COS125 - Precept 11 (Functions II)

1 Array Mutation

Download `precept11.zip` from the precepts webpage, unzip and open the project folder.

Open `ReverseArray.java` and implement two functions to reverse an array:

- `void reverseInPlace(int[] a)` reverses the array `a` in-place, i.e., without creating a new array; and
- `int[] reversedCopy(int[] a)` copies the values of `a` in reverse order into a new array `b`, then returns `b`.

Both functions should also handle invalid inputs and corner cases: if the argument is `null`, then they should not throw errors, and `reversedCopy()` should return `null`. Both should also work as expected when `a.length` is 0.

Also, remember to leave a comment before each function in a class!

2 Recursion

2.1 Fibonacci

The Fibonacci function (or Fibonacci sequence) $f\colon \mathbb{N} \to \mathbb{N}$ is defined as follows:

$$f(n) = \begin{cases} 0, & \text{if } n = 0; \\ 1, & \text{if } n = 1; \\ f(n-1) + f(n-2), & \text{otherwise.} \end{cases}$$

Open `Fibonacci.java` and implement two functions to compute the Fibonacci function: a recursive version, and an iterative version (that uses a `for` loop instead).

2.2 Collatz

The Collatz function $c\colon \mathbb{N}_{>0} \to \mathbb{N}_{>0}$ is defined as follows:[1]

[1] Strictly speaking, we don't know if c is a well-defined function – indeed, it is well-defined if and only if the Collatz conjecture is true (and, if so, c is identically one).

$$c(n) = \begin{cases} 1, & \text{if } n = 1; \\ c(3n+1), & \text{if } n \neq 1 \text{ is odd;} \\ c(n/2), & \text{if } n \text{ is even.} \end{cases}$$

Open `Collatz.java` and implement two functions to compute the Collatz function: a recursive version, and an iterative version (that uses a `while` loop instead).

3 Efficiency

3.1 In-place vs. copied reversion

Test `reverseInPlace()` and `reversedCopy()` with arrays of increasing size. What is the size at which your program throws an error? (And which error is it?)

Which version is more efficient, and why? Does the ratio of the elapsed times reveal something?

3.2 Recursion vs. Iteration

What is the largest integer with which you can run `fibonacciRecursive()` before it takes over 10 seconds?

How long does `fibonacciIterative()` take to terminate in that case, and what accounts for the difference?

Now, test `collatzRecursive()` and `collatzIterative()` with the number `837799`.[2] What happens? Can you figure out and fix the problem? (Hint: print the value of `n` at each iteration.)

---

[2] This is the $n \leq 1,000,000$ that requires the largest number of Collatz iterations to stabilize: 524 of them.

Bonus: Factorial and really large numbers

The order of growth of $n!$ is the largest we've seen: it beats even 2^n (and C^n for any constant C)![3] Factorials of even relatively small integers easily overflow `long`.

Open `Factorial.java` and write an iterative function to compute the factorial of a number (using a `for` loop instead of recursion). Then, replace the `int` return type of both iterative and recursive methods with `BigInteger` (and adapt the function appropriately, so it returns a `BigInteger`).

What is the largest number you can run `factorialRecursive()` with before it throws an error? (And which error is it?) How does its efficiency compare with that of `factorialRecursive()`?

---

[3] Indeed, Stirling's approximation shows that the order of growth is $\sqrt{n} \cdot (n/e)^n$, where e is the base of the natural logarithm.