Name: ____

Algorithms and Data Structures
Jacobs University Bremen
Dr. Florian Rabe

Quiz 7
given: 2017-05-16

You have 20 minutes.

Problem 1

Points: 5+2+3

Consider the following recursive function:

```
fun $product(x:\mathbb{Z}):\mathbb{Z}$ =
    if $x == 0$
        $1$
    else
        $x \cdot product(x-1)$
```

1. Convert it to a dynamic program by completing the blanks below.
2. Give the Θ-class of the space complexity of the resulting dynamic program.
3. Explain in which situations a dynamic program performs better than the recursive variant. Does it matter in this case?

```
fun $product(x:\mathbb{Z}):\mathbb{Z}$ =
    $results$ := new $Array[\mathbb{Z}](x)$
    for $i$ in $0,\dots,x$

        ___________

    $results$[___________]
```

Solution:

1.

```
fun $product(x:\mathbb{Z}):\mathbb{Z}$ =
    $results$ := new $Array[\mathbb{Z}](x)$
    for $i$ in $0,\dots,x$
        if $i == 0$
            $results[i] := 1$
        else
            $results[i] := results[i-1] \cdot i$
    $results[x]$
```

2. $\Theta(x)$ (dynamic programs have high space complexity because they have to store all intermediate results for later use)
3. Dynamic programs perform better than the recursive variant if recursions into other arguments than the next-smaller one are needed (e.g., Fibonacci). In this case, it does not matter because we only recurse into the next-smaller argument, i.e., $x-1$.

Name: ______________________________

Problem 2

Points: 3+2+5

Assume a fixed list D of denominations of coins, sorted decreasingly. We want to find the smallest set of coins whose denominations add up to n.

We use the following greedy-style algorithm:

```
                                                        precondition: $D$ is sorted by $\geq$
fun $greedyCoinChange(D : List[\mathbb{Z}], n : \mathbb{Z}) : List[\mathbb{Z}]$ =
    $solution := Nil$
    for $x$ in $D$
        while $x + sumList(solution) \leq n$
            $solution := prepend(x, solution)$
    $solution$

fun $sumList(l : List[\mathbb{Z}]) : \mathbb{Z}$ =
    return the sum of all elements in $l$
```

1. Give the result for running $greedyCoinChange([50, 20, 10, 5, 2, 1], 14)$.
2. For fixed D, the algorithm's run time is $\Theta(n^2)$. Give a simple improvement that makes the run time $\Theta(n)$.
3. The algorithm is not correct in general. Give an example where $greedyCoinChange(D, n)$ returns a non-optimal result.

Solution: Note that this is not quite an instance of the generic greedy algorithm from the lecture notes: it is a variant where every element can be picked arbitrarily often. Thus, our matroid theorem is useless here.

1. $[10, 2, 2]$
2. The quadratic behavior is caused by the function *sumList*, which takes linear time. As typical for greedy algorithms, the direct implementation can still be optimized. In this case, we see that we keep adding elements to *solution* and then call $sumList(solution)$, which requires redoing the previous summation. We optimize by keeping an additional variable in which we store the sum:

```
                                                  precondition: $D$ is sorted by $\geq$
fun $greedyCoinChange(D : List[\mathbb{Z}], n : \mathbb{Z}) : List[\mathbb{Z}]$ =
    $solution := Nil$
    $sum := 0$
    for $x$ in $D$
        while $x + sum \leq n$
            $solution := prepend(x, solution)$
            $sum := x + sum$
    $solution$
```

   To prove correctness, we use the loop invariant $sum = sumList(solution)$.

3. $greedyCoinChange([10, 9, 1], 18)$ returns $[10, 1, 1, 1, 1, 1, 1, 1, 1]$ instead of the optimal $[9, 9]$. (Using 0 in D makes the algorithm run forever, which also violates the specification, but it is not technically a correct answer to the question. Using a non-sorted D is not a correct answer because algorithms are allowed to do anything on input that violates the precondition.)