UNIDAD DE APRENDIZAJE
DE PROGRAMACIÓN AVANZADA

CRÉDITOS INSTITUCIONALES: 9

MANUAL DE PRÁCTICAS DE LABORATORIO: PROGRAMACIÓN AVANZADA

PROGRAMA DE INGENIERÍA EN COMPUTACIÓN

CU UAEM VALLE DE CHALCO

AUTORES:
MTRO. MARCO ALBERTO MENDOZA PÉREZ
MTRA. MARISOL HERNÁNDEZ HERNÁNDEZ
MTRO. RODOLFO MELGAREJO SALGADO

PERIODO DE REALIZACIÓN:
MARZO - JUNIO 2019

ÍNDICE DE CONTENIDO

PRESENTACIÓN

Las presentes prácticas de laboratorio fueron desarrolladas en apego al programa de la Unidad de Aprendizaje de Programación Avanzada, con la finalidad de que los alumnos puedan poner en práctica los conocimientos teóricos que se van adquiriendo a lo largo del curso. Esta Unidad de Aprendizaje consta de la siguiente estructura:

1.- Paradigmas de programación modular y recursiva.
2.- Análisis de algoritmos de ordenamiento y búsqueda.
3.- Diseño de algoritmos empleando diversas técnicas.

Este manual consta de 4 Prácticas, cuya complejidad es acorde al tema que abarca. En este sentido estas prácticas pertenecen a los temas: 2.2.- Método de la Burbuja, 2.5.- Método de Ordenamiento Rápido, 2.7.- Método de Búsqueda Secuencial y 2.8.- Método de Búsqueda Binaria.

Cada Práctica consta de Titulo, Objetivos, Introducción, Material y Equipo a Utilizar, Duración, Desarrollo y Evaluación. El alumno debe replicar de forma práctica lo que se encuentra en el apartado de Desarrollo y resolver lo que se pide en el apartado de Evaluación.

De cada Práctica, el alumno deberá elaborar su Reporte de Práctica correspondiente, el cual lo debe entregar de forma: digital o impresa, con los siguientes puntos: Carátula, Índice, Introducción, Desarrollo (evidencias originales “algoritmos y códigos” con explicación), Resultados (evidencias originales “ejecución de códigos” con explicación), Conclusiones Individuales y Bibliografía/Referencias Electrónicas (ambas en formato APA).

PRÁCTICA 1

MÉTODO DE ORDENAMIENTO DE LA BURBUJA

OBJETIVOS:

Conocer el funcionamiento de este método de ordenamiento y programar su algoritmo en Lenguaje C.

INTRODUCCIÓN:

Para desarrollar la práctica, se va a hacer uso de la siguiente información:

Ordenar significa reagrupar o reorganizar un conjunto de datos u objetos en una secuencia especifica.

Los objetos ordenados aparecen en cualquier lugar. Directorios telefónicos, registros de pacientes en un hospital, registros de pasajeros de una aerolínea, registros de huéspedes de un hotel, registros de alumnos inscritos en una escuela, son tan solo algunos ejemplos de objetos ordenados con los cuales el ser humano se encuentra frecuentemente. La ordenación es una actividad fundamental y relevante en la vida.

Formalmente se define el ordenamiento de la siguiente manera:
Sea A una lista o arreglo de N elementos: A_1, A_2, A_3, …, A_N

Ordenar significa permutar estos elementos de tal forma que los mismos queden de acuerdo con una distribución preestablecida.
Ascendente: $A_1 <= A_2 <= A_3 <= \ldots <= A_N$
Descendente: $A_1 >= A_2 >= A_3 >= \ldots >= A_N$

En el procesamiento de datos, los métodos de ordenamiento se clasifican en 2 tipos:

a) Ordenamiento de arreglos u ordenación interna: Los elementos o componentes del arreglo, lista enlazada o árbol se encuentran en la memoria principal de la computadora.
b) Ordenamiento de archivos u ordenación externa: Los elementos se encuentran en archivos almacenados en dispositivos de almacenamiento secundario como discos, cintas, etc.

El método de ordenamiento de la burbuja, es uno de los más simples, es tan fácil como *comparar* todos los elementos de un arreglo contra todos, si se cumple que uno es mayor o menor a otro, entonces los *intercambia* de posición.

Veamos a continuación su algoritmo, para ordenar un arreglo de menor a mayor (ascendente), partiendo de que los datos a ordenar están en un arreglo de N elementos:

1. Comparamos el primer elemento con el segundo, el segundo con el tercero, el tercero con el cuarto, etc. Cuando el resultado de una *comparación* sea "mayor que", se *intercambian* los valores de los elementos comparados. Con esto conseguimos llevar el valor mayor a la posición n.
2. Repetimos el punto 1, ahora para los n-1 primeros elementos del arreglo y así sucesivamente.
3. Repetimos el punto 1, ahora para los n-2 primeros elementos del arreglo y así sucesivamente.
4. El proceso termina después de repetir el punto 1, n-1 veces, o cuando al finalizar la ejecución del punto 1 no haya existido ningún cambio.

Ejemplo: Suponga se desea ordenar los elementos que se encuentran en el arreglo A utilizando el método de la burbuja.

int A[6] = {40, 21, 4, 9, 10, 35};

Primera pasada
{40, 21, 4, 9, 10, 35}
{21, 40, 4, 9, 10, 35} si (A[i] > A[i+1]) ← Si (40 > 21), se intercambia el 21 por el 40
{21, 4, 40, 9, 10, 35} si (A[i] > A[i+1]) ← Si (40 > 4), se intercambia el 4 por el 40
{21, 4, 9, 40, 10, 35} si (A[i] > A[i+1]) ← Si (40 > 9), se intercambia el 9 por el 40
{21, 4, 9, 10, 40, 35} si (A[i] > A[i+1]) ← Si (40 > 10), se intercambia el 10 por el 40
{21, 4, 9, 10, 35, 40} si (A[i] > A[i+1]) ← Si (40 > 35), se intercambia el 35 por el 40

Segunda pasada
{21, 4, 9, 10, 35, 40}
{4, 21, 9, 10, 35, 40} si (A[i] > A[i+1]) ← Si (21 > 4), se intercambia el 4 por el 21
{4, 9, 21, 10, 35, 40} si (A[i] > A[i+1]) ← Si (21 > 9), se intercambia el 9 por el 21
{4, 9, 10, 21, 35, 40} si (A[i] > A[i+1]) ← Si (21 > 10), se intercambia el 10 por el 21
{4, 9, 10, 21, 35, 40} si (A[i] > A[i+1]) ← Si (21 > 35), no hay intercambio
{4, 9, 10, 21, 35, 40} si (A[i] > A[i+1]) ← Si (35 > 40), no se comparan

Este algoritmo es estable porque nunca intercambia registros con claves iguales. Solo requiere de una variable adicional para realizar los intercambios. Es muy lento, su rendimiento es muy bajo, realiza numerosas comparaciones y numerosos intercambios.

