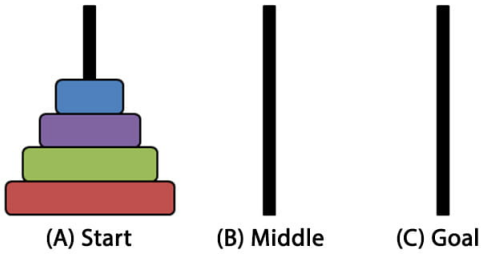
Discrete Mathematics in Computer Science

M. Helmert, G. Röger
S. Eriksson
University of Basel
Computer Science
Fall Term 2021

Exercise Sheet 9

Due: Thursday, November 25, 2021

Exercise 9.1 (2 marks)



In the Towers of Hanoi puzzle we have three pegs A, B and C, and n discs $\{1, \ldots, n\}$, where the name of the disc denotes its size. In the beginning all discs are placed on A in decreasing size, with the biggest disc at the bottom. We can move a disc i from peg X to peg Y if it is on the top of X and peg Y either has no discs or its top disc is bigger than i. The goal is to move all discs to peg C.
If we call the following algorithm with $X = A$, $Y = B$ and $Z = C$, it solves the problem recursively by first moving discs $\{1, \ldots, n-1\}$ from X to Y (using Z to temporarily store discs), moving n from X to Z and finally moving discs $\{1, \ldots, n-1\}$ from Y to Z (using X to temporarily store discs):

procedure TowersOfHanoi(n, X, Y, Z)
  if $n = 1$ **then**
    move disc n from X to Z
  else if $n > 1$ **then**
    TowersOfHanoi($n-1$, X, Z, Y) ▷ move discs $\{1, \ldots, n-1\}$ from X to Y
    move disc n from X to Z
    TowersOfHanoi($n-1$, Y, X, Z) ▷ move discs $\{1, \ldots, n-1\}$ from Y to Z
  end if
end procedure

The number of moves needed by the algorithm to solve a Towers of Hanoi puzzle with n discs can be described by the following recursive function:

$$moves(0) = 0$$
$$moves(n) = 2 \cdot moves(n-1) + 1$$

Specify $moves(n)$ in closed form and prove your answer.

Exercise 9.2 (3 marks)

(a) Consider a naive recursive implementation of $F(n)$ that computes the Fibonacci numbers by returning 0 for $n = 0$, returning 1 for $n = 1$, and returning $F(n-1) + F(n-2)$ otherwise. How many times is function F called when computing $F(n)$ for $n > 1$? You do not need to justify your answer.

(b) Memoization is an optimization technique for recursive functions where the intermediate results are stored in a table to avoid recomputing them again:

```
procedure RecursiveFunction(x, table)
    if base case then
        return base value
    else if table contains entry for x then
        return table[x]
    else
        table[x] ← recursive call(s) to RecursiveFunction
        return table[x]
    end if
end procedure
```

How does your answer to (a) change, if $F(n)$ uses memoization? You do not need to justify your answer.

(c) Assume each call to F takes one nanosecond excluding the time needed for recursive calls. Specify for $n = 10$, $n = 40$ and $n = 50$ how much time is spent in total to compute $F(n)$ without and with memoization.

Hint: To answer (a) and (b) we recommend to implement the function in your favourite programming language with a function call counter. This way you can get some data points on how many function calls are needed for a specific n. From this you can then find a general pattern.
An implementation is however not mandatory and should not be handed in (it will not be graded).

Exercise 9.3 (3 marks)

We consider a Fibonacci-like sequence that is defined as follows:

$$\begin{aligned} F'(0) &= 4 \\ F'(1) &= 2 \\ F'(n) &= F'(n-1) + F'(n-2) \text{ for } n \geq 2 \end{aligned}$$

The goal of this exercise is to derive a closed form expression of F' using the strategy on slide 9 (handout version) of Chapter D2. Since one error can have a significant impact we split the exercise into three parts and for the latter two provide the solution to the previous task. In order to get marks for a subtask you thus need to provide the full calculation process.

(a) Derive an equation for the generating function g and simplify it as much as possible (in particular, the simplified equation should no longer contain an infinite sum).

As in the lecture, you can assume without proof that the power series of the F'-numbers converges for $|x| < \varepsilon$ for sufficiently small values of $\varepsilon > 0$.

(b) Solve the equation $g(x) = x^2 g(x) + xg(x) - 2x + 4$ for $g(x)$.

(c) Derive the power series representation for $g(x) = \frac{4-2x}{1-x-x^2}$ by using partial fractial decomposition and from this specify the closed form of L.

Hint: Unlike for the Fibonacci series, you can use the partial fractial decomposition directly on $g(x)$.

Exercise 9.4 (2 marks)

The following recursive algorithm computes the largest element of a given array when using the index of the first array element as parameter *left* and using the index of the last array element as parameter *right*.

```
procedure MAX(array, left, right)
    if left = right then
        return array[left]
    else
        middle ← ⌊(left + right)/2⌋
        max1 ← MAX(array, left, middle)
        max2 ← MAX(array, middle + 1, right)
        if max1 > max2 then
            return max1
        else
            return max2
        end if
    end if
end procedure
```

Compute the asymptotic runtime of MAX with the Master theorem (slide 27 handout version, Chapter D2).

Submission rules:

Upload a single PDF file (ending .pdf) generated using LaTeX. Put the names of all group members on top of the first page. Use page numbers or put your names on each page. Make sure your PDF has size A4 (fits the page size if printed on A4).