2.1.2 Ordenamiento por mezcla “mergesort”

Este ordenamiento es susceptible de incluirlo con Divide y Vencerás, así como también se incluye como recursivo. Este algoritmo lo que busca es dividir una lista siempre a la mitad, si la lista está vacía, o tiene un solo ítem, por definición se dice que esta ordenada. Si la lista tiene más ítems se sigue el mismo procedimiento recursivamente, partiendo a la mitad y ordenando hasta llegar a la condición de paro que es cuando tenemos una lista que no se puede partir a la mitad y que ya mencionamos esta ordenada.

Un ejemplo sencillo se muestra en la figura 2.1.2.1

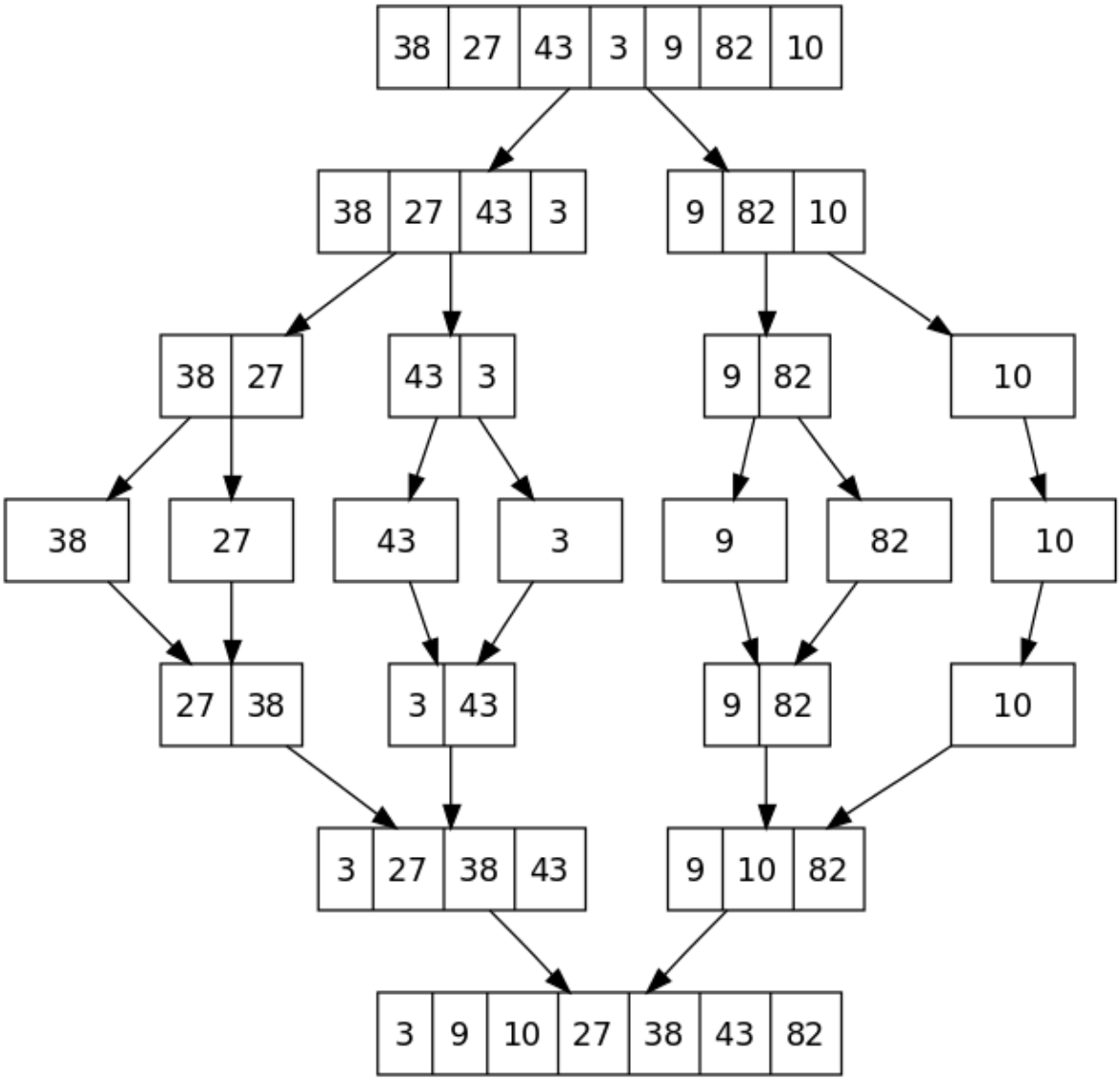

Figura 2.1.2.1 Ordenando por el algoritmo por mezcla

Este algoritmo es susceptible de ser mejorado, de tal forma que existe un algoritmo nombrado "tiled merge sort" el cuál deja de particionar las listas o arreglos cuando ha alcanzado un tamaño S (el tamaño de una página de memoria) de tal forma que para ese último particionado se emplea un algoritmo local para dejar de emplear más memoria en intercambios.

Análisis

Aunque heapsort tiene los mismos límites de tiempo que el algoritmo mergesort, requiere sólo Θ(1) espacio auxiliar en lugar del Θ(*n*) de mergesort, y es a menudo más rápido en implementaciones prácticas. Quicksort, sin embargo, es considerado por mucho como el más rápido algoritmo de ordenamiento de propósito general. Mergesort es un algoritmo de ordenamiento estable, paraleliza mejor, y es más eficiente manejando medios secuenciales de acceso lento. Mergesort es la mejor opción para ordenar una lista enlazada, es relativamente fácil implementar mergesort de manera que sólo requiera Θ(1) espacio extra, y el mal rendimiento de las listas enlazadas ante el acceso aleatorio hace que otros algoritmos (como quicksort) den un bajo rendimiento, y para otros (como heapsort) sea algo imposible.

La programación para ordenar listas de números enteros se muestra en el código 2.1.2.1.

Código 2.1.2.1 Algoritmo de ordenación por mezcla

```
/**
 *
 * @author sdelaot
 */
public class OrdenamientoPorMezcla {
    private int [] lista;
    public OrdenamientoPorMezcla( int cuantosItems ) {
        lista = new int[cuantosItems];
    }
    public OrdenamientoPorMezcla( int [] items ) {
        if( items==null ) {
            items = new int[10];
            }
        lista = items;
    }
    public void llenarArreglo() {
        int limiteSuperior = lista.length;
        for( int n=0; n<limiteSuperior; n++ ) {
            int numero = (int)(Math.random()*limiteSuperior);
            if( Math.random()*10>5 ) {
                numero *= -1;
                }
```

```
            lista[n] = numero;
            }
      }
      public void mostrarLista( String mensaje ) {
         System.out.println( " " + mensaje );
         for( int n=0; n<lista.length; n++ ) {
            System.out.print( " " + lista[n] );
            if( n!=0 && n%20==0 ) {
               System.out.println();
               }
            }
         System.out.println();
