# COMP26120
# Academic Session: 2018-19

# Lab Exercise 1: Algorithm Design Workout

Duration: 1 lab session (See online for policy on deadlines and extensions)

You should do all your work in your COMP26120/ex1 directory. You will produce a text file named *answer.txt* and submit this text file in this directory using the `submit` command. See the course website for further details on submission and marking of coursework.

## Learning Objectives

At the end of this lab you should be able to:

1. Explain why we use pseudocode as a language-independent method for describing algorithms.

2. Use pseudocode to represent algorithms and explain what given pseudocode means in English.

3. Design algorithms for some simple problems.

4. Describe one or more generic techniques for algorithmic problem-solving, such as divide-and-conquer.

5. Reason (informally) about the correctness and time complexity of algorithms e.g. to present reasoned arguments for each based on a description of the algorithms execution

## Introduction

This lab is about examining problems, devising solutions to those problems, and then analysing those solutions. You are encouraged to be *creative* as well as thinking *analytically* about the problem and your solutions. There is never only one way to solve a problem, although there may be an optimal solution (however, it is not necessary to find this solution). **It is important to remember that you should be providing your answers in pseudocode and not any real programming language.**

Select any **one** of the following problems. For your selected problem, give

1. **Two** algorithms for solving the problem, in **pseudocode**. The first algorithm can be the first thing you think of that works. The second one should be substantially different to the first and should improve upon the first one (i.e. it should use fewer operations).
2. A description why **each** algorithm is correct.
3. A description of the number of operations each algorithm uses, explaining why you think this is the **worst case**. Your answer should be in terms of $n$ unless stated otherwise in the problem.

**Please try to think these up for yourself before scouring the web.** If you do find the solution online or in a book, please acknowledge this (you will not lose marks). Try and learn from the way the solution was constructed. *You will need to be able to explain your submitted solutions.*

**You should write your answer in a flat text file called *answer.txt* and put it in your *COMP26120/ex1* directory. Clearly state the letter of the problem you have chosen to tackle.**

# Problem sets

Clearly some of these problems are more difficult than others. There is no extra points for difficultly, although some of the more challenging problems may have more obviously different solutions. Once you have devised solutions for one problem you may want to have a go at another **but you may only submit one solution**.

---

## A. Find the *fixed point* of an array

**Input:** an array A of **distinct** integers in ascending order. (Remember that integers can be negative!) The number of integers in A is $n$.
**Output:** one position $i$ in the list, such that A[$i$]=$i$, if any exists. Otherwise: "No".

**Hint:** for your second algorithm, you may like to read up on "binary search".

---

## B. Majority Element

**Input:** An array of integers A, of length $n$.
**Output:** An integer $k$ such that $k$ appears in more than half of the positions of A if such a $k$ exists. Otherwise "No".

**Hint**: For the second algorithm you could consider trading space for time.

---

## C. Greatest common divisor

**Input:** Two positive integers $u$ and $v$, where $u > v$
**Output:** The greatest number that divides both $u$ and $v$

**Hint:** if you get stuck for a second algorithm, look up Euclid's algorithm. (But this does not need to be one of your methods).

**Note: The complexity of your algorithms should be expressed in terms of $u$ for this problem.**

---

## D. Computing Statistics

**Input:** An array of integers A, of length $n$
**Output:** The *mean*, *variance*, and *standard deviation* of the values in A

**Hint:** It is possible to do this in a single pass using a *recurrence relation*

---

## E. Choose Without Replacement

**Input:** An array of integers A, of length $n$ and a value $k$ such that $k \neq n$.
**Output:** $k$ unique items from A chosen randomly *(without bias)* i.e. select $k$ things from $n$ without replacement.

**Hint**: For the first algorithm you may want to just keep track of what you have chosen. For the second algorithm you will need to be cleverer than this. In the worst case analysis consider the possible relationship between $n$ and $k$.

