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): 2

Integrante(s): Calzada Martinez Jonathan Omar

No. de Equipo de cómputo empleado 35

Semestre: 2019-2

Fecha de entrega: 15/02/2019

Obervaciones:

CALIFICACIÓN: _____

Objetivo:

El estudiante identificará la estructura de los algoritmos de ordenamiento SelectionSort, InsertionSort y HeapSort.

Introducción:

Los algoritmos de ordenamiento nos permite, como su nombre lo dice, ordenar información de una manera especial basándonos en un criterio de ordenamiento.

En la computadoción el ordenamiento de datos juega un rol muy importante porque es algo que se utiliza para todo, siempre estamos ordenando cosas a veces sin darnos cuenta y estos algoritmos han ayudado a resolver problemas, así como crear maneras de hacernos la vida más simple .Se han desarrollado muchas técnicas en este ámbito, cada una con características específicas, y con ventajas y desventajas sobre las demás.

En esta ocasión estudiaremos los algoritmos SelectionSort, InsertionSort así como el de HeapSort

Desarrollo:

a) Con respecto al enfoque visto en clase el algoritmo trabaja de la misma manera en como se vio en la clase ya que se almacenan los n valores a ordenar en un arreglo (o en una lista) de n elementos, Insertion Sort va construyendo un trozo ordenado del arreglo al extremo izquierdo, y en cada iteración del programa, le agrega un nuevo elemento a ese grupo. De este modo, se va formando un sub-arreglo (o sublista) ordenado al lado izquierdo, el cual crece 1 elemento en cada repetición, de modo que al llegar al final del arreglo (o lista), éste estará totalmente ordenado.Para realizar el algoritmo el programa utiliza una variable auxiliar, la cual almacena el elemento que está siendo insertado. Se recorre la sublista de atrás hacia adelante para insertar el elemento en su lugar y así quedar ordenado respecto a los elementos que se encuentran antes en la lista, los cuales fueron ordenados en el paso anterior.

Considero que este es el momento de la iteración.

```
while ( (i > -1) && ( array [i] > key ) )
    array [i+1] = array [i];
    i--;
}
array[i+1] = key;
```

En la siguente imagen se muestra el arreglo original (desordenado) y el ordenado.

```
run:
Arreglo Original
10 15 4 3 44 12 59 23
Arreglo ordenado
3 4 10 12 15 23 44 59
BUILD SUCCESSFUL (total time: 0 seconds)
```

b)

El en InsertionSort solo se le manda el arreglo para que pueda hacer el

ordenamiento, a diferencia de selectionSort que de igual manera se le manda el arreglo a la funcion pero tambien se llama a un modulo para que ejecute la funcion.

Esto debido a que selección compara todo y necesita crear esa instancia.

```
//InsertionSort.insertionSort(arr2);
SelectionSort seleccion = new SelectionSort();
seleccion.selectionSort(arr1);
```

c)
El algoritmo mostrado de SelectorSort es el mismo que se estudio en clase. Consiste en encontrar el menor de entre todos los elementos de la lista y hacer el cambio con el que está en la primera posición. Luego el segundo mas pequeño, y así sucesivamente hasta ordenarlo todo.

El cambio que hay solo es al terminar el ciclo en donde se compara, se crea una variable llamada numeroMin en donde es se le asigna el valor más pequeño en de entre la lista y ahora trabajar con ese.

```
Arreglo Original
10 15 4 3 44 12 59 23
Iteración 1
Índice del mínimo 3
Valor 2
2 14 3 9 43 11 58 22
Iteración 2
Índice del mínimo 2
Valor 3
2 3 14 9 43 11 58 22
Iteración 3
Índice del mínimo 3
Valor 9
2 3 9 14 43 11 58 22
Iteración 4
Índice del mínimo 5
Valor 11
2 3 9 11 43 14 58 22
Iteración 5
Índice del mínimo 5
Valor 14
2 3 9 11 14 43 58 22
Iteración 6
Índice del mínimo 7
Valor 22
2 3 9 11 14 22 58 43
Iteración 7
Índice del mínimo 7
```

c) A diferencia de C, en java no empizan con un encabezado en donde se encuentran las bibliotecas principales. Se muestra una clase y en ella trabaja como si fuese la funcion principal, comparandolo con C con el (Int main) se encuentra un metodo en donde se utiliza para imprimir el arreglo que es llamada en cada una de las funciones utilizadas de ordenamiento. Este metodo esta hecha con public static void main.

