|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación salas A y B

*Profesor: Asignatura:*

*Grupo: No de Práctica(s): Integrante(s):*

Tista Garcia Edgar

Estructura de Datos y Algoritmos II

5

2

Calzada Martinez Jonathan Omar

*No. de Equipo de cómputo empleado*

*Semestre:*

35

2019-2

15/02/2019

*Fecha de entrega: 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.

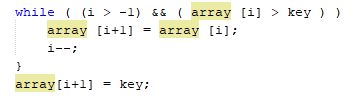
En la computadoción el ordenamiento de datos juega un rol muy importante porque es algo que se utiliza para todo, siempre estamos ordenando cosas a veces sin darnos cuenta y estos algoritmos han ayudado a resolver problemas, así como crear maneras de hacernos la vida más simple .Se han desarrollado muchas 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

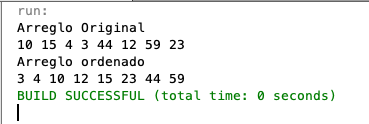
**Desarrollo:**

1. 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.



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

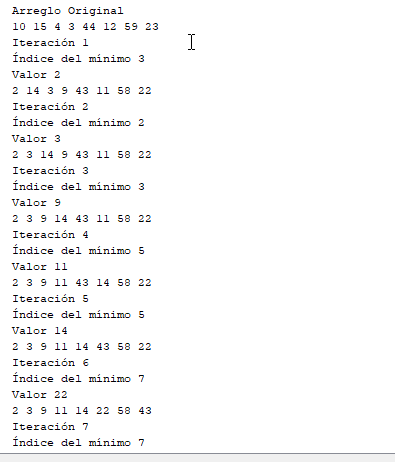


b)

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.



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.

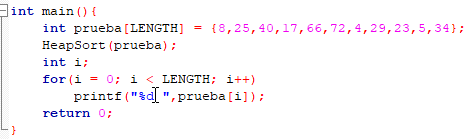
* 1. A diferencia de C, en java no empizan con un encabezado en donde se encuentran las bibliotecas principales. Se muestra una clase y en ella trabaja como si fuese la funcion principal, comparandolo con C con el (Int main ) se encuentra un metodo en donde se utiliza para imprimir el arreglo que es 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.

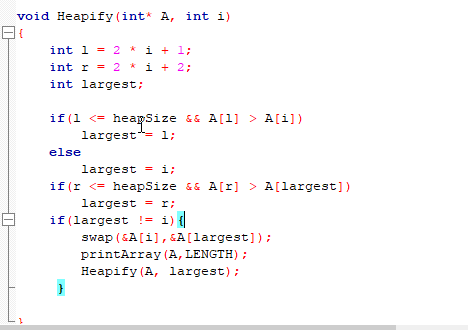
EJERCICIO 2

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

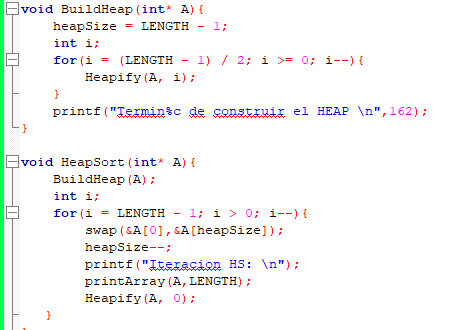
Primero tenermos la función principal main en donde se crea una arreglo llamado prueba de 11 elementos, se llama a la función HeapSort y se le manda el tamaño del arreglo y luego se crea un for para imprmir el arreglo.



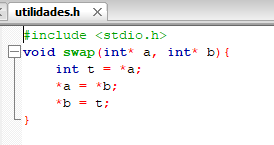
La función Heapyfy hace que subárbol con raíz en el índice i se convierte en un heap



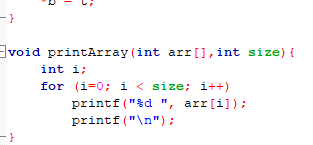
La función BuilHeap crea el heap, llamando a heapify para hacer los intercambios de las raíces de los subárboles y la función Heapsort es la que llama a la función crearheap para que se cree el algoritmo, y solo en esta realizar la iteración para imprmirlos datos.



