Método de la ingeniería:

***Fase 1: identificación del problema.***

* Una empresa de microprocesadores quiere implementar 3 métodos de ordenamiento para que sean operaciones nativas de su coprocesador matemático y así mejorar el rendimiento del procesador.
* La empresa no tiene un prototipo de pruebas de software para medir la eficiencia de los algoritmos que desea implementar.
* La solución es la creación de un software que permita

1. Ingresar los valores a ordenar por medio de la interfaz
2. General aleatoriamente los números tanto enteros como flotantes
3. La generación aleatoria debe permitir general en secuencia, general valores ordenados inversamente, y generarlos completamente aleatorios

* Problema:
  + La empresa requiere el desarrollo de un software de prueba para algoritmos de ordenamiento.
  + Definición matemática:

Se crea un conjunto de entrada “A” tal que:

Y se espera una permutación de salida A’ tal que:

Encontrando el algoritmo que transforme A en A’ en el menor tiempo posible.

***Fase 2: Recopilación de la información necesaria.***

Para resolver este problema necesitamos información sobre los siguientes conceptos: Coprocesador, microprocesador y coma flotante. También necesitamos conocer los diferentes tipos de algoritmos de ordenamiento.

Conceptos:

**Coprocesador**

Un coprocesador es un microprocesador utilizado como respaldo para el procesador principal así evitando que el este tenga que realizar tareas de computo intensivo, el coprocesador puede realizar varias tareas como operaciones aritméticas en coma flotante, procesamiento gráfico, procesamiento de señales, procesamiento de texto entre otras.

**Microprocesador**

El microprocesador se puede definir como el “cerebro” del sistema, entre sus funciones principales incluye el manejo del sistema operativo, la ejecución de aplicaciones y la coordinación de diferentes dispositivos del equipo.

Es el encargado de ejecutar los programas, desde el sistema operativo hasta las aplicaciones de usuario; sólo ejecuta instrucciones programadas en lenguaje de bajo nivel, realizando operaciones aritméticas y lógicas simples, tales como sumar, restar, multiplicar, dividir, las lógicas binarias y accesos a memoria.

**Coma flotante**

La representación de coma flotante es una forma de notación científica usada en los microprocesadores con la cual se pueden representar números racionales extremadamente grandes y pequeños de una manera muy eficiente y compacta, y con la que se pueden realizar operaciones aritméticas.

Fuente: www.wikipedia.org

Fase 3: Búsqueda de soluciones creativas.

Para la solución de este problema necesitamos enfocarnos en dos componentes:

* Entradas.
* Algoritmos de ordenamiento.

Entradas:

Para generar las entradas de nuestros métodos de ordenamiento debemos dar dos opciones: ingresarlos por el usuario y generarlas aleatoriamente con parámetros recibidos del usuario como: los valores estén ya ordenados, que los valores estén ordenados inversamente, que los valores estén en orden completamente aleatorio o que los valores estén desordenados en un % indicado por el usuario.

Una manera de recibir los datos y parámetros ingresados por el usuario es realizando una interfaz gráfica donde por medio de distintos botones de chequeo el usuario podrá elegir la opción que prefiera. También, podrá ingresar un archivo de texto con el conjunto de números que desea ordenar.

Algoritmos de ordenamiento:

El problema de ordenar un conjunto de datos es un problema clásico, ya existen bastantes algoritmos de ordenamiento los cuales podemos utilizar para nuestro software e intentar crear uno nuevo es un trabajo bastante tedioso. Por esta razón, listaremos algunos de los algoritmos de ordenamiento ya existentes que tendremos en cuenta para nuestra búsqueda de soluciones y usando la técnica de “lista de atributos”

se explicará brevemente como funciona cada uno de estos.

Primero listaremos los algoritmos clásicos para ordenar:

1. Ordenamiento burbuja (Bubble sort): Funciona comparando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.

Ejemplo:

|  |  |  |
| --- | --- | --- |
| Primer iteration:  (5 1 4 2 8) → (1 5 4 2 8)  (1 5 4 2 8) → (1 4 5 2 8) (1 4 5 2 8) → (1 4 2 5 8) (1 4 2 5 8) → (1 4 2 5 8) | Segundo iteration:  (1 4 2 5 8) → (1 4 2 5 8) (1 4 2 5 8) → (1 2 4 5 8) (1 2 4 5 8) → (1 2 4 5 8) (1 2 4 5 8) → (1 2 4 5 8) | Tercera iteración: (1 2 4 5 8) → (1 2 4 5 8) (1 2 4 5 8) → (1 2 4 5 8) (1 2 4 5 8) → (1 2 4 5 8) (1 2 4 5 8) → (1 2 4 5 8) |

1. Ordenamiento por inserción (Insertion sort): Inicialmente se tiene un solo elemento, que obviamente es un conjunto ordenado. Después, cuando hay k elementos ordenados de menor a mayor, se toma el elemento k+1 y se compara con todos los elementos ya ordenados, deteniéndose cuando se encuentra un elemento menor (todos los elementos mayores han sido desplazados una posición a la derecha) o cuando ya no se encuentran elementos (todos los elementos fueron desplazados y este es el más pequeño). En este punto se inserta el elemento k+1 debiendo desplazarse los demás elementos.

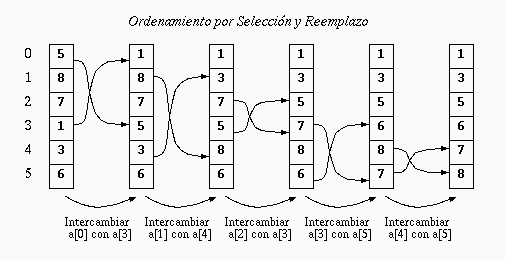
Ejemplo:



1. Ordenamiento por selección (Selection sort): el algoritmo encuentra repetidamente el elemento mínimo (teniendo en cuenta el orden ascendente) de una parte sin clasificar y ponerlo al principio. El algoritmo mantiene dos submatrices en una matriz dada: El subcampo que ya está ordenado y otro subcampo restante que no está ordenado.

En cada iteración de ordenación de selección, el elemento mínimo (considerando orden ascendente) del subarreglo sin clasificar se selecciona y se mueve al subcampo ordenado.

