El algoritmo "Mergesort" es un algoritmo de ordenamiento externo estable basado en la técnica divide y vencerás. Al aplicar "Divide y Vencerás"
lo asociamos a una estructura para una organización de tareas recursivo.

Para el caso de un mapeo directo de tares, este método se encarga de tomar como referencia el array a ordenar y ordena los elementos desde el indice menor(inclusivo) al mayor(exclusivo).
Una ordenación completa sería usando la llamada sort(A,0,A.length) siendo A el array a ordenar.

```
static void sort(final int[] A, final int lo, final int hi){
  int n = hi - lo;
  //if not large enough to do in parallel, sort sequentially
  if (n <= THRESHOLD){
    Arrays.sort(A,lo,hi);
    return;
  }
  else{//split array
    final int pivot = (hi+lo)/2;

    //create and start new thread to sort lower half
    Thread t = new Thread()
    { public void run()
      { sort(A, lo, pivot); }
  };
  t.start();

  //sort upper half in current thread
  sort(A,pivot,hi);

  //wait for other thread
  try{t. join();}
  catch (InterruptedException e){Thread.dumpStack();}

  //merge sorted arrays
  int [] ws = new int [n] ;
  System.arraycopy(A,lo,ws,0,n);
  int wpivot = pivot - lo;
  int wlo = 0;
  int whi = wpivot;
  for (int i = lo; i != hi; i++)
  { if((wlo < wpivot) && (whi >= n II ws[wlo] <= ws[whi]))
    { A[i] = ws[wlo++]; }
    else { A[i] = ws[whi++];
  }
 }
}
```

El anterior código muestra el algoritmo paralelo de mergesort donde cada tarea le corresponde a una hebra, este código se encuentra implementado en Java.

Para el caso de un mapeo indirecto de tares, se usará el framework FJTask, en lugar de crear una nueva hebra para ejecutar cada tarea se crea una instancia o subclase de FJTask. Dicho paquete mapeará dinamicamente los objetos FJTasl a un conjunto estático de subprocesos para su ejecución.
Los FJTask son más ligeros que las hebras y estos son más baratos de crear y destruir.

```
-----------------------------------------------------------
int groupSize = 4; //number of threads
FJTaskRunnerGroup group = new FJTaskRunnerGroup(groupSize);
group.invoke(new FJTask()
    { public void run()
      { synchronized(this)
        { sort(A,0, A.length); }
      }
    });
-----------------------------------------
static void sort(final int[] A,final int lo, final int hi) {
  int n = hi - lo;
  if (n <= THRESHOLD){ Arrays.sort(A,lo,hi); return; }
  else {
    //split array
    final int pivot = (hi+lo)/2;

    //override run method in FJTask to execute run method
    FJTask t = new FJTask()
    { public void run()
      { sort(A, lo, pivot); }
    }

    //fork new task to sort lower half of array
    t.fork();

    //perform sort on upper half in current task
    sort(A,pivot,hi);

    //join with forked task
    t.join();

    //merge sorted arrays as before, code omitted
  }
}
-----------------------------------------------------------
```

Ejemplo de Mergesort usando FJTask

Dentro de la documentación del FJTask se incluyen varias aplicaciones Fork/Join, como OpenMP se basa en un modelo Fork/Join se esperaría un uso intensivo de este patrón, pero no es el caso. La mayoria de sus programadores utilizan Paralelismo de Bucles ya

que
OpenMP proporciona el apoyo suficiente para el verdadero anidamiento de las regiones paralelas.

En conclusión la mayoría de los algoritmo paralelos que usan Divide y Venceras utilizan modelos Fork/Join.