Instituto Tecnológico de Tijuana

Proyecto final: metodos de ordenamiento

Materia:
Estructura de Datos

Profesor(a):
Ray Brunett Parra Galaviz

Alumno(a):
Jiménez Mayoral Gloria Alejandra – 17212146

Fecha:
30 de noviembre de 2018

Métodos de ordenamiento

El programa realizado se encargará de llenar un arreglo con 50 números aleatorios escogidos de entre el 1 al 999 y posteriormente, se podrán ordenar dichos números en orden ascendente mediante cuatro métodos: Bubblesort, Quicksort, Shellsort o Mergesort. Al final de cada operación se mostrará el tiempo que tomó cada método en ordenar los datos del arreglo.
El código del programa está escrito en lenguaje C++.

```
1   #include<iostream>
2   #include<stdlib.h>
3   #include<ctime>
4   #define TAM 50
5   using namespace std;
```

En las primeras líneas de código, se agregan las librerías para el uso de ciertas funciones. La librería iostream para ingresar(cout) y leer(cin) datos. stdlib.h para poder utilizar la función de generador de números aleatorios y la función de pausa y por último, la librería ctime para poder hacer uso de un timer. Después se definirá el tamaño del arreglo en una variable TAM que en este caso será de 50 espacios. Finalmente el using namespace para poder leer y meter datos.

```
7   void bubble(int [TAM]);
8   void quicksort(int [TAM], int, int);
9   void shellsort(int []);
10  void merge(int [], int, int, int);
11  void mergesort(int [], int, int);
```

Al iniciar el código se definen las funciones que se utilizarán y se llamarán de forma recursiva. Cada uno es para los métodos a utilizar; bubble, quick, shell y merge. Todas reciben como parámetros el arreglo de tamaño TAM y las variables de tipo entero a utilizar.

```
13  int main()
14  {
15      clock_t start;
16      double duration;
17      start = clock();
18
```

A partir de la línea 13 se inicia la función principal main().
A continuación, se declaran unas variables utilizando el el reloj de la computadora y éstas servirán para poder analizar el tiempo al final de cada método.

```
19      int array[TAM], op;
20      cout<<"Arreglo generado"<<endl;
21      srand(time(0));
22      for(int i = 0; i<TAM; i++)
23      {
24          array[i] = 1 + rand() % 999;
25          cout<<" "<<array[i];
26      }
```

Se declara el arreglo de tipo entero con el tamaño TAM de 50 y una variable “op” para guardar la opción del usuario al elegir uno de los métodos. Después se generarán los números del arreglo con un bucle for para irlo llenando. Estos se generan gracias al rand(), se escogerán números del 1 al 999 para llenarlo y se imprimen en pantalla. La línea 21 contiene una clase condición para evitar que el arreglo siempre sea el mismo cada que el código compile.

```
27  cout<<"\n\n\n";
28  cout<<"1.Ordenar por bubblesort. \n2.Ordenar por quicksort. \n3.Ordenar por shellsort. \n4.Ordenar por mergesort
29  cout<<"\n5.Salir. \n\nTu opcion es: ";
30  cin>>op;
```

Después de que se generó el arreglo, se presentará al usuario las opciones de ordenamiento y una opción 5 para salir del programa. Su respuesta se guardará en la variable op.

```
31      do
32      {
33          switch(op)
34          {
35              case 1:
36                  system("cls");
37                  bubble(array);
38                  cout<<"Elementos ordenados por el metodo bubblesort"<<endl;
39                  for(int i = 0; i<TAM; i++)
40                  {
41                  cout<<array[i]<<" ";
42                  }
43                  duration = (clock()-start)/(double) CLOCKS_PER_SEC;
44                  cout<<"\n\n\n";
45                  cout<<"La operacion tomo "<<duration<< "segundos"<<endl;
46                  system("pause");
47              break;
```

Las opciones se llevarán a cabo con una función switch y un ciclo do-while para terminar el programa cuando el usuario decida. En el case 1 irá implementado el método burbuja. Se llama a éste y posteriormente se imprimen los elementos ya ordenamos mediante un bucle for. Al final de dichas operaciones, se agrega la función duration para marcar el tiempo que tomó el método para realizar el ordenamiento de los datos y también se imprime en pantalla. Al final se agrega una pausa para que el programa no se cierre de inmediato y termina el case 1.

```
48              case 2:
49                  system("cls");
50                  quicksort(array, 0, 49);
51                  cout<<"Elementos ordenados por el metodo quicksort"<<endl;
52                  for(int i = 0; i<TAM; i++)
53                  {
54                  cout<<array[i]<<" ";
55                  }
56                  duration = (clock()-start)/(double) CLOCKS_PER_SEC;
57                  cout<<"\n\n\n";
58                  cout<<"La operacion tomo "<<duration<< "segundos"<<endl;
59                  system("pause");
60              break;
```