Siguiendo con la comparación de C y Java la forma de crear un for/ para es de la misma manera al igual que las instrucciones de inserción como por ejemplo min=j; o las de impresión que terminan con " ; "; en las de impresión solo cambia la sintaxis que es System.out.prentln.

EJERCICIO 2

a) Codifica, compila y ejecuta el programa. // Comenta las funciones del programa.

Primero tenermos la función principal main en donde se crea una arreglo llamado prueba de 11 elementos, se llama a la función HeapSort y se le manda el tamaño del arreglo y luego se crea un for para imprmir el arreglo.

```
int main(){
    int prueba[LENGTH] = {8,25,40,17,66,72,4,29,23,5,34};
    HeapSort(prueba);
    int i;
    for(i = 0; i < LENGTH; i++)
        printf("%d ",prueba[i]);
    return 0;
}
```

La función Heapyfy hace que subárbol con raíz en el índice i se convierte en un heap

```c
void Heapify(int* A, int i)
{
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    int largest;

    if(l <= heapSize && A[l] > A[i])
        largest = l;
    else
        largest = i;
    if(r <= heapSize && A[r] > A[largest])
        largest = r;
    if(largest != i){
        swap(&A[i],&A[largest]);
        printArray(A,LENGTH);
        Heapify(A, largest);
     }

}
```

La función BuilHeap crea el heap, llamando a heapify para hacer los intercambios de las raíces de los subárboles y la función Heapsort es la que llama a la función crearheap para que se cree el algoritmo, y solo en esta realizar la iteración para imprmirlos datos.

```c
void BuildHeap(int* A){
    heapSize = LENGTH - 1;
    int i;
    for(i = (LENGTH - 1) / 2; i >= 0; i--){
        Heapify(A, i);
    }
    printf("Termin%c de construir el HEAP \n",162);
}

void HeapSort(int* A){
    BuildHeap(A);
    int i;
    for(i = LENGTH - 1; i > 0; i--){
        swap(&A[0],&A[heapSize]);
        heapSize--;
        printf("Iteracion HS: \n");
        printArray(A,LENGTH);
        Heapify(A, 0);
   }
}
```

La función swap realiza el intercambio por medio de apuntadores.

utilidades.h

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

La función printArray realiza la impresión de las iteraciones mantandole el arreglo y el tamaño