      }
      public void mostrarLista( int [] unaLista ) {
         for( int n=0; n<unaLista.length; n++ ) {
            System.out.print( " " + unaLista[n] );
            if( n!=0 && n%20==0 ) {
               System.out.println();
               }
            }
         System.out.println();
      }
      public int [] ordenarPorMezcla( int[] unaLista ) {
         int i, j, k;
         if( unaLista.length>1 ){
            int nElementosIzq = unaLista.length / 2;
            int nElementosDer = unaLista.length - nElementosIzq;
            int [] listaIzquierda = new int[nElementosIzq];
            int [] listaDerecha = new int[nElementosDer];
            // Dividir
            // Copiando de unaLista a listaIzquierda
            llenarLista( unaLista, listaIzquierda, 0, nElementosIzq );
            // Copiando unaLista a listaDerecha
            llenarLista( unaLista, listaDerecha, nElementosIzq, nElementosIzq+nElementosDer
);
            // Recursividad
            listaIzquierda = ordenarPorMezcla(listaIzquierda);
            listaDerecha = ordenarPorMezcla(listaDerecha);
            // Unir
            intercambiar( unaLista, listaIzquierda, listaDerecha );
            }
         // Muestra como se va haciendo el proceso de ordenamiento
         //this.mostrarLista(unaLista);
         return unaLista;
      }

      private void llenarLista( int[] arreglo, int[] arreglo2, int inicio, int fin ) {
```

```
        int contador = 0;
        for( int i=inicio; i<fin; i++ ){
            arreglo2[contador] = arreglo[i];
            contador++;
            }
    }

    private void intercambiar( int[] unaLista, int[] listaIzquierda, int[] listaDerecha ) {
            int i = 0;
        int j = 0;
        int k = 0;
        while(listaIzquierda.length!=j && listaDerecha.length!=k ) {
            if( listaIzquierda[j]<listaDerecha[k] ) {
                unaLista[i] = listaIzquierda[j];
                i++;
                j++;
                }
            else {
                unaLista[i] = listaDerecha[k];
                i++;
                k++;
                }
            }
        // lista final = unir listas
        while( listaIzquierda.length!=j ) {
            unaLista[i] = listaIzquierda[j];
            i++;
            j++;
            }
        while( listaDerecha.length!=k ) {
            unaLista[i] = listaDerecha[k];
            i++;
            k++;
            }
    }
    public int[] getLista() {
        return lista;
    }
    public void setLista(int[] lista) {
        this.lista = lista;
    }
}

/**
 *
 * @author sdelaot
 */
```

```
public class Ordenamiento {
   public static void main(String[] args) {
      OrdenamientoPorMezcla ordenamiento = new OrdenamientoPorMezcla( 100 );
      ordenamiento.llenarArreglo();
      ordenamiento.mostrarLista( "Lista sin ordenar" );
      int [] lista = ordenamiento.ordenarPorMezcla( ordenamiento.getLista());
      ordenamiento.setLista(lista);
      ordenamiento.mostrarLista( "Lista ordenada" );
   }
}
```

