Diego Lamus

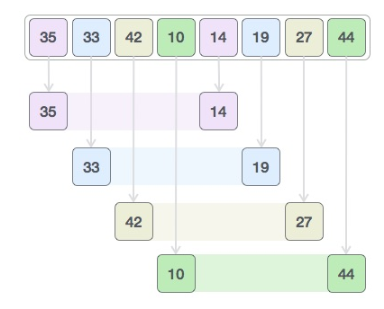
Eileen Guerrero

Análisis de algoritmos:

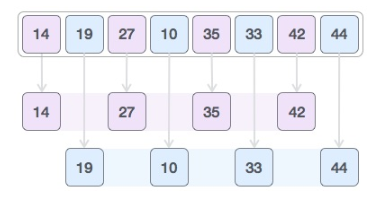
* ShellSort
* BubbleSort

**ShellSort**

El algoritmo ShellSort su forma de ordenar es tomando parejas, comparando y ordenado. Cada pareja se toma en diferentes intervalos del vector como se muestra en la **imagen 1**.









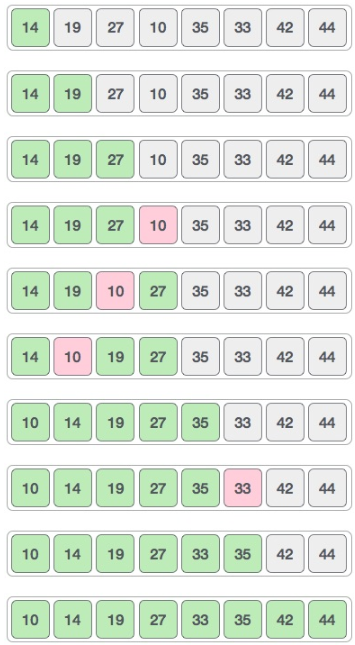


Imagen 1.

No podemos usar ninguna estrategia de divide y vencerás, es decir, particionando el problema, pues al unir las pequeñas soluciones, tendríamos nuevamente que recorrer todos los datos, pues la comparación se hace de a parejas como se muestra en el siguiente código en java.

**public** **final** String[] sort(**final** String[] cadena)

{

**int** salto, i;

String aux;

**boolean** cambios;

**for**(salto=cadena.length/2; salto!=0; salto/=2){

cambios=**true**;

**while**(cambios){ // Mientras se intercambie algÃºn elemento

cambios=**false**;

**for**(i=salto; i< cadena.length; i++) // se da una pasada

**if**(cadena[i-salto].compareTo(cadena[i])>0){ // y si estÃ¡n desordenados

aux=cadena[i]; // se reordenan

cadena[i]=cadena[i-salto];

cadena[i-salto]=aux;

cambios=**true**; // y se marca como cambio.

}

}

}

**return** cadena;

}

Es clara la dependencia de datos. Por tal razón la única estructura es SISD (Single Instruction Single Data).

**BubbleSort**

En el caso del algoritmo **BubbleSort** hay dependencia de datos y es imposible idear una forma de distribuirlo, pues el ordenamiento se hace recorriendo el arreglo de elementos de forma secuencial y comparando i con i-1, siendo i el índice con el cual me muevo dentro del arreglo.



Imagen 2.

Como este método tiene dependencia de datos, no hay ninguna estructura de instrucciones y datos que permita la implementación en paralelo.

La única forma seria con la estructura **SISD**, pues no hay forma de dividir el arreglo, ya que el ordenamiento se hace de forma secuencial.

En el siguiente código se puede apreciar la dependencia de datos.

**public** **final** String[] sort(**final** String[] cadena)

{

**int** n = cadena.length;

String temp = "";

**for**(**int** i=0; i < n; i++){

**for**(**int** j=1; j < (n-i); j++)

**if**( cadena[j-1].compareTo(cadena[j]) > 0){

//swap elements

temp = cadena[j-1];

cadena[j-1] = cadena[j];

cadena[j] = temp;

}

}

**return** cadena;

}

No hay forma de dividir el problema y efectuar ninguna estrategia en paralelo.

Si efectuamos cualquier tipo de división de los archivos para ordenar por partes, al momento de intentar unir los fragmentos, el método **BubbleSort** lo que hará será recorrer nuevamente todo el vector para ordenarlo. Por tal razón, el algoritmo **BubbleSort** no es posible desplegarlo de forma distribuida.