MATERIAL Y EQUIPO A UTILIZAR:

Para la realización de esta práctica son necesarios los siguientes componentes:

Equipo:

- Computadora.

Software:

- IDE CodeBlocks.

Material:

- Práctica digital o impresa.

DURACIÓN:

- 420 minutos.

DESARROLLO:

Reproducir en el IDE CodeBlocks, el código en Lenguaje C de la Figura 1.1.

```
#include<stdio.h>
#define T 10
main(){
int a[T]={2,6,4,8,10,12,89,68,45,37};
int i,pasada,variable;
printf("Arreglo original: \n");
for(i=0; i<=T-1;i++)
   printf("%4d",a[i]);
for(pasada=1; pasada<=T-1;pasada++)
   for(i=0; i<=T-2;i++)
if(a[i]>a[i+1]){
   variable=a[i];
   a[i]=a[i+1];
   a[i+1]=variable;
}
printf("\nArreglo en orden ascendente: \n");
for(i=0; i<=T-1;i++)
   printf("%4d",a[i]);
printf("\n");
return 0;
}
```

Figura 1.1 Programa en Lenguaje C del método de ordenamiento de la burbuja.

En la Figura 1.2, se muestra la ejecución del código en Lenguaje C del método de ordenamiento de la burbuja.

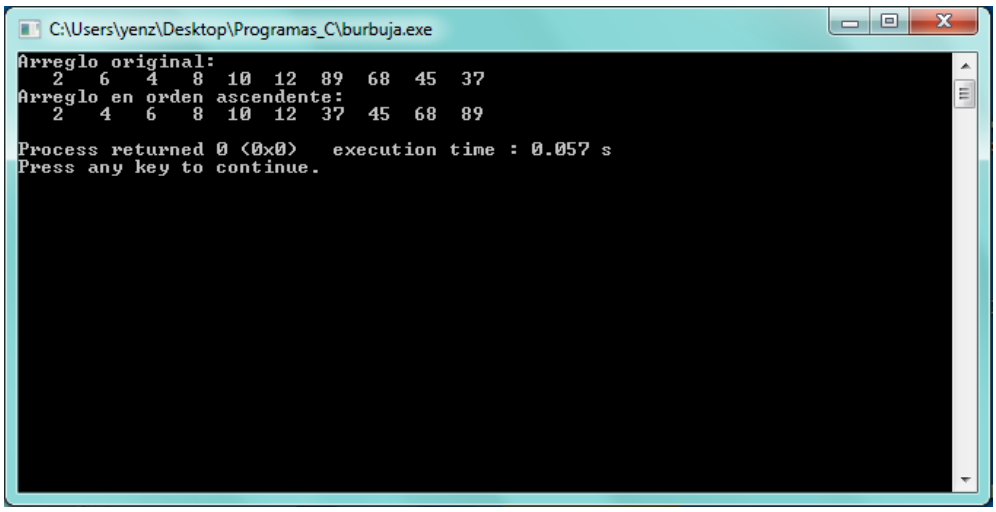

Figura 1.2 Ejecución del programa de ordenamiento de la burbuja.

EVALUACIÓN:

Del programa de la Figura 1.1., realiza los siguientes programas para mejorar el rendimiento de este tipo de ordenación:

1.- Coloca en tres nuevas funciones los códigos para que el usuario ingrese 10 números en el arreglo, el método de ordenamiento burbuja y si deseas volver a ingresar valores o salir del programa, y posteriormente mándalos llamar desde la función main().

2.- Después de la primera pasada el número más alto está garantizado que deberá aparecer en el elemento numerado más alto dentro del arreglo; después de la segunda pasada, los dos números más altos estarán en su lugar, y así en lo sucesivo. En vez de hacer nueve comparaciones en cada pasada, 7 en la tercera, y así sucesivamente.

3.- Los datos en el arreglo pudieran estar ya en el orden apropiado o en un orden casi apropiado, por lo tanto, ¿Por qué hacer nueve pasadas si menos pudieran ser suficientes?. Modifique la ordenación para verificar, al final de cada pasada, si se han hecho intercambios. Si no ha habido intercambios, entonces los datos deben de estar ya en el orden apropiado y, por lo tanto, el programa debe darse por terminado. Si ha habido intercambios, entonces por lo menos se requiere de una pasada adicional.

Nota: Leer la práctica, trabajar en los apartados de Desarrollo y Evaluación; para posteriormente elaborar el Reporte de Práctica y por último entregarlo a su profesor en forma digital o impresa en la fecha acordada con él.

PRÁCTICA 2

MÉTODO DE ORDENAMIENTO RÁPIDO

OBJETIVOS:

Conocer el funcionamiento de este método de ordenamiento y programar su algoritmo en Lenguaje C.

INTRODUCCIÓN:

Para desarrollar la práctica, se va a hacer uso de la siguiente información:

Ordenar significa reagrupar o reorganizar un conjunto de datos u objetos en una secuencia especifica.

Los objetos ordenados aparecen en cualquier lugar. Directorios telefónicos, registros de pacientes en un hospital, registros de pasajeros de una aerolínea, registros de huéspedes de un hotel, registros de alumnos inscritos en una escuela, son tan solo algunos ejemplos de objetos ordenados con los cuales el ser humano se encuentra frecuentemente. La ordenación es una actividad fundamental y relevante en la vida.

