Problem Set 9

Question 1. Compare and contrast dynamic programming and the divide-and-conquer technique.

Question 2. The Python-code for the coin-collecting robot from lecture:

```
def coin_robot(C):
    F = np.zeros((C.shape[0]+1,C.shape[1]+1))
    for i in range(1, F.shape[0]):
        for k in range(1, F.shape[1]):
            above = F[i-1,k]
            left = F[i,k-1]
            F[i,k] = max(above, left)+C[i-1,k-1]

    return F[C.shape[0], C.shape[1]]
```

(a) What is the (worst-case) time and space complexity of the algorithm?

(b) Augment the code for the coin-collecting robot so that it also outputs the path.

Question 3. *Longest path in a DAG* (Levitin 8.1.10)

(a) Design an efficient algorithm for finding the length of the longest path in a DAG (Directed Acyclic Graph).

(b) Show how to reduce the coin-row problem from the lecture to the problem of finding the longest path in a dag.

Question 4. A palindrome is a non-empty string over some alphabet that reads the same forward and backward. Examples of palindromes are: all strings of length 1, civic, ABBA, racecar, and aibohphobia (fear of palindromes). A *subsequence* of a string s is a string that can be derived from s by deleting some elements without changing the order of the remaining elements. E.g., `acd` is a subsequence of `abcde`. We want to solve the following problem: given a string s of length n, find the **length** of the longest palindrome that is a subsequence of s. For example, given the input `character`, the output should be 5, i.e., the length of `carac`.

(a) What is the simplest algorithm you can think of? What is its complexity? (writing the algorithm explicitly is not required).

(b) Let $L[i, j]$ be the length of the longest palindrome of the sub-string $s[i, \ldots, j]$. Explain how $L[i, j]$ can be recursively computed.

Hint: If $s[i] = s[j]$, we have found a palindrome subsequence of length 2, then we can look for palindromes in $s[i + 1, \ldots, j - 1]$.

(c) We want to use bottom-up dynamic programming to solve the problem in $O(n^2)$ time. Give a non-recursive, bottom-up pseudocode algorithm that returns the length of the longest palindrome subsequence of a given string s. The algorithm should rely on the bottom-up computation of $L[i, j]$ of point b).