ST444: Exercise Sheet 2

Part I: Python

1. Read in an integer n, then decide if it is within 10 of 100 or 200.
2. Read in an integer n (say from 1 to 4000), then decide if it represents a leap year or not.
3. We want make a package of M (integer) kilos of chocolate. We have small bars (1 kilo each), medium bars (5 kilos each) and big bars (10 kilos each). Read in the integer M and print the minimum number of bars to use. Here you can take for granted that the greedy algorithm works.

   [Bonus: prove that the greedy approach works in this setting.]

Part II: Basic algorithms

1. Given n observations $X_1, \ldots, X_n$, how many basic operations are needed to find the sample variance? What is its complexity using the Big-O notation?
2. Given an unsorted list of distinct integers $X_1, \ldots, X_n$ and a new integer X_{n+1}, how many basic operations are needed to determine whether X_{n+1} has already appeared in the list before? What is its complexity in the worst case in terms of the Big-O notation?
3. Now given a *sorted* list of distinct integers $X_1, \ldots, X_n$ and a new integer X_{n+1}, how many basic operations are needed to determine whether X_{n+1} has already appeared in the list before (and if yes, at which position)? Does the fact that the list is sorted make your algorithm faster? What is its complexity in the worst scenario in terms of the Big-O notation? Is your proposed algorithm optimal?
4. Given n positive integers $X_1, \ldots, X_n$, find a dynamic programming algorithm that tells us if it is possible to divide them into two sets with equal sums.
5. ($\star$) Recall the matrix multiplication example (i.e. Example 6) from Lecture 1. Let n be an positive integer, and let $a_1, \ldots, a_{n+1}$ be $n+1$ positive integers. Suppose that you are given n matrices, where the i-th matrix $\mathbf{B}_i$ is $a_i \times a_{i+1}$. How to determine the optimal parenthesization of a product of these n matrices, $\mathbf{B}_1\mathbf{B}_2 \cdots \mathbf{B}_n$?
6. ($\star$) Revisit the closest pair of points problem and understand its solution by Divide and Conquer (Wikipedia/google should provide you with a good starting point for your search). What is its complexity in terms of the Big-O notation?