Formalmente se define el ordenamiento de la siguiente manera:
Sea A una lista o arreglo de N elementos: A_1, A_2, A_3, …, A_N

Ordenar significa permutar estos elementos de tal forma que los mismos queden de acuerdo con una distribución preestablecida.
Ascendente: $A_1 <= A_2 <= A_3 <= \ldots <= A_N$
Descendente: $A_1 >= A_2 >= A_3 >= \ldots >= A_N$

En el procesamiento de datos, los métodos de ordenamiento se clasifican en 2 tipos:

a) Ordenamiento de arreglos u ordenación interna: Los elementos o componentes del arreglo, lista enlazada o árbol se encuentran en la memoria principal de la computadora.
b) Ordenamiento de archivos u ordenación externa: Los elementos se encuentran en archivos almacenados en dispositivos de almacenamiento secundario como discos, cintas, etc.

El método de ordenamiento rápido, es actualmente el más eficiente y veloz de los métodos de ordenación interna. Este método es una mejora sustancial del método de intercambio directo y recibe el nombre de quicksort (rápido) por la velocidad con que ordena los elementos del arreglo.

Fue desarrollado por C. Antony R. Hoare en 1960. El algoritmo original es recursivo, pero se utilizan versiones iterativas para mejorar su rendimiento (los algoritmos recursivos son en general más lentos que los iterativos y consumen más recursos).

La idea central de este algoritmo consiste en lo siguiente:

1. Se toma un elemento X de una posición cualquiera del arreglo.
2. Se trata de ubicar a X en la posición correcta del arreglo, de tal forma que todos los elementos que se encuentren a su izquierda sean menores o iguales a X y todos los elementos que se encuentren a su derecha sean mayores o iguales a X.
3. Se repiten los pasos anteriores, pero ahora para los conjuntos de datos que se encuentran a la izquierda y a la derecha de la posición correcta de X en el arreglo.
4. El proceso termina cuando todos los elementos se encuentran en su posición correcta en el arreglo.

Ejemplo: Suponga se desea ordenar los elementos que se encuentran en el arreglo A utilizando el método quicksort.

int A[8] = {15, 67, 8, 16, 44, 27, 12, 35};

Se selecciona A[0], por lo tanto X <- 15.
Las comparaciones que se realizan son las siguientes:

Primera pasada
Recorrido de derecha a izquierda (A[0] <- A[7])
A[7] >= X (35 >= 15) Si, por lo tanto, No hay intercambio
A[6] >= X (12 >= 15) No, por lo tanto, Si hay intercambio

Va quedando:
A = {12, 67, 8, 16, 44, 27, 15, 35}

Recorrido de izquierda a derecha (A[1] -> A[6])
A[1] <= X (67 <= 15) No, por lo tanto, Si hay intercambio

Después de la primera pasada, queda:
A = {12, 15, 8, 16, 44, 27, 67, 35}

Segunda pasada
Recorrido de derecha a izquierda (A[1] <- A[5])
A[5] >= X (27 >= 15) Si, por lo tanto, No hay intercambio
A[4] >= X (44 >= 15) Si, por lo tanto, No hay intercambio
A[3] >= X (16 >= 15) Si, por lo tanto, No hay intercambio

A[2] >= X (8 >= 15) No, por lo tanto, Si hay intercambio

Va quedando:
A = {12, 8, 15, 16, 44, 27, 67, 35}
Recorrido de izquierda a derecha (A[1] -> A[4])
A[1] <= X (8 <= 15) Si, por lo tanto, No hay intercambio

Después de la segunda pasada, queda:
A = {12, 8, 15, 16, 44, 27, 67, 35}

P*asadas siguientes:*
A = {12, 8, 15, 16, 35, 27, 44, 67}

A = {12, 8, 15, 16, 27, 35, 44, 67}

A = {8, 12, 15, 16, 27, 35, 44, 67}

MATERIAL Y EQUIPO A UTILIZAR:

Para la realización de esta práctica son necesarios los siguientes componentes:
Equipo:

- Computadora.

Software:

- IDE CodeBlocks.

Material:

- Práctica digital o impresa.

DURACIÓN:

- 420 minutos.

DESARROLLO:

Reproducir en el IDE CodeBlocks, el código en el Lenguaje de programación C del método de ordenamiento rápido de la Figura 2.1.

```
#include <stdlib.h>
#include <stdio.h>
void intercambio(int* a, int* b) { //Función para intercambiar los valores de 2 elementos.
   int temp = *a;
   *a = *b;
   *b = temp;
}
```

```
void quicksort(int* izq, int* der) { //Función recursiva para ordenar el arreglo.
   if (der < izq)
      return;
   int pivote = *izq;
   int* ulti = der;
   int* prim = izq;
   while (izq < der) {
      while (*izq <= pivote && izq < der+1)
         izq++;
      while (*der > pivote && der > izq-1)
         der--;
      if (izq < der)
         intercambio(izq, der); }
   intercambio(prim, der); //Llamamos estas 3 funciones.
   quicksort(prim, der-1);
   quicksort(der+1, ulti);
}
int main(void) {
   int i, tam;
   printf("Ingresa el tamaño del arreglo:\n"); //Ingresamos el tamaño del arreglo.
   scanf("%d", &tam);
   int arreglo[tam];

   printf("Ingresa valores para el arreglo:\n"); //Llenamos el arreglo.
   for (i = 0; i < tam; i++)
      scanf("%d", &arreglo[i]);
   printf("\n");

   printf("Arreglo Original:\n"); //Mostramos el arreglo original.
   for (i = 0; i < tam; i++)
      printf("%d ", arreglo[i]);
   printf("\n");

   quicksort(&arreglo[0], &arreglo[tam-1]); //Llamamos a la función quicksort.

   printf("Arreglo Ordenado:\n"); //Mostramos el arreglo ordenado.
   for (i = 0; i < tam; i++)
      printf("%d ", arreglo[i]);
   printf("\n\n");
}
```

Figura 2.1 Programa en Lenguaje C del método de ordenamiento rápido.

En la Figura 2.2, se muestra la ejecución del código del método de ordenamiento rápido.