---

## F. Word Cloud Problem

You may have seen word clouds in the media. Some examples are here. They visually represent the important words in a speech or written article. The words that get used most frequently are printed in a larger font, whilst words of diminishing frequency get smaller fonts. Usually, common words like "the", and "and" are excluded. However, we consider the problem of generating a word cloud for all the words. A key operation to generate a word cloud is:

**Input:** A list W of words, having length $n$ (i.e. $n$ words)
**Output:** A list of the frequencies of all the words in W, written out in any order, e.g. the=21, potato=1, toy=3, story=3, head=1

**Hint:** For the second algorithm you could sort the words first. You may assume that this can be done efficiently. For your efficiency calculation, just count the basic operations used after the sorting.

### G. Minimum Number of Coins

**Input:** A list of coin values, which are 1,2,5,10,20,50,100,200. An integer $T$.
**Output:** The minimum number of coins needed to make the number $T$ from the coins.

It is assumed that there is no limit to the number of coins available, i.e. every coin has an infinite supply. For example, for $T$=20,001. The answer would be 101. (100x the 200-value coin and 1x the 1-value coin).

**Hint:** One algorithm to solve this problem efficiently is to use an algorithmic technique called dynamic programming (this has nothing to do with computer programming, it a mathematical method). Later in the course you will do this for a related problem, but it is too difficult (and not needed) for this problem.

Instead, consider using enumeration: trying out all possible coin combinations of 1 coin, 2 coins, etc., until a combination that works is found. And for the second algorithm, consider using a *greedy* method. (Look this up in Goodrich and Roberto, p259-262).

**Note: The complexity of your algorithms should be expressed in terms of *T* for this problem.**

---

# Submission

Use *submit* in your *COMP26120/ex1* directory. Your text file should identify the problem you selected.

Remember: It should give two solutions, i.e. two pieces of pseudocode. For each one, you should attempt to state why the code is correct, and also attempt to calculate the worst-case number of operations used.

# Marking Scheme

The marks are awarded according to the following marking rubric. Similar marking rubrics will be used throughout this course. The marking scheme is formed of a sequence of questions where the answer to each question is associated with a particular mark. These are the questions that your markers will be asking themselves during marking.

| Is Algorithm 1 correct? | |
|---|---|
| The pseudocode correctly solves the problem and the argument for correctness is non-trivial and makes sense | (3) |
| A good attempt has been made at a solution and it appears correct but is not well explained | (2) |
| An attempt has been made at some pseudocode but it is clearly flawed | (1) |
| The solution looks correct but the student cannot explain how the algorithm works sufficiently | (1) |
| No attempt made | (0) |

| Is Algorithm 2 correct? | |
|---|---|
| The solution is sufficiently different from Algorithm 1. The pseudocode correctly solves the problem, and the argument for correctness is non-trivial and makes sense | (3) |
| The solution is sufficiently different from Algorithm 1. A good attempt has been made at a solution and it appears correct but is not well explained | (2) |
| An attempt has been made at some pseudocode but it is clearly flawed | (1) |
| The solution is not sufficiently different from Algorithm 1 | (1) |
| The solution looks correct but the student cannot explain how the algorithm works sufficiently | (1) |
| No attempt made | (0) |

| Has efficiency been calculated correctly? | |
|---|---|
| The efficiency of both algorithms has been calculated correctly and the student can explain how they reached this answer | (3) |
| Some mistakes have been made in calculating the efficiency but in general the right approach has been taken and the answer is almost right. The student can explain what they did. | (2) |
| A reasonable attempt has been made to compute the number of operations of one of the algorithms as a function of the input but large mistakes have been made | (1) |
| An attempt has been made but the student cannot explain it sufficiently | (1) |
| No attempt made | (0) |

| Does Algorithm 2 improve on Algorithm 1? | |
|---|---|
| It is demonstrated that the second algorithm is more efficient than the first algorithm | (1) |
| The second algorithm does not improve on the first | (0) |
| No attempt is made to demonstrate the second algorithm is more efficient | (0) |