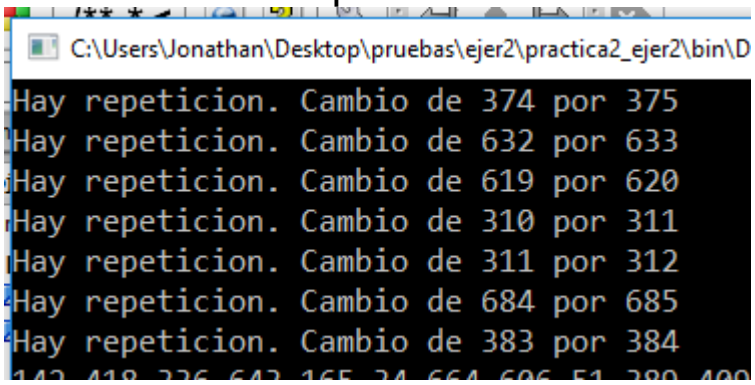
```c
}

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

En esta función se imprime el sub arreglo que se crea. Mandandole como parámetros el arreglo, menor y el mayor

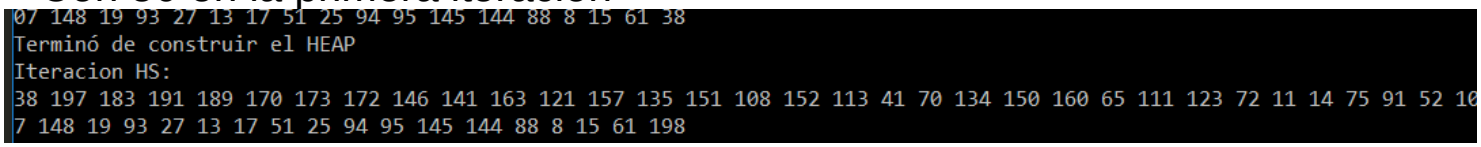
```c
void printSubArray(int arr[],int low, int high){
    int i;
    printf("Sub array :  ");
    for (i=low; i <= high; i++)
       printf("%d ", arr[i]);
    printf("\n");
}
```

B) hacer que funcione con 50,100,500.

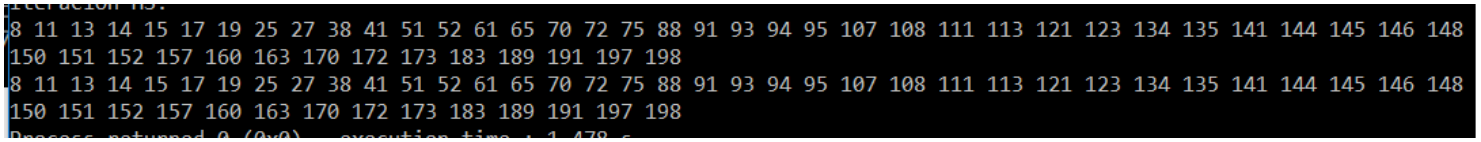
Nota: Al igual que en la practica pasada se implementaron las líneas necesarias de código para poder intercambiar los números que ya estaban en el nuevo arreglo creado de forma aleatoria y de esta manera que no existan nueros repetidos.



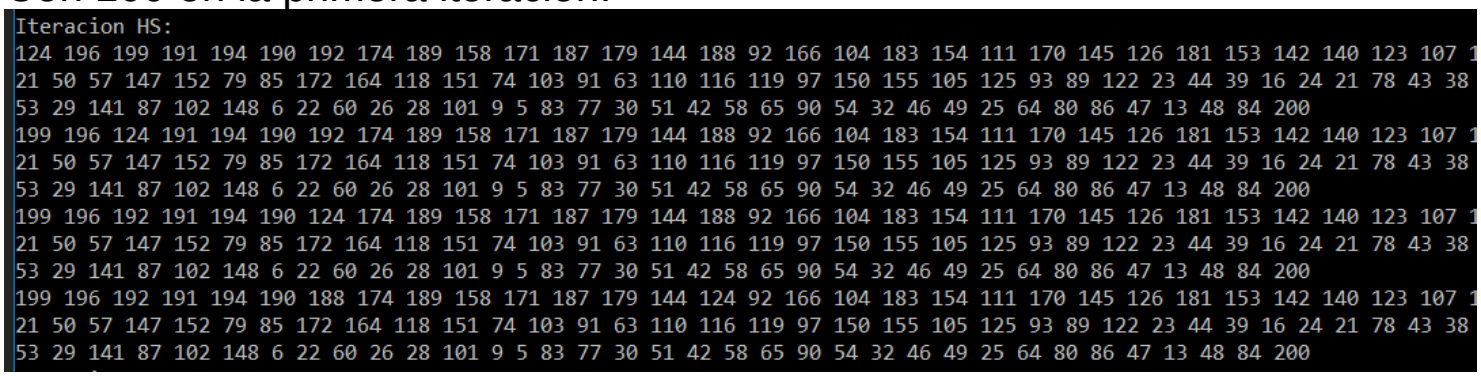
Con 50 en la primera iteracion



Listado ordenado. Tardo. 1.5 segundos



Con 100 en la primera iteracion.



Listado ordenado. Tardo 5.071 segundos

```
Iteracion HS:
5 6 9 13 16 21 22 23 24 25 26 28 29 30 32 38 39 42 43 44 46 47 48 49 50 51 53 54 57 58 60 63 64 65 74 77 78 79 80 83 84
85 86 87 89 90 91 92 93 97 101 102 103 104 105 107 110 111 116 118 119 121 122 123 124 125 126 140 141 142 144 145 147 1
48 150 151 152 153 154 155 158 164 166 170 171 172 174 179 181 183 187 188 189 190 191 192 194 196 199 200
5 6 9 13 16 21 22 23 24 25 26 28 29 30 32 38 39 42 43 44 46 47 48 49 50 51 53 54 57 58 60 63 64 65 74 77 78 79 80 83 84
85 86 87 89 90 91 92 93 97 101 102 103 104 105 107 110 111 116 118 119 121 122 123 124 125 126 140 141 142 144 145 147 1
48 150 151 152 153 154 155 158 164 166 170 171 172 174 179 181 183 187 188 189 190 191 192 194 196 199 200
Process returned 0 (0x0)   execution time : 4.497 s
Press any key to continue.
```

Con 500 en la primera iteracion.