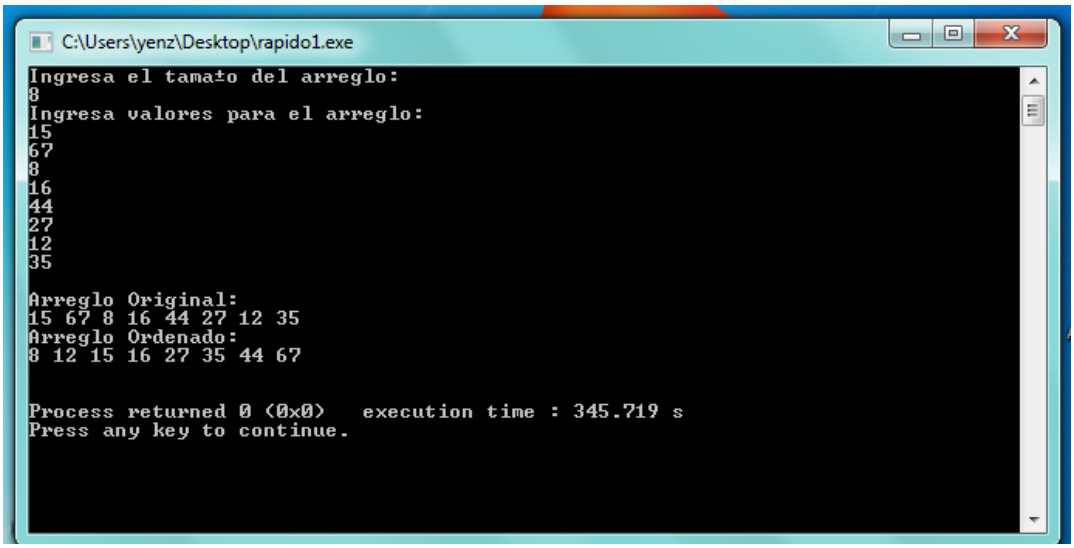

Figura 2.2 Ejecución del programa de ordenamiento rápido.

En el Lenguaje de programación C, hay una función que ya realiza el método de ordenamiento expuesto en la Figura 2.1, sin tener que desarrollarla. Esta función se encuentra incluida en la librería stdlib.h y se llama qsort. Reproducir en el IDE CodeBlocks, el código de la Figura 2.3.

```
#include <stdlib.h>
#include <stdio.h>

int comparacion(const void *i, const void *j) { //Función para comparar 2 elementos
return *(int *)i - *(int *)j;                   //del arreglo.
}

int main(void) {
   int i, tam;
   printf("Ingresa el tamaño del arreglo:\n"); //Ingresamos el tamaño del arreglo.
   scanf("%d", &tam);
   int arreglo[tam];

   printf("Ingresa valores para el arreglo:\n"); //Llenamos el arreglo.
   for (i = 0; i < tam; i++)
      scanf("%d", &arreglo[i]);
   printf("\n");

   printf("Arreglo Original:\n"); //Mostramos el arreglo original.
```

```
   for (i = 0; i < tam; i++)
      printf("%d ", arreglo[i]);
   printf("\n");

   qsort(arreglo, tam, sizeof(int), comparacion); //Hacemos el llamado a la función qsort.

   printf("Arreglo Ordenado:\n"); //Mostramos el arreglo ordenado.
   for (i = 0; i < tam; i++)
      printf("%d ", arreglo[i]);
   printf("\n\n");
}
```

Figura 2.3 Programa en Lenguaje C del método de ordenamiento rápido, con la función qsort incluida en la librería stdlib.h.

En la Figura 2.4, se muestra la ejecución del código anterior.

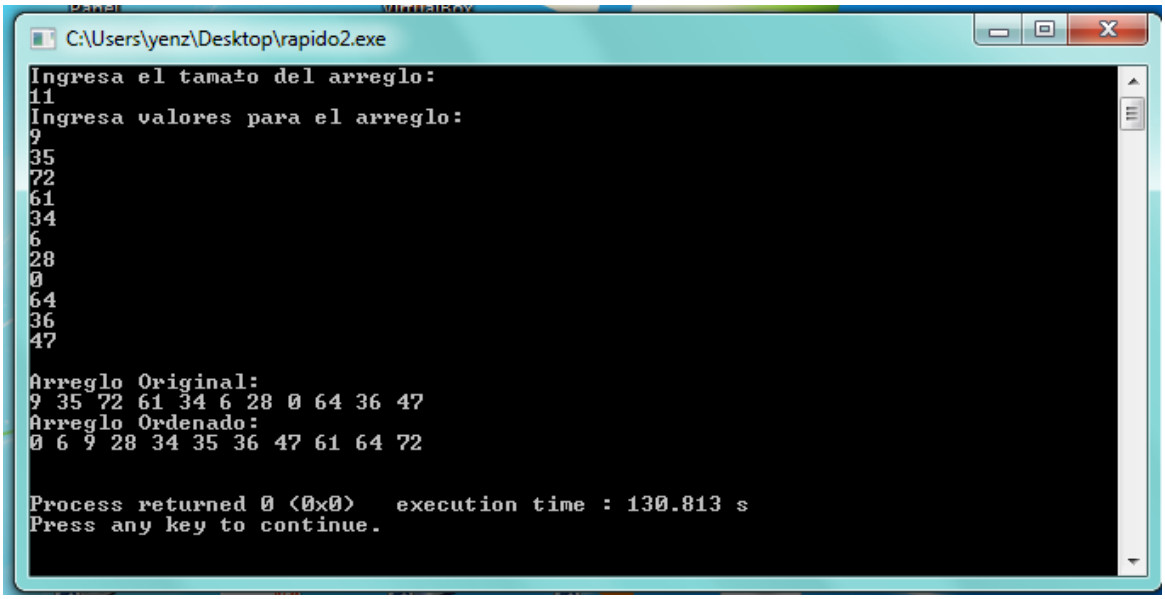

Figura 2.4 Ejecución del programa de ordenamiento rápido, con la función qsort.

EVALUACIÓN:
Selecciona uno de los programas anteriores y realiza el siguiente programa:

1.- Se tienen 3 arreglos paralelos. El primero de ellos almacena los Números de Cuenta de N alumnos; el segundo, los Promedios Generales de los N alumnos, y el tercero, el Total de Materias Aprobadas por cada alumno. Los elementos de los arreglos se corresponden.

- Agrega una función que permita al usuario ingresar el Numero de alumnos en el sistema y de cada uno de estos su Número de Cuenta (condicionar que acepte entre 6 y 7 dígitos, solo números), Promedio General (condicionar que acepte un numero decimal después del punto) y Total de Materias Aprobadas (condicionar que acepte entre 0 y 20, solo números).
- Agrega una función que ordene los valores de los arreglos ingresados por el usuario, de tal manera que queden ordenados en forma descendente por el Total de Materias Aprobadas y se visualicen en pantalla.
- Agrega una función que ordene los valores de los arreglos ingresados por el usuario, de tal manera que queden ordenados en forma ascendente por el Número de Cuenta y se visualicen en pantalla.
- Agrega una función que permita al usuario ingresar la cantidad de valores que va a generar el sistema de forma aleatoria para los arreglos Número de Cuenta (condicionar que acepte entre 6 y 7 dígitos, solo números), Promedio General (condicionar que acepte un numero decimal después del punto) y Total de Materias Aprobadas (condicionar que acepte entre 0 y 20, solo números). En esa misma función que ordene los arreglos, de tal manera que queden ordenados en forma ascendente por el Promedio General y se visualicen en pantalla.

Nota: Leer la práctica, trabajar en los apartados de Desarrollo y Evaluación; para posteriormente elaborar el Reporte de Práctica y por último entregarlo a su profesor en forma digital o impresa en la fecha acordada con él.

PRÁCTICA 3

MÉTODO DE BÚSQUEDA SECUENCIAL

OBJETIVOS:

Conocer el funcionamiento de este método de búsqueda y programar su algoritmo en Lenguaje C.

INTRODUCCIÓN:

Para desarrollar la práctica, se va a hacer uso de la siguiente información:

Buscar significa recuperar datos previamente almacenados. La búsqueda es una actividad relevante en la vida diaria del ser humano. Se realizan tareas esenciales, como la búsqueda de números telefónicos en un directorio, ofertas laborales en un periódico o en sitios de internet, libros en una biblioteca, etc.

La operación de búsqueda puede llevarse a cabo sobre elementos ordenados y sobre elementos desordenados. En el primer caso, la búsqueda se facilita, y por lo tanto se ocupará menos tiempo que si se trabajara con elementos desordenados.

Los métodos de búsqueda pueden clasificarse en 2 tipos:

a) Búsqueda interna: Cuando todos los elementos se encuentran en la memoria principal de la computadora (almacenados en arreglos o listas ligadas).
b) Búsqueda externa: Cuando todos los elementos se encuentran en memoria secundaria (archivos almacenados en dispositivos como discos magnéticos y cintas).

La búsqueda secuencial o lineal consiste en revisar elemento por elemento hasta encontrar el dato buscado, o hasta llegar al final de la lista de datos disponibles.
Cuando se habla de búsqueda en arreglos debe distinguirse entre arreglos desordenados y arreglos ordenados.

La búsqueda secuencial en arreglos desordenados consiste, en recorrer el arreglo de izquierda a derecha hasta que se encuentre el elemento buscado o se termine el arreglo, lo que ocurra primero. Por lo tanto, en promedio, el programa tendrá que comparar el valor buscado con la mitad de los elementos del arreglo.

La búsqueda secuencial en arreglos ordenados es similar al caso anterior. Sin embargo, el orden que existe entre los elementos del arreglo permite incluir una nueva condición que hace más eficiente el proceso.

El método de búsqueda secuencial funciona bien para arreglos pequeños o para arreglos no ordenados. Sin embargo, para la búsqueda de arreglos extensos, el sistema lineal es ineficiente. Si el arreglo esta ordenado, se puede utilizar el método de alta velocidad de búsqueda binaria.

MATERIAL Y EQUIPO A UTILIZAR:

Para la realización de esta práctica son necesarios los siguientes componentes:
Equipo:

- Computadora.

Software:

- IDE CodeBlocks.

Material:

- Práctica digital o impresa.

DURACIÓN:

- 420 minutos.

DESARROLLO:

Reproducir en el IDE CodeBlocks, el código en el Lenguaje de programación C del método de búsqueda secuencial de la Figura 3.1.

```
#include<stdio.h>
#define T 100
main(){
int a[T], x, buscar, ii=0;
for(x=0; x<T; x++)
a[x]=2*x;
printf("Ingresa el elemento que deseas buscar en el arreglo: \n");
scanf("%d",&buscar);
  for(x=0; x<T; x++){
      if(a[x] == buscar){
         printf("El numero %d se encuentra en la posición %d del arreglo \n",buscar,x);
         ii=ii+1;
      } }
      if(ii==0){
         printf("El numero %d no se encontró en el arreglo \n",buscar);
      }
return 0;
}
```

Figura 3.1 Programa en Lenguaje C del método de búsqueda secuencial.

En las Figuras 3.2 y 3.3, se muestran 2 resultados de ejecución del código búsqueda secuencial.

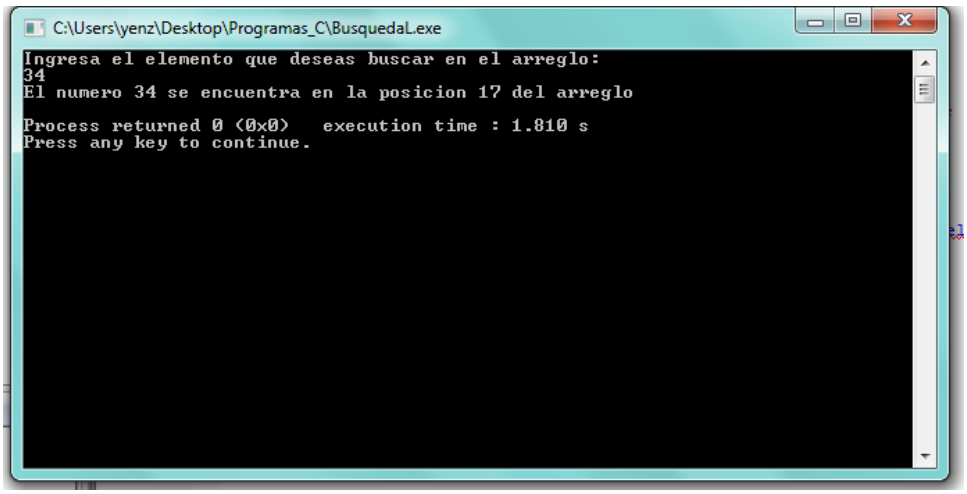

Figura 3.2 Ejecución del programa para el caso de cuando se encontró un elemento en el arreglo.

