Tarea I

Daniel Monge Arce - B85017

I. Introducción

En este trabajo se implementan los métodos de ordenamiento denominados: selección, inserción, mezcla, montículos, rápido y residuos y se prueban en distintos arreglos de tamaños diferentes, estos son de 50.000, 150.000, 100.000 y 200.000 números aleatorios que serán acomodados de menor a mayor. Los arreglos utilizados para tomar los tiempos constan de números negativos y positivos.

Los códigos estan escritos en lenguaje de programación C++, y ejecutados en una computadora 3 veces cada método por cada tamaño de arreglo, es decir 12 veces cada algoritmo. Además se toman los tiempos de ejeccucion de cada uno y se comparan con respecto a los demás algoritmos para poder determinar sus diferencias.

II. Metodología

Para lograr lo propuesto, se escribieron los códigos de selección e inserción a mano, con ayuda de distintos foros se logró escribir el código del método mezcla, luego montículos y rápido, por último el más complejo personalmente, residuos para lograr acomodar los números negativos. Primeramente, se escribió el código de selección, que consta en encontrar el número menor de la lista, intercambiarlo por el primero, buscar el siguiente menor e intercambiarlo por el segundo, sucesivamente hasta acomodar el arreglo completo. Para este algoritmo se utilizó un método auxiliar de intercambio con un temporal.

Seguidamente, se escribió el código de inserción, empieza la primera posición y se comprara con la segunda, si es mayor, se desplaza y continuamos con el siguiente elemento, sucesivamente hasta colocar el último elemento en su posición ordenada.

Luego, el código de mezcla, siendo un código más complejo, fue necesario visitar distintas páginas web para lograr que funcioné, si el arreglo es de tamaño 0 ó 1, ya está acomodado, de lo contrario, se divide el arreglo desordenado y se trabajan separadas de manera recursiva, una vez ordenados, se unen los arreglos y se crea un arreglo total con los elementos ya ordenados.

Para el algoritmo de montículos, se requiere almacenar los elementos del arreglo en un montículo para luego extraer el nodo que viene siendo el nodo raíz o cima, se realiza hasta obtener un arreglo ordenado, se implementó los métodos auxiliares: heap() e intercambiar(), para lograr el buen funcionamiento del algoritmo.

Con respecto al algoritmo rápido, se implementó de forma recursiva, es decir llama al mismo método dentro de él. Primero utilizamos un pivote siendo el dato final/el más derecha del arreglo y lo compara con el arreglo buscando si es menor al pivote. Por medio del método auxiliar intercambiar(), se van acomodando los datos y el pivote va cambiando.

Por último, el algoritmo de residuos, requirió mucho tiempo para implementar su buen funcionamiento con números negativos. Primero obtenemos el arreglo a ordenar y lo partimos en 2 arreglos, uno de positivos y otro de negativos, al arreglo de negativos le multiplicamos -1 a cada dato para hacerlo positivo, y buscamos el número mayor de los 2 arreglos. Una vez tengamos los datos podemos hacerle el procedimiento de residuos y el "paso mágico.[a] los 2 arreglos. Una vez acomodados, volvemos a poner el signo a los negativos mutiplicando -1 a todos y usamos el método auxiliar invertir() para darle vuelta al arreglo, así tener los negativos ordenados de manera correcta, y por último concatenamos los 2 arreglos al arreglo original.

Cuadro I
Tiempo de ejecución de los algoritmos.

		Tiempo (s)			
		Corrida			
Algoritmo	Tam. (k)	1	2	3	Prom.
Selección	50	3,107	3,077	3,083	3,089
	100	12,325	12,322	12,316	12,321
	150	28,657	29,948	29,094	29,233
	200	51,019	51,150	51,592	51,253
Inserción	50	1,56	1,579	1,585	1,574
	100	6,410	6,262	6,196	6,289
	150	13,279	13,316	13,201	13,265
	200	23,252	23,285	23,245	23,260
Mezcla	50	0,005	0,005	0,005	0,005
	100	0,009	0,009	0,010	0,009
	150	0,015	0,015	0,014	0,014
	200	0,019	0,019	0,019	0,019
Montículos	50	0,017	0,017	0,017	0,017
	100	0,036	0,037	0,036	0,036
	150	0,055	0,055	0,055	0,055
	200	0,077	0,077	0,083	0,079
Rápido	50	0,007	0,007	0,007	0,007
	100	0,017	0,017	0,017	0,017
	150	0,029	0,029	0,030	0,029
	200	0,045	0,043	0,045	0,044
Residuos	50	0,004	0,004	0,004	0,004
	100	0,009	0,009	0,08	0,008
	150	0,013	0,012	0,012	0,012
	200	0,017	0,017	0,017	0,019

III. Resultados

Los tiempos de ejecución de las X corridas de los algoritmos se muestran en el cuadro I.

