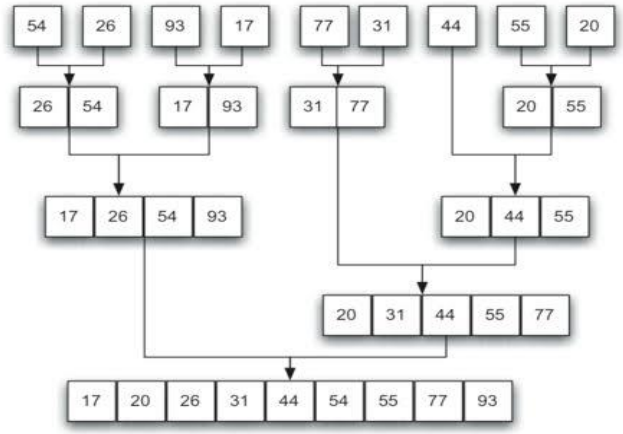
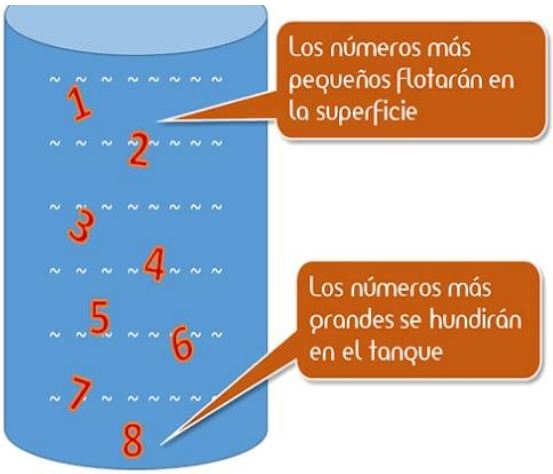
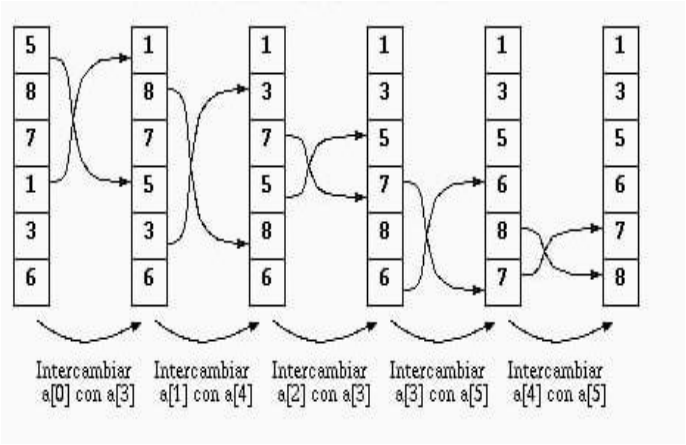
Algoritmos y Estructura de Datos

Práctica 2:

ORDENAMIENTO POR INSERCIÓN, BURBUJA, SELECCIÓN Y MEZCLA.







Grupo: **2CM2**

Profra. Cecilia Alborante Morato

Integrantes:

- Guevara Badillo Areli Alejandra
- Hernández Simón Rebeca

14 de Marzo, 2023

INTRODUCCIÓN

Los algoritmos de ordenamiento han sido fuente de gran interés e investigación desde el inicio de la computación, aunque su resolución es relativamente simple a lo largo de la historia se han diseñado numerosas técnicas para lograr el algoritmo que logre ordenar una lista de la manera más rápida y eficiente. Un algoritmo es la operación de arreglar los elementos en algún orden secuencial de acuerdo con un criterio de ordenamiento. El propósito principal de un ordenamiento es el de facilitar las búsquedas de los miembros del conjunto ordenado.

Para esta práctica se desarrollaron 4 tipos de ordenamiento:

inserción: El ordenamiento por inserción analiza de izquierda a derecha los datos de una sucesión, este se repite para cada elemento de la lista hasta que todos queden ordenados. La idea fundamental de este es ir comparando el elemento actual con el elemento de enfrente y si es mayor intercambia su posición. Se sigue comparando y recorriendo el elemento hasta que llegue a la posición que le corresponde.

Burbuja: Este hace varias pasadas a lo largo de una lista. Compara el elemento actual con su sucesor y si este es mayor los intercambia haciendo que el número más grande vaya quedando atrás acomodándose. En esencia, cada elemento “burbujea” hasta el lugar al que pertenece. El algoritmo será recorrer el arreglo comparando a[i] con a[i+1] para permutarlos si no están en orden.

