NOTE TO STUDENTS: This file contains sample solutions to the term test together with the marking scheme for each question. Please read the solutions and the marking schemes carefully. Make sure that you understand why the solutions given here are correct, that you understand the mistakes that you made (if any), and that you understand *why* your mistakes were mistakes.

Remember that although you may not agree completely with the marking scheme given here it was followed the same way for all students. We will remark your test only if you clearly demonstrate that the marking scheme was not followed correctly.

For all remarking requests, please submit your request **in writing** directly to your instructor. For all other questions, please don't hesitate to ask your instructor during office hours or by e-mail.

## Question 1. [10 MARKS]

Consider the following **Gas Station** problem: you are driving from Toronto to Montréal and you want to fill up your gas tank as few times as possible along the way (to get there as quickly as possible). Formally, you are given input $d > 0$ (the distance you can drive with a full tank of gas), and locations $0 = g_0 < g_1 < g_2 < \cdots < g_n < g_{n+1}$ (the locations of gas stations along the way, with $g_0$ representing Toronto and $g_{n+1}$ representing Montréal) such that $g_{i+1} - g_i \leqslant d$ for $i = 0, 1, \ldots, n$ (*i.e.*, it is possible to get from each location to the next one on a full tank of gas). You want to find a list of stations $i_1 < i_2 < \cdots < i_k$ such that $g_{i_{j+1}} - g_{i_j} \leqslant d$ for $j = 0, 1, \ldots, k$ (letting $i_0 = 0$ and $i_{k+1} = n + 1$), and $k$ is as small as possible. Assume that the tank is empty initially.

### Part (a) [5 MARKS]

Design a Greedy Algorithm to solve the gas station problem. Include a clear, concise, high-level English description of the main idea behind your algorithm. Also calculate the time complexity of your algorithm.

SAMPLE SOLUTION:

**Main idea:** Go as far as possible without stopping.
**Algorithm:**

```
lastChosen := 0
S := {g_0}
for j := 1, ..., n do:
    if g_{j+1} > g_{lastChosen} + d then
        S := S ∪ {g_j}
        lastChosen := j
    end if
end for
return S
```

**Runtime:** $\Theta(n)$.

MARKING SCHEME:

A. 2 marks: idea is probably correct but description is unclear enough that there is some ambiguity
B. 2 marks: correct algorithm
C. 1 mark: correct runtime computation

### Part (b) [5 MARKS]

Prove the correctness of your algorithm.

SAMPLE SOLUTION:

We will prove by induction that for every $0 \leqslant j \leqslant n$, the set of gas stations amongst $\{g_0, g_1, \ldots, g_j\}$ chosen by the greedy algorithm is the same as the set of gas stations chosen by some optimal algorithm $\mathcal{O}$. And hence, the greedy algorithm is itself optimal.

**Base Case:** $j = 0$: At the start of the journey, you need to have your tank full. Thus, every (optimal) solution will include $g_0$.

**Ind. Hyp.:** For some $j \geqslant 0$, assume that there is an optimal solution $\mathcal{O}$ such that $\mathcal{S}$ and $\mathcal{O}$ choose the same gas stations from amongst $\{g_0, g_1, \ldots, g_j\}$.

**Ind. Step:** At stage $j + 1$, there are two possibilities: either,

**Case 1:** The optimal solution $\mathcal{O}$ does not pick $g_{j+1}$; or,

**Case 2:** The optimal solution $\mathcal{O}$ picks $g_{j+1}$.

If Case 1 happens, then $\mathcal{O}$ picks $g_k$ for some $k \geqslant j + 2$, which implies $g_{j+2} - g_j \leqslant d$, and hence, the greedy algorithm won't pick $g_{j+1}$ either. Thus, $\mathcal{S}$ and $\mathcal{O}$ agree on all the stations chosen from amongst $\{g_0, \ldots, g_{j+1}\}$.

If Case 2 happens, then there are two further sub-possibilities: either,

**SubCase 1:** $\mathcal{S}$ picks $g_{j+1}$, in which case again $\mathcal{S}$ and $\mathcal{O}$ agree on all chosen stations from amongst $\{g_0, g_1, \ldots, g_{j+1}\}$; or,