La forma de las curvas no fueron las esperadas porque pensé que iban a haber cambios más drásticos según los tamaños de los arreglos, pero se puede notar una línea con un crecimiento

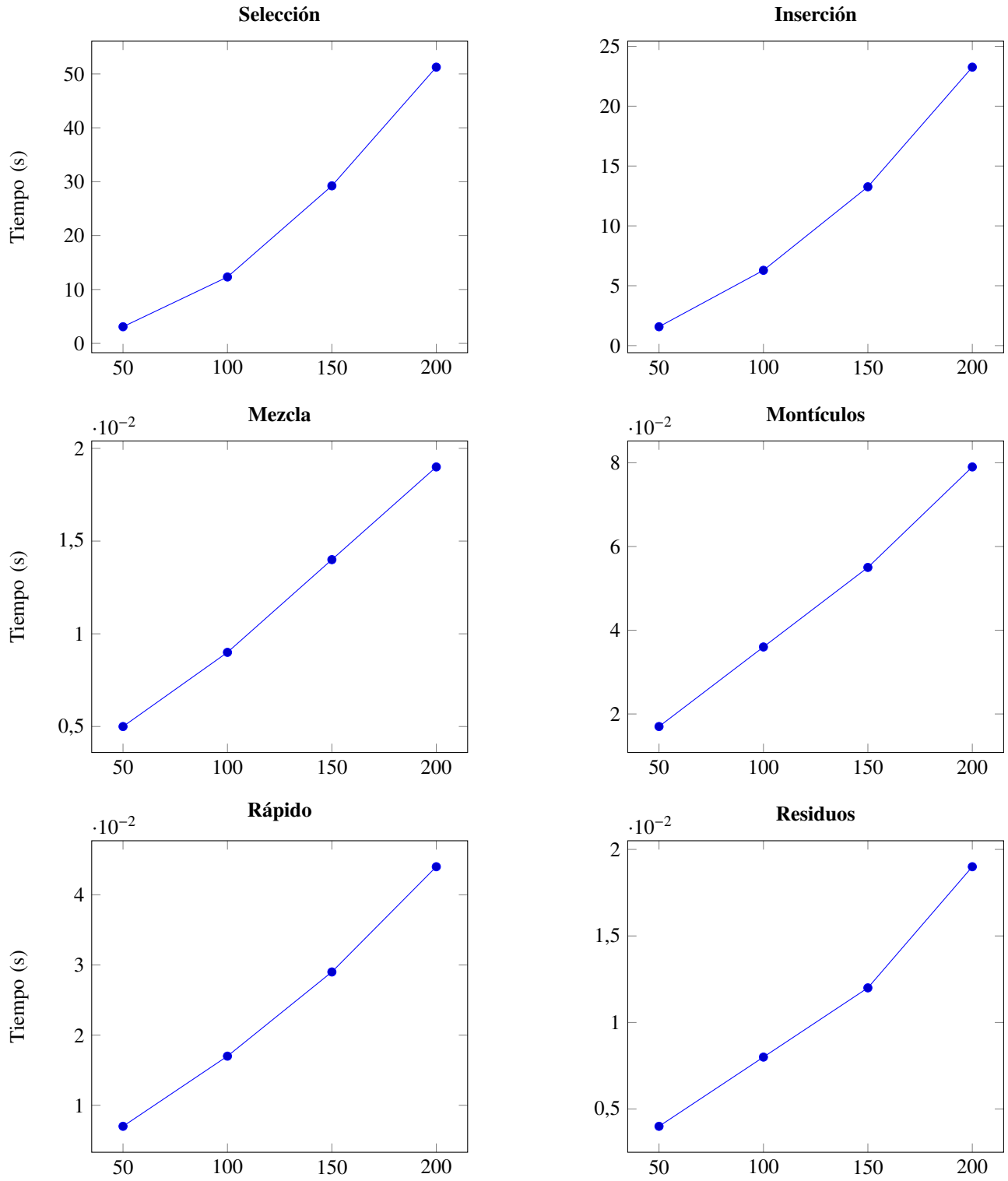

Figura 1. Tiempos promedio de ejecución de los algoritmos de ordenamiento por selección, inserción, mezcla.