Selección: Básicamente, este ordenamiento permuta. Entre los elementos busca al mas pequeño y lo pone en primer lugar, y al que estaba en esa posición lo intercambia al lugar en el que estaba el mas pequeño. Ya que permutaron, sigue con la siguiente posición y busca ahora al elemento más pequeño (descartando al de la posición inicial del arreglo porque este ya contiene su elemento definitivo) y los cambia, así con todo el arreglo hasta que todos los elementos estén ordenados.

Mezcla: En teoría, este método de ordenamiento ocupa recursividad donde sus casos base es si la longitud de la lista es 0 o 1, si esto se cumple ya esta ordenada, si no entonces se divide la lista en dos sublistas, estas deben ser a la mitad, si llegase a ser un numero impar entonces se aplica la fórmula: (inicio + (fin-inicio) /2). Cada sublista se ordena recursivamente y al final solo mezcla las dos sublistas en una sola lista ordenada.

DESARROLLO

Planteamiento de problema:

Codificar los algoritmos de ordenamiento por inserción, burbuja, selección y mezcla en lenguaje de programación C.

Empezamos declarando todas las funciones que se ocuparán a lo largo del programa.

```
 6  int entrada(void);//pide en numero de datos a ordenar
 7  void llenado(int* , int);//llena el arreglo
 8  void menu(int* , int);//muestra el menu de opciones
 9  void insercion(int* , int);//ordena por insercion
10  void burbuja(int* , int);//ordena por burbuja
11  void seleccion(int* , int);//ordena por seleccion
12  void mezclaRec(int* , int, int);//hace las divisiones el arreglo
13  void mezcla(int*, int, int, int);//ordena el arreglo por partes
14  void imprimir(int*, int);//imprime el arreglo ordenado
```

La función *entrada* pide el numero de datos a ordenar y devuelve el numero ingresado.

```
38  int entrada(){
39      int n;
40
41      printf("Dame el numero datos a ordenar?: ");
42      scanf("%d", &n);
43
44      return n;//devuelve el numero de datos que el usuario quiere ordenar
45  }
```

La función *llenado* llena el arreglo con números aleatorios de 1 a 99. A esta función se le manda el arreglo y su tamaño. Con un for lo llena y ahí mismo muestra los datos desordenados.

```
93  void llenado(int* numeros, int n){
94      int i;
95      srand(time(NULL));//inicializa la funcion rand
96
97      for(i = 0; i < n; i++){
98          numeros[i] = (rand()%98+1);//se lleva cada indice con un numero aleatorio
99      }
100     printf("\nNumeros sin Ordenar\n");
101     imprimir(numeros, n);//imprime el arreglo sin ordenar
102 }
```

La función *menú* contiene el menú en donde seleccionaremos el tipo de ordenamiento que deseemos. Se le manda el arreglo y su tamaño.

```
47  void menu(int* numeros, int n){
48      int op;
49
50      do{
51          printf("\n<>Menu Metodos de Ordenamiento<>");
52
53          printf("\n1. Ordenamiento por Insercion");
54          printf("\n2. Ordenamiento por Burbuja");
55          printf("\n3. Ordenamiento por Seleccion");
56          printf("\n4. Ordenamiento por Mezcla");
57
58          printf("\n\nElije la opcion de ordenamiento:");
59          scanf("%d", &op);//recibe la opcion que eligio el usuario
60
```

Con un switch se hace el menú. Dentro de los paréntesis de switch estará la opción que el usuario escogió y si es igual a un caso que está ahí ejecutará el ordenamiento. ¿Cómo lo ejecutará? En cada caso estará la función *llenado*, la función con el tipo de ordenamiento y la función *imprimir* para que siga este orden:

- Llene.
- Ordene.
- Imprima.

```
61          switch(op){
62              case 1: llenado(numeros, n);//llama a la funcion llenado
63                      insercion(numeros,n);
64                      printf("\n\nNumeros Ordenados por Insercion\n");
65                      imprimir(numeros, n);//imprime los numero ordenado
66                      break;
67
68              case 2: llenado(numeros, n);//llama a la funcion llenado
69                      burbuja(numeros,n);
70                      printf("\n\nNumeros Ordenados por Burbuja\n");
71                      imprimir(numeros, n);//imprime los numero ordenado
72                      break;
73
74              case 3: llenado(numeros, n);//llama a la funcion llenado
75                      seleccion(numeros,n);
76                      printf("\n\nNumeros Ordenados por Seleccion\n");
77                      imprimir(numeros, n);//imprime los numero ordenado
78                      break;
79
80              case 4: llenado(numeros, n);//llama a la funcion llenado
81                      mezclaRec(numeros,0,n-1);
82                      printf("\n\nNumeros Ordenados por Mezcla\n");
83                      imprimir(numeros, n);//imprime los numero ordenado
84                      break;
85
86              default: printf("Opcion no valida, elija de nuevo!!\n");
87                       system("pause");
88                       system("cls");
89          }
```

También se encuentra un default por si la opción que el usuario ingresó es incorrecta.

