# Assignment 2

**Due by Friday, June 1**

**Acknowledgments.** Acknowledge all the sources you used to complete the assignment. DO NOT COPY! Please read `http://www.student.cs.uwaterloo.ca/~cs341/` for general instructions and policies.

**For all algorithm design questions, you must give the algorithm, argue the correctness, and analyze time complexity.**

## 1 Long integers [10 marks]

Having learned how to multiply $n$-bit integers in time $\Theta(n^{\log_2 3})$, we decide to put this knowledge to good use by developing a software library for performing arithmetic operations on arbitrary precision nonnegative integers. We call the new data type `LongInt` and have developed four algorithms for the library. In the descriptions, a *primitive integer* is an integer $x$ in the range $0 \leq x \leq 10$.

- LongInt($x$) — Given a primitive integer $x$, it creates a `LongInt` with the value $x$.
  Time complexity: $\Theta(1)$.
- Add($A$,$B$) — Given two `LongInts` $A$ and $B$, returns a `LongInt` with the value $A + B$.
  Time complexity: $\Theta(\max\{\log A, \log B\})$.
- MulWithInt($A$,$x$) — Given a `LongInt` $A$ and a primitive integer $x$, returns a `LongInt` with the value $Ax$.
  Time complexity: $\Theta(\log A)$.
- Mul($A$,$B$) — Given two `LongInts`, returns a `LongInt` with the value $AB$.
  Time complexity: $\Theta(\max\{(\log A)^{log_2 3}, (\log B)^{log_2 3}\})$.

We want to add an algorithm Parse to the library that converts strings to `LongInts`. Specifically, the input to the algorithm is an array $s$ of $n \geq 1$ integers in the range from 0 to 9 and we want the output of the algorithm to be the `LongInt` with value $s[0] + 10\, s[1] + 10^2\, s[2] + ... + 10^{n-1}\, s[n-1]$. For example, on input $s = [3, 1, 4, 5, 6]$ the algorithm should generate the `LongInt` with value 65413.

(a) Consider the following implementation of the Parse algorithm.

**Algorithm 1:** Parse($s[0 \dots n-1]$)

```
S ← LongInt(0);
M ← LongInt(1);
for i = 0, ..., n − 1 do
    S ← Add(S, MulWithInt(M, s[i]));
    M ← MulWithInt(M, 10);
return S;
```

In terms of $n$, give a big-$\Theta$ bound for the time complexity of the Parse algorithm. Note: arithmetic operations on primitive types like array indices and loop variables all have time complexity $\Theta(1)$.

(b) Use the divide and conquer approach to design a different Parse algorithm that solves the same problem but has time complexity $\Theta(n^{\log_2 3})$.

## 2 Diameter of a tree [10 marks]

The *diameter* of a tree is the length of the longest simple path between nodes of the tree. Design an efficient divide and conquer algorithm to compute the diameter of a rooted binary tree. Your algorithm may use the following operations on trees that each have time complexity $\Theta(1)$.

- Root($T$) — Returns the root node of the tree $T$.
- LeftChild($T$, $v$) — Returns the node that is the left child of $v$ in $T$; or $\emptyