UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INFORMÁTICA

**LABORATORIO N°1**

**Programación en C: Arreglos, Archivos, Iteración,**

**Recursión, Algoritmos de Ordenamiento.**

**POR: RODRIGO CERDA R.**

**PROFESOR: IRENE ZUCCAR PARRINI**

**AYUDANTE: AUGUSTO TOLEDO**

**Santiago, 16 de Septiembre 2016**

TABLA DE CONTENIDOS

* Introducción 3
* Estructuras 3
* Algoritmos utilizados 4
  + Inserción 4
  + Burbuja 5
  + Quicksort 5
  + Mergesort 7
* Experimentos 9
  + Arreglos especiales 9
  + Otros arreglos 11
* Evaluación de resultados 12
  + Arreglos especiales 12
  + Otros arreglos 14
* Conclusiones 21
* Manual de usuario 22
* Referencias bibliográficas 24

INTRODUCCION

Un conjunto de instrucciones que resuelva un problema puntual, se denomina algoritmo. Pueden existir variados algoritmos que realicen la misma tarea, por lo que resulta importante contar con herramientas que logren esclarecer que camino es el más eficiente.

En el presente informe se desarrolla el ordenamiento de distintos arreglos numéricos generados dentro del mismo programa, utilizando cuatro algoritmos distintos, en el lenguaje de programación “C” y el entorno “CODEBLOCKS”.

En la utilización de cada algoritmo se monitorearán variables como el tiempo de ejecución, la cantidad de comparaciones entre elementos y la cantidad de intercambios entre elementos, con el fin de contar con datos concretos que permitan encontrar el mejor algoritmo en cada caso.

ESTRUCTURAS

Se definen dos estructuras, una contenida dentro de otra, con el propósito de almacenar en ellas el arreglo ordenado, más los datos necesarios para analizar el rendimiento del algoritmo.

Estructura Datos: Contiene tres elementos:

* double contadorTiempo (almacena el tiempo que le toma al algoritmo en ordenar el arreglo)
* int contadorComparaciones (almacena la cantidad de comparaciones entre elementos)
* int contadorIntercambios (almacena la cantidad de intercambios entre elementos)

Estructura Resultados: Contiene tres elementos:

* int \*arreglo (Puntero hacia el arreglo)
* Datos datos (Estructura anterior, contenedora de la información de monitoreo)
* int size (Cantidad de elementos dentro del arreglo)

Cada algoritmo de ordenamiento solicita como parámetro un puntero a variable tipo “Resultados”, por lo que se emplean cuatro variables “Resultados” distintas para almacenar la información necesaria.

ALGORITMOS UTILIZADOS

* Algoritmo de Inserción: De manera iterativa, compara el elemento actual con el anterior, si el primero es menor que el segundo, los intercambia.

void Insercion(Resultados \*r){

r->datos.contadorComparaciones=0;

r->datos.contadorIntercambios=0;

r->datos.contadorTiempo=0;

clock\_t tiempoini, tiempofin;

tiempoini = clock();

int a;

int i,j;

for(i=1;i < r->size; i++){

j=i;

r->datos.contadorComparaciones++;

while( (r->arreglo[j] < r->arreglo[j-1]) & j>0){

intercambio(&r->arreglo[j],&r->arreglo[j-1]);

r->datos.contadorIntercambios++;

j--;

}

}

tiempofin=clock();

r->datos.contadorTiempo=(double)tiempofin-tiempoini;

}

* Algoritmo de Burbuja:

void Burbuja(Resultados \*r){

r->datos.contadorComparaciones=0;

r->datos.contadorIntercambios=0;

r->datos.contadorTiempo=0;

clock\_t tiempoini, tiempofin;

tiempoini= clock();

int i,j;

for (i=0;i<r->size;i++){

for(j=r->size-1;j>=i+1;j--){

r->datos.contadorComparaciones++;

if(r->arreglo[j]<r->arreglo[j-1]){

r->datos.contadorIntercambios++;

intercambio(&r->arreglo[j],&r->arreglo[j-1]);

}

}

}

tiempofin=clock();

r->datos.contadorTiempo=(double)tiempofin-tiempoini;

}

* Algoritmo Quicksort:

Para el usuario, lo importante es que la función reciba el arreglo que se desea ordenar, por lo que se utiliza una función auxiliar que esconda del usuario otros parámetros necesarios, como el primer y último elemento.

void Quicksort(Resultados \*r){

r->datos.contadorComparaciones=0;

r->datos.contadorIntercambios=0;

r->datos.contadorTiempo=0;

clock\_t tiempoini, tiempofin;

tiempoini= clock();

QuicksortAux(0, r->size-1, r);

tiempofin=clock();

r->datos.contadorTiempo=(double)tiempofin-tiempoini;