El case 2, llama al método quicksort recibiendo como parámetros el arreglo y las posiciones inicial y final de éste. Se imprimen los números ordenados con un bucle for y al final se implementa la función duration para mostrar el tiempo que el método tomó. Termina el case 2.

```
61              case 3:
62                  system("cls");
63                  shellsort(array);
64                  cout<<"Elementos ordenados por el metodo shellsort"<<endl;
65                  for(int i = 0; i<TAM; i++)
66                  {
67                  cout<<array[i]<<" ";
68                  }
69                  duration = (clock()-start)/(double) CLOCKS_PER_SEC;
70                  cout<<"\n\n\n";
71                  cout<<"La operacion tomo "<<duration<< "segundos"<<endl;
72                  system("pause");
73              break;
```

El case 3 llama al método shellsort el cual recibe el arreglo como parámetro.
Después de igual forma, imprime los elementos ordenados con un bucle for y la función duration para mostrar los segundos que demoró. Termina el case 3.

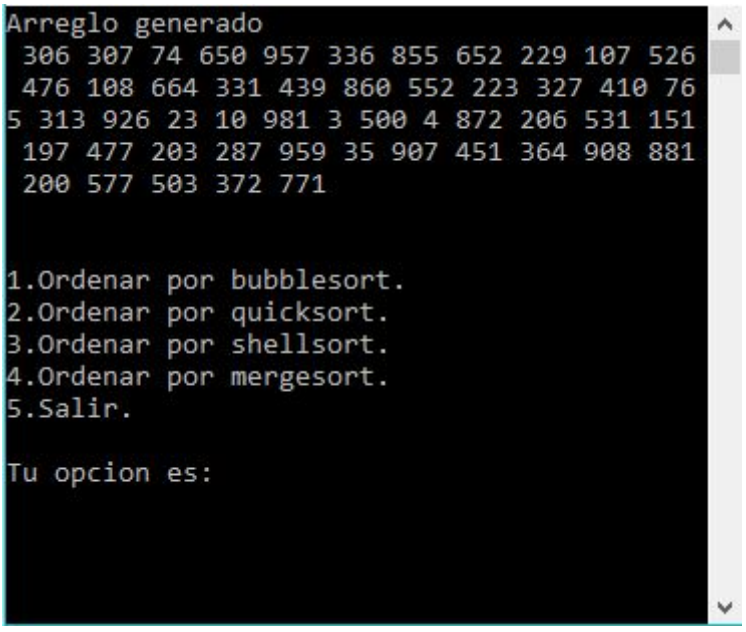
```
74              case 4:
75                  system("cls");
76                  cout<<"\n\nElementos ordenados por el metodo mergesort"<<endl;
77                  mergesort(array,0,49);
78                  for(int i=0;i<TAM;i++)
79                  {
80                  cout<<array[i]<<" ";
81                  }
82                  duration = (clock()-start)/(double) CLOCKS_PER_SEC;
83                  cout<<"\n\n\n";
84                  cout<<"La operacion tomo "<<duration<< "segundos"<<endl;
85                  system("pause");
86              break;
```

Finalmente, el case 4 llama al método mergesort que recibe como parámetros el arreglo y las posiciones inicial y final. Se imprimen los datos ordenados con el bucle for y la operación duration con los segundos que tomó la método. Fin del case 4.

```
87              case 5:
88              break;
89              default:
90              cout<<"Error!";
91              return 0;
92              break;
93          }
94      }while(op!=5);
95  }
```

El case 5 servirá simplemente para terminar el programa y salir del mismo.
Un caso default para cuando el usuario ingrese un número o dato fuera del rango de opciones, se le mostrará un mensaje de error y se termina el programa. Por último termina el ciclo do-while si la opción es ésta última de salir.

El resultado obtenido al iniciar el programa es el siguiente.



```
96   void bubble(int array[TAM])
97   {
98       int aux;
99       for(int i=0;i<TAM;i++)
100      {
101          for(int act=0;act<TAM;act++)
102          {
103              if(array[act]>array[act+1])
104              {
105                  aux=array[act];
106                  array[act]=array[act+1];
107                  array[act+1]=aux;
108              }
109          }
110      }
111  }
```

El primer método implementado es bubblesort; recibe un arreglo de enteros y dentro del mismo declaramos una variable de tipo entero llamada auxiliar.