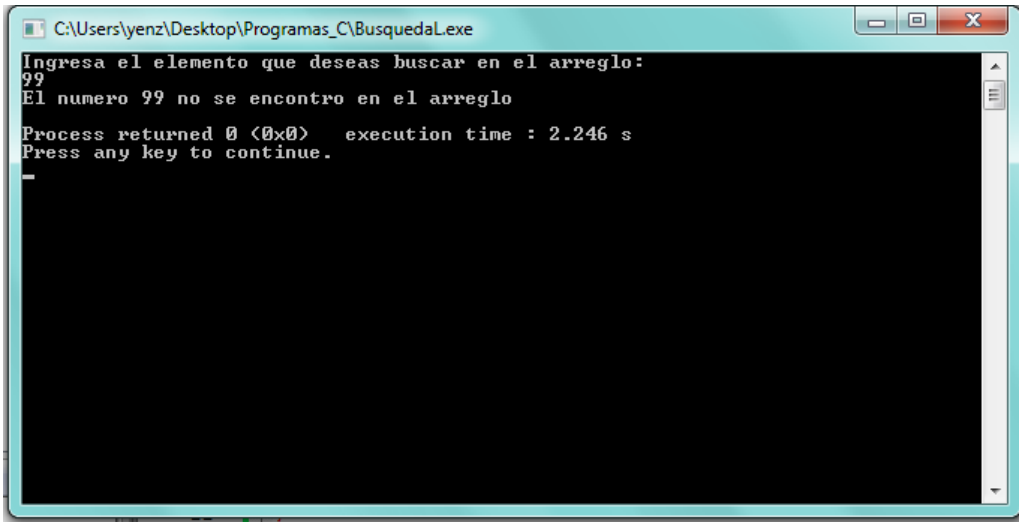

Figura 3.3 Ejecución del programa para el caso de cuando no se encontró el elemento en el arreglo.

En la Figura 3.4, se presenta otra forma de programar en Lenguaje C el método de búsqueda secuencial. En este programa, se utilizan punteros y el código de búsqueda secuencial se coloca en una nueva función, la cual se manda a llamar desde la función main().

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int BusquedaSecuencial(int *parray, int buscar, int nelem);
```

```
void main(){
int *pdatos, nelem, dbuscar,d,result;
printf("Cuantos elementos deseas en el arreglo: ");
scanf("%d",&nelem);
pdatos=(int *)malloc(nelem * sizeof(int));
if(pdatos==NULL){
   printf("Insuficiente espacio de memoria");
   exit(-1);
}
for(d=0; d<nelem; d++)
{
   printf("Elemento[%d]: ",d);
   scanf("%d",(pdatos+d));
}

printf("\nQue elemento deseas buscar en el arreglo: ");
scanf("%d",&dbuscar);
result=BusquedaSecuencial(pdatos,dbuscar,nelem);
if(result !=-1)
   printf("\n%d Se encuentra en la posición %d del arreglo: \n",dbuscar,result);
else
   printf("\n%d No se encontró en el arreglo\n",dbuscar);
}

