Algorithms and Data Structures II (1DL231)
Uppsala University – Autumn 2017
Assignment 1

Prepared by Pierre Flener, Carlos Pérez Penichet, Huu-Phuc Vo, and Kjell Winblad

— Deadline: 13:00 on Friday 2017-11-17 —

It is strongly recommended to read the *Submission Instructions* and *Grading Rules* at the end of this document ***before*** attempting to solve the following problems.

Problem 1: The Birthday Present Problem (*mandatory*!)

Given a set P of prices for items in a gift shop, as well as a total amount t of money collected by a group of people towards buying birthday presents for a common friend, the birthday present problem is to determine whether there exists a subset $P' \subseteq P$ whose sum is exactly t, since we do not want to distribute any excess money back to the donors. For example, if $P = \{1, 7, 2, 32, 56, 35, 234, 12332\}$ and $t = 299$, then the subset $P' = \{2, 7, 56, 234\}$ is a solution, but there is no solution for $t = 11$. Perform the following tasks:

a. Give a recursive equation for the required Boolean. Use it to justify that dynamic programming is applicable to the birthday present problem.

b. Design and implement an efficient dynamic programming algorithm for the birthday present problem as a Python function $\mathit{birthday\_present}(P, n, t)$ that returns *True* if and only if there exists a subset P' with sum $t \geq 0$ of the set P, which is given as an array of n non-negative integers.

c. Extend your function from task b to return such a subset P' as a list, empty if none exists. Implement the extended algorithm as a Python function $\mathit{birthday\_present\_subset}(P, n, t)$.

d. Compute the worst-case time complexity of your (extended) algorithm.

Solo teams may omit task c.

Problem 2: Integer Sort

It can be proven that, for an arbitrary array of n elements, *any* sorting algorithm takes $\Omega(n \cdot \lg n)$ time in the worst case. In this respect, algorithms like MERGE-SORT are optimal. On the other hand, if the array to be sorted satisfies some additional properties, then asymptotically faster algorithms may exist. For example, consider the following algorithm for the in-place sorting of an array A of n elements, each one known in advance to belong to the *integer* interval $0 \mathinner{.\,.} k$:

a. Create an auxiliary integer array $Y[0 \mathinner{.\,.} k]$ and initialise it with zeros.

b. Scan A for all indices i: if $A[i] = x$, then increment $Y[x]$ by 1.

c. Scan Y for all indices x: if $Y[x] = t > 0$, then record value x a total of t times into A.

Let us call this algorithm INTEGER-SORT. Perform the following tasks:

a. Implement INTEGER-SORT as a Python function $integer\_sort(A, k)$. The elements of A are assumed to belong to the integer interval $0\,..\,k$. The resulting array must be non-decreasingly ordered.

b. Compute the worst-case time complexity of INTEGER-SORT. Note that the input is characterised by *two* numbers, namely $|A|$ and k.

c. Compute the worst-case time complexity of INTEGER-SORT when $k = \mathcal{O}(|A|)$.

Solo teams must perform all tasks.

Problem 3: Binary Multiplication

Consider the problem of computing the product of two integers, which are represented in binary notation as two arrays A and B of n binary digits in the set $\{0, 1\}$. Let the product be represented by an array P of $2 \cdot n$ binary digits. Let the rightmost element of every array represent the least significant digit. Perform the following tasks:

a. Following the divide-and-conquer approach, design and implement a *recursive* efficient algorithm for this problem as a Python function $binary\_mult(A, B)$.

   Divide the two given arrays into two halves each, recursively multiply the halves in a proper fashion, and then combine the partial results using summations and shifts: two points will be deducted from your score if your algorithm is more complicated than that.

b. Make an educated guess at the worst-case time complexity of your algorithm, using the recursion tree method.

c. Check your guess of task b by using the substitution method. If there is a contradiction, then return to task b.

d. Discuss whether the Master Theorem is applicable to your algorithm. If yes, then apply it and check whether its answer is the same as the one you obtained in tasks b and c.

Solo teams may omit tasks b and c.

Submission Instructions

All task answers, including documented source code, ***must*** be in a ***single*** report in ***PDF*** format; all other formats are rejected. Furthermore:

- Identify the team members and state the team number inside the report.
- Certify that your report and all its uploaded attachments were produced solely by your team, except where explicitly stated otherwise and clearly referenced, that each teammate can individually explain any part starting from the moment of submitting your report, and that your report and attachments are not and will not be freely accessible on a public repository. Without such an honour declaration, your report will ***not*** be graded.
- State the problem number and task identifier for each answer in the report.

- Take Section 1 of the demo report at `http://user.it.uu.se/~pierref/courses/AD2/demoReport` as a ***strict*** guideline for document structure and content, as well as an indication of its expected quality.

- Comment each function according to the AD2 coding convention at the *Student Portal*.

- Write clear task answers, source code, and comments.

- Justify all task answers, except where explicitly not required.

- State any assumptions you make that are not in this document. Any legally re-used help function of Python can be assumed to have the complexity given in the textbook, even if an analysis of its source code would reveal that it has a worse complexity.

- Thoroughly proofread, spellcheck, and grammar-check your report.

- Match exactly the uppercase, lowercase, and layout conventions of any filenames and I/O texts imposed by the tasks, as we will process your source code automatically.

- Make a sanity check of all your Python functions on *random* tests via the server at `http://ad2checker.it.uu.se`. Beware: passing random tests does ***not*** guarantee that your code will pass the *boundary* and *stress* tests of our grading test suite (see below), hence you are ***strongly*** encouraged to devise your own *boundary* and *stress* tests.

Only ***one*** of the teammates submits the solution files (one PDF report with answers to ***all*** the tasks, plus up to three commented Python source code files, which are ***also*** imported into the report), ***without*** folder structure and ***without*** compression, via the *Student Portal*, whose clock may differ from yours, by the given ***hard*** deadline.

Grading Rules

For each problem: ***If*** the requested source code exists in a file with exactly the name of the corresponding skeleton code, ***and*** it depends only on the libraries imported by the skeleton code, ***and*** it runs without runtime errors under version 2.7.∗ of Python (use `python`), ***and*** it produces correct outputs for some of our grading tests in reasonable time on the Linux computers of the IT department, ***and*** it has the comments prescribed by the AD2 coding convention at the *Student Portal*, ***and*** it features a serious attempt at algorithm analysis, ***then*** you get at least 1 point (of 5), ***otherwise*** you get 0 points. Furthermore:

- If your function has a ***reasonable*** algorithm and ***passes most*** of our grading tests, ***and*** your report is ***complete***, then you get 4 or 5 points, depending also on the quality of the Python source code comments and the report part for this problem; you are not invited to the grading session for this problem.

- If your function has an ***unreasonable*** algorithm *or* ***fails many*** of our grading tests, ***or*** your report is ***incomplete***, then you get an initial mark of 1 or 2 points, depending also on the quality of the Python source code comments and the report part for this problem; you are invited to the grading session for this problem, where you can try and increase your initial mark by 1 point into your final mark.

However, ***if*** the coding convention is insufficiently followed ***or*** the assistants figure out a minor fix that is needed to make your source code run as per our instructions above, ***then***, instead of giving 0 points up front, the assistants may at their discretion deduct 1 point from the mark earned otherwise.

Considering that there are three help sessions for each assignment, you must get minimum 5 points (of 15) on each assignment until the end of its grading session, including minimum 1 point (of 5) on each mandatory problem, and minimum 23 points (of 45) over all three assignments; otherwise you fail the *Assignments* part of the course.