Learning objective: parallelize algorithm using the fork-join mechanism (with a divide-and-conquer approach).

Exercise 1: fork-join prediction
Let consider the following fork-join task.

```java
class RecAction extends RecursiveAction {
    int work = 0;
    RecAction(int work) {
        this.work = work;
    }
    protected void compute() {
        if (work <= 1000) {
            System.out.println("Working on " +
                    Thread.currentThread() + " with work " + work);
            Thread.sleep(1000);
        } else {
            System.out.println("Splitting work on " +
                    Thread.currentThread() + " with work " + work);
            RecAction otherAction = new RecAction(work / 2);
            otherAction.fork();

            RecAction currAction = new RecAction(work / 2);
            currAction.compute();
        }
    }
}
```

How to invoke it with a ForkJoinPool with 2 threads? What would be the expected output with input 4000? When is any thread stealing tasks?

Exercise 2: fork-join Fibonacci
Write a Fibonacci implementation relying on a fork-join pool.

Exercise 3: fork-join array increment
Implement an algorithm that increments each value of an array by one and that relies on a fork-join pool.

Exercise 4: fork-join sum of squares
Implement an algorithm that sums the squares of all values in an array using a fork-join pool.

Exercise 5: fork-join array sort
Implement the merge sort using a fork-join pool.