El primer bucle for servirá para ir recorriendo el arreglo y el segundo para el intercambio de elementos. La condicional if se llevará a cabo de la siguiente manera: si el elemento actual es mayor que el actual + 1, es decir, mayor que el siguiente, se almacena ese valor mayor en la auxiliar siguiente para poder recorrerlo al final y así con todas las posiciones hasta que quede totalmente ordenado de forma ascendente.

```
Elementos ordenados por el metodo bubblesort
18 25 49 101 137 159 177 208 209 223 235 244 249 297 305 336 339
 376 433 438 450 476 484 490 502 543 583 595 617 630 646 681 683
 752 765 769 777 787 788 886 887 899 914 930 955 973 979 982 991
 997
```

```
112  void quicksort(int array[TAM], int first, int last)
113  {
114      int pivote, centro, i, j, aux;
115      i = first;
116      j = last;
117      centro = ((first + last)/2);
118      pivote = array[centro];
119      do
120      {
121          while(array[i]<pivote)
122          {
123              i++;
124          }
125          while(array[j]>pivote)
126          {
127              j--;
128          }
129          if(i<=j)
130          {
131              aux = array[i];
132              array[i] = array[j];
133              array[j] = aux;
134              i++;
135              j--;
136          }
137      }while(i<=j);
```

El siguiente método de ordenamiento es quicksort, éste recibe un arreglo y dos variables de tipo entero que serán las primera y segunda posición.

Se declaran las variables a utilizar: *i* tomará las posiciones de izquierda a derecha, es decir, de menor a mayor, y *j* tomará las posiciones de derecha a izquierda, es decir, de mayor a menor. La variable centro será para dividir el arreglo y encontrar el centro del mismo para tomar el valor de esa posición como pivote. El pivote es el valor en la posición del centro del arreglo. Mientras el valor *i* sea menor que el pivote, i aumenta para ir buscando el siguiente elemento de izquierda a derecha.Y mientras el valor de *j* sea mayor al pivote, j disminuye para ir cambiando de posición de derecha a izquierda. Para intercambiar los elementos al respectivo lado izquierdo o derecho dependiendo si son mayores o menores se utiliza una condicional if.La aux. guardará el valor de la posición *i*. Ahora *i* pasa ser igual a la posición *j* para intercambiar para avanzar en el arreglo. Por último se implementan las condiciones para ubicar la posición de los índices. La primera condición si el primer elemento first es menor que la posición de j la posición first será la posición 0 y la última será j. Y la otra condición dice que si la posición i es menor que last. la primera posición ahora será i 0 y la última será last.

```
Elementos ordenados por el metodo quicksort
101 104 109 132 140 146 175 199 223 225 268 313 332 334 338 365 371 380 398 411 4
61 469 497 511 543 578 583 595 621 639 661 751 754 765 802 848 852 871 872 877 89
3 915 923 924 927 940 958 963 970 982
```

El penúltimo método es shellsort y funciona de la siguiente manera: éste recibe únicamente el arreglo de enteros. Se declaran las variables a utilizar de tipo entero y una variable de tipo booleana. Los saltos que se harán a lo largo del arreglo se inicializan con el tamaño TAM del arreglo. Mientras que el intervalo de saltos sea mayor a uno se llevan a cabo las instrucciones. Esto porque el último salto será de un espacio y el recorrido habrá concluido. El intervalo de saltos estará definido dividiendo entre 2 la variable saltos que anteriormente tenía el valor del tamaño del arreglo. Esto define cuántos espacios habrá entre cada comparación. Mientras la variable booleana sea true, se ejecutan las instrucciones.

```
148  void shellsort(int array[])
149  {
150      int saltos, i, aux;
151      bool b;
152      saltos = TAM;
153      while(saltos>1)
154      {
155          saltos=(saltos/2);
156          b = true;
157          while(b==true)
158          {
159              b=false;
160              i = 0;
161              while((i+saltos)<=TAM)
162              {
163                  if(array[i]>array[i+saltos])
164                  {
165                      aux = array[i];
166                      array[i] = array[i+saltos];
167                      array[i+saltos] = aux;
168                      b = true;
169                  }
170                  i++;
171              }
172          }
173      }
174  }
```

Dentro de otro ciclo while, en el cual se comparan el elemento a distancia con el tamaño del arreglo se llevará a cabo una comparación.