Todo este menú se encuentra dentro de un do-while para que se repita el menú en caso de que el usuario no elija una opción válida.

Función *Inserción*

En el caso uno tenemos el ordenamiento por inserción, le mandamos nuestro arreglo ya lleno y el tamaño de este.

Declaramos 3 variables: ‘i’, ‘j’ y ‘aux’. Tenemos un for que recorrerá desde 1 hasta el numero de elementos que se le ordenó. La condición dice que si ‘j’ es mayor 0 y ‘números[j-1]’ es mayor a auxiliar entonces j decrementa y se actualizan los valores. Es decir, el primer elemento se compara con el de enfrente y si es mayor se intercambian de posición y se aumenta el contador.

```
103
104 void insercion(int* numeros, int n){
105     int i, j, aux;
106
107     for(i = 1; i < n; i++){//indice principal
108         j = i;//iguala el valor de j a i
109         aux = numeros[i];//le da valor a la variable auxiliar
110
111         while((j > 0) && (numeros[j-1] > aux)){//verificara si j es mayor a 0
112                                                 //y si el antecesor del numero a comparar es mayor
113             numeros[j] = numeros[j-1];//pone al antecesor en el lugar del numero comparado
114             j--;//decrementa uno a j
115         }
116         //cuando el while ya no se cumpla
117         numeros[j] = aux;// coloca el numero comparado en el lugar que le corresponde
118     }
119 }
120
```

Función *Burbuja*

En la segunda opción tenemos a esta función, también se le manda el arreglo y el número de datos. Se tienen dos ciclos anidados, esto representa las ‘pasadas’. La condición de la línea 126 dice que si el numero de la posición [j -1] es mayor al de la posición en j entonces hace un intercambio que es lo que se lleva a cabo desde la línea 127-129, aquí es donde se ocupa la variable auxiliar para que nos ayude a guardar un valor momentáneamente. Cuando termine de acomodar el primer elemento que sería el más grande, ahora aumenta ‘i’ y sigue con el segundo elemento, aquí estaríamos en la segunda pasada.

```
121 void burbuja(int* numeros, int n){
122     int i, j, aux;
123
124     for(i = 0; i < n; i++){
125         for(j=1; j < n; j++){
126             if(numeros[j-1] > numeros[j]){//si el antecesor del numero es mayor
127                 aux = numeros[j];//guarda el numero menor en auxiliar
128                 numeros[j] = numeros[j-1];//guarda el numero mas grande en su lugar
129                 numeros[j-1] = aux;//guarda el menor en su lugar
130             }
131         }
132     }
133 }
```

Función *Selección*

En la tercera opción tenemos al ordenamiento por selección. Le mandamos el arreglo y el tamaño. Declaramos las variables que vamos a ocupar y empezamos a escribir el primer for. Lo que hará este es recorrer todo el arreglo y después poner una variable 'min' que guarde el elemento más pequeño. El segundo for recorrerá de i+1 hasta n y dentro de ese for irá un condicional y lo que hará será comparar con los otros elementos y si encuentra a uno menor que al que habíamos guardado en un primer momento, lo sustituye. Ya que encontró al primer número mínimo, lo que hará será una permutación con el numero de la primera posición y con la posición donde se encontraba el numero mas pequeño. Así con todo el arreglo.

```
135 void seleccion(int* numeros, int n){
136     int i, j, aux, min;
137 
138     for(i = 0; i < n - 1; i++){
139         min = i;//declara que el numero es el menor
140         for(j = i + 1; j < n; j++){//busca si hay un numero mas pequenio que min
141             if(numeros[j] < numeros[min]){
142                 min = j;//si encuentra un numero mas pequenio guarda la ubicacion de ese numero
143             }
144         }
145             aux = numeros[i];//guarda el numero de la posicion i en un auxiliar
146             numeros[i] = numeros[min];//pone el numero menor en la posicion i
147             numeros[min] = aux;//pone el numero de i en el lugar del menor
148     }
149 }
```

Función *Mezcla*

Como se había comentado anteriormente, este tipo de ordenamiento ocupa recursividad. Divide una lista desordenada en sublistas, ordena los elementos por separado y después los vuelve a juntar, pero ya ordenados.