Ejemplo:

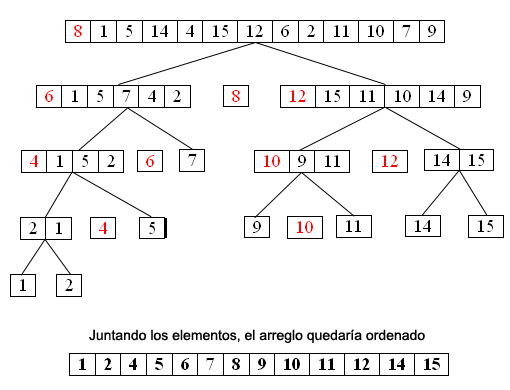


Ahora listaremos algunos de los algoritmos rápidos más populares:

1. Algoritmo Quicksort (ordenación rápida): El algoritmo trabaja de la siguiente forma:

* Elegir un elemento de la lista de elementos a ordenar, al que llamaremos pivote.
* Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
* La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
* Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

Ejemplo:

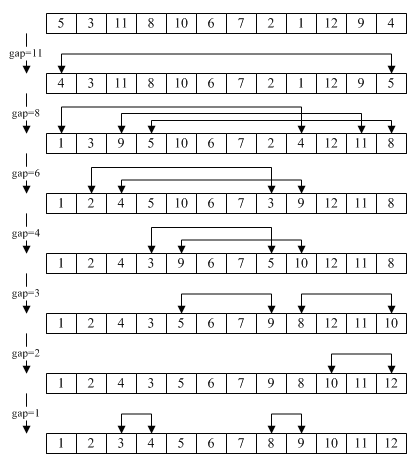


1. Algoritmo merge sort (ordenación por mezcla): Este algoritmo funciona de la siguiente manera:
   1. Dividir la lista al medio, formando dos sublistas de (aproximadamente) el mismo tamaño cada una.
   2. Ordenar cada una de esas dos sublistas (usando este mismo método).
   3. Una vez que se ordenaron ambas sublistas, intercalarlas de manera ordenada.

Ejemplo:



1. Comb sort : Comb Sort mejora Bubble Sort al usar brechas de tamaño mayor de 1. La brecha comienza con un valor grande y se reduce en un factor de 1.3 en cada iteración hasta que alcanza el valor 1. Así Comb Sort elimina más de una inversión que cuenta con una intercambiar y funciona mejor que Bubble Sort.

Ejemplo:

1. Algoritmo Heapsort (ordenación por montículo): El algoritmo heapsort se puede dividir en dos partes:

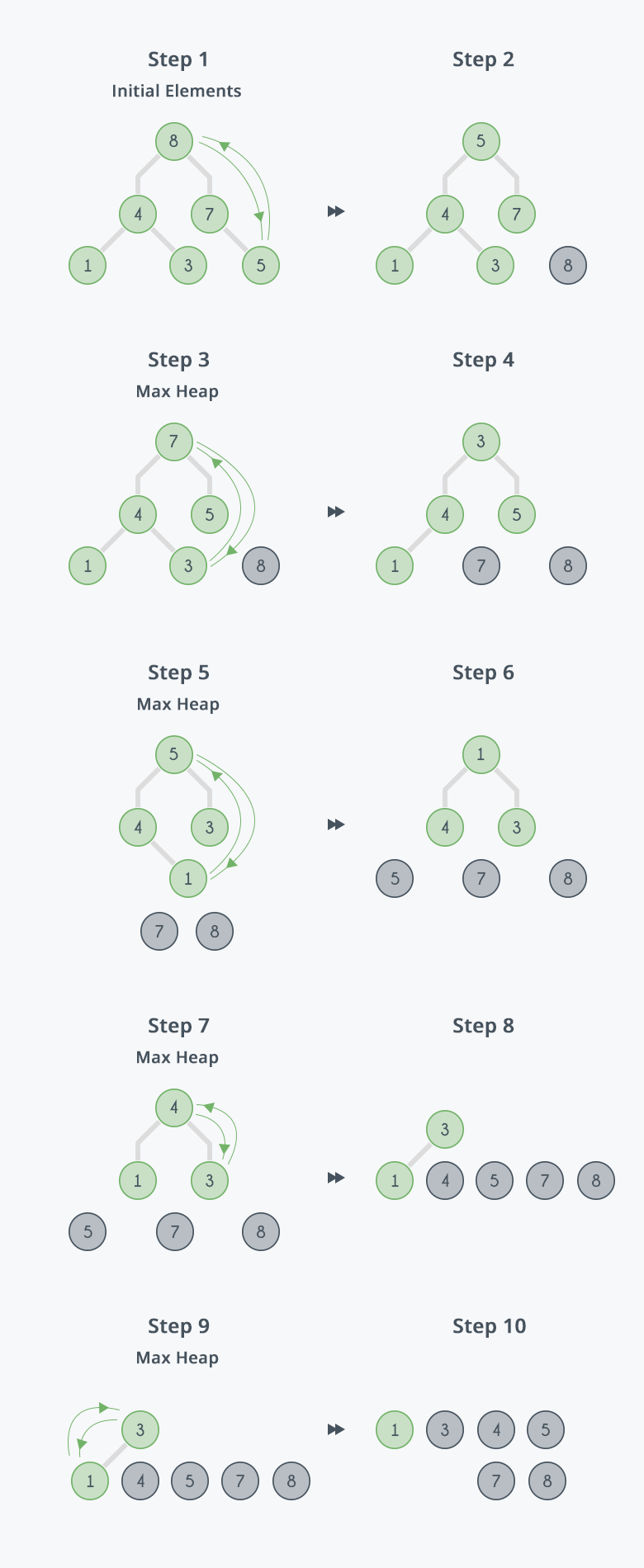
En el primer paso, un montón se construye de los datos. El montón a menudo se coloca en un arreglo con el diseño de un árbol binario completo. Cada índice del arreglo representa un nodo; el índice del elemento primario del nodo, rama secundaria izquierda o rama secundaria derecha son expresiones simples. Para un arreglo basado en cero, el nodo raíz se almacena en el índice 0

En el segundo paso, se crea un arreglo ordenado eliminando repetidamente el elemento más grande del montón (la raíz del montón) e insertándolo en el arreglo. El montón se actualiza después de cada eliminación para mantener la propiedad de montón. Una vez que todos los objetos se han eliminado del montón, el resultado es un arreglo ordenado.