```
Iteracion HS:
1 133 132 127 131 126 128 125 117 115 123 124 92 114 118 116 111 112 110 109 85 121 106 122 68 79 91 84 94 113 107 97 99 56 89 96 98 90 40 108 105 55
 58 81 88 87 48 101 86 52 60 54 77 75 21 82 53 74 43 93 31 5 49 42 47 45 16 29 27 62 7 57 70 13 10 33 46 25 11 63 64 12 95 41 30 34 15 59 76 78 37 0
26 17 18 6 80 19 83 44 24 14 50 20 35 134 137 138 139 140 142 147 148 149 151 152 153 154 155 156 157 158 159 160 161 162 163 164 169 171 173 176 179
 180 181 182 183 184 185 186 187 188 189 190 191 193 195 196 198 199 202 204 205 210 211 214 216 217 220 223 225 226 228 232 233 234 235 236 238 239
240 241 242 243 244 245 246 247 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 265 268 270 271 274 275 276 278 279 280 283 285 286 287 2
89 290 292 293 294 295 296 297 298 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 321 322 328 329 330 331 332 333 334 336 33
8 339 340 341 344 346 349 350 351 352 353 355 356 357 358 359 360 361 365 366 367 368 369 370 372 374 375 377 378 379 380 381 382 383 386 387 388 389
 390 391 392 393 394 395 396 397 398 399 400 401 403 404 405 406 407 408 409 411 412 413 414 415 416 418 419 420 421 422 423 424 425 429 430 431 433
434 436 437 439 443 444 447 448 449 450 451 452 453 455 456 457 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 4
80 481 482 483 485 486 487 488 491 492 493 494 495 496 497 499 502 503 504 507 509 510 511 512 513 514 515 516 517 519 522 523 524 525 526 527 528 52
9 530 531 532 533 534 535 536 537 538 539 540 543 544 545 546 547 548 549 550 554 555 558 559 560 566 567 568 570 571 572 573 574 575 576 578 580 581
 582 583 585 590 591 592 593 594 597 598 603 604 605 606 607 608 609 610 611 612 614 615 616 617 625 626 628 630 631 632 637 638 639 640 641 642 643
646 649 650 655 657 658 659 660 661 665 667 669 672 674 675 676 677 684 685 686 687 688 689 690 691 692 694 695 697 698 699 700
133 1 132 127 131 126 128 125 117 115 123 124 92 114 118 116 111 112 110 109 85 121 106 122 68 79 91 84 94 113 107 97 99 56 89 96 98 90 40 108 105 55
 58 81 88 87 48 101 86 52 60 54 77 75 21 82 53 74 43 93 31 5 49 42 47 45 16 29 27 62 7 57 70 13 10 33 46 25 11 63 64 12 95 41 30 34 15 59 76 78 37 0
26 17 18 6 80 19 83 44 24 14 50 20 35 134 137 138 139 140 142 147 148 149 151 152 153 154 155 156 157 158 159 160 161 162 163 164 169 171 173 176 179
 180 181 182 183 184 185 186 187 188 189 190 191 193 195 196 198 199 202 204 205 210 211 214 216 217 220 223 225 226 228 232 233 234 235 236 238 239
240 241 242 243 244 245 246 247 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 265 268 270 271 274 275 276 278 279 280 283 285 286 287 2
89 290 292 293 294 295 296 297 298 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 321 322 328 329 330 331 332 333 334 336 33
8 339 340 341 344 346 349 350 351 352 353 355 356 357 358 359 360 361 365 366 367 368 369 370 372 374 375 377 378 379 380 381 382 383 386 387 388 389
 390 391 392 393 394 395 396 397 398 399 400 401 403 404 405 406 407 408 409 411 412 413 414 415 416 418 419 420 421 422 423 424 425 429 430 431 433
434 436 437 439 443 444 447 448 449 450 451 452 453 455 456 457 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 4
80 481 482 483 485 486 487 488 491 492 493 494 495 496 497 499 502 503 504 507 509 510 511 512 513 514 515 516 517 519 522 523 524 525 526 527 528 52
9 530 531 532 533 534 535 536 537 538 539 540 543 544 545 546 547 548 549 550 554 555 558 559 560 566 567 568 570 571 572 573 574 575 576 578 580 581
 582 583 585 590 591 592 593 594 597 598 603 604 605 606 607 608 609 610 611 612 614 615 616 617 625 626 628 630 631 632 637 638 639 640 641 642 643
646 649 650 655 657 658 659 660 661 665 667 669 672 674 675 676 677 684 685 686 687 688 689 690 691 692 694 695 697 698 699 700
133 131 132 127 1 126 128 125 117 115 123 124 92 114 118 116 111 112 110 109 85 121 106 122 68 79 91 84 94 113 107 97 99 56 89 96 98 90 40 108 105 55
 58 81 88 87 48 101 86 52 60 54 77 75 21 82 53 74 43 93 31 5 49 42 47 45 16 29 27 62 7 57 70 13 10 33 46 25 11 63 64 12 95 41 30 34 15 59 76 78 37 0
26 17 18 6 80 19 83 44 24 14 50 20 35 134 137 138 139 140 142 147 148 149 151 152 153 154 155 156 157 158 159 160 161 162 163 164 169 171 173 176 179
 180 181 182 183 184 185 186 187 188 189 190 191 193 195 196 198 199 202 204 205 210 211 214 216 217 220 223 225 226 228 232 233 234 235 236 238 239
240 241 242 243 244 245 246 247 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 265 268 270 271 274 275 276 278 279 280 283 285 286 287 2
89 290 292 293 294 295 296 297 298 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 321 322 328 329 330 331 332 333 334 336 33
```

Listado ordenado. Tardo 97.6 seg. =1.62 min