Para que esto se lleve a cabo necesitamos declarar esta función:

```
151 void mezclaRec(int* numeros, int inicio, int fin){
152     int i;
153 
154     if(inicio < fin){//si el indice del inicio es menor que el indice del final
155             int mitad = inicio + (fin - inicio)/2;//declara la mitad del arreglo
156             mezclaRec(numeros, inicio, mitad);//llama recursivamente mandando inicio y mitad
157             mezclaRec(numeros,mitad + 1, fin);//llama recursivamente mandando mitad y fin
158             mezcla(numeros,inicio,mitad, fin);//llama mezcla la cual hara la funcion de ordenar los numeros
159     }
160 }
```

Contendrá un condicional en primer lugar, si el índice del inicio es menor que el del final entonces se hace la operación para dividir a la mitad la lista y la guardamos en la variable 'mitad' y ahora llama a esa misma función *mezclaRec* pero mandando el arreglo, el inicio y la mitad; y a otra, pero mandando el arreglo, la mitad + 1 y el fin. Después de que estén ordenados se manda a llamar a la función *mezcla* para que ordene y junte los números.

En la función *mezcla* se llevará acabo lo siguiente:

Se declaran dos variables que contendrán el número de elementos de la parte izquierda (n1) y de la parte derecha (n2) y después se declaran los primeros arreglos con sus respectivos tamaños. L[n1] hace referencia al arreglo de lado izquierdo y R[n2] hace referencia al arreglo de lado derecho.

En el for de la línea 171 lo que hará será recorrer y llenar el arreglo de lado izquierdo con la primera mitad del arreglo original, por eso el for va de 0 a n1. El arreglo de la línea 175 hace lo mismo pero con el arreglo de lado derecho, lo llena con la segunda mitad del arreglo original, por eso va de 0 a n2. Las variables se reinician y 'k' se declara con el valor del inicio.

```
161
162 void mezcla(int* numeros, int inicio, int mitad, int fin){
163     int i,j,k;
164
165     int n1 = mitad - inicio + 1;//declara el tamanio del primer arreglo
166     int n2 = fin - mitad;//declara el tamanio del segundo arreglo
167
168     int L[n1];//declara el primer arreglo tamanio n1
169     int R[n2];//declara el segundo arreglo tamanio n2
170
171     for(i = 0; i < n1; i++)
172         L[i] = numeros[inicio + i];
173         //llena el primer arreglo con la primera mitad del arreglo original
174
175     for(j = 0; j < n2; j++)
176         R[j] = numeros[mitad + j + 1];
177         //llena el segundo arreglo con la segund mitad del arreglo original
178
179     i = 0;//reinicia i
180     j = 0;//reinicia j
181     k = inicio;//declara k con el valor del inicio
```

El while de la línea 183 se encargará de mezclar los elementos de los arreglos izquierdo y derecho, para que esto se cumpla el elemento de la posición 'i' debe ser menor al elemento de la posición 'j'. El elemento de la posición del arreglo izquierdo se guardará en la posición del arreglo original e 'i' se aumenta, si esto no se cumple entonces el elemento de la posición de arreglo derecho se guardará en la posición del arreglo original y 'j' se aumenta. Fuera del ciclo while 'k' se aumenta.

```
183     while(i < n1 && j < n2){
184         if(L[i] <= R[j]){//si el numero en R es menor que el de L
185             numeros[k] = L[i];//coloca el numero de L en el indice k del arreglo original
186             i++;//aumenta i + 1
187         }
188         else {// si el numero de L es menor que el de R
189             numeros[k] = R[j];//coloca el numero de R en el indice k del arreglo original
190             j++;//aunmenta j + 1
191         }
192         k++;//aumenta k + 1
193     }
```

Para finalizar, se tendrán dos ciclos while, uno para el lado izquierdo y otro para el lado derecho. Estos while verifican si alguna sublista aun no ha terminado de ordenar sus elementos. Si 'j' o 'i' es menor al tamaño significa que el arreglo aun no ha terminado y debe vaciar los elementos que le quedan.

```
194
195     while(i < n1){//mientras i sea menor al tamanio del primer arreglo
196         numeros[k] = L[i];//pone el numero en L de i en el indice k del arreglo original
197         i++;//aumenta i + 1
198         k++;//aumenta k + 1
199     }
200
201     while(j < n2){//mientras j sea menor al tamanio del segundo arreglo
202         numeros[k] = R[j];//pone el numero en R de j en el indice k del arreglo original
203         j++;//aumenta j + 1
204         k++;//aumenta k + 1
205     }
206 }
207
```

