4CCS1DST – Data Structures
2016/17

Lecture 1:

Solutions to exercises

Exercise 1(a)

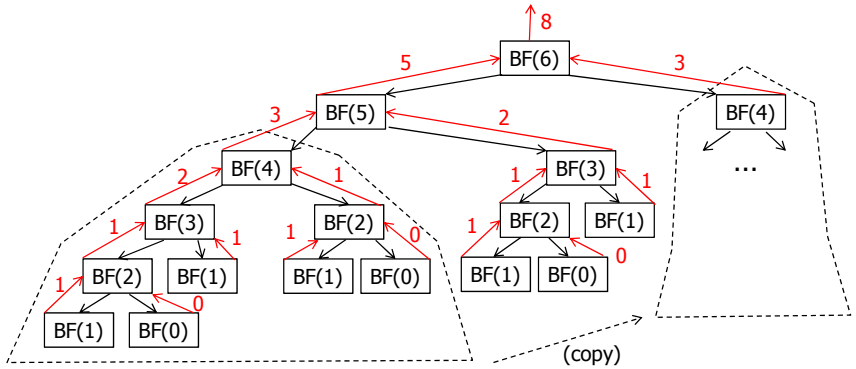
```
Algorithm BinaryFib(k):
if k <= 1 then
  return k
else
  return BinaryFib(k - 1) + BinaryFib(k - 2)
```

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

Draw the recursion trace of the call BinaryFib(6).

Show the values returned from each recursive call.

Exercise 1(a)

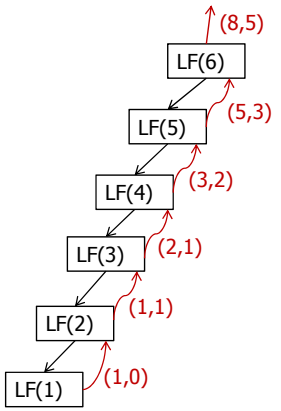


Exercise 1(b)

```
Algorithm LinearFib(k):
if k <= 1 then
        return ( k, 0 )
   else
        ( i, j ) = LinearFib(k – 1)
        return ( i + j, i )
```

Draw the recursion trace of the call LinearFib(6).

Show the values returned from each recursive call.



Exercise 2(a)

```
Algorithm ReverseArray(A, i,  j ):
if i  <  j  then
          Swap A[i] and A[ j]
          ReverseArray(A, i + 1,  j - 1)
return
```

Implement in Java.

```
public static void reverseArray(Object [ ] A, int i, int j) {
    if (i < j) {
       swap(A, i, j);
       reverseArray(A, i+1, j-1);
    }
}

private static void swap(Object [ ] A, int i, int j){
       Object  tmp = A[i];    A[i] = A[j];    A[j] = tmp;
}
```


Exercise 2(b)

```
Algorithm IterativeReverseArray(A, i, j ):
while i <  j do
    Swap A[i ] and A[ j ]
    i  = i + 1
    j  = j – 1
return
```

Implement in Java.

```
public static void iterativeReverseArray(Object [ ] A, int i, int j) {
    while ( i < j ) {
       swap(A, i, j);
       i++;  j--;
    }
}
private static void swap(Object [ ] A, int i, int j) … // as previously
```


Exercise 3(a)

$P_0 = P_1 = P_2 = 1,$

$P_n = (P_{n-3} * P_{n-1}) + 1,$ for $n \geq 3.$

Calculate P_6

$P_3 = P_0 * P_2 + 1 = 1 * 1 + 1 = 2$

$P_4 = P_1 * P_3 + 1 = 1 * 2 + 1 = 3$

$P_5 = P_2 * P_4 + 1 = 1 * 3 + 1 = 4$

$P_6 = P_3 * P_5 + 1 = 2 * 4 + 1 = \mathbf{9}$

Exercise 3(b)

$P_0 = P_1 = P_2 = 1,$

$P_n = (P_{n-3} * P_{n-1}) + 1,$ for $n \geq 3.$

Write a straight recursive Java method:

```
public  static  int  recP( int  n)
```

```
public  static  int  recP( int n )  {
                 if ( n <= 2 )  return 1;
                 else  return ( recP(n-3) * recP(n-1) ) + 1;
        }
```

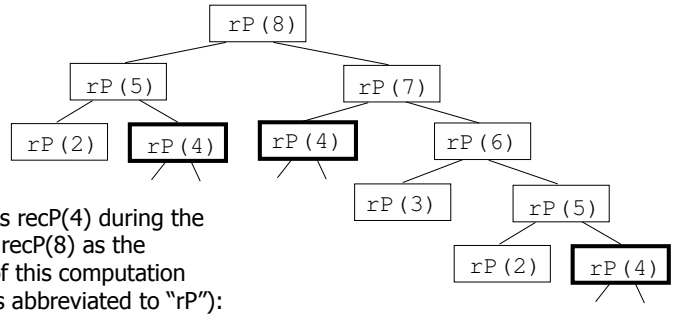
Exercise 3(c)

$P_0 = P_1 = P_2 = 1,$

$P_n = (P_{n-3} * P_{n-1}) + 1,$ for $n \geq 3.$

Consider the computation of recP(8), including all calls at all levels of recursion.

How many calls recP(4) are there during this computation?



Answer:
There are 3 calls recP(4) during the computation of recP(8) as the recursion tree of this computation shows ("recP" is abbreviated to "rP"):

Exercise 3(d)

Write an iterative (non-recursive) Java method

public static int iterP(**int** n)

which computes the number P_n.

Exercise 3(d) - cont.

```
public static int iterP(int n) {
  if (n <= 2) return 1;
  else {
    int prePrevious = 1;    // previous-previous P number (init. P(0))
    int previous = 1;       // previous P number (init. P(1))
    int current = 1;        // current P number (init. P(2))
    int next;               // next P number
    for (int i = 3; i <= n; i++) {
      // current is P(i-1); previous is P(i-2); prePrevious is P(i-3)
      next = (prePrevious * current) + 1;          // next is P(i)
      prePrevious = previous;                      // prePrevious is P(i-2)
      previous = current;                          // previous is P(i-1)
      current = next;                              // current is P(i)
      // current is P(i); previous is P(i-1); prePrevious is P(i-2)
    }
    return current;
  }
}
```


Exercise 4

- Consider the following Java method:

```
public static int tlum(int n, int m) {
    // assume both n and m are at least 1
    if ( m == 1 ) return n;
    else return (n + tlum(n, m-1));
}
```

- What are the values returned by the calls tlum(5,3) and tlum(6,15)?
- What does tlum(n, m) return?

Answer:

Expanding the recursion for the call tlum(5,3), you should observe that this method keeps adding n's (the first argument). Altogether, m n's are added, where m is the second argument. Thus this method multiplies n by m. The calls tlum(5,3) and tlum(6,15) return 15 and 90, respectively.

Exercise 5

```
public static void drawTicks(int tickLength) {
  if (tickLength > 0) {              // stop when length drops to 0
    drawTicks(tickLength - 1);       // recursively draw left ticks
    drawOneTick(tickLength);         // draw center tick
    drawTicks(tickLength - 1);       // recursively draw right ticks
  }
```

Show output for:

drawTicks(0), drawTicks(1), drawTicks(2), drawTicks(3), drawTicks(4)

[nothing]