Ejemplo:

Arreglo : {, 8 , 4 , 7 , 1 , 3 , 5 }

Pos: 0 , 1 , 2 , 3 , 4 , 5 , 6



También existen algoritmos de ordenamiento llamados imprácticos como el bogo sort o el Random sort los cuales trabajan generando todas las posibles permutaciones existentes en la lista que se desea arreglar. En el peor caso nunca terminan, por esto no los tendremos en cuenta para este trabajo.

Fase 4 – Fase 5: Transición de la formación de ideas a los diseños preliminares y Evaluación y selección de la mejor solución.

Teniendo en cuenta nuestros conocimientos adquiridos en clase y aterrizándolos en el problema, sabemos que un coprocesador tiene capacidad mínima de memoria y el tiempo para realizar las tareas en este debe ser el más optimo así que decidimos usar una tabla de criterios para elegir las mejores opciones para el diseño del software:

Criterios:

|  |  |
| --- | --- |
| * + - 1. Complejidad temporal.          1. 5 constante          2. 4 logarítmica          3. 3 n \* log n          4. 2 cuadrática          5. 1 factorial | * + - 1. Complejidad Espacial.          1. 5 constante          2. 4 logarítmica          3. 3 n \* log n          4. 2 cuadrática          5. 1 factorial |
| * + - 1. Estabilidad.          1. 2 estable          2. 1 inestable | * + - 1. Complejidad de programación          1. 3 poco complejo          2. 2 complejo          3. 1 bastante complejo |

|  |  |  |  |
| --- | --- | --- | --- |
| Algoritmo | C. temporal | C. espacial | Estabilidad |
| Bubble sort | O(n2) | O(1) | Estable |
| Insertion sort | O(n2) | O(1) | Estable |
| Selección sort | O(n2) | O(1) | Inestable |
| Quick sort | O(n2) | O(Logn) | Inestable |
| Merge sort | O(n\*Log n) | O(n) | Estable |
| Comb sort | O(n2) | O(1) | Inestable |
| Heap sort | O(n\*Log n) | O(1) | Inestable |
| Bogo sort | O(n\*n!) |  | Impracticos |
| Random sort | O(n!) |  | Impracticos |

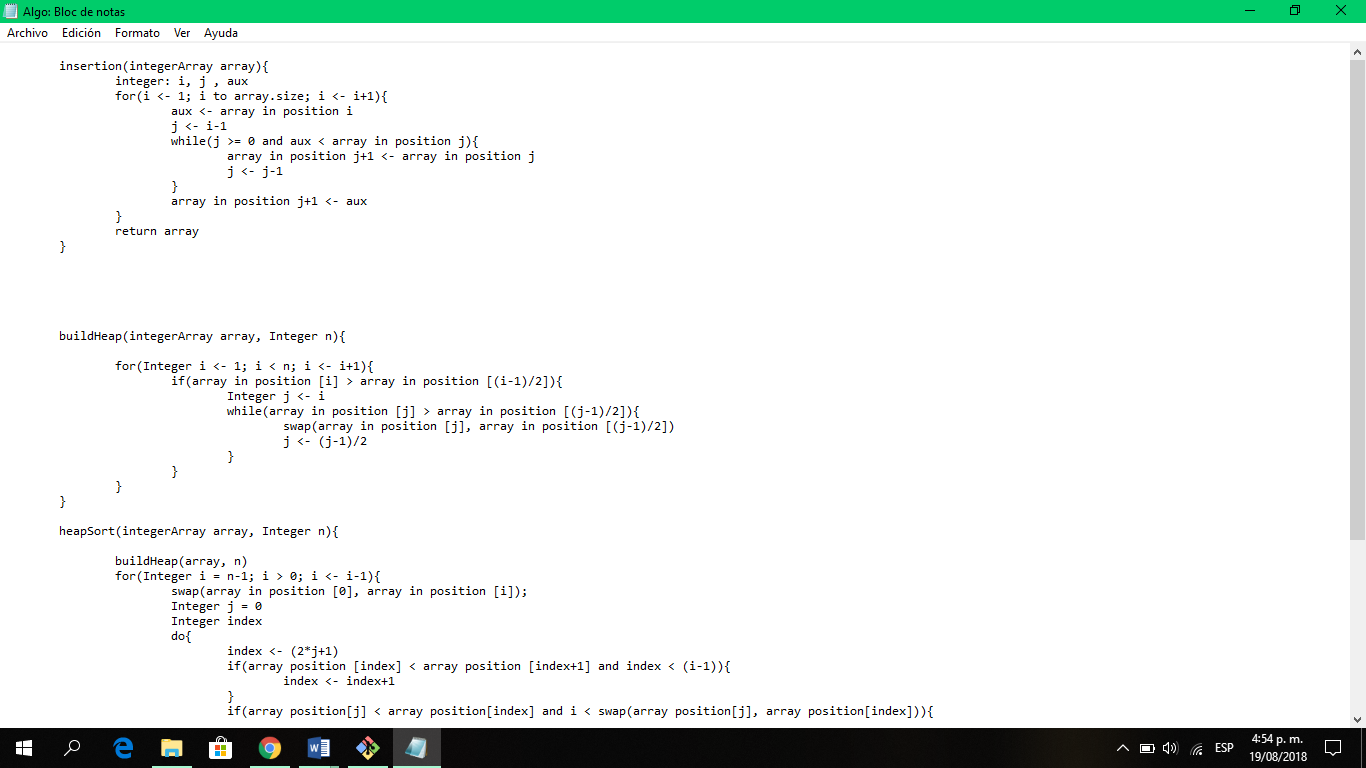
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algoritmo** | **Criterio 1** | **Criterio 2** | **Criterio 3** | **Criterio 4** | **Total** |
| Bubble sort | 2 | 5 | 2 | 2 | 11 |
| Insertion sort | 2 | 5 | 2 | 3 | 12 |
| Selección sort | 2 | 5 | 1 | 2 | 10 |
| Quick sort | 2 | 4 | 1 | 1 | 8 |
| Merge sort | 3 | 3 | 2 | 2 | 10 |
| Comb sort | 2 | 5 | 1 | 1 | 9 |
| Heap sort | 3 | 5 | 1 | 1 | 10 |

Según la tabla realizada los algoritmos que usaremos para este proyecto será:

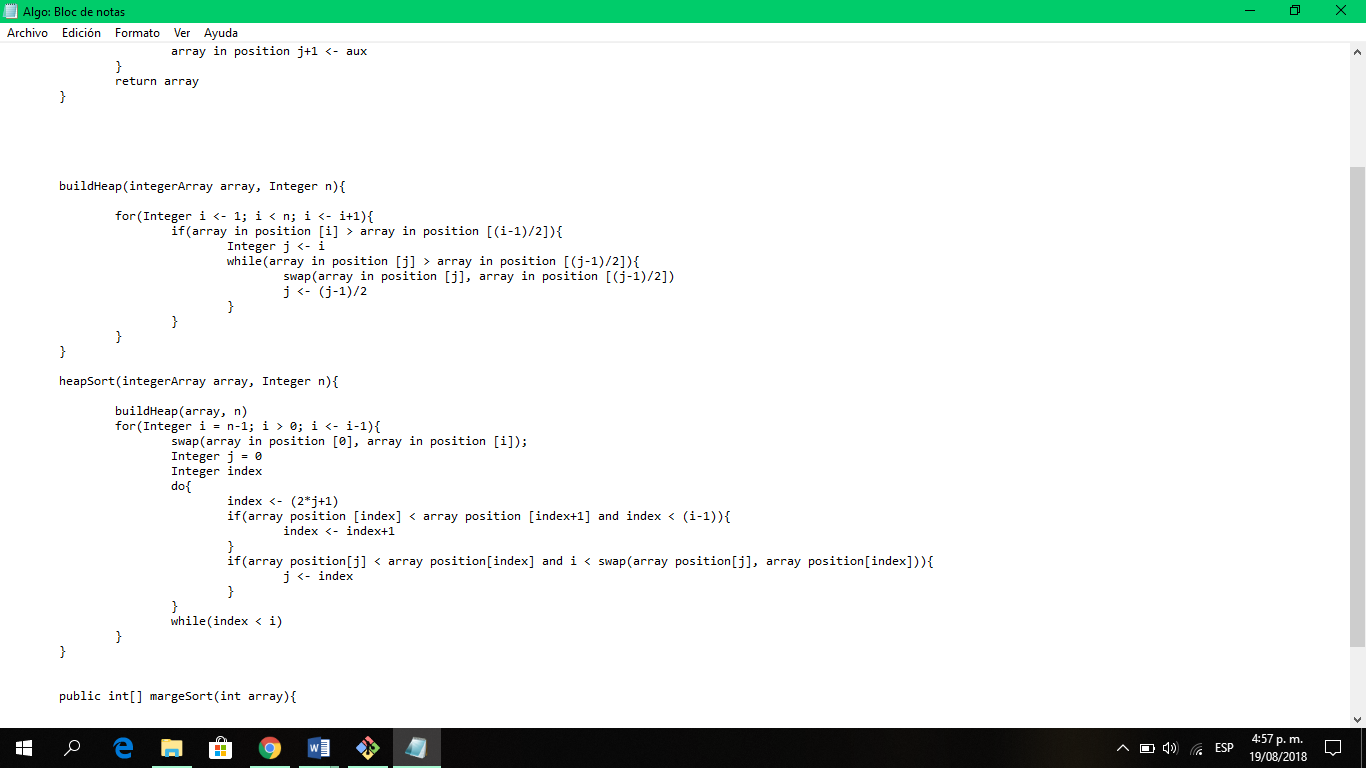
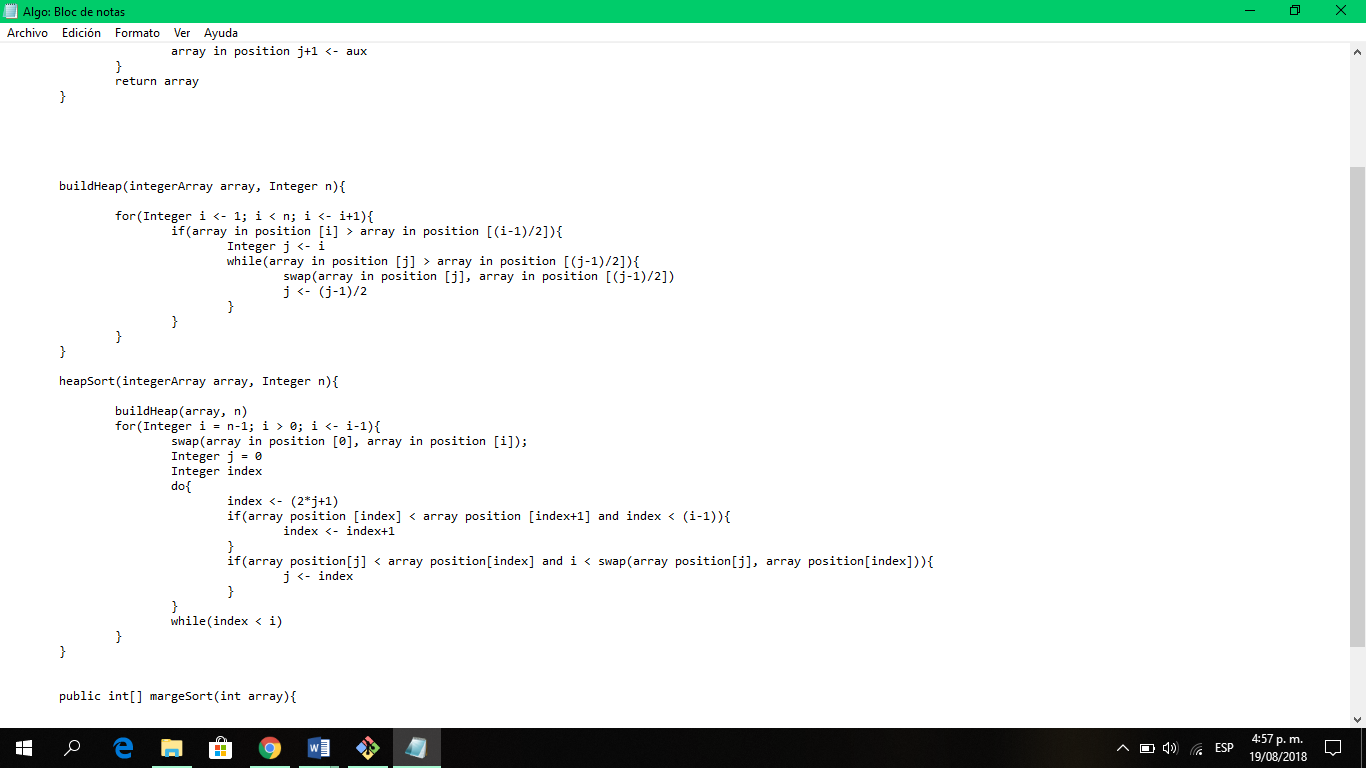
* Insertion sort
* Merge sort
* Heap sort