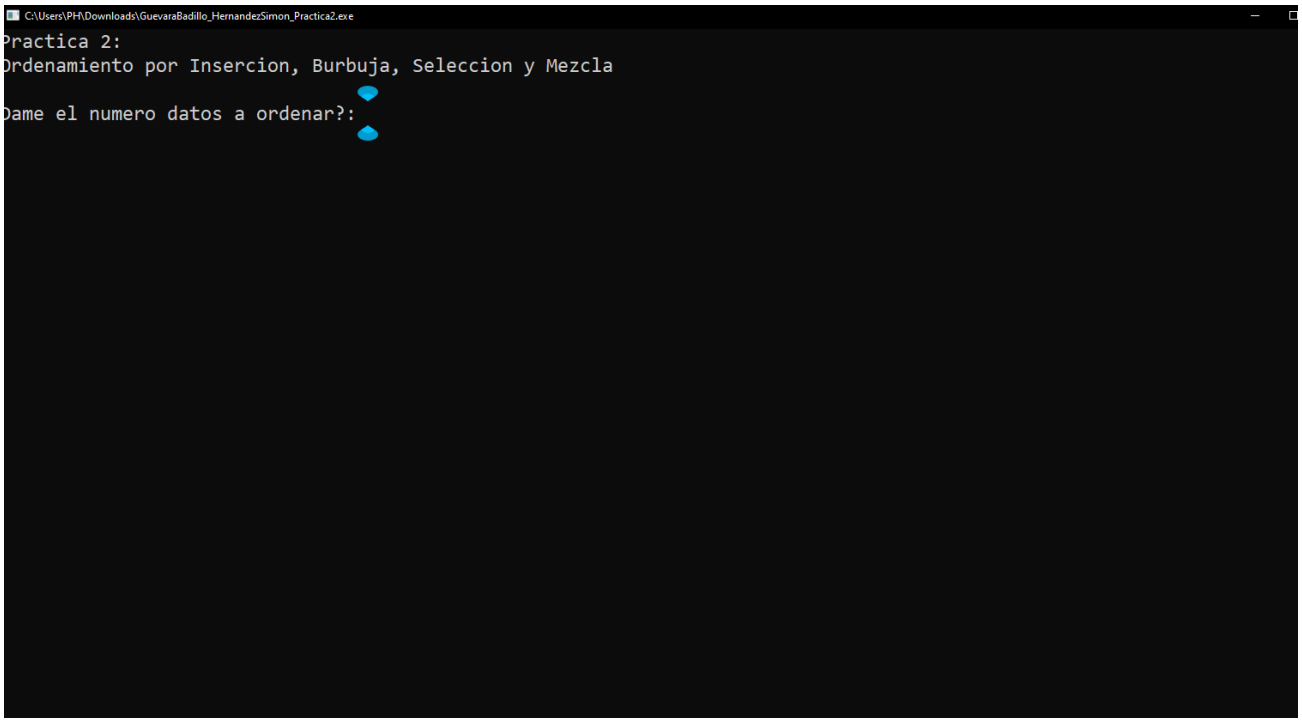
Función *Imprimir*

Esta función lo único que hará será imprimir el arreglo, se le manda el arreglo y su tamaño. Esta función está en cada opción del menú para que imprima con respecto a aquel método de ordenamiento que se le ordenó.

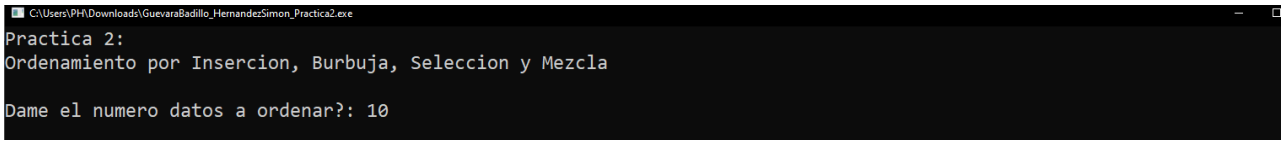
```
208 void imprimir(int* numeros, int n){
209     int i;
210     for(i = 0; i < n ; i++)
211         printf("%d\t", numeros[i]);
212 }
213
```