En el método ordenarPor mezcla al final del mismo encontrará dos líneas de código, ambas comentadas, una es de un comentario y otra es para que usted vea como trabaja el algoritmo, ellas son

```
// Muestra como se va haciendo el proceso de ordenamiento
//this.mostrarLista(unaLista);
```

Solo hay que descomentar, compilar de nuevo y ejecutar

```
// Muestra como se va haciendo el proceso de ordenamiento
this.mostrarLista(unaLista);
```

TAREA.

Implementar el algoritmo es fácil para cualquier tipo de dato primitivo, sin embargo, no lo es tanto cuando se trata de datos o información que viene reunida, por ejemplo, revise el siguiente diagrama de clases de la figura 2.1.2.2 y cree la aplicación que ordene al menos 40 estructuras (o como se dice en base de datos, 40 registros) empleando el algoritmo que acaba de estudiar hay que ordenarlos, se puede ordenar por nombre, paterno, materno, edad, colonia, cp, entidad federativa o municipio y/o alcaldía.

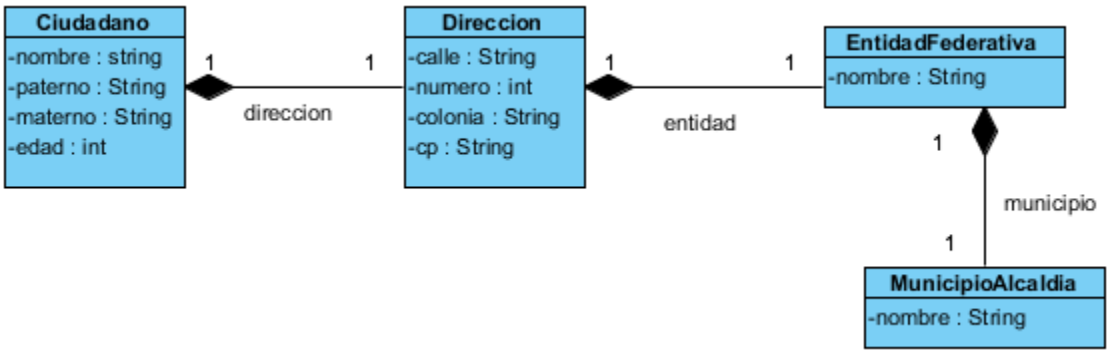

Figura 2.1.2.2 Diagrama de clases de la tarea.