***Fase 6: Preparación de informes y especificaciones.***

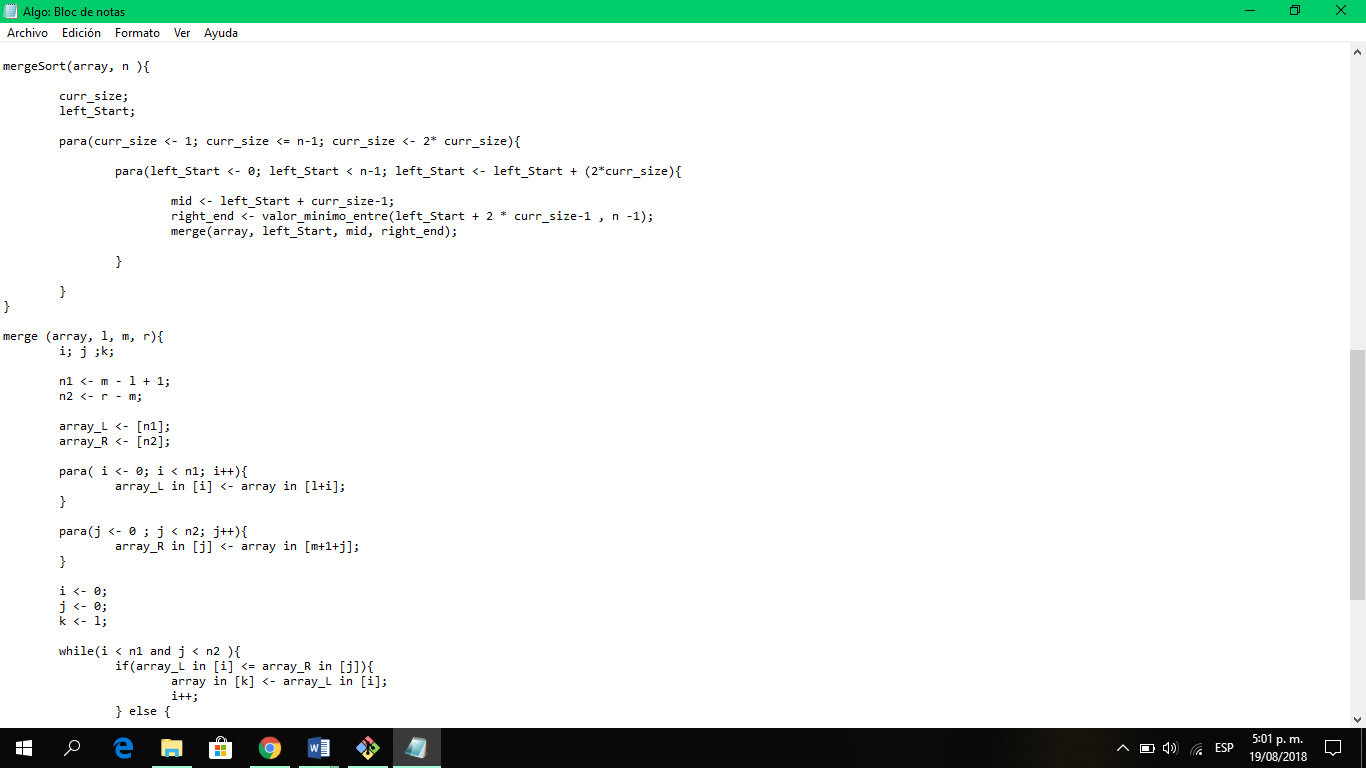
* *Pseudocodigo insertionSort*

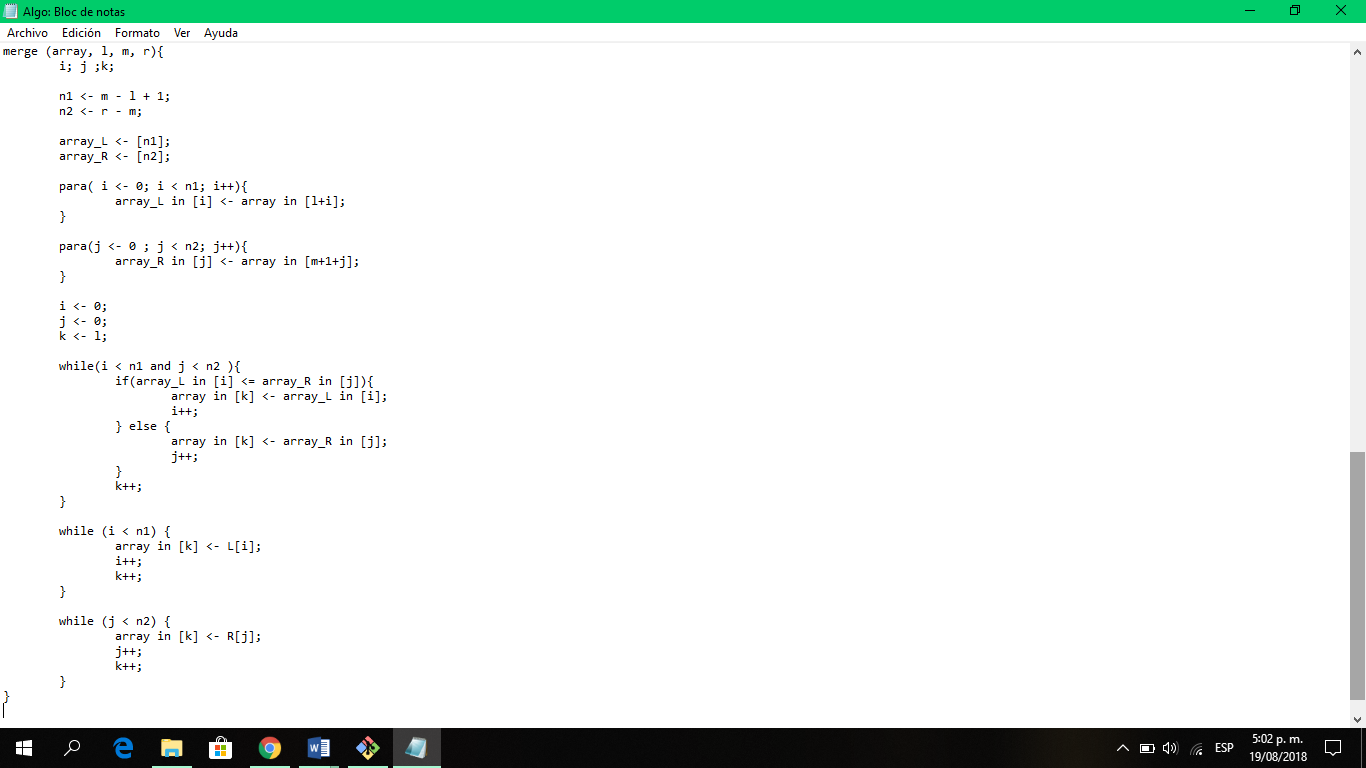