RESULTADOS

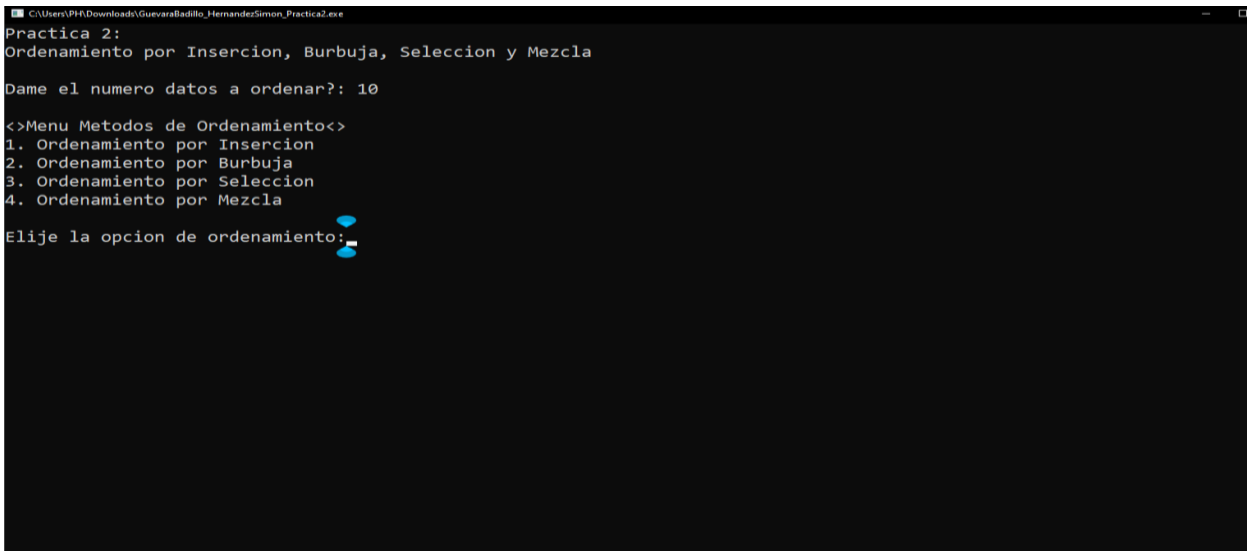
El programa inicia preguntando el número de datos que quiere ordenar el usuario.



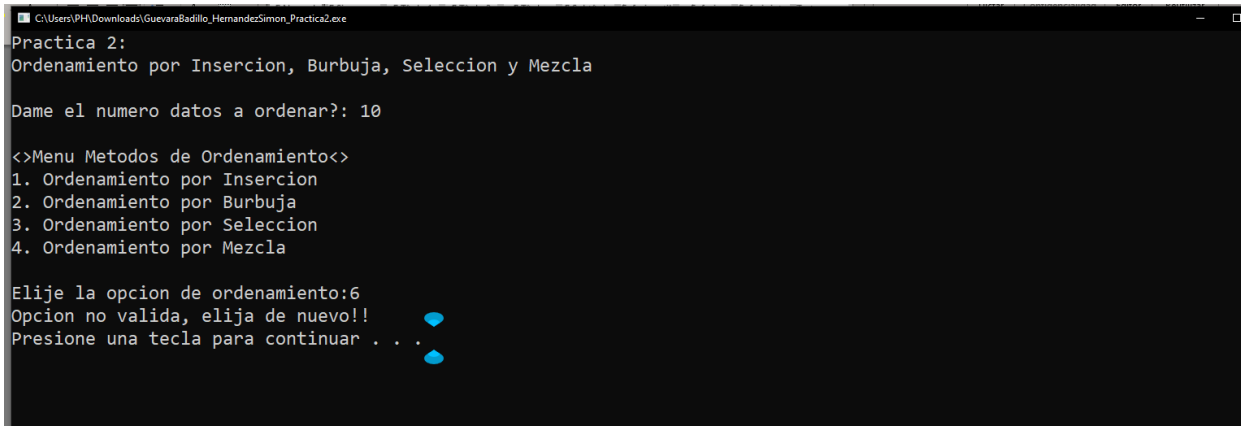
Supongamos que queremos 10 elementos a ordenar…



A continuación, aparecerá el menú donde tú podrás elegir el método de ordenamiento. Las opciones van de 1 a 4.