int BusquedaSecuencial(int *parray, int buscar, int elem)
{
  int i;
  for(i=0; i<elem; i++)
  {
  if(*(parray+i)==buscar)
  return (i);
  }
  return(-1);
}
```

Figura 3.4 Otra forma de programar en Lenguaje C el método de búsqueda secuencial.

En las Figuras 3.5 y 3.6, se muestran 2 resultados de ejecución del código anterior.

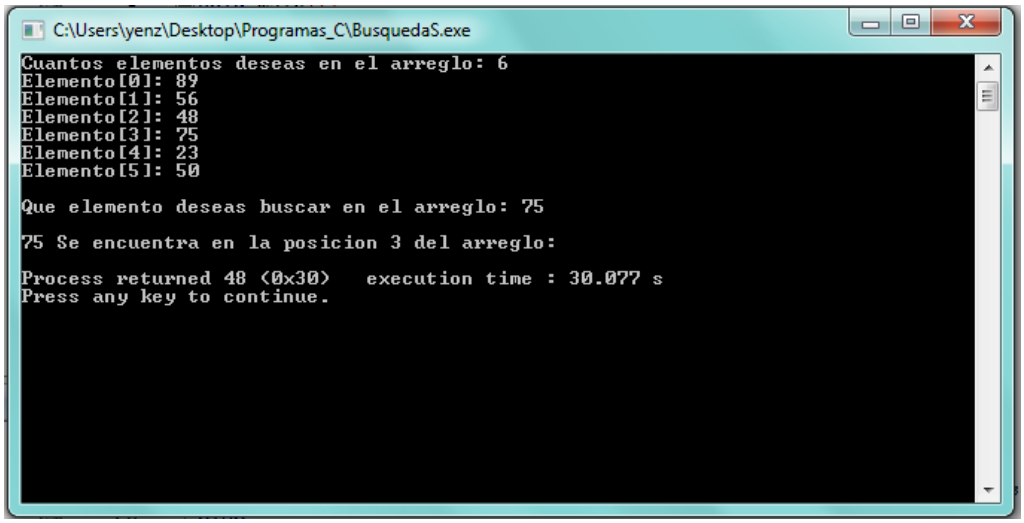

Figura 3.5 Ejecución del programa para el caso de cuando se encontró un elemento en el arreglo.

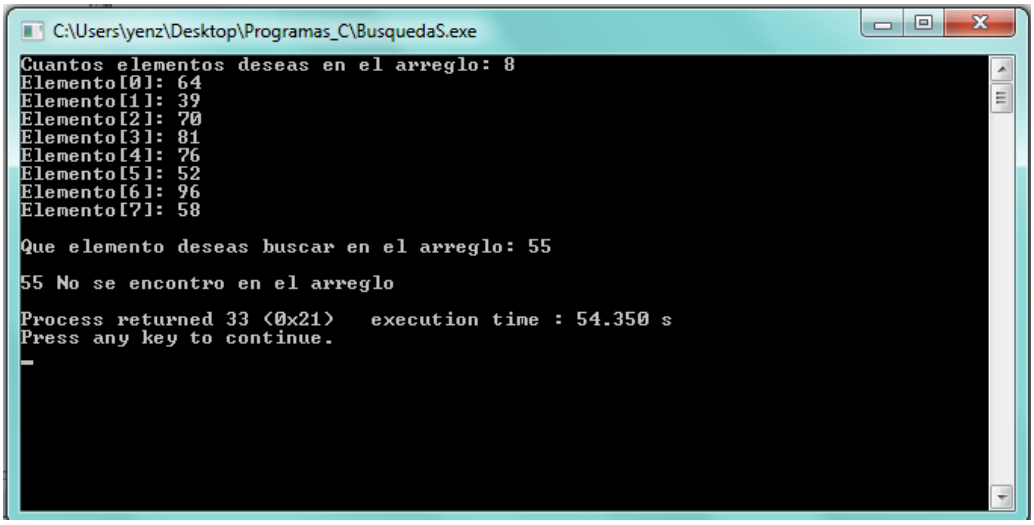

Figura 3.6 Ejecución del programa para el caso de cuando no se encontró el elemento en el arreglo.

EVALUACIÓN:
A partir de los 2 programas anteriores, realiza los siguientes programas en Lenguaje C:

1.- Modifica el programa de la Figura 3.1. Coloca en una nueva función el código del método de búsqueda secuencial y posteriormente mándalo llamar desde la función main().

2.- Selecciona uno de los programas anteriores y realiza lo que se te pide. Dado un arreglo de N elementos que contiene la siguiente información: Número de cuenta, Nombre del alumno, Promedio y Numero de materias aprobadas.

Elabora un menú que lea el Nombre o Número de cuenta de un alumno y arroje como resultado el Promedio y el Número de materias aprobadas por dicho alumno. Si el Nombre o Número de cuenta dado no está en el arreglo, envíe un mensaje adecuado.

a) Considera que el arreglo esta desordenado.
b) Considera que el arreglo esta ordenado. Para esto utiliza uno de los métodos de ordenamiento antes vistos y ordénalo por Nombre o Número de cuenta.

Notas:
1.- Visualiza los tiempos de ejecución del programa de búsqueda para arreglos desordenados y ordenados (compáralos).
2.- Que el programa cada vez que termine de ejecutarse, permita regresar al menú.
3.- Leer la práctica, trabajar en los apartados de Desarrollo y Evaluación; para posteriormente elaborar el Reporte de Práctica y por último entregarlo a su profesor en forma digital o impresa en la fecha acordada con él.

PRÁCTICA 4

MÉTODO DE BÚSQUEDA BINARIA

OBJETIVOS:
Conocer el funcionamiento de este método de búsqueda y programar su algoritmo en Lenguaje C.

INTRODUCCIÓN:
Para desarrollar la práctica, se va a hacer uso de la siguiente información:

Buscar significa recuperar datos previamente almacenados. La búsqueda es una actividad relevante en la vida diaria del ser humano. Se realizan tareas esenciales, como la búsqueda de números telefónicos en un directorio, ofertas laborales en un periódico o en sitios de internet, libros en una biblioteca, etc.

La operación de búsqueda puede llevarse a cabo sobre elementos ordenados y sobre elementos desordenados. En el primer caso, la búsqueda se facilita, y por lo tanto se ocupará menos tiempo que si se trabajara con elementos desordenados.

Los métodos de búsqueda pueden clasificarse en 2 tipos:

a) Búsqueda interna: Cuando todos los elementos se encuentran en la memoria principal de la computadora (almacenados en arreglos o listas ligadas).
b) Búsqueda externa: Cuando todos los elementos se encuentran en memoria secundaria (archivos almacenados en dispositivos como discos magnéticos y cintas).

Una búsqueda más eficiente puede hacerse sobre un arreglo ordenado. Para esto se puede utilizar el método de Búsqueda Binaria, el cual pertenece a la clasificación de Búsqueda interna.

La Búsqueda Binaria, compara si el valor buscado está en la mitad superior o inferior. En la que esté, subdivido nuevamente, y así sucesivamente hasta encontrar el valor. Por lo tanto, la velocidad de ejecución de este método, depende logarítmicamente del tamaño del arreglo.

En el método de búsqueda binaria, después de cada una de las comparaciones, elimina la mitad de los elementos en el arreglo bajo búsqueda. El algoritmo localiza el elemento medio del arreglo y lo compara con el valor buscado. Si son iguales, la clave de búsqueda ha sido encontrada y se regresa el subíndice del arreglo correspondiente a dicho elemento. Si no son iguales, el problema se reduce a buscar en una mitad del arreglo. Si el valor buscado es menor que el elemento medio del arreglo, se seguirá buscando en la primera parte del arreglo, de lo contrario se buscara en la segunda parte. Si el valor buscado no se encuentra en el subarreglo especificado (es la porción del arreglo original), el algoritmo se repite en una cuarta parte del arreglo original. La búsqueda continua, hasta que el valor buscado es igual al elemento del medio del subarreglo, o hasta que el subarreglo ha quedado reducido a un elemento diferente a el valor buscado (es decir, el valor buscado no ha sido encontrado).

En el peor escenario, utilizando la búsqueda binaria, la búsqueda de un arreglo de 1024 elementos sólo tomará 10 comparaciones. La división repetida de 1024 entre 2, da los 10 valores: 512, 256, 128, 64, 32, 16, 8, 4, 2 y 1. El numero 1024 (2^{10}) se divide entre 2 sólo diez veces para obtener el valor de 1. En el algoritmo de búsqueda binaria la división por 2 es equivalente a una comparación. Un arreglo de 1048576 (2^{20}) elementos toma un máximo de 20 comparaciones, para encontrar el valor buscado.

MATERIAL Y EQUIPO A UTILIZAR:

Para la realización de esta práctica son necesarios los siguientes componentes:

Equipo:

- Computadora.

Software:

- IDE CodeBlocks.

Material:

- Práctica digital o impresa.

DURACIÓN:

- 420 minutos.

DESARROLLO:

Reproducir en el IDE CodeBlocks, el código en el Lenguaje de programación C del método de búsqueda binaria de la Figura 4.1.