* *Pseudocodigo heapSort*



* *Pseudocodigo mergeSort*

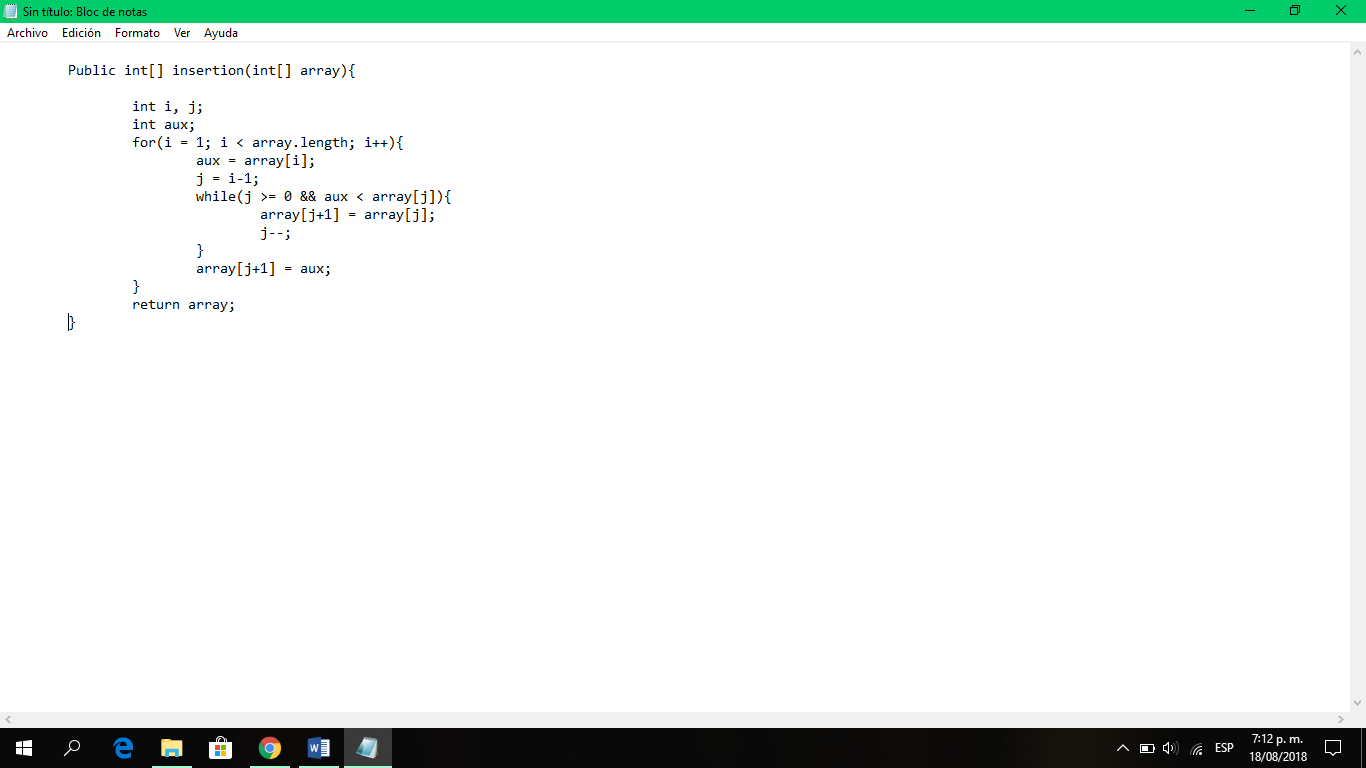




***Fase 7: implementación del diseño.