Project 2: Experimenting with Greedy Algorithms (Part A) and Dynamic Programming (Part B)[[1]](#footnote-1)

# Part A carries 30 points.

# Part B carries 30 points.

## General Notes

* Submit all source code, program output, and project report electronically on Canvas as a Zip file.
* Allowed programming languages: Java, C++, Python, or others *with prior approval from the instructor*.

## Academic Honesty Policy: For this project, you are not allowed to discuss with peer students. Also, you are not allowed to copy things from online resources, e.g., source code. We will use a plagiarism detection tool on all submitted source code. It is capable of finding portions of copied code from other students in the course, other sections of the course, previous offerings of the course, as well as Internet sources.

## Instructions

In this project you will implement solution of one problem using dynamic programming as well as using a greedy strategy, and then analyze the performance of those algorithms on sample input data. You will then attempt to infer the complexity of each algorithm and provide a report describing each algorithm’s observed performance and a comparison to the theoretical/expected **performance** and **correctness**.

This project focuses on the ***Change-Making*** ***Problem*** that we discuss in class (refer to Chapter 16 Greedy Algorithm lecture ppt on Canvas).

***Change-Making*** ***Problem***: Let’s consider the problem of making change for *n* cents using the fewest number of coins. Assume that each coin’s value is an integer. Assume that the coin system consists of *k* denominations of coin, and they are d1, d2, …, dk.

An example: Say your company manages vending machines and one of your responsibility is to feed these machines with coins and to make sure that these machines do not run out of coins. So, you may program these machines in such way that they always try to *make changes* (to the customer) in minimum number of coins.

For instance, to return 63 cents to the customer assuming US coin systems (i.e., **penny, nickel, dime, and quarter**), a vending machine may return 2 quarters, 1 dime, and 3 pennies (where 63 = 2×25 + 1×10 + 3×1), which is in fact the solution having the minimum number (which is 6) of coins.

Note that in the above example, n = 63, k = 4, d array is [1, 5, 10, 25] representing the US coin system.

To solve the ***Change-Making*** ***Problem***, let’s implement three different solutions:

1. A straight-forward recursive solution (which is included in **Part A** of this project).
   * This uses the basic “divide and conquer” approach. This is expected to take exponential amount of time (and in that respect, it is comparable to recursive *CutRod* algorithm as in the textbook page 363).
2. A solution using Greedy Algorithm (which is in **Part A** of this project)
   * This follows a greedy strategy: in making change, first use as many counts of highest-denomination-coin as possible, and so on. This algorithm should take *O(k)* time to complete.
3. The bottom-up dynamic programming solution (which is in **Part B** of this project)
   * This algorithm should take *O(k.n)* time to complete.

So, in total, you would implement 3 **algorithms**. The emphasis of the project is on experimental analysis of running time and the correctness of the solution. Track the running time and try to record it in units that are fine-grained enough to allow meaningful comparison. For example, the unit for time should be micro- or possibly even nano-seconds on fast machines. Moreover, your algorithms will be tested for correctness, so ensure the alogorithms work!

## Expected Input Format

Each of your programs should accept a set of **command line arguments**: (a) an integer n representing the total change in cents, (b) number of denominations k, (c) k-size integer array d (i.e., d[1, 2, …, k]) that represents the denomination set. This should assume general coin system; **not** specific to US coin system where k = 4.

## Test Data

1. Test each of your solutions on the following input: vary n over 11, 23, 31, 51, 73, 83, 91, 99 while assuming **US coin system**, i.e., k = 4 and d[1] = 1 (i.e. penny), d[2] = 5 (i.e. nickel), d[3] = 10 (i.e. dime), and d[4] = 25 (i.e. quarter).
2. Test each of your solutions on a **weird coin system** which has 5 coins (penny, nickel, dime, 23 cents, and quarter), i.e., k = 5, and d[1] = 1, d[2] = 5, d[3] = 10, d[4] = 23, d[5] = 25. Test for **n = 69**.

**Expected Output Format**

The output should contain the following lines: (1) the first line has a list of integers representing the coins used, and (2) the second line shows the time required to compute the solution. For recursive solution, reporting item (1) is optional, and instead of this you may report the coin-count.

## What to Submit

You should submit an archive (ZIP, TAR/GZip, etc) of all files, including the following:

1. Source code of the solutions.
2. Makefiles (or simple scripts) to compile the source code.
3. A README.txt file including instructions on how to compile and run your source code.
4. A project report (PDF or Word) containing (at a minimum) the following:
   1. Runs showing output (the results, including the running time) for each input data.
   2. (Required for **CS 5120**): A graphical representation of the running time (graphs).
   3. Pseudocode for the algorithms. Theoretical performance (e.g., theta) for each algorithm.
   4. Comment on the correctness of each solution on Test Data (ii).
   5. Comparison of the different solutions.
   6. Are the theoretical and actual performance results consistent? Any anomalies and/or surprises?
   7. Analysis and Discussion of ***your results*** vis-à-vis expectations.

1. The **Part C** of Project 2 will be announced in near future. [↑](#footnote-ref-1)