Project3 Fork-Join Sorting Application

问题

> Implement the preceding project (Multithreaded Sorting Application) using Java's fork-join parallelism API. This project will be developed in two different versions. Each version will implement a different divide-and-conquer sorting algorithm
>
> 1. Quicksort
> 2. Mergesort
>
> The Quicksort implementation will use the Quicksort algorithm for dividing the list of elements to be sorted into a left half and a right half based on the two evenly sized halves. For both the quicksort and Mergesort algorithms, when the list to be sorted falls within some threshold value (for example, the list is size 100 or fewer), directly apply a simple algorithm such as the Selection
> or Insertion sort. Most data structures texts describe these two well-known, divide-and-conquer sorting algorithms. The class `SumTask` shown in Section 4.5.2.1 extends `RecursiveTask`,which is a result-bearing `ForkJoinTask`. As this assignment will involve sorting the array that is passed to the task, but not returning any values, you will instead create a class that extends `RecursiveAction`, a non result-bearing `ForkJoinTask` (see Figure 4.19). The objects passed to each sorting algorithm are required to implement Java's Comparable interface, and this will need to be reflected in the class
> definition for each sorting algorithm. The source code download for this text includes Java code that provides the foundations for beginning this project.

实现

使用`Java`运行程序，已经编译完成类。

结果展示

Quicksort

快速排序原理：

1. 获得待排序数组a
2. 选取一个合适的数字p(一般来说就选取数组或是子数组的第一个元素)作为排序基准
3. 将待排序数组a中比基准p小的放在p的左边，比基准p大的放在p的右边
4. 从第3步获得的两个子数组sub1跟sub2
5. 判断sub1或sub2中是否只有一个元素，如果只有一个元素则返回此元素，否则就将sub1（或是sub2）代回到第1步中继续执行

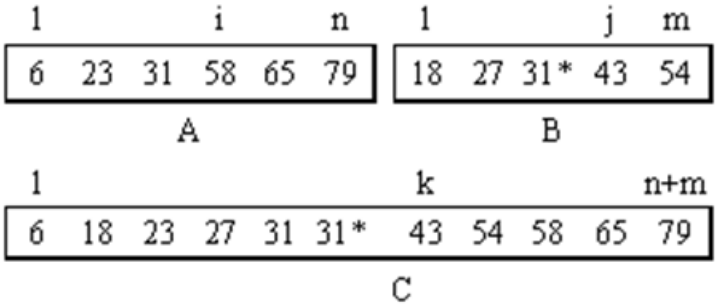
```
protected void compute() {
    if (serialThresholdMet()) {
        Arrays.sort(a, left, right + 1);
    } else {
        int pivotIndex = partition(a, left, right);
        ForkJoinTask t1 = null;
```

```
        if (left < pivotIndex)
            t1 = new Main(a, left, pivotIndex).fork();
        if (pivotIndex + 1 < right)
            new Main(a, pivotIndex + 1, right).invoke();

        if (t1 != null)
            t1.join();c
    }
}
```

Mergesort

归并将两个或两个以上的有序表合并成一个新的有序表。如下图所示，有两个已经排好序的有序表A[1]$_{A[n]和B[1]}$B[m]（在图中只给出了它们的关键字），通过归并把它们合成一个有序表C[1]~C[m+n]。



基本思想：归并（Merge）排序法是将两个（或两个以上）有序表合并成一个新的有序表，即把待排序序列分为若干个子序列，每个子序列是有序的。然后再把有序子序列合并为整体有序序列。

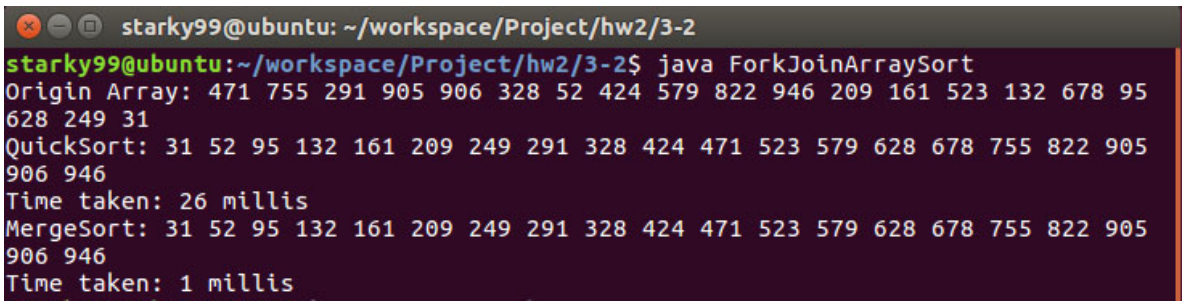
```
public static void invokeAll(ForkJoinTask<?> t1, ForkJoinTask<?> t2) {
    int s1, s2;
    t2.fork();
    if ((s1 = t1.doInvoke() & DONE_MASK) != NORMAL)
        t1.reportException(s1);
    if ((s2 = t2.doJoin() & DONE_MASK) != NORMAL)
        t2.reportException(s2);
}
```

```
@Override
protected void compute() {
    if (serialThresholdMet()) {
        Arrays.sort(a, left, right + 1);
    } else {
        int pivotIndex = partition(a, left, right);
        ForkJoinTask t1 = null;

        if (left < pivotIndex)
            t1 = new Main(a, left, pivotIndex).fork();
        if (pivotIndex + 1 < right)
            new Main(a, pivotIndex + 1, right).invoke();

        if (t1 != null)
            t1.join();
    }
}
```

结果展示


starky99@ubuntu: ~/workspace/Project/hw2/3-2
starky99@ubuntu:~/workspace/Project/hw2/3-2$ java ForkJoinArraySort
Origin Array: 471 755 291 905 906 328 52 424 579 822 946 209 161 523 132 678 95
628 249 31
QuickSort: 31 52 95 132 161 209 249 291 328 424 471 523 579 628 678 755 822 905
906 946
Time taken: 26 millis
MergeSort: 31 52 95 132 161 209 249 291 328 424 471 523 579 628 678 755 822 905
906 946
Time taken: 1 millis