CS2030 Programming Methodology
Semester 1 2018/2019

2 November 2018
Tutorial 8
Java's Fork/Join Framework

1. Consider the following `RecursiveTask` called `BinSearch` that finds an item within a sorted array using binary search.

```
class BinSearch extends RecursiveTask<Boolean> {
    int low;
    int high;
    int toFind;
    int[] array;

    BinSearch(int low, int high, int toFind, int[] array) {
        this.low = low;
        this.high = high;
        this.toFind = toFind;
        this.array = array;
    }

    protected Boolean compute() {
        if (high - low <= 1) {
            return array[low] == toFind;
        } else {
            int middle = (low + high)/2;
            if (array[middle] > toFind) {
                BinSearch left = new BinSearch(low, middle, toFind, array);
                left.fork();
                return left.join();
            } else {
                BinSearch right = new BinSearch(middle, high, toFind, array);
                return right.compute();
            }
        }
    }
}
```

As an example,

```
int[] array = {1, 2, 3, 4, 6};
new BinSearch(0, array.length, 3, array).compute(); // return true
new BinSearch(0, array.length, 5, array).compute(); // return false
```

Assuming that we have a large number of parallel processors in the system and we never run into stack overflow, comment on how `BinSearch` behaves in the following situations:

i. Replace the statements

```
left.fork();
return left.join();
```

with

```
return left.compute();
```

ii. Swap the order of `fork()` and `join()`, i.e. replace

```
left.fork();
return left.join()
```

with

```
left.join();
return left.fork();
```

iii. Searching for the largest element versus searching for the smallest element in the input array.

2. Given below is the classic recursive method to obtain the n^{th} term of the Fibonacci sequence *without memoization*

$$0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$$

```
static int fib(int n) {
    if (n <= 1) {
        return n;
    } else {
        return fib(n-1) + fib(n-2);
    }
}
```

(a) Parallelize the above implementation by transforming the above to a recursive task and inherit from `java.util.concurrent.RecursiveTask`

(b) Explore different variants and combinations of `fork`, `join` and `compute` invocations.