Carátula para entrega de prácticas	
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Tista Garcia Edgar

Asignatura: Estructura de Datos y Algoritmos II

Grupo: 5

No de Práctica(s): 1

Integrante(s): Calzada Martinez Jonathan Omar

No. de Equipo de cómputo empleado

Semestre: 2019-2

Fecha de entrega: 7/02/2019

Obervaciones:

CALIFICACIÓN: _____

Introduccion:

Un algoritmo de ordenamiento es utilizado como su nombre lo dice a ordenar información, se puede hacer de mayor a menor o menor a mayor, existen diversas maneras para poder realizar los ordenamientos, pero dependiendo la cantidad de información a ordenar, así como hardware es como se elige al que mejor nos conviene ya sea por la cantidad de tiempo o la cantidad de recursos utilizados de la computadora.

Para esta practica estudiaremos el algoritmo de la Burbuja (BubbleSor) y el algoritmo quickSor de orden rápido implementado distintas cantidades de información.

Desarrollo:

En la librerìa de ordenamiento se encuentran las funciones de ordenamientos quickSort y bublleSort, en ellas se encuentra el algoritmo correspondiente para ser llamado en la funcion principal segun se requiera o se necesite.

```
void bubbleSort(int a[], int size){
    int intercambios=0;
    int i,j,n;
    n= size;
    for(i=n-1;i>0;i--){
        int cambios=0;
        for(j=0; j<i; j++){
            if(a[j]>a[j+1]){
                swap(&a[j], &a[j+1]);
                cambios = 1;
            }
            printArray(a,size);
        }
        if(cambios==0)
            break;

        printArray(a,size);
        printf("Fin Iteracion");
    }
}
```

También en la biblioteca de ordenamiento encontramos una función llamada partición donde agarra el valor que se va a utilizar de pivote para empezar a hacer el ordenamiento.

Biblioteca de UTILIDADES
En la biblioteca de utilidades encontramos funciones que utilizan apuntadores para poder hacer los cambios de información en las variables a la hora de hacer los intercambios en los ordenamientos.

Así también imprime el los resultados obtenido de los ordenamientos ya sea utilizando el algoritmo de Burbuja o quickSor

```c
#include <stdio.h>
void swap(int* a, int* b){
    int t = *a;
    *a = *b;
    *b = t;
}

void printArray(int arr[],int size){
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
```

En la función principal encontramos un arreglo en donde lo inicializamos con valores arbitrarios dados por el profesor.

```c
int main () {
    int arr[] = {30,60,23,1,56,77,21,11,78,40};
    int size = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr,size);
    //quickSort(arr,0,size-1);
    printArray(arr,size);

    return 0;
```

Utilizando el código con la función Burbuja obtenernos estos resultados

```
30 60 23 1 56 77 21 11 78 40
30 23 60 1 56 77 21 11 78 40
30 23 1 60 56 77 21 11 78 40
30 23 1 56 60 77 21 11 78 40
30 23 1 56 60 77 21 11 78 40
30 23 1 56 60 21 77 11 78 40
30 23 1 56 60 21 11 77 78 40
30 23 1 56 60 21 11 77 78 40
30 23 1 56 60 21 11 77 40 78
30 23 1 56 60 21 11 77 40 78
Fin Iteracion23 30 1 56 60 21 11 77 40 78
23 1 30 56 60 21 11 77 40 78
23 1 30 56 60 21 11 77 40 78
23 1 30 56 60 21 11 77 40 78
23 1 30 56 21 60 11 77 40 78
23 1 30 56 21 11 60 77 40 78
23 1 30 56 21 11 60 77 40 78
23 1 30 56 21 11 60 40 77 78
```