**SubCase 2:** $\mathcal{S}$ does not pick $g_{j+1}$. Let's assume
$\mathcal{S} := \{g_{i_0}, \ldots, g_{i_s}, g_{i_{s+1}}, \ldots, g_{i_k}\}$ and $\mathcal{O} := \{g_{i_0}, \ldots, g_{i_s}, g_{i'_{s+1}}, g_{i'_{s+2}}, \ldots, g_{i'_l}\}$,
with $\{g_{i_0}, \ldots, g_{i_s}\} \subseteq \{g_0, \ldots, g_j\}$ and $g_{i'_{s+1}} = g_{j+1} \neq g_{i_{s+1}}$. Since the greedy algorithm picks the farthest next stop possible, it follows that $g_{i'_{s+1}} < g_{i_{s+1}}$.
Define $\mathcal{O}' := \{g_{i_0}, \ldots, g_{i_s}, g_{i_{s+1}}, g_{i'_{s+2}}, \ldots, g_{i'_l}\}$, essentially replacing $g_{i'_{s+1}}$ by $g_{i_{s+1}}$ in $\mathcal{O}$. Clearly, $\mathcal{O}'$ is a solution since it is possible to get from $g_{i_s}$ to $g_{i_{s+1}}$ (as $g_{i_{s+1}}$ was picked by the greedy algorithm), and it is possible to get from $g_{i_{s+1}}$ to $g_{i'_{s+2}}$ (as $g_{i_{s+1}} > g_{i'_{s+1}}$, and it is possible to get from $g_{i'_{s+1}}$ to $g_{i'_{s+2}}$ as both $g_{i'_{s+1}}$ and $g_{i'_{s+2}}$ were picked by the optimal algorithm $\mathcal{O}$).
Since $\mathcal{O}$ and $\mathcal{O}'$ have the same size, it follows that $\mathcal{O}'$ is also optimal.
Moreover, $\mathcal{S}$ and $\mathcal{O}'$ share the same chosen stations from amongst $\{g_0, \ldots, g_{j+1}\}$, as desired.

MARKING SCHEME:

A. 2 marks: correct format/structure
B. 3 marks: correct idea

## Question 2. [15 MARKS]

A new "healthy fast-food" restaurant sells *Tofu Nuggets* in boxes that contain either 3, 10 or 25 nuggets. You visit the restaurant with a group of your friends. Can you purchase exactly the right amount of nuggets so that everyone gets to try one but nobody has to try more than one?

We formalize the "Tofu Nugget" problem as follows.

- **Input:** An integer $N \geqslant 0$ (the size of your group).
- **Question:** Determine whether or not you can purchase exactly $N$ Tofu Nuggets (given that you can only purchase boxes of 3, 10 or 25). And if it is possible, find a way of doing so.

Write an algorithm that solves this problem by using dynamic programming, and analyze its running time. Follow the 5-steps "dynamic programming paradigm" covered in class. In particular, describe carefully the recursive structure of the problem in order to justify the correctness of your recurrence.

SAMPLE SOLUTION:

**Recursive Structure:** If $N = 3x + 10y + 25z$ for some $x, y, z \in \mathbb{N}$, then either

- $x > 0$ and $N = 3 + 3(x-1) + 10y + 25z$, or
- $y > 0$ and $N = 10 + 3x + 10(y-1) + 25z$, or
- $z > 0$ and $N = 25 + 3x + 10y + 25(z-1)$.

**Array Definition:** For $i = -24, -23, \ldots, -1, 0, 1, \ldots, N$, define $A[i] =$ True if $i$ nuggets can be purchased exactly (False otherwise).

**Recurrence using Array:** $A[-24] = \cdots = A[-1] =$ `False`
$A[0] =$ `True` $(0 = 3 \cdot 0 + 10 \cdot 0 + 25 \cdot 0)$
$A[i] = A[i-3] \vee A[i-10] \vee A[i-25]$, for $i = 1, \ldots, N$ (from argument above)

**Bottom-up Algorithm:**
**function** tofuNugget($N$):

```
Define array A
for i := −24, ..., −1 do:
    A[i] := False
end for
A[0] := True
for i := 1, ..., N do:
    A[i] := A[i − 3] ∨ A[i − 10] ∨ A[i − 25]
end for
return A[N]
```

**Runtime**: $\Theta(N + 25)$.

**Finding a solution:**
**function** tofuNuggetSolution($A$, $N$):

```
if A[N] = False then
    print "no solution possible"
    return
set i := N
set x := y := z := 0
while i > 0 do:
```

```
        if A[i − 3] = True then
            i := i − 3
            x := x + 1
        else if A[i − 5] = True then
            i := i − 5
            y := y + 1
        else
            i := i − 7
            z := z + 1
        end if
    return x, y, z
```

**Runtime**: $\Theta(N)$.

Marking Scheme:

A. 4 marks: **Structure:** clear attempt to define an array indexed by subproblems, to give a recurrence for the array values (with justification), and to write an iterative algorithm based on the recurrence — even if these are done incorrectly

B. 4 marks: **Idea:** answer contains the argument that one of the box sizes must be used in any solution, so we can simply try each one in turn — even if this is poorly structured or written up

C. 7 marks: **Details:** correct array definition, correct recurrence with appropriate base cases and justification, correct algorithm (even if based on wrong recurrence), and correct runtime (even if algorithm is wrong)