***

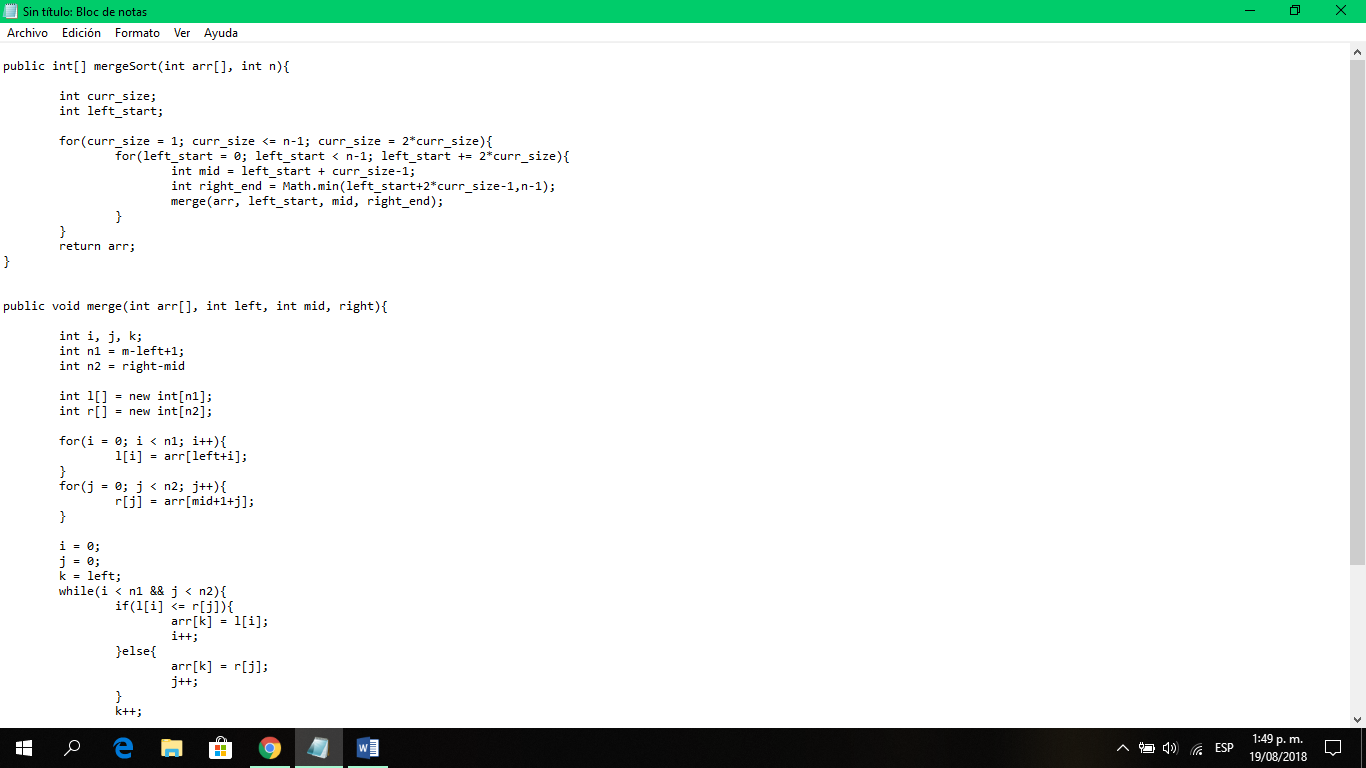
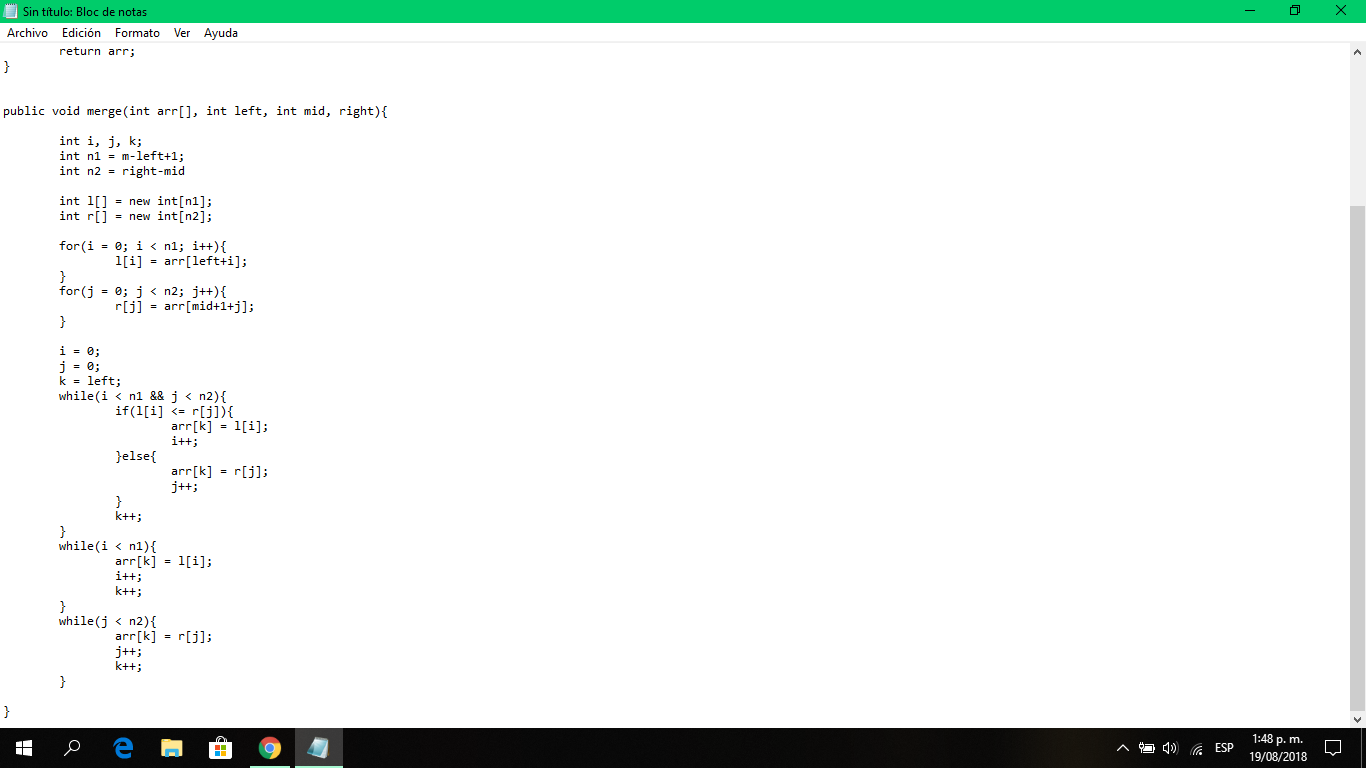
* *Ordenar un arreglo de tamaño pequeño pasado como parametro*