```
#include<stdio.h>
#define T 10
int main() {
int buscar,i,j,k,a[T];
```

```
printf("Ingresa 10 elementos en el arreglo:\n");
for(i=0;i<T;i++)
scanf("%d",&a[i]);
printf("Fin del llenado.\n");
printf("Ingresa un número a buscar: ");
scanf("%d",&buscar);
i=0;
j=T-1;
do {
k=(i+j)/2;
if(a[k]<=buscar)
i=k+1;
if(a[k]>=buscar)
j=k-1;
} while (i<=j);
printf("El elemento %d está en la posición %d\n",a[k],k);
return 0;
}
```

Figura 4.1 Programa en Lenguaje C del método de búsqueda binaria.

En la Figura 4.2, se muestra la ejecución del código anterior

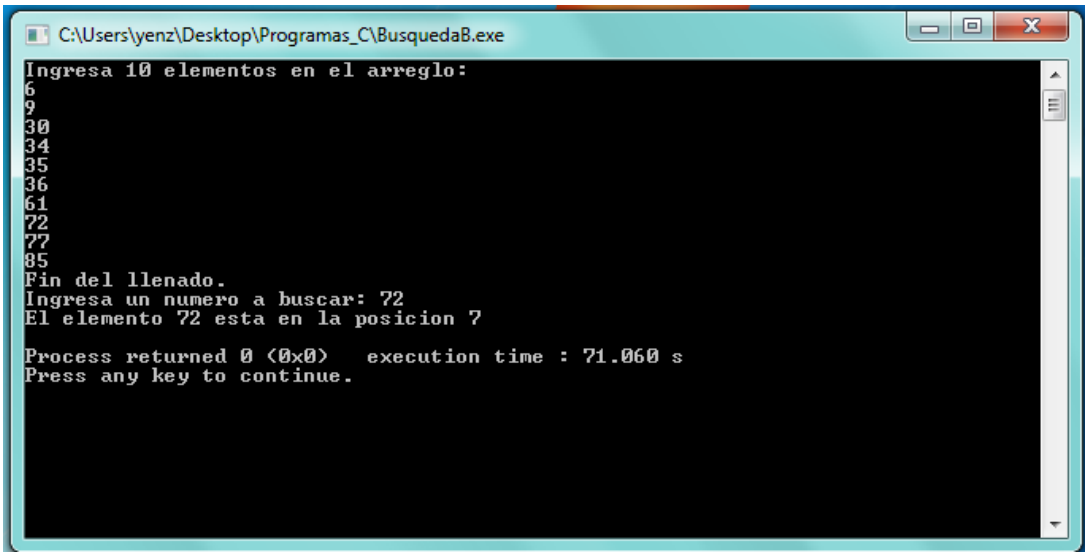

Figura 4.2 Ejecución del programa de búsqueda binaria

EVALUACIÓN:

1.- Modifica el programa de la Figura 4.1. Coloca en tres nuevas funciones los códigos para que el usuario ingrese la cantidad de números aleatorios que el programa va a generar, el método de ordenamiento burbuja y la búsqueda binaria junto con su número de comparaciones, y posteriormente mándalos llamar desde la función main(). La idea es que cuando ejecuten el programa, el usuario ingrese la cantidad de números aleatorios que el programa va a generar y visualizar en pantalla, después el programa los deberá ordenar y visualizar de forma ascendente y por último el programa deberá solicitar un número a buscar en el arreglo (lo debe mostrar en pantalla junto con su posición y número de comparaciones o en caso contrario debe mandar un mensaje de no encontrado en el arreglo).

2.- Código Morse. Quizás el más famoso de todos los sistemas de codificación, fue desarrollado por Samuel Morse en 1832, para uso en el sistema telegráfico. El código Morse asigna una serie de puntos y rayas a cada letra del alfabeto, a cada digito y a unos cuantos caracteres especiales (punto, coma, punto y coma, y dos puntos). En los sistemas orientados a sonido, el punto representa un sonido corto y la raya representa un sonido largo.

La separación entre palabras se indica por un espacio, o por la ausencia de un punto o de una raya. En un sistema orientado a sonidos, un espacio queda indicado por un corto periodo de tiempo, en el cual ningún sonido se transmite.

Escriba un programa que lea una frase en lengua española y que cifre la frase en código Morse. También escriba un programa que lea una frase en código Morse y la convierta en el equivalente en lengua española. Utiliza un espacio en blanco entre cada letra codificada Morse y dos espacios en blanco entre cada palabra codificada en Morse. Utilizar la versión internacional del código Morse.

Nota: Leer la práctica, trabajar en los apartados de Desarrollo y Evaluación; para posteriormente elaborar el Reporte de Práctica y por último entregarlo a su profesor en forma digital o impresa en la fecha acordada con él.

BIBLIOGRAFÍA

1. Deitel, P. J. y Deitel, H. M. (2014). *COMO PROGRAMAR EN C++*. 9a. Edición. México: Pearson Educación.
2. Domínguez, Edgar (2014). *Programación Estructurada: Raptor y Lenguaje C*. México: Alfaomega.
3. Guardati, Silvia y Cairó, Osvaldo (2006). *Estructuras de Datos*. 3a. Edición. México: McGraw-Hill.
4. Hsiang, L. Y. (2015). *Intermediate C Programming*. CRC Press.
5. Joyanes, A. L. (2006). *Programación en C+: algoritmos, estructuras de datos y objetos*. México: McGraw-Hill.
6. Kochan, Stephen G. (2014). *Programming in C*. 4/e. Addison-Wesley Professional.
7. Koffman, Elliot y Wolfgang, Paul (2008). *Estructura de datos con C++: Objetos, abstracciones y diseño*. McGraw-Hill.
8. Lee R., Teng. S., Chang R. y Tsai Y. (2007). *Introducción al diseño de algoritmos*. McGraw- Hill.
9. Márquez G., Osorio S. y Olvera N. (2011). *INTRODUCCIÓN A LA PROGRAMACIÓN ESTRUCTURADA EN C*. PEARSON.