Small Group Tutorial 1, 23-27/1/2017 Solutions

1. Fibonacci numbers can also be computed using the following recursive formula:

$$
\begin{aligned}
fib(0) &= 0, \\
fib(1) &= 1, \\
fib(2) &= 1, \\
fib(n) &= \begin{cases} (fib((n+1)/2))^2 + (fib((n-1)/2))^2, & n \geq 3 \text{ and } n \text{ is odd}, \\ (fib((n/2)+1) + fib((n/2)-1)) * fib(n/2), & n \geq 3 \text{ and } n \text{ is even}. \end{cases}
\end{aligned}
$$

(a) Using the above formula, calculate $fib(3)$, $fib(4)$, $fib(5)$, $fib(6)$, and $fib(11)$. Verify if the calculated value $fib(11)$ is correct by writing down the elements of the Fibonacci sequence until $fib(11)$, using the recursive formula "$fib(n) = fib(n-1) + fib(n-2)$".

Answer

Apply one of the two recursive steps, depending whether the argument is an even or odd number:

$fib(3) = (fib(2))^2 + (fib(1))^2 = 1^2 + 1^2 = 2,$
$fib(4) = (fib(3) + fib(1)) * fib(2) = (2+1) * 1 = 3,$
$fib(5) = (fib(3))^2 + (fib(2))^2 = 2^2 + 1^2 = 5,$
$fib(6) = (fib(4) + fib(2)) * fib(3) = (3+1) * 2 = 8,$
$fib(11) = (fib(6))^2 + (fib(5))^2 = 8^2 + 5^2 = 89.$

Fibonacci sequence until $fib(11)$ computed using the recursive formula"$fib(n) = fib(n-1) + fib(n-2)$":

n:	0	1	2	3	4	5	6	7	8	9	10	11
$f(n)$	0	1	1	2	3	5	8	13	21	34	55	89

(b) Write a recursive Java method

```
public static long fib(int k) { ... }
```

which computes Fibonacci number $fib(k)$ using the above recursive formula.

Answer

```
public static long fib(int k) {
   // k >= 0
   if (k <= 1) return k;        // base case: k equals  0 or 1;
   else if (k == 2) return 1; // base case: k equals 2
   else {  // check if k is even or odd and apply the recursive formula
      if ( k % 2 == 1 )
      {  // k is an odd number greater than 2
         long f1 = fib((k+1)/2);
         long f2 = fib((k-1)/2);
         return f1 * f1 + f2 * f2;
      }
      else // k is an even number greater than 2
         return  (fib(k/2 + 1) + fib(k/2 - 1)) * fib(k/2);
   }
}
```

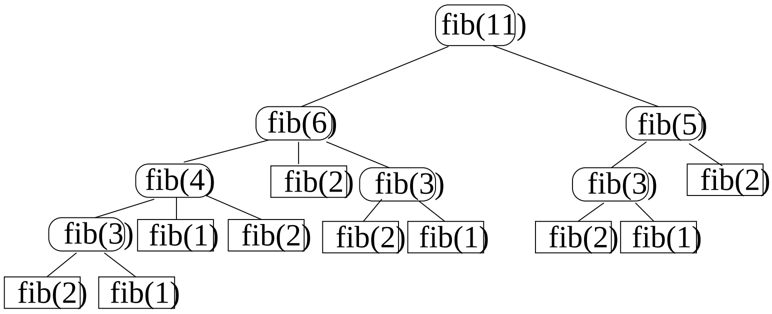
Instead of lines 8-10, we could have:

```
    return (fib((k+1)/2) * fib((k+1)/2)) + (fib((k-1)/2) * fib((k-1)/2));
```

but this way the computation of $fib((k+1)/2)$ and $fib((k-1)/2)$ would be unnecessarily repeated.

(c) Draw the recursion tree (the recursion trace) of the computation `fib(11)`.

