# Project 3-2: Fork-Join Sorting Application

白骐硕 518030910102

## 任务概述

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:

- Quicksort
- Mergesort

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.

## 具体实现

### Mergesort

Mergesort类继承于RecursiveAction类，其中有三个属性：

- array 为待排序的数组
- begin 为数组开始位置的下标
- end 为数组结束位置的下标

在Mergesort类中，我们需要重写子线程的工作函数 `compute()`，具体细节如下文所述。

当数组长度 (end-begin) 小于阈值时，直接采用插入排序。阈值设置为5。代码如下：

```
if (end - begin < THRESHOLD) {
    // simple sort
    int tmp = 0;
    for (int i = begin+1; i <= end; i++)
    {
        tmp = array[i];
        int j;
        for (j=i-1; j>=begin && array[j] > tmp; --j ) {
            array[j+1] = array[j];
        }
        array[j+1] = tmp;
    }
}
```

若数组长度大于阈值，则采用mergesort，将原数组分成两部分分别创建一个新的Mergesort类对其进行处理，并 `fork()` 创建子线程进行执行，`join()` 两个子线程任务的结果。最后用编写好的merge函数，对两段排序好的子序列进行合并。代码如下：

```
// mergesort
int mid = (begin+end)/2;

MergeSort leftTask = new MergeSort(array, begin, mid);
MergeSort rightTask = new MergeSort(array, mid+1, end);

leftTask.fork();
rightTask.fork();

leftTask.join();
rightTask.join();

merge(begin,mid,end);
```

函数 `merge(...)` 的具体实现如下：

```
private void merge(int left, int mid, int right) {
    int temp [] = new int[right - left + 1];
    int x = left;
    int y = mid + 1;
    int z = 0;
    while (x <= mid && y <= right) {
        if (array[x] <= array[y]) {
            temp[z++] = array[x++];
        } else {
            temp[z++] = array[y++];
        }
    }
    while (y <= right) {
        temp[z++] = array[y++];
    }
    while (x <= mid) {
        temp[z++] = array[x++];
    }

    for (z = 0; z < temp.length; z++) {
        array[left + z] = temp[z];
    }
}
```

### Quicksort

Quicksort类的大体结构与Mergesort基本相同，只是在划分子序列的方式上有所不同。

Quicksort类的属性与Mergesort相同，下面主要介绍 `compute()` 的具体实现。

当数组长度小于阈值的处理方法与前述的Mergesort相同，当数组长度大于阈值时采用快速排序的方法进行分治。

```
// quicksort
int pivot = array[begin];
int k = begin+1;

for (int i = begin + 1; i<= end; ++i) {
    if(array[i] < pivot) {
```

```
            int tmp= array[i];
            array[i] = array[k];
            array[k] = tmp;
            k+=1;
        }
    }
    array[begin] = array[k-1];
    array[k-1] = pivot;

    QuickSort leftTask = new QuickSort(array, begin, k-2);
    QuickSort rightTask = new QuickSort(array, k, end);

    leftTask.fork();
    rightTask.fork();

    leftTask.join();
    rightTask.join();
```

### Main

在Main函数中，主要的工作分为三个部分：

- 从 `.txt` 文件中读取数组
- 调用Quicksort类对输入数组进行排序，并输出结果
- 调用Mergesort类对输入数组进行排序，并输出结果

首先，从文件中读入数组：

```
public static int array_size = 20;

public static int[] readArray(String filename){
    int[] array = new int[array_size];
    try{
        BufferedReader in = new BufferedReader(new FileReader(filename));
        String str;
        str = in.readLine();
        String[] strArr = str.split(" ");
        array_size = strArr.length;
        array = new int[array_size];
        int i = 0;
        for(String s : strArr){
            array[i] = Integer.parseInt(s);
            ++i;
        }
    }catch (IOException e){
    }finally{
        return array;
    }
}
```

然后，为了使用Quicksort类和Mergesort类进行多线程排序，需要创建 ForkJoinPool对象 pool 执行多线程，并使用 `pool.invoke(task)` 方法执行排序任务。（其中task即为排序类的一个实例化）

```
ForkJoinPool pool = new ForkJoinPool();
```

```
//QuickSort
QuickSort quicksort = new QuickSort(array1,0,array1.length-1);
pool.invoke(quicksort);
System.out.println("QuickSort: ");
for (int value : array1) {
    System.out.printf("%d ", value);
}
System.out.print("\n");

//MergeSort
MergeSort mergesort = new MergeSort(array2,0,array2.length-1);
pool.invoke(mergesort);
System.out.println("MergeSort: ");
for (int value : array2) {
    System.out.printf("%d ", value);
}
System.out.print("\n");
```

## 实验结果

测试文本中的待排序数组为：

6 14 11 5 3 0 7 12 15 16 10 1 8 30 45 33 18 15 10 9

结果为：

```
qsbai@qsbai-virtual-machine:~/os-prj/3/3-2$ java ForkJoin_sort.java
Original Array:
6 14 11 5 3 0 7 12 15 16 10 1 8 30 45 33 18 15 10 9

QuickSort:
0 1 3 5 6 7 8 9 10 10 11 12 14 15 15 16 18 30 33 45
MergeSort:
0 1 3 5 6 7 8 9 10 10 11 12 14 15 15 16 18 30 33 45
```