}

void QuicksortAux(int prim, int ult, Resultados \*r){

int corte;

if (ult - prim>=1){

corte= particion(prim,ult,r);

QuicksortAux(prim,corte - 1,r);

QuicksortAux(corte +1, ult,r);

}

}

int particion(int prim, int ult, Resultados \*r){

int pivote, corte, k;

pivote = r->arreglo[prim];

corte=prim;

for (k=prim +1;k<=ult;k++){

r->datos.contadorComparaciones++;

if (r->arreglo[k]< pivote){

corte=corte+1;

intercambio(&r->arreglo[corte],&r->arreglo[k]);

r->datos.contadorIntercambios++;

}

}

r->datos.contadorIntercambios++;

intercambio(&r->arreglo[prim],&r->arreglo[corte]);

return corte;

}

* Algoritmo Mergesort:

Para el usuario, lo importante es que la función reciba el arreglo que se desea ordenar, por lo que se utiliza una función auxiliar que esconda del usuario otros parámetros necesarios, como el primer y último elemento.

void Mergesort(Resultados \*r){

r->datos.contadorComparaciones=0;

r->datos.contadorIntercambios=0;

r->datos.contadorTiempo=0;

clock\_t tiempoini, tiempofin;

tiempoini= clock();

MergesortAux(r, 0, r->size-1);

tiempofin=clock();

r->datos.contadorTiempo=(double)tiempofin-tiempoini;

}

void MergesortAux(Resultados \*r,int ini, int fi){

int med;

if(ini < fi){

med=(ini+fi) / 2;

MergesortAux(r,ini,med);

MergesortAux(r,(med+1),fi);

Merge(r,ini,med,fi);

}

}

void Merge(Resultados \*r,int inicio, int medio, int fin){

int h,i,j,k,l;

int arrayAux[fin-inicio+1];

h = 0;

i = inicio;

j = medio + 1;

while ( (i<=medio)&&(j <= fin) ) {

r->datos.contadorComparaciones++;

if (r->arreglo[i]<r->arreglo[j]){

arrayAux[h]=r->arreglo[i];

i++;

}else{

r->datos.contadorIntercambios++;

arrayAux[h]=r->arreglo[j];

j++;

}

h++;

}

if (i> medio){

for (k=j;k<=fin;k++){

arrayAux[h]=r->arreglo[k];

h++;

}

}else{

for(k=i;k<=medio;k++){

arrayAux[h]=r->arreglo[k];

h++;

}

}

l=0;

for (k=inicio;k<=fin;k++){

r->arreglo[k]=arrayAux[l];

l++;

}

}

EXPERIMENTOS

Para realizar las mediciones, se generan como muestra veinte arreglos, de los cuales, cuatro son considerados especiales, por tener un orden definido previamente.

* Arreglo ascendente: Arreglo ordenado en forma creciente. La función que lo genera parte por un numero al azar,y los elementos avanzan desde el sumado a otro número al azar entre 1 y 40:

void arrayAscendente(int \*array, int c){

int i;

array[0]=rand();

for(i=1;i<c;i++){

array[i]=array[i-1]+rand() %40 +1;

}

}

* Arreglo Descendente: Arreglo ordenado en forma decreciente. La función que lo genera tiene un procedimiento similar a la función anterior, partiendo por un numero al azar, y avanza restando otro número al azar.

void arrayDescendente(int \*array,int c){

int i;

array[0]=rand();

for(i=1;i<c;i++){

array[i]=array[i-1]-rand() % 30 - 1;

}

}

* Arreglo Intercalado: Los elementos pares están ordenados de forma creciente, y los impares de manera decreciente. Para esta función, se toma como elemento inicial el número 100, y después de él se va sumando o restando 2 según sea par o impar.

void arrayIntercalado(int \*array,int c){

int i;

array[0]=100;

array[1]=array[0]-rand();

for(i=2;i<c;i++){

if (i%2==0){

array[i]=array[i-2]+2;

}else{

array[i]=array[i-2]-2;

}

}

}

* Arreglo Mitades: La primera mitad del arreglo contiene elementos menores y desordenados, la segunda mitad contiene elementos mayores y desordenados.

void arrayMitades(int \*array,int c){

int i,o, numeroInicial, acum,cont;

numeroInicial= rand();

for (i=0;i<c;i++){

cont=0;

if(i<c/2){

acum=numeroInicial-rand();

for (o=0;o<=i;o++){

if (array[o]==acum) //para revisar que el número no este repetido

cont=1;

}

if (cont!=1){

array[i]= acum;

}else{

i--;

}

}else{

acum=numeroInicial+rand();

for (o=(c/2);o<=i;o++){

if (array[o]==acum) //para revisar que el número no este repetido

cont=1;

}

if (cont!=1){

array[i]= acum;

}else{

i--;

}

}

}

}

Además de los cuatro arreglos especiales mencionados anteriormente, se evalúan dieciséis arreglos más, con números aleatorios. La función que los genera elige un numero al azar, se elige su signo positivo o negativo con otro número al azar, y se revisa que no exista en el arreglo y así guardarlo.

void arraySimple(int \*array,int c){

int i,o, numeroInicial, acum,cont,signo;

for (i=0; i < c; i++){

cont=0;

numeroInicial= rand();

signo= rand()%2;

if (signo==1){

numeroInicial=numeroInicial\*(-1);

}

for (o=0;o<=i;o++){

if(array[o]==numeroInicial)

cont=1;

}

if (cont!=1){

array[i]=numeroInicial;

}else{

i--;

}

}

}

EVALUACION DE RESULTADOS

* ARREGLOS ESPECIALES:

***Tabla 1.*** *Valores de monitoreo para algoritmos de inserción y burbuja en cada arreglo especial.*



***Tabla 2.*** *Valores de monitoreo para algoritmos quicksort y mergesort en cada arreglo especial.*



***Figura 1.*** *Gráfico de tiempo de ejecución para cada algoritmo en cada arreglo especial.*

***Figura 2.*** *Gráfico de cantidad de comparaciones entre elementos del arreglo para cada*

*Algoritmo en cada arreglo especial.*

***Figura 3.*** *Gráfico de cantidad de intercambios entre elementos del arreglo para cada*

*Algoritmo en cada arreglo especial.*

* OTROS ARREGLOS:

***Tabla 3.*** *Valores de monitoreo para algoritmo de inserción y burbuja en el resto de arreglos.*



***Tabla 4.*** *Valores de monitoreo para algoritmo quicksort y mergesort en el resto de arreglos.*



***Figura 4.*** *Gráfico de tiempo de ejecución para cada algoritmo en arreglos simples del 1 al 4.*

***Figura 5.*** *Gráfico de tiempo de ejecución para cada algoritmo en arreglos simples del 5 al 8.*

***Figura 6.*** *Gráfico de tiempo de ejecución para cada algoritmo en arreglos simples del 9 al 12.*

***Figura 7.*** *Gráfico de tiempo de ejecución para cada algoritmo en arreglos simples del 13 al 16.*

***Figura 8.*** *Gráfico cantidad de comparaciones para cada algoritmo en arreglos simples del 1 al 4.*

***Figura 9.*** *Gráfico cantidad de comparaciones para cada algoritmo en arreglos simples del 5 al 8.*

***Figura 10.*** *Gráfico cantidad de comparaciones para cada algoritmo en arreglos simples del 9 al 12.*

***Figura 11.*** *Gráfico cantidad de comparaciones para cada algoritmo en arreglos simples del 13 al 16.*

***Figura 12.*** *Gráfico cantidad de intercambios para cada algoritmo en arreglos simples del 1 al 4.*

***Figura 13.*** *Gráfico cantidad de intercambios para cada algoritmo en arreglos simples del 5 al 8.*

***Figura 14.*** *Gráfico cantidad de intercambios para cada algoritmo en arreglos simples del 9 al 12.*

***Figura 15.*** *Gráfico cantidad de intercambios para cada algoritmo en arreglos simples del 13 al 16.*

CONCLUSIONES

Del análisis de los datos obtenidos en el ítem anterior, se desprenden las siguientes conclusiones:

* Arreglo ascendente: (el mejor caso)

Para este caso, el algoritmo de inserción realizo la tarea en menor tiempo. Lo cual es comprensible, ya que, al estar todo ordenado, el algoritmo solo confirmo que el elemento n, sea menor a n+1, es decir, no realizó intercambio alguno entre elementos, por lo que su ejecución fue rápida.

En este punto, el algoritmo burbuja fue el más lento, ya que recorre el arreglo en oportunidades.

Burbuja y quicksort, son los algoritmos que realizan mayor cantidad de comparaciones. Siendo quicksort el que menos hizo ( n comparaciones).

El único algoritmo que realizó intercambios fue el de quicksort. Por lo que su rendimiento en el mejor de los casos es deficiente.

* Arreglo descendente: (el peor caso)

En este caso, el algoritmo que tomo menos tiempo en realizar el ordenamiento es mergesort, siendo el más óptimo en el peor de los casos a la hora de ordenar.

Inserción y burbuja tienen un rendimiento del tiempo deficiente en el peor de los casos, debido a que por su manera de operar deben comparar e intercambiar elementos miles de veces más que el resto de algoritmos.

En el caso de las comparaciones, mergesort realiza cerca de 60.000, contra las casi 50.000.000 del resto de los algoritmos.

Para los intercambios, también es mergesort quien gana en eficiencia, ya que realiza alrededor de 60.000, contra los casi 50.000.000 de inserción y burbuja, y 25.000.000 de quicksort.

* Arreglo intercalado:

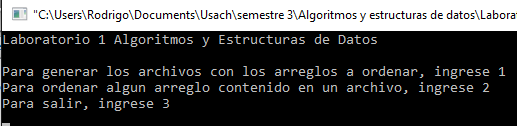
Para este caso, es mergesort el algoritmo que nuevamente resulta ser más eficiente que el resto, siendo el de burbuja el más ineficiente.

* Arreglo mitades:

En este caso gana también mergesort en eficiencia, a la vez que el menos eficiente resulta ser burbuja nuevamente. Con la particularidad de que en esta oportunidad los algoritmos de inserción y quicksort resultan bastante eficientes.

En conclusión, en caso de tener que elegir un algoritmo de ordenamiento, se opta por elegir mergesort. Ya que resulta veloz en su ejecución, con menor cantidad de comparaciones e intercambios que el resto.

MANUAL DE USUARIO:



***Figura 16.****Menú principal de la aplicación.*

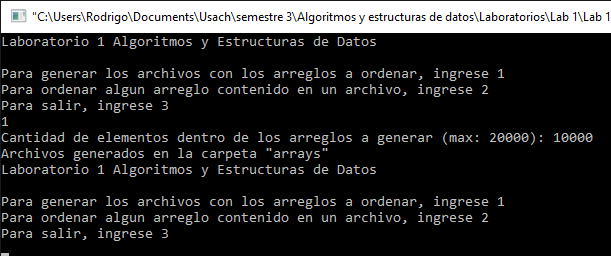
El programa se presenta con un menú principal, con tres alternativas enumeradas:

1. Generar arreglos: Opción para generar los 4 arreglos especiales más los 16 simples solicitados en el enunciado.

Los arreglos generados son almacenados en la carpeta “arrays” dentro de la carpeta principal, con nombres estandarizados, por lo tanto, cada vez que se solicite la creación de arreglos nuevos, los arreglos anteriores se perderán.

El programa permite generar arreglos de diferentes tamaños, siendo 20.000 el límite impuesto en la programación.

El laboratorio ha sido entregado con los archivos ya generados, ya que son los que se utilizaron para las pruebas de ordenamiento.



***Figura 17.*** *Vista del menú de generación de arreglos de prueba.*

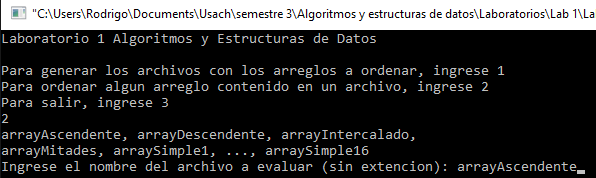
1. Ordenar arreglo contenido en un archivo: Opción para realizar las pruebas de ordenamiento en algún arreglo contenido en un archivo.

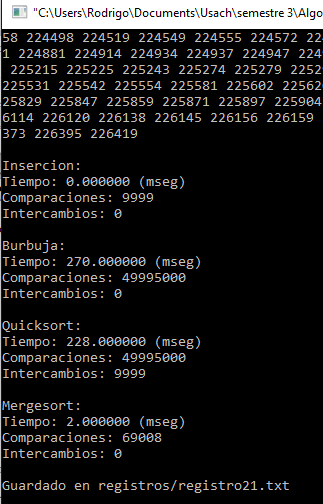
El usuario deberá ingresar el nombre del archivo (sin extensión) desde donde se desea leer el arreglo a ordenar.

En caso de ingresar datos erróneos, vuelve a menú principal.

En caso de encontrar el archivo solicitado, ejecuta los algoritmos de ordenamiento, mostrando los resultados por pantalla, y almacenándolos en la carpeta “registros” dentro de la carpeta principal.

Cada prueba que se haga, generará un registro único, que se diferenciará por un id numérico en su nombre.





***Figura 18 y 19.*** *Vista de menú de ordenamiento y vista de resultados de ordenamiento.*

1. Salir: Opción para terminar el programa.

Consideraciones:

Si bien se logra resguardar el programa ante el posible ingreso de datos erróneos por parte del usuario, además de resguardos en caso de no existir las carpetas contenedoras, o el archivo contador de registros, resulta fuera del alcance de este laboratorio el estudio de casos en los que los archivos a leer han sido adulterados.

El laboratorio es entregado con los arreglos en sus archivos contenedores, sin embargo, se puede solicitar al programa generar nuevos arreglos, que reemplacen los entregados.

El laboratorio se entrega con los reportes para cada arreglo, utilizados en la evaluación de resultados del presente informe.

REFERENCIAS BIBLIOGRAFICAS:

* *Stack Overflow*. (s.f.). Obtenido de http://stackoverflow.com/tags/c/info
* Zuccar, I. (2016). Apuntes Algoritmos y estructuras de Datos. Universidad de Santiago