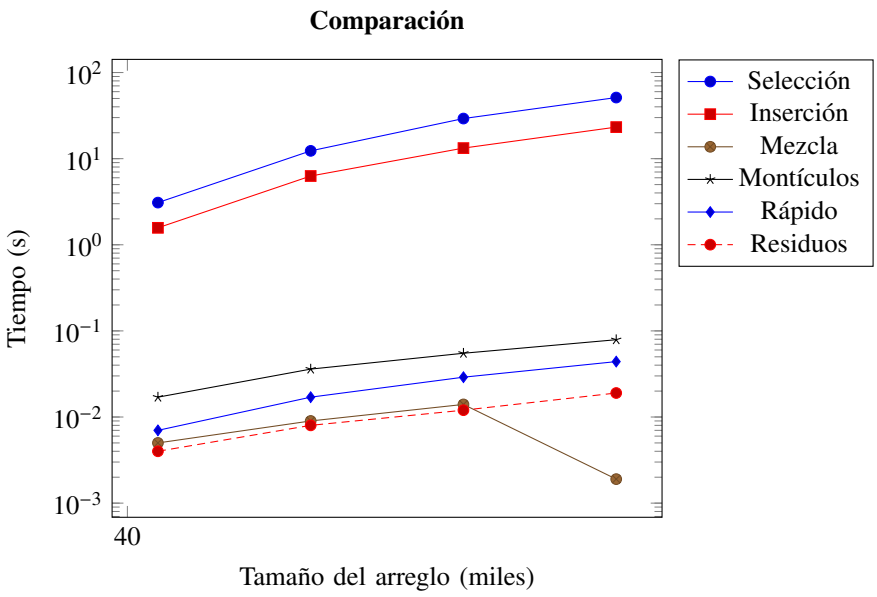

Figura 2. Gráfico comparativo de los tiempos promedio de ejecución de los algoritmos.

semi-constante y podría mantenerse igual o cambiar de forma abrupta con tamaños mucho mayores o menores a los tomados para estas pruebas.

Cabe denotar que los resultados se dieron un una computadora de escritorio con un procesador: Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz, 3201 Mhz, 4 Core(s), 4 Logical Processor(s), por lo que los tiempos pueden varias según la máquina que se utilice.

IV. Conclusiones

A partir de los resultados se puede concluir que el método de mezcla, montículos, rápido y redisuos dan tiempos que no podemos diferenciar como seres humanos, pero a nivel de computadoras se diferencian por decimales de segundos. Tambien se denota que el tamaño del arreglo es directamente proporcional al tiempo de ejecución, es decir, entre más grande el arreglo, más tiempo demora la máquina en procesar y ordenar los datos. Por otra parte, los algoritmos de selección e insersión tienen tiempos sumamente altos en comparación a los otros algoritmos, tan grandes que si se los podemos sentir, por ejemplo esperar alrededor de 51 segundos para ordenar 200.000 números con selección es muy lento y poco eficiente, una persona probablemente prefiere invertir 0.019 segundos que es un instante para ordenar esos 200.000 números pero con el algoritmo de Residuos. Algo que me llamo la atención fue la similitud de tiempos de ejecucción con respecto a Mezcla y Residuos, al ser diferentes dan casi los mismos tiempos al realizar los 3 tamaños de arreglos que utilicé.

Referencias

Sitios utilizados para entender mejor los algoritmos:

https://www.toptal.com/developers/sorting-algorithms
https://www.geeksforgeeks.org/merge-sort/
https://www.toptal.com/developers/sorting-algorithms
https://es.wikipedia.org/wiki/Heapsort
https://es.wikipedia.org/wiki/Quicksort
https://es.wikipedia.org/wiki/Ordenamiento$_Radix$
http://www.algostructure.com/sorting/
https://www.hackerearth.com/pt-br/practice/algorithms/sorting/radix-sort/tutorial/

Apéndice A
Código de los algoritmos

Algoritmo 1 Algoritmo de Ordenamiento Selección.

```
void seleccion(int * arreglo, int tamano) {
        int contador;
        int contador_sec;
        int minimo;

        for (contador = 0; contador < tamano-1; contador++) {
                //entontrar el numero minimo
                minimo = contador;
                for (contador_sec = contador + 1; contador_sec < tamano; contador_sec++)
                        if (arreglo[contador_sec] < arreglo[minimo]) {
                                minimo = contador_sec;
                        }
                //Intercambiar minimo con el primer elemento
                intercambiar(&arreglo[minimo], &arreglo[contador]);
        }
};
```

