Algorithms

Exercise 2 (revised): Strategies for algorithm design

1 I heard you like cards

1. Suppose that you are playing a card game in which each card has a value represented by a natural number. We have a deck of cards which is represented by a set A such that $|A| = n$. A player draws k-many cards from A so that they have a set of cards $P_i \subseteq A$ and $|P_i| = k$. We always choose k such that $1 \leq k \leq n$.

   Our goal is to calculate the strengths of *all* possible hands for a set A and a given k. For example, suppose that $A = \{ 2, 6, 2, 8 \}$ and that $k = 3$. Some possible subsets of A with cardinality 3 include $\{ 2, 6, 8 \}$ whose strength is 8 and $\{ 2, 6, 2 \}$ whose strength is 6. The solution for this particular problem is 30 since we have four subsets of cardinality 3 with strengths 8, 8, 8 and 6.[1]

   (a) Describe a brute-force solution to this problem.

   (b) Implement your algorithm in a language of your choice

   (c) What is the worst-case time complexity of your solution?

   (d) Suggest a better strategy for designing an algorithm to solve this problem and describe a new algorithm using this strategy.

   (e) Implement your new algorithm in a language of your choice.

   (f) Compare your new algorithm with your old algorithm in terms of time and space complexity.

2 Shooting yourself in the foot

2. As many of you have noted, last week's implementation of multiplication could be improved by choosing the "counter" to be whichever one of the two operands is smaller. This would change the time complexity of the generated code from $\mathcal{O}([\![e_0]\!]n)$ to $\mathcal{O}(min([\![e_0]\!]n, [\![e_1]\!]n))$. Since the value of n is not known at compile-time, we cannot statically determine which of the two operands will be smaller.

   (a) Your special treat for this week is therefore the following: using only the instructions provided in last week's specification, modify the compilation function so that the generated code for multiplication dynamically determines which of the two operands should be used as a counter. For reference, the previous specification of the relevant case of the compilation function is given below:

[1] This problem is based on one presented as part of the Facebook Hacker Cup 2013.

$$
\begin{aligned}
C[\![e_0 * e_1]\!] \;=\; & \textbf{PUSH } 0; \\
& C[\![e_1]\!]; \\
& C[\![e_0]\!]; \\
& \textbf{JMPZ } 7; \\
& \textbf{PUSH } 1; \\
& \textbf{SUB}; \\
& \textbf{LOAD } -2; \\
& \textbf{LOAD } -2; \\
& \textbf{ADD}; \\
& \textbf{STORE } -3; \\
& \textbf{JMP } -8; \\
& \textbf{POP}; \\
& \textbf{POP}
\end{aligned}
$$

(b) What is the average/worst-case time complexity of the generated code now?

3 The Countdown problem

3. You are given a list of positive natural numbers (*i.e.* elements of $\mathbb{N}^+$) and the four arithmetic operators $+$, $-$, $*$, and $/$. Additionally, you are given a target number. Your goal is to construct an expression which evaluates to the target number using the given operators and the numbers from the list. You may use each number at most once. For example, suppose that you are given the numbers $[1, 3, 7, 10, 25, 50]$ and your target is 765, then one solution is $(25 - 10) * (50 + 1)$. There are 780 possible solutions for this instance of the problem.

   To simplify the problem, we also say that all *intermediate* results must also be positive natural numbers. For example, $4/2$ is okay, but $4/3$ is not. $4 - 2$ is okay, but $4 - 5$ is not.

   (a) What would a brute-force solution look like?

   (b) What are likely problems you will run into with a brute-force solution?

   (c) Suggest better strategies for the design of your algorithm.

   (d) Suggest techniques which could reduce the search space.

   (e) In a programming language of your choice[2], try to implement a program which solves the countdown problem. Your program should accept a list of positive natural numbers and a target number as input. It should then display all solutions.

[2]Some recommended languages for this task are: Haskell, Python, C#, etc. It is of course possible to solve this problem in other languages, but it is more difficult in *e.g.* ML or Java.