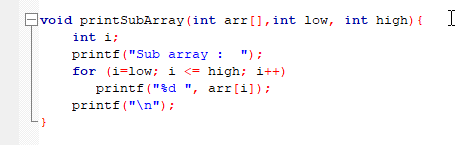
La función swap realiza el intercambio por medio de apuntadores.



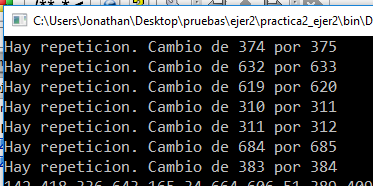
La función printArray realiza la impresión de las iteraciones mantandole el arreglo y el tamaño



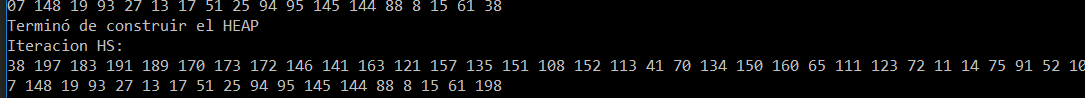
En esta función se imprime el sub arreglo que se crea. Mandandole como parámetros el arreglo, menor y el mayor



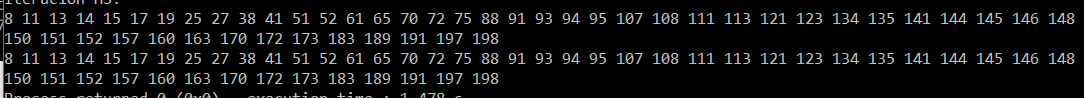
B) hacer que funcione con 50,100,500.   
 Nota: Al igual que en la practica pasada se implementaron las líneas necesarias de código para poder intercambiar los números que ya estaban en el nuevo arreglo creado de forma aleatoria y de esta manera que no existan nueros repetidos.