<u>Answer</u>



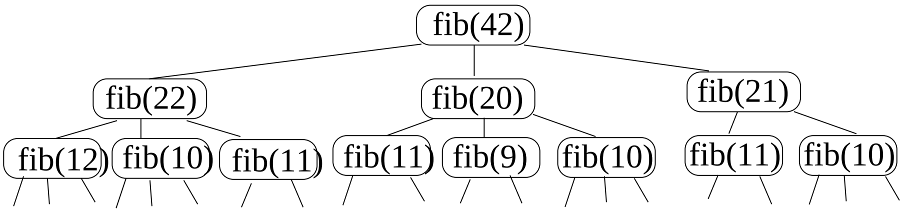
(d) In the tests of the recursive and iterative methods for computing Fibonacci numbers presented in Lecture 1, slide 43, the following computational times were observed:

```
> java FibonacciTest 42
fib(42) = 267914296 [computed iteratively in 0 ms]
fib(42) = 267914296 [computed by linear recursion in 0 ms]
fib(42) = 267914296 [computed by binary recursion in 2785 ms]
```

What do you think about the efficiency of your recursive method from Question 1b? How much time do you think this method would take to compute $fib(42)$?

<u>Answer</u>

The recursive method from Question 1b is fast, despite the fact that some computation is repeated (for example, during the computation of `fib(11)`, `fib(3)` is computed 3 times – see the recursion tree given in Question 1c). The recursion tree for the computation of `fib(42)` is almost small enough to be drawn by hand. The top part of this tree is drawn below. The 8 nodes at the lowest level shown (`fib(12)`, ...) would need to be completed with the recursion trees of the size similar to the size of the recursion tree for `fib(11)` given in Question 1c. Thus the recursion tree for `fib(42)` would be only a few times bigger than the recursion tree for `fib(11)`.



Using a test similar to the tests presented in class, we would likely measure that our new method computes $fib(42)$ in 0 ms.

2. Write a recursive method and an iterative method to determine whether a given element x occurs in the array `list` at index f or higher. The recursive method should check if x is in the array at index f (the base case), and if not, then it should recursively check if x is at index f+1 or higher.

   Compare two objects using method "`equals`," not the operator "`==`" (since we are not interested whether two objects are physically the same, but whether they represent the same data).

```java
public class Search {

  public static boolean searchRecursive (Object[] list, int f, Object x) {
    // recursive method for checking if x is in array list at index f or higher
    // GIVE YOUR CODE HERE
  }

  public static boolean searchIterative (Object[] list, int f, Object x) {
    // iterative method for checking if x is in array list at index f or higher
    // GIVE YOUR CODE HERE
  }

  public static void main(String[] args) {
    // test the methods
    for (int i = 0; i < args.length; i++) {
      System.out.print(args[i] + " ");
    }
    System.out.println();

    // check if the first argument in the command line is repeated
    System.out.println("the 1st argument is repeated: ");
    System.out.println(searchRecursive(args, 1, args[0]));
    System.out.println(searchIterative(args, 1, args[0]));
  }
}
```

Answer

```java
  public static boolean searchRecursive (Object[] list, int f, Object x) {
    // recursive method for checking if x is in array list at index f or higher
    if ( list.length <= f )  return false;       // base step
    else if ( list[f].equals(x) ) return true;     // base step
    else  return searchRecursive(list, f+1, x);    // recursive step
  }

  public static boolean searchIterative (Object[] list, int f, Object x) {
    // iterative method for checking if x is in array list at index f or higher
    for (int i = f; i < list.length; i++) {
      if ( list[i].equals(x) ) return true;
    }
    return false;
  }
```

3. Consider the following Java method:

```java
public static int geom(int x, int n) {
    // assume n is at least 0
    if ( n <= 0 ) return 1;
    else return 1 + (x * geom(x, n-1));
}
```

What is the number returned by the call $geom(2, 3)$?

What is the number returned by $geom(x, n)$?

<u>Answer</u>

The call $geom(x, 3)$ returns:

$$\begin{aligned} geom(x,3) &= 1 + x * geom(x,2) \\ &= 1 + x * (1 + x * geom(x,1)) \\ &= 1 + x * (1 + x * (1 + x * geom(x,0))) \\ &= 1 + x * (1 + x * (1 + x)) \\ &= 1 + x + x^2 + x^3, \end{aligned}$$

so $geom(2, 3)$ returns 15 $(= 1 + 2 + 4 + 8)$.

The call $geom(x, n)$ returns the value of the expression:

$$1 + x + x^2 + x^3 ... + x^n,$$

(the sum of the geometric series).