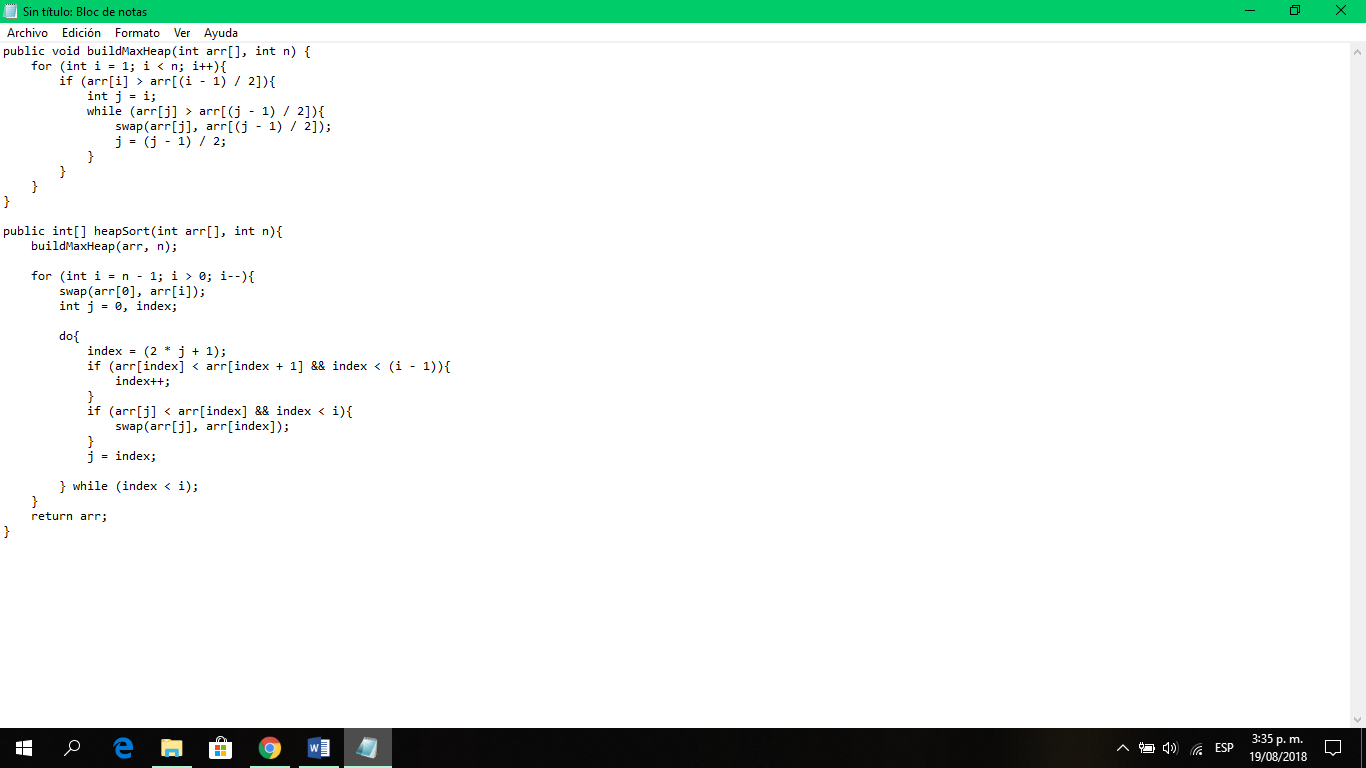


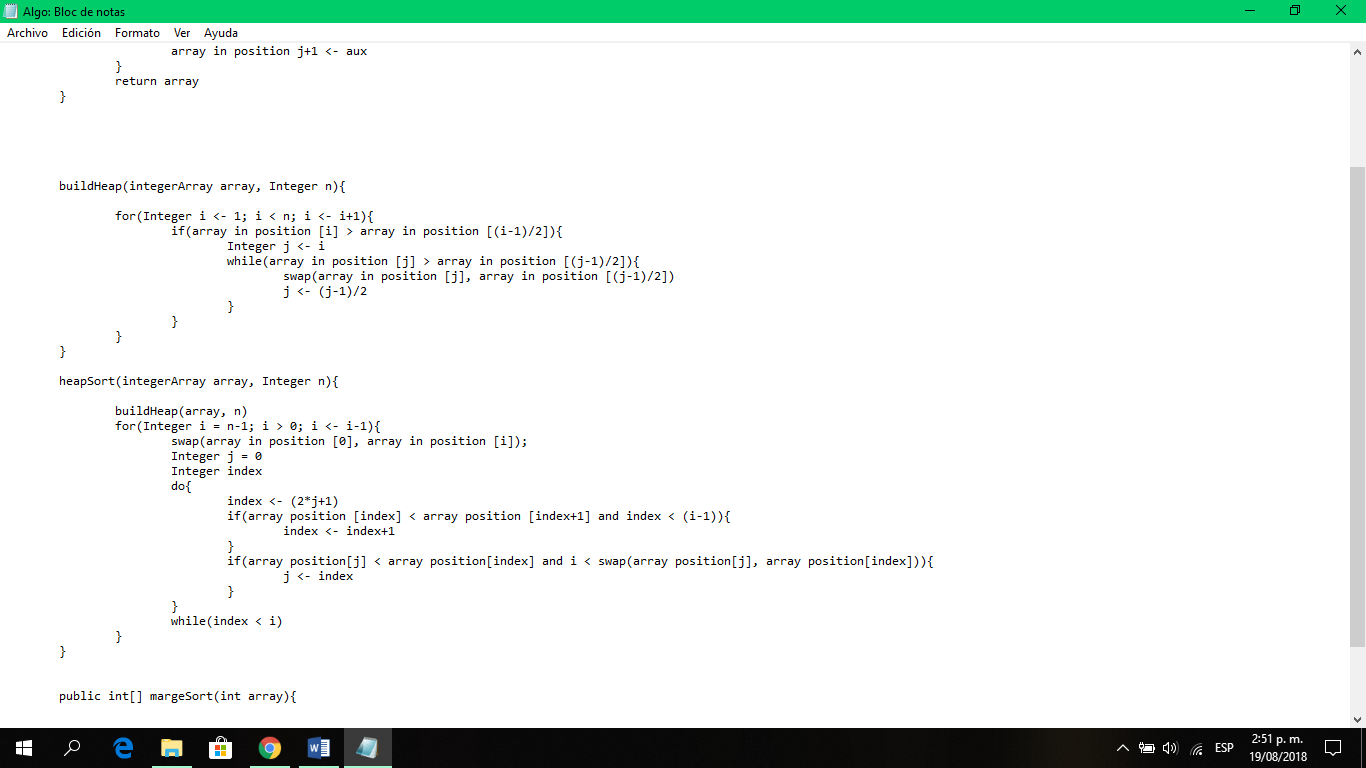
* *Ordenar un arreglo de tamaño grande con un algoritmo eficiente y estable*





* Ordenar un arreglo pasado como parámetro con un algoritmo eficiente y rápido





Fuentes:

<https://www.geeksforgeeks.org/>  
[www.wikipedia.org](http://www.wikipedia.org)  
[www.hackerearth.com](http://www.hackerearth.com)  
www.programiz.com