Si el primer elemento del arreglo a comparar es mayor que el elemento a la distancia del salto, se procede a intercambiar de lado dicho valor.

```
Elementos ordenados por el metodo shellsort
55 62 96 128 134 164 211 212 253 256 263 278 295 303 319 322 372 378
 383 414 425 484 494 501 518 524 553 578 605 650 697 703 754 761 769
 782 802 816 817 841 855 899 900 909 921 941 967 990 998 999
```

```
175  void merge(int array[], int p, int q, int r)
176  {
177      int n1 = q - p + 1;
178      int n2 = r - q;
179      int k = p;
180      int i = 0;
181      int j = 0;
182      int L[n1], R[n2];
183      for(int i = 0; i < n1; i++)
184      {
185          L[i] = array[p+i];
186      }
187      for(int j = 0; j < n2; j++)
188      {
189          R[j] = array[q+1+j];
190      }
```

El último método es mergesort. Este se divide en dos métodos, el primero, merge, se encargará de acomodar los elementos en dos subarreglos.

Se declaran dos variables para dividir en mitades el arreglo n1 y n2, a su vez, éstas servirán para almacenar los elementos de dos arreglos temporales que se crean a partir de la división del arreglo principal. L para el lado izquierdo y R para el lado derecho. Estos subarreglos se llenan gracias a un bucle for para cada uno. El índice i será para el lado izquierdo y el índice j para el lado derecho.

Para acomodar los valores en el arreglo principal se lleva a cabo un ciclo while y una condicional if. Si el elemento del lado izquierdo es menor que el del derecho, se copia el valor de i en la posición k del arreglo final. Y si no, r se considera como el menor y j incrementa y se posiciona en k.

```
191      while(i < n1 and j < n2)
192      {
193          if(L[i]<=R[j])
194          {
195              array[k] = L[i];
196              i++;
197          }else
198          {
199              array[k] = R[j];
200              j++;
201          }
202          k++;
203      }
```

```
217  void mergesort(int array[], int p, int r)
218  {
219      if(p<r)
220      {
221          int q = ((p+r)/2);
222          mergesort(array,p,q);
223          mergesort(array,q+1,r);
224          merge(array,p,q,r);
225      }
226  }
```

La segunda parte del método, se encargará de dividir el arreglo en dos y posicionar los elementos en cada lado llamando a los métodos de manera recursiva. Primero se arregla la primera subsecuencia. Después se arregla la segunda subsecuencia. Por último se unen esas dos subsecuencias mediante el método merge.

```
Elementos ordenados por el metodo mergesort
28 59 83 116 118 133 143 196 198 234 236 282 298 315 345 350 375 3
81 413 420 432 449 466 481 482 483 532 534 538 540 565 567 586 586
 593 600 605 628 669 700 717 738 740 791 844 862 891 912 919 927
```

La siguiente tabla muestra los resultados de las operaciones de los métodos probado 30 veces cada uno con diferentes arreglos. El tiempo está dado en segundos.

Método	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
Bubblesort	1.61 s	1.40 s	1.83 s	1.55 s	1.16 s	0.89 s	1.01 s	1.36 s	1.04 s	1.11 s	1.25 s	1.74 s	1.47 s	1.07 s	1.03 s
Quicksort	2.32 s	1.71 s	1.92 s	1.97 s	1.56 s	1.34 s	1.51 s	1.40 s	1.42 s	1.94 s	1.59 s	1.72 s	1.67 s	1.52 s	1.69 s
Shellsort	2.93 s	2.54 s	2.47 s	2.07 s	2.79 s	1.27 s	1.49 s	3.73 s	1.77 s	1.20 s	1.22 s	1.51 s	1.46 s	1.99 s	1.43 s
Mergesort	1.96 s	1.46 s	1.43 s	0.82 s	1.88 s	1.87 s	2.52 s	2.05 s	1.12 s	1.13 s	2.12 s	0.80 s	1.14 s	1.33 s	2.03 s

Método	P16	P17	P18	P19	P20	P21	P22	P23	P24	P25	P26	P27	P28	P29	P30
Bubblesort	1.23 s	0.89 s	1.02 s	1.01 s	1.29 s	1.55 s	1.06 s	0.899 s	0.76 s	1.48 s	1.07 s	0.89 s	0.87 s	0.78 s	1.31 s
Quicksort	1.99 s	1.27 s	1.72 s	1.27 s	1.26 s	1.11 s	1.47 s	1.82 s	1.58 s	1.11 s	1.15 s	1.21 s	1.33 s	1.27 s	1.54 s
Shellsort	2.27 s	2.22 s	1.98 s	1.80 s	1.82 s	1.09 s	1.84 s	2.27 s	2.19 s	2.07 s	1.38 s	1.22 s	1.53 s	1.85 s	2.55 s
Mergesort	1.78 s	1.25 s	1.32 s	1.72 s	1.68 s	1.95 s	0.78 s	0.75 s	0.88 s	1.18 s	1.15 s	0.82 s	1.92 s	0.63 s	0.71 s