Algoritmo 2 Algoritmo de Ordenamiento Inserción.

```
void insercion(int * arreglo, int tamano){

        int contador;
        int llave;
        int contador_sec;

        for (contador = 1; contador < tamano; contador++) {
                llave = arreglo[contador];
                contador_sec = contador - 1;
                while (contador_sec >= 0 && arreglo[contador_sec] > llave) {
                                arreglo[contador_sec + 1] = arreglo[contador_sec];
                                contador_sec = contador_sec - 1;
                }
                arreglo[contador_sec + 1] = llave;
        }
};
```

Algoritmo 3 Algoritmo de Ordenamiento Mezcla.

```
void mergesort(int * arreglo, int tamano){
        //Buscar si el arreglo es vacio
        if (tamano == 0)
                return;
        //Crear segundo arreglo
        int arreglo2[tamano];
    for (int i = 0; i < tamano; i++)
        arreglo2[i] = arreglo[i];
    acomodar(arreglo2, 0, tamano, arreglo);
    for (int i = 0; i < tamano; i++)
        arreglo[i] = arreglo2[i];
        }
};
```

Algoritmo 4 Algoritmo de Ordenamiento Montículos.

```
void heapsort(int * arreglo, int tamano){
        //Acomodar arreglo
    for (int i = tamano / 2 - 1; i >= 0; i--) {
       heap(arreglo, tamano, i);
        }
    // Extraer cada numero del arreglo
    for (int i = tamano - 1; i > 0; i--) {
       //Intercambiar al final
       intercambiar(&arreglo[0], &arreglo[i]);
       //Llamar al maximo en el arreglo reducido
       heap(arreglo, i, 0);
        }
};
```

Algoritmo 5 Algoritmo de Ordenamiento Rápido.

```
void quicksort(int * arreglo, int tamano){
       int pivote = arreglo[tamano-1];
   int indiceMayor = 0;
   for (int i = 0; i < tamano-1; i++){
     if (arreglo[i] < pivote){
       intercambiar(&arreglo[i], &arreglo[indiceMayor]);
       indiceMayor++;
       }
   }
   intercambiar(&arreglo[indiceMayor], &arreglo[tamano-1]);
   if (indiceMayor > 1){
       quicksort(arreglo, indiceMayor);
       }
   if (tamano-indiceMayor-1 > 1) {
     quicksort(arreglo+indiceMayor+1, tamano-indiceMayor-1);
       }
}
```

Algoritmo 6 Algoritmo de Ordenamiento Residuos.

```
void radixsort(int * arreglo, int tamano){
      int i, q, maxPos, maxNeg, j, k, cantidadNegativos = 0;

   //Busca cuantos numeros son negativos en el arreglo
   for(i = 0; i < tamano; i++) {
     if(arreglo[i] < 0)
       cantidadNegativos++;
   }

   //Crear 2 arreglos para positivos y negativos
   int arregloPos[tamano - cantidadNegativos];
   int arregloNeg[cantidadNegativos];

     for(i = 0; i < tamano; i++) {
       if(arreglo[i] < 0) {
         arregloNeg[j] = arreglo[i];
         j++;
     }
       else {
         arregloPos[k] = arreglo[i];
         k++;
       }
     }

                        //Quita el signo a los numeros
     for (i = 0; i < cantidadNegativos; i++) {
       arregloNeg[i] = arregloNeg[i] * -1;
     }

     //Busca los valores maximos de cada arreglo
     maxNeg = getMaximo(arregloNeg, cantidadNegativos);
     maxPos = getMaximo(arregloPos, tamano - cantidadNegativos);

         //ordena los numeros del arreglo positivo
     for (int exp = 1; maxPos/exp > 0; exp *= 10) {
       residuos(arregloPos, tamano - cantidadNegativos, exp);
     }
         //ordena los numeros del arreglo negativo
     for (int exp = 1; maxNeg/exp > 0; exp *= 10) {
       residuos(arregloNeg, cantidadNegativos, exp);
     }
                        //Invierte los numeros negativos para luego poner signos
     invertir(arregloNeg, cantidadNegativos);

     //Vuelve a poner el signo a los numeros negativos ya ordenados
     for(i = 0; i < cantidadNegativos; i++) {
       arregloNeg[i] = arregloNeg[i] * -1;
     }

                        //Junta los 2 arreglos en el arreglo original
     for(i = 0; i < cantidadNegativos; i++) {
       arreglo[i] = arregloNeg[i];
     }
     for(i = cantidadNegativos; i < tamano; i++) {
       arreglo[i] = arregloPos[q];
       q++;
     }
};
```