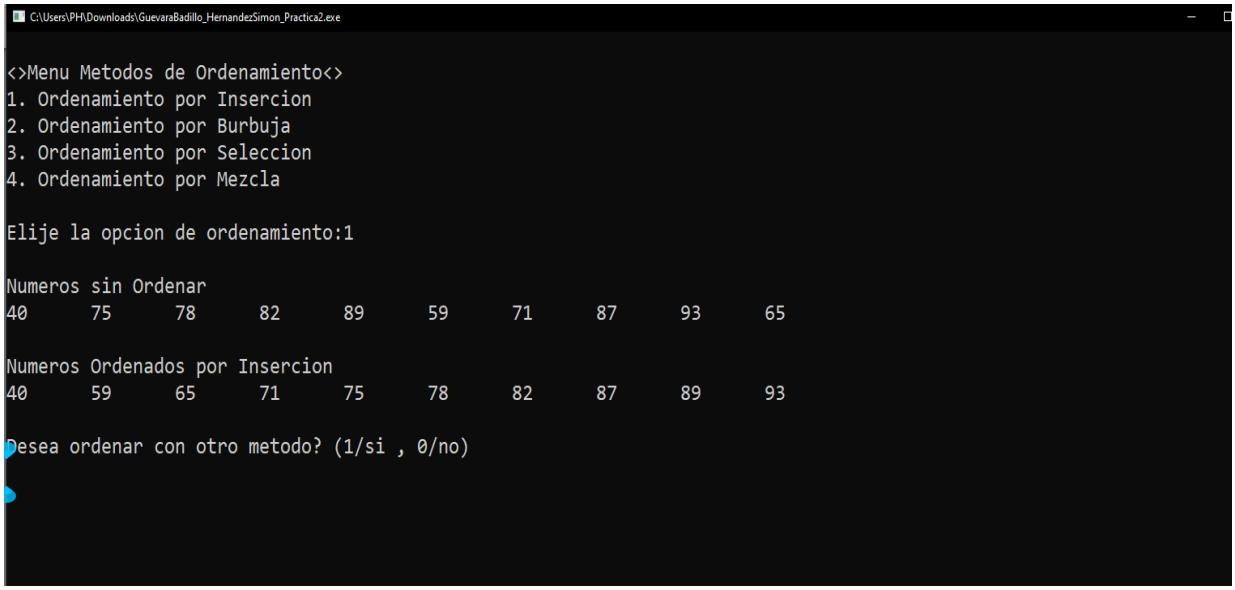


Si el usuario ingresa una opción no valida aparecerá el siguiente mensaje:

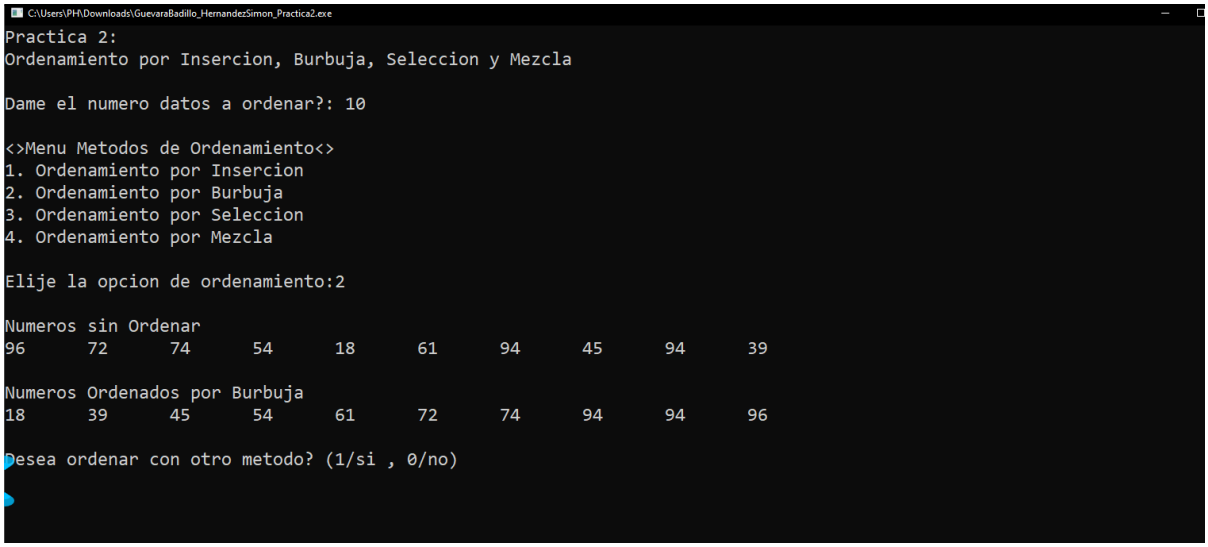


Y te dará la opción de elegir de nuevo

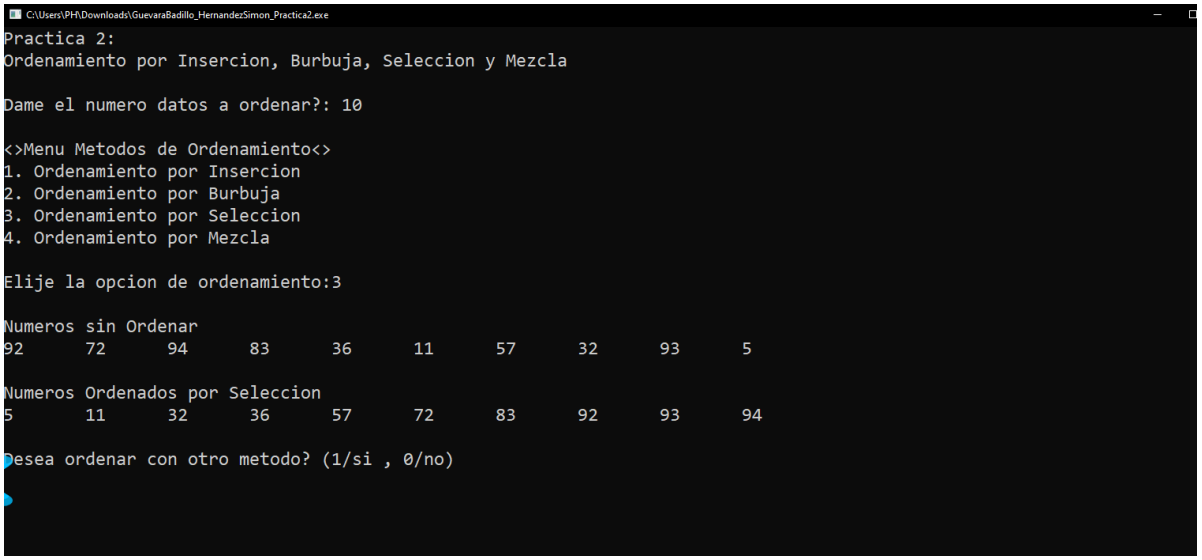
1) Ordenamiento por Inserción:



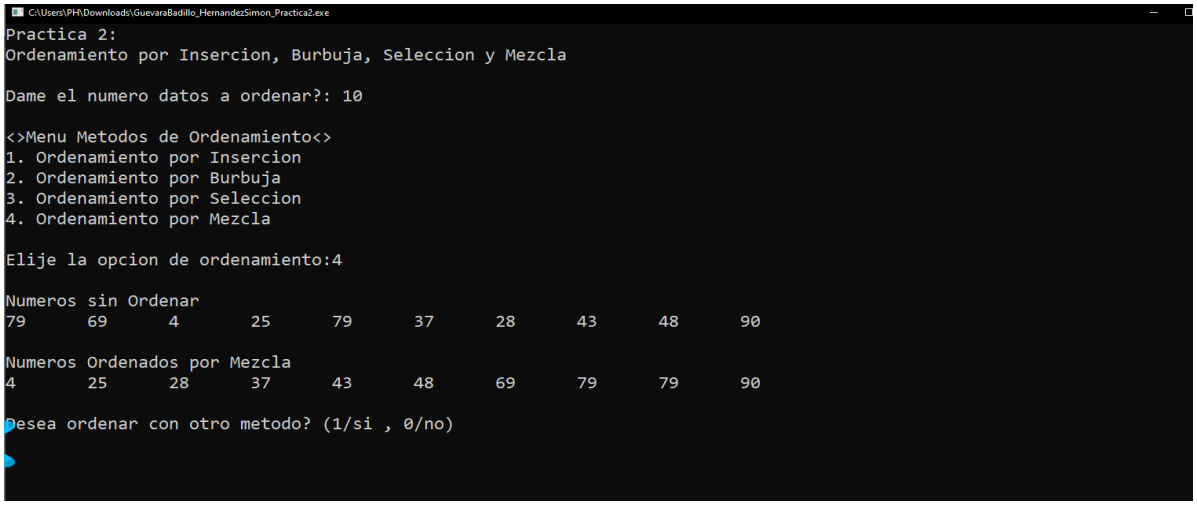
2) Ordenamiento por Burbuja:



3) Ordenamiento por Selección:



4) Ordenamiento por Mezcla:



Al final de cada ordenamiento te preguntará si quiere ordenarlo por otro método. Si tu respuesta es afirmativa (1), el programa te vuelve a preguntar el número de datos a ordenar y el método de ordenamiento, si tu respuesta es negativa (0) el programa termina ahí.