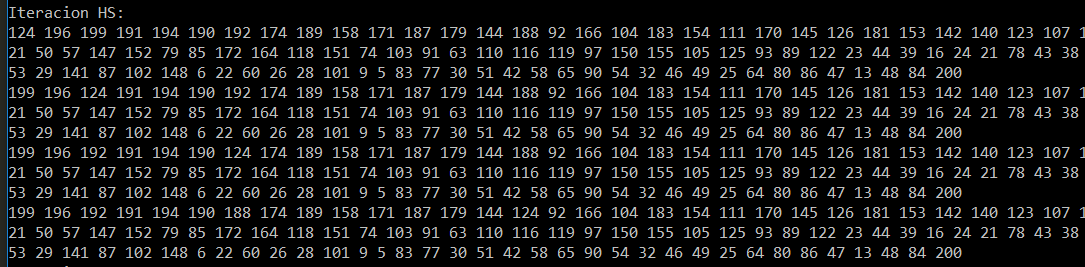
Con 50 en la primera iteracion



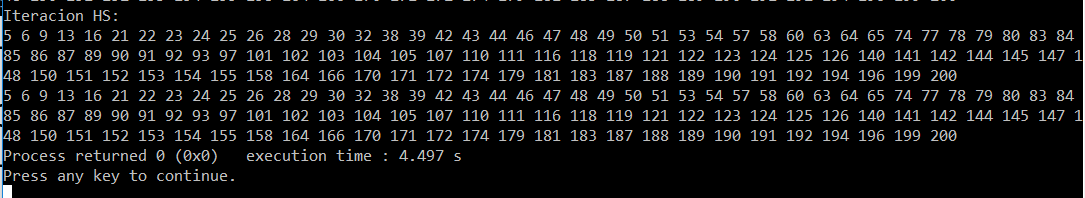
Listado ordenado. Tardo. 1.5 segundos



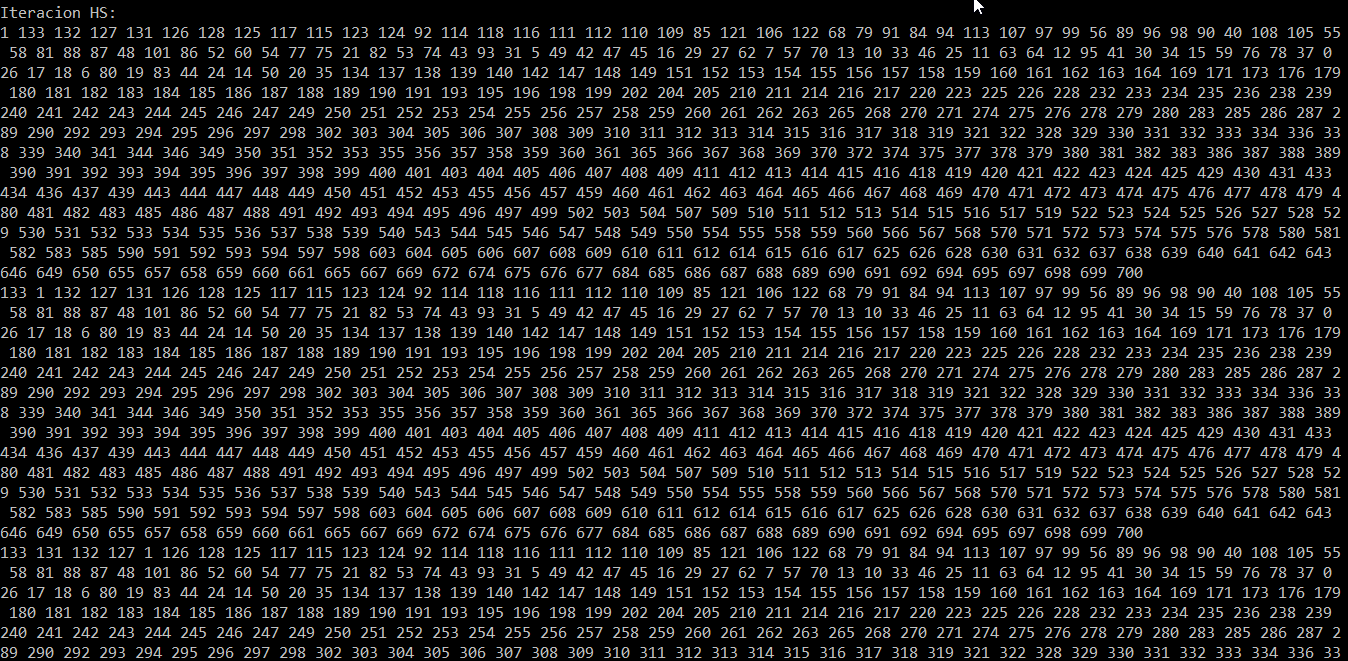
Con 100 en la primera iteracion.



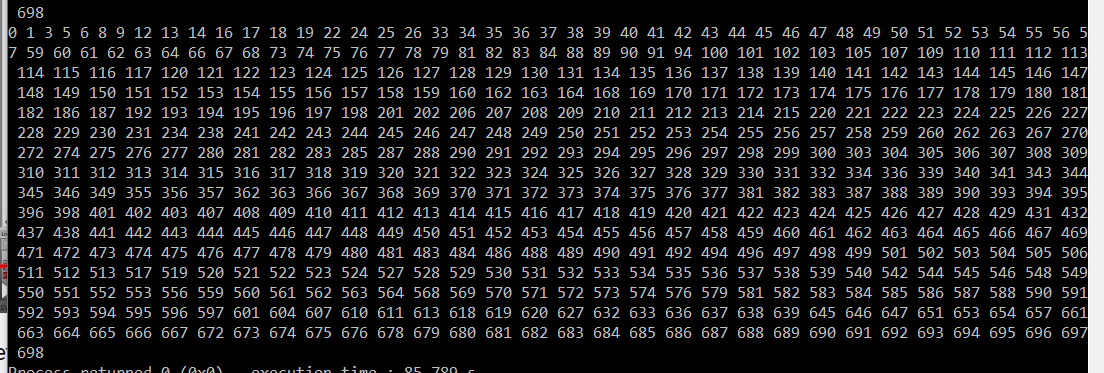
Listado ordenado. Tardo 5.071 segundos



Con 500 en la primera iteracion.



Listado ordenado. Tardo 97.6 seg. =1.62 min



**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.