**ForkJoin JDK implementation**

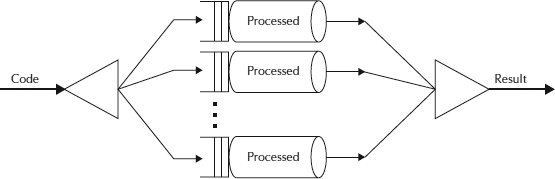
**java.util.Arrays class for its parallelSort() method**

**Stream.parallel()**

[**Fork and Join framework**](http://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html) distribute the work across multiple cores in processor and then join them to return the result set

Basically the **Fork-Join breaks the task at hand into mini-tasks** until the mini-task is simple enough that it can be solved without further breakups. It’s **like a**[**divide-and-conquer algorithm**](http://en.wikipedia.org/wiki/Divide_and_conquer_algorithms).

One important concept to note in this framework is that **ideally no worker thread is idle**. They implement a **work-stealing algorithm** in that idle workers steal the work from those workers who are busy.



1) [Core Classes used in Fork/Join Framework](http://howtodoinjava.com/java-7/forkjoin-framework-tutorial-forkjoinpool-example/#Core_Classes)

i) [ForkJoinPool](http://howtodoinjava.com/java-7/forkjoin-framework-tutorial-forkjoinpool-example/#ForkJoinPool)

ii) [ForkJoinTask](http://howtodoinjava.com/java-7/forkjoin-framework-tutorial-forkjoinpool-example/#ForkJoinTask)

The ForkJoinPool is basically a specialized implementation of ExecutorService implementing the work-stealing algorithm.

The ForkJoinPool is basically a specialized implementation of ExecutorService implementing the work-stealing algorithm.

We create an instance of ForkJoinPool by providing the target parallelism level i.e. the number of processors as shown below:

**ForkJoinPool pool = new ForkJoinPool(numberOfProcessors);**

**Where numberOfProcessors = Runtime.getRunTime().availableProcessors();**

If you use a no-argument constructor, by default, it creates a pool of size that equals the number of available processors obtained using above technique.