```
 698
0 1 3 5 6 8 9 12 13 14 16 17 18 19 22 24 25 26 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 5
7 59 60 61 62 63 64 66 67 68 73 74 75 76 77 78 79 81 82 83 84 88 89 90 91 94 100 101 102 103 105 107 109 110 111 112 113
 114 115 116 117 120 121 122 123 124 125 126 127 128 129 130 131 134 135 136 137 138 139 140 141 142 143 144 145 146 147
 148 149 150 151 152 153 154 155 156 157 158 159 160 162 163 164 168 169 170 171 172 173 174 175 176 177 178 179 180 181
 182 186 187 192 193 194 195 196 197 198 201 202 206 207 208 209 210 211 212 213 214 215 220 221 222 223 224 225 226 227
 228 229 230 231 234 238 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 262 263 267 270
 272 274 275 276 277 280 281 282 283 285 287 288 290 291 292 293 294 295 296 297 298 299 300 303 304 305 306 307 308 309
 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 334 336 339 340 341 343 344
 345 346 349 355 356 357 362 363 366 367 368 369 370 371 372 373 374 375 376 377 381 382 383 387 388 389 390 393 394 395
 396 398 401 402 403 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 431 432
 437 438 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 469
 471 472 473 474 475 476 477 478 479 480 481 483 484 486 488 489 490 491 492 494 496 497 498 499 501 502 503 504 505 506
 511 512 513 517 519 520 521 522 523 524 527 528 529 530 531 532 533 534 535 536 537 538 539 540 542 544 545 546 548 549
 550 551 552 553 556 559 560 561 562 563 564 568 569 570 571 572 573 574 576 579 581 582 583 584 585 586 587 588 590 591
 592 593 594 595 596 597 601 604 607 610 611 613 618 619 620 627 632 633 636 637 638 639 645 646 647 651 653 654 657 661
 663 664 665 666 667 672 673 674 675 676 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697
 698
Process returned 0 (0x0)   execution time : 85.789 s
```

Conclusiones:

Hemos visto que exiten muchos tipos de algoritmos para ayudarnos a ordenar información de todo tipo, y entre ellos debemos saber identificar cual es el que mejor nos conviene según sea el caso porque como hemos visto aun teniendo la misma cantidad de información o el mismo tipo de datos hay unos que tardan más y otros que tardan menos, o unos que nos gastan muchos recursos computacionales y otros no tanto (depende el hardware).

El algoritmo de ordenamiento selectionSort consiste en encontrar el menor de los elementos del arreglo e intercambiarlo con el que esta en la primera posición.
Y se realiza lo mismo para cada iteracion del algoritmo. Este algoritmo tarda lo mismo para un arreglo ordenado como para uno que no lo esta. Y es necesario crear un heap y eliminar la raíz.

El algoritmo de insertionSort se va construyendo un pedazo ordenado del arreglo al extremo izquierdo y en cada iteracion del programa le agrega un nuevo elemento a ese grupo y de esa manera se va creando la lista o el arreglo, en donde va creciendo en 1 por cada iteracion de modo que al llegar al final la lista esta completamente ordenada. Este algoritmo trabaja con una complejidad de n en el mejor de los casos y para el caso promedio y el peor trabaja con una complejidad de n(2)

En el caso del algoritmo HeapSort este se basa en la comparación usando el Heap. De esta manera en cada iteracion se elimina la raíz y se verifica que el resto de la estructura se siga conservando de la misma manera es de complejidad o(nlogn) todos los casos.

Se cumplido el objetivo ya que se entendió la menera de trabajar de estos algoritmos que estudiamos, cada uno de ellos se analizo y entendiendo las diferencias, ventajas y desventajas de cada uno de ellos.

En la parte teórica le había entendido bien al algoritmo de heapSort y en el código me costo un poco identificar las partes pero al final se logro cumplir con el objetivo de la practica.