Assignment 2 **Julien Corman, Werner Nutt**
Flavio Vella

Recursion and Algorithms for Arrays

Instructions: This coursework counts 200%, since there will be no lectures and labs during the week from 22 to 26 October.
The coursework consists of 4 questions, each of which requires a conceptual answer and a piece of Java code. The conceptual answers must be submitted in PDF format via OLE. The modalities for code submission are identical to Coursework 1, namely:

- All exercises must be implemented as static methods. Method stubs are available on Codeboard. Your implementation must respect the methods' signatures (but of course, you are free to create and call additional methods if needed).
- The code must be submitted via Codeboard, and a copy of it (.java file) via OLE.

1. Recursion

A palindrome is a word or phrase that reads the same forwards and backwards (examples: 'racecar', 'radar', 'noon').
By extension, we call a palindrome any string that reads the same from left to right and from right to left. For this exercise, we require the forward and backward strings to be identical. So 'rats live on no evil star' is considered as a palindrome, but 'Racecar' or 'never odd or even' are not.
Develop a recursive algorithm that takes as input a string and decides whether the string is a palindrome.

1. Write down your algorithm in pseudocode.
2. Implement your algorithm (method `isPalindrome` of class `Assignment2`).

(Weight: 30% of this CW)

2. Maximal Length of Ascents in Arrays

Consider an array $A[1..n]$ of integers. A *subarray of A* is a contiguous segment of A. We denote the subarray from position k to position l (both included) as $A[k..l]$.
The subarray $A[k..l]$ is an *ascent* if $A[j] \leq A[j+1]$ for all j where $k \leq j < l$. In other words, an ascent is a nondecreasing segment of A.
We want to compute the maximal length of an ascent in A. For instance, for the array $A = [3, 1, 4, 2, 4, 4, 5, 3]$, the maximal length of an ascent would be 4, because the subarray $A[4..7] = [2, 4, 4, 5]$ is the longest ascent in that array.
We are interested in an *efficient* algorithm.

1. Write down pseudocode for an function that takes an array A of integers as input and returns the maximal length of an ascent in A.
2. Explain why your algorithm is correct.
3. Implement your algorithm (method `computeMaxAscentLength` of class `Assignment2`).

(Weight: 40% of this CW)

3. Dividing Arrays into Segments

Let A be an array of integers. We want to rearrange the numbers in A in such a way that all even numbers come before the odd numbers. For instance, for the input array

$$[3, 2, 5, 7, 4, 6, 9, 11, 8]$$

a possible rearrangement is

$$[2, 4, 6, 8, 3, 5, 9, 11, 7].$$

Another possible rearrangement is

$$[6, 2, 8, 4, 9, 3, 5, 7, 11].$$

We say that an algorithm is *in-place* if it does not use additional data structures, except for a fixed number of additional integer variables.

1. Write pseudocode for an efficient in-place procedure that takes as input an integer array A and rearranges it in the way described above.
2. Explain why your algorithm is correct.
3. If the array A has length n, how many assignments will your algorithm perform in the worst case? Which is the worst case?
4. Implement your algorithm (method `divideInTwo` of class `Assignment2`).

(Weight: 50% of this CW)

4. Array of Averages

Design an *efficient* algorithm that achieves the following task. Given an array $A[1..n]$ of floating point numbers, it returns a two-dimensional array, say M, of size $n \times n$, in which the entry $M[i][j]$ for $i \leq j$ contains the *average* of the array entries $A[i]$ through $A[j]$ (both included). That is, if $i \leq j$, then

$$M[i][j] = \frac{A[i] + \cdots + A[j]}{j - i + 1},$$

whereas for $i > j$ we have that $M[i][j] = 0$.
For instance, for the array

$$A = [2, 3]$$

the output matrix (2 dimensional array) must be

$$M = [[2, 2.5], [0, 3]]$$

The explanation is the following:

$$\begin{aligned}
M[1][1] &= A[1]/1 = 2 \\
M[1][2] &= (A[1] + A[2])/2 = 2.5 \\
M[2][1] &= 0 \text{ (because } 1 > 0) \\
M[2][2] &= A[2]/1 = 3
\end{aligned}$$

1. Describe your idea for an algorithm that creates this matrix.
2. Write down the algorithm in pseudocode.
3. How many assignments will your algorithm perform for an input of size n?
4. Implement your algorithm (method `computeMatrixOfAverages` of class `Assignment2`).
5. Run experiments to check whether the actual running time of your implemented algorithm corresponds to your analysis of the number of assignments. Write code to measure the running time of your Java program for random inputs of size $n = 10$, 100, 1000, etc. Report on your experiments and findings.

(Weight: 80% of this CW)

Deliverables. Submit two copies of your code:

- one via Codeboard (instructions are available here),
- one via the OLE submission page of your lab (together with the other deliverables).

The other questions must be answerd in a PDF document.
Combine all deliverables into one zip file, which you submit via the OLE submission page of your lab. Please include name, student ID and email address in your submission.

Submission until Thursday, 1 November 2018, 23:55, to Codeboard and the OLE submission page of:

Lab A / Lab B