 203, 488, 205, 273, 171, 77, 109, 132, 120, 206, 252, 4, 376, 51, 222, 156, 378, 185, 480, 300, 399, 30, 93, 90, 285, 17, 460, 384, 367, 55, 385, 227, 182, 410, 385, 470, 140, 271, 440, 63, 256, 17, 387, 407, 129, 155, 227, 502, 412, 404, 411, 291, 459, 234, 395, 267, 110, 346, 75, 174, 467, 178, 443, 357, 205, 121, 90, 137, 161, 376, 162, 213, 407, 284, 402, 116, 303, 100, 96, 208, 196, 191, 395, 16, 479, 150, 353, 415, 90, 184, 111, 233, 168, 277, 325, 443, 263, 344, 8, 357, 368, 59, 429, 126, 201, 483, 466, 471, 477, 330, 88, 46, 197, 77, 348, 22, 339, 460, 394, 437, 44, 47, 108, 474, 385, 458, 351,

1. Quick con 1.0894700000000001E-4s. 2. Inserción con 2.3058600000000002E-4s. 3. Merge con 2.4072600000000003E-4s. 4. Shell con 4.2800300000000004E-4s. 5. Burbuja con 6.16808E-4s.	1. Quick con 7.559E-5s. 2. Shell con 1.33032E-4s. 3. Merge con 1.66341E-4s. 4. Inserción con 1.9808100000000002E-4s. 5. Burbuja con 0.0015309420000000002s.
1. Quick con 2.6225E-5s. 2. Merge con 6.7964E-5s. 3. Shell con 1.2233800000000002E-4s. 4. Inserción con 3.06829E-4s. 5. Burbuja con 4.86999E-4s.	1. Quick con 1.7755000000000002E-5s. 2. Merge con 3.1952E-5s. 3. Shell con 1.1743100000000001E-4s. 4. Inserción con 3.13209E-4s. 5. Burbuja con 8.1172E-4s.
1. Quick con 2.4919000000000003E-5s. 2. Merge con 4.7877E-5s. 3. Shell con 1.2356000000000002E-4s. 4. Burbuja con 2.08332E-4s. 5. Inserción con 3.01448E-4s.	1. Quick con 2.2329000000000003E-5s. 2. Merge con 4.3268E-5s. 3. Shell con 1.2423300000000001E-4s. 4. Burbuja con 1.8064800000000002E-4s. 5. Inserción con 3.04484E-4s.
1. Quick con 2.4727000000000002E-5s. 2. Merge con 5.0663000000000006E-5s. 3. Inserción con 8.8868E-5s. 4. Shell con 1.13658E-4s. 5. Burbuja con 1.8352600000000002E-4s.	1. Quick con 2.6971000000000003E-5s. 2. Merge con 4.6668E-5s. 3. Inserción con 6.8273E-5s. 4. Shell con 1.23774E-4s. 5. Burbuja con 1.9493400000000001E-4s
1. Quick con 2.6132000000000003E-5s. 2. Merge con 4.5237E-5s. 3. Inserción con 6.168100000000001E-5s. 4. Shell con 1.2313E-4s. 5. Burbuja con 1.47119E-4s.	1. Quick con 2.2341000000000002E-5s. 2. Merge con 5.3846000000000006E-5s. 3. Inserción con 6.273E-5s. 4. Shell con 1.29514E-4s. 5. Burbuja con 1.8291500000000002E-4s.
Promedio	1. Quick con 2.1642843750000004E-5s. 2. Merge con 5.216331250000001E-5s. 3. Inserción con 6.0769687500000004E-5s. 4. Shell con 1.254666875E-4s. 5. Burbuja con 1.7719890625E-4s.

Conclusión.

El método de ordenamiento Quick resultó ser el que menos tiempo hace en ordenar los vectores seguido del método merge, y los menos eficientes fue el burbuja e inserción.