Utilizando la función quickSort obtenernos estos resultados

```
1  //
2  //  main.c
3  //  prsactica1
4  //
5  //  Created by Alumno on 2/5/19.
6  //  Copyright © 2019 alumno. All rights reserved.
7  //
8
9  #include <stdio.h>
10 #include "ordenamientos.h"
11
12
13 int main () {
14     int arr[] = {30,60,23,1,56,77,21,11,78,40};
15     int size = sizeof(arr)/sizeof(arr[0]);
16     //bubbleSort(arr,size);
```

```
Program ended with exit code: 0Pivote:40
Sub array :  30 23 1 21 11
Sub array :  60 56 78 77
Pivote:11
Sub array :  1
Sub array :  30 21 23
Pivote:23
Sub array :  21
Sub array :  30
Pivote:77
Sub array :  60 56
Sub array :  78
Pivote:56
Sub array :
Sub array :  60
1 11 21 23 30 40 56 60 77 78
Program ended with exit code: 0
```

C) Agrega las instrucciones necesarias para contabilizar las comparaciones y los intercambios realizados

Para el algoritmo BubbleSort Se le agrego un variable llamada como para las contabilizaciones y una variable cambio que contabiliza los cambios hechos, colocados de la siguiente manera.

Esto se hizo con el arreglo proporcionado por el profesor.

```
void bubbleSort(int a[], int size){
    int intercambios=0;
    int comp=0,cambi=0;
    int i,j,n;
    n= size;
    for(i=n-1;i>0;i--){
        int cambios=0;
        for(j=0; j<i; j++){
        comp=comp+1;
            if(a[j]>a[j+1]){
                swap(&a[j], &a[j+1]);
                cambios = 1;
        cambi++;
            }
            printArray(a,size);
        }
        if(cambios==0)
            break;
```

Como resultado:

```
          Fin Iteracion
1 11 21 23 30 40 56 60 77 78
1 11 21 23 30 40 56 60 77 78
1 11 21 23 30 40 56 60 77 78

 comparaciones 42,  cambios  21
        1 11 21 23 30 40 56 60 77 78
```

En total hubo 42 comparaciones y 21 cambios.

Para el algoritmo de quickSort

Insertar un contador en el algoritmo de quickSort no se pudo, en esta parte tengo que estudiar más ya que no se logro encontrar la manera de insertar el contador y así mismo que imprima la cantidad de comparaciones y cambios.

C.2) Realiza pruebas con arreglos de diferentes tamaños de entrada 10,50,100,500 elementos y verifica el crecimiento del numero de operaciones.

Para poder realizar el ejercicio con números aleatorios se utilizó la función srand para poner el rango de valores.

```
srand (time(NULL));
for (i=0; i<=10; i++) {
    arr[i] = rand()%50;
}


    repet=1;
```

Después se implemento un while para detectar y cambiar un numero repetido. Se hizo de esta manera:

```
    repet=1;
    while (repet==1){
        repet=0;
    for (i=0; i<=9; i++) {
        for (j=i+1; j<=9; j++) {

            if (arr[i] == arr[j] && arr[i] !=29) { //Repetición
                printf("Hay repeticion. Cambio de %d por %d \n", arr[i], arr[i]+1);
                arr[i] = arr[i] +1;
                repet = 1;

            }
            if (arr[i] == arr[j] && arr[i] ==29) { //Repetición
                arr[i] = arr[i] -rand()%28+1;
                printf("Hay repeticion de 29. Cambio de 29 por %d \n", arr[i]);
                repet = 1;
            }
        }
    }
}
```

Realizando la prueba con 10 números para burbuja:

```
srand (time(NULL));
for (i=0; i<=10; i++) {
    arr[i] = rand()%29;
}


    repet=1;
    while (repet==1){
        repet=0;
    for (i=0; i<=10; i++) {
        for (j=i+1; j<=9; j++) {
```

```
5 4 6 7 12 13 14 18 19 24
5 4 6 7 12 13 14 18 19 24
        Fin Iteracion
4 5 6 7 12 13 14 18 19 24
4 5 6 7 12 13 14 18 19 24
        Fin Iteracion

comparaciones 45,  cambios  23
       4 5 6 7 12 13 14 18 19 24
```

Realizando la prueba con 50 números para burbuja:

```
int arr[50]; //Indices de 0 a 9, re
int i, j;

srand (time(NULL));
for (i=0; i<=50; i++) {
    arr[i] = rand()%90;
}
```

```
5 76 77 79 82 85 86 87 88

 comparaciones 1224,  cambios  646
        0 1 2 9 11 12 22 23 24 25 26
 71 73 75 76 77 79 82 85 86 87 88

Process returned 0 (0x0)   execution
```

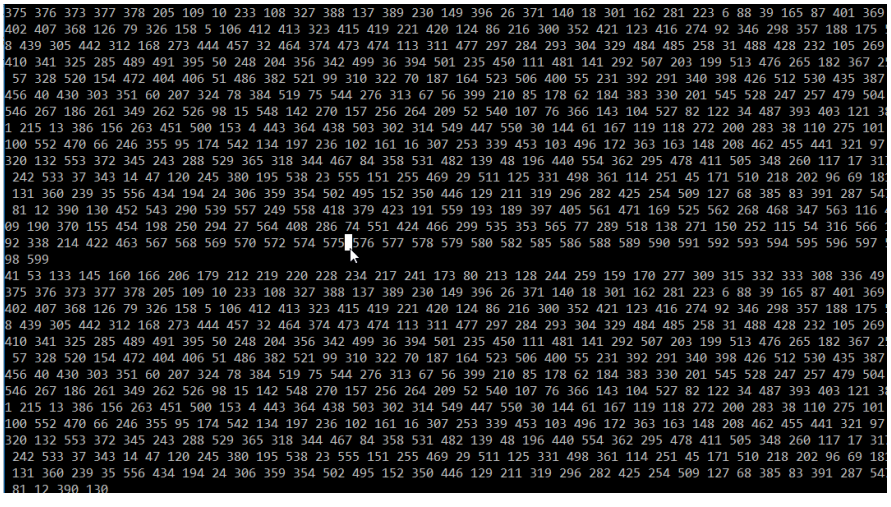
Realizando la prueba con 100 números para burbuja:

```
int main (){
int repet=0;
int arr[100]; //Indices de 0 a 9, re
int i, j;

srand (time(NULL));
for (i=0; i<=100; i++) {
    arr[i] = rand()%119;
}


    repet=1;
    while (repet==1){
```

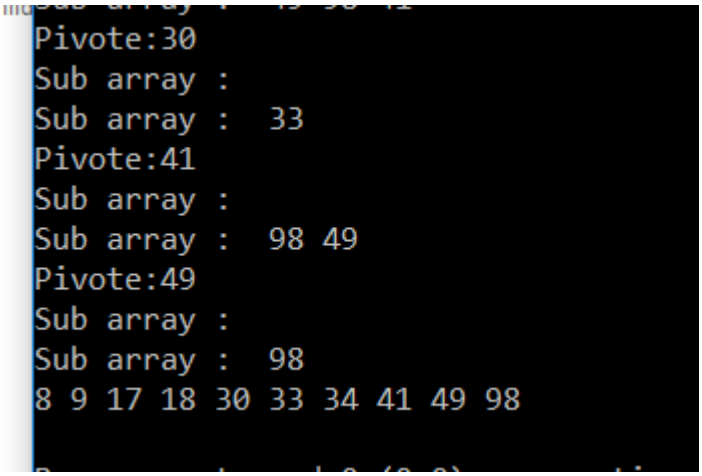
```
53 54 58 59 60 61 62 63 64 65 66 67 69 70 71 72 7
 101 102 103 104 105 106 107 108 109 110 111 112

 comparaciones 4914,  cambios  2728
        0 1 9 11 12 13 14 15 16 17 18 19 20 21 22
0 51 52 53 54 58 59 60 61 62 63 64 65 66 67 69 70
8 99 100 101 102 103 104 105 106 107 108 109 110

Process returned 0 (0x0)   execution time : 44.04
Press any key to continue.
```

Realizando la prueba con 500 números para burbuja:

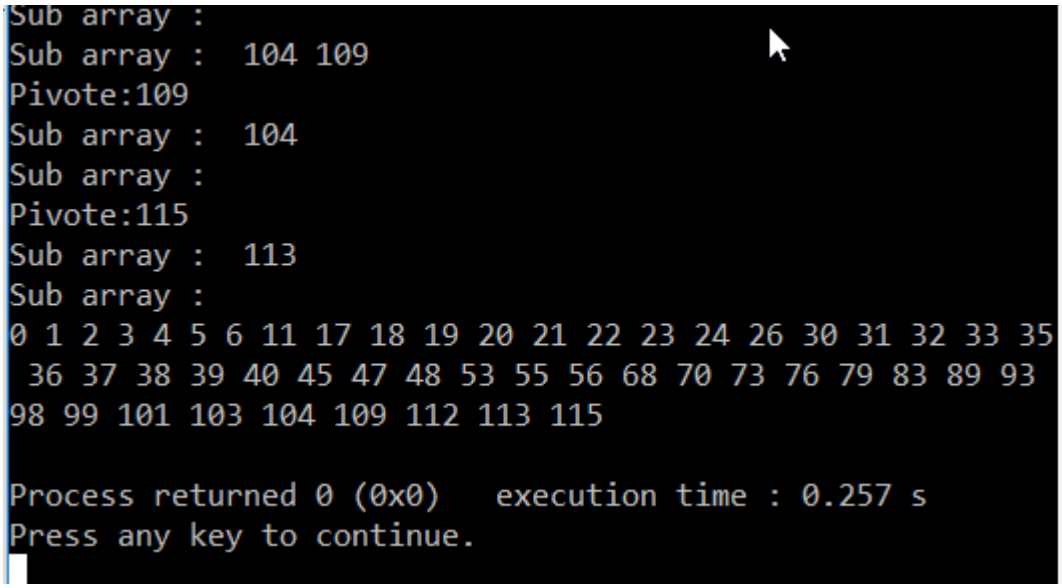
Para esta prueba debido a la complejidad el algoritmo Burbuja tardo más de 10 minutos haciendo el intercambio y cerre el ejecutable, la imagen es la siguiente.



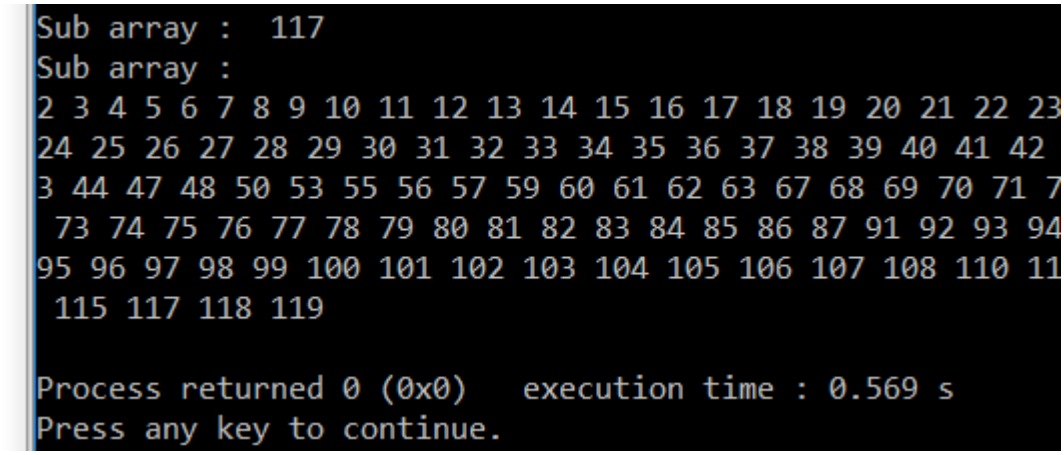
Realizando la prueba con 10 números para quickSort:



Realizando la prueba con 50 números para quickSor:



Realizando la prueba con 100 números para quickSort:



Realizando la prueba con 500 números para quickSort:

```
Sub array :  599
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 84 85
91 92 93 94 95 96 97 98 99 100 103 106 107 108 109 110 111 112 113 114 117 120 121 122 123 124 125 126 127 129 130 133 1
34 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 161 163 164 167 170 1
71 172 173 175 176 177 178 179 180 181 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 2
03 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 2
33 234 235 236 237 238 239 240 241 242 243 245 246 247 249 253 255 256 257 258 259 260 261 262 263 264 265 266 267 268 2
69 270 271 272 273 274 275 276 277 278 279 280 281 282 284 285 287 288 290 291 292 293 294 295 296 297 298 299 300 308 3
09 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 336 337 338 339 340 341 3
42 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 368 371 372 373 376 377 3
80 382 383 384 385 386 387 388 390 391 393 394 397 398 399 400 402 403 405 407 409 410 411 412 413 415 416 417 418 419 4
20 421 422 424 425 426 427 428 429 431 432 434 435 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 4
54 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 476 477 479 482 483 484 485 486 487 4
88 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 513 514 516 517 518 520 521 523 525 5
26 527 528 529 530 531 532 533 534 535 539 540 541 544 545 548 549 550 551 552 553 556 557 559 560 561 562 563 564 566 5
67 568 569 570 572 574 576 577 578 579 580 581 582 583 584 588 589 592 593 594 595 596 597 598 599
```

3) QuickSort -

Compila y ejecuta el proyecto.

```
    // Driver program
    public static void main(String args[])
    {
        int arr[] = {87,4,32,15,8,12,10,30,22};
        int n = arr.length;
        QuickSort ob = new QuickSort();
        ob.sort(arr, 0, n-1);
        System.out.println("Arreglo ordenado");
        printArray(arr);
    }

}
```

```
Output - Quicksort (run)
init:
deps-jar:
Updating property file: C:\Users\Jonathan\De
compile:
run:
Arreglo ordenado
4 8 10 12 15 22 30 32 87
BUILD SUCCESSFUL (total time: 1 second)
```

Comentarios acerca del lenguaje Java:

Me doy cuneta que el lenguaje no cambia mucho. Muchas cosas son similares a C y a otros lenguajes como por ejemplo la declaración las variables el “for” y las asignaciones como i++; o arr[i]=temp; , así como también las funciones son parecidas a C.

Lo que si cambia aun que no mucho es la sintaxis al querer imprimir en pantalla, se le tiene que agregar System.out.printl, es un poco mas largo a comparación de C.

public static void main(String args[]) esta línea me causa un poquito de ruido porque aun no comprendo bien su uso.

Para compilar no hay diferencia solo los iconos que son diferentes ha y que la ejecución se hace en la parte baja del compilador de Netbeans.

Conclusiones:

Los algoritmos de ordenamiento tienen su grado de complejidad, así como ventajas y desventajas según sea el caso de los datos a ordenar como por ejemplo la cantidad de datos.

Para la mayoría de los casos siempre existe un algoritmo optimo que nos va a resolver el problema de la mejor manera.

El algoritmo de la burbuja su mejor caso es cuando la lista o los datos están ordenados, el peor de los casos es cuando no están ordenados son muchos datos. Por ejemplo, cuando hice la prueba para 500 números tardo más de 10 minutos en hacer el orden y mejor cerré la ejecución ya que era demasiado tiempo. Esto se debe a que trabaja con una complejidad de n^(2)

El algoritmo de quickSort en las pruebas fue muy rápido comparado con el de la Burbuja incluso en la prueba de 500 elementos.

La eficiencia de este algoritmo depende de la posición en donde se encuentre le pivote elegido. El mejor de los casos en cuando se encuentra en la mitad de los datos y el peor cuando se encuentra al último.

En java este algoritmo no cambio casi nada me refiero a las instrucciones y funciones ya que el algoritmo es el mismo.

Se cumplió el objetivo de la practica ya que se logró identificar la diferencia entre un algoritmo y otro, así como su complejidad, así como se logro entender la manera en como trabaja cada uno de los algoritmos vistos en la práctica.

Solo me hace falta estudiar la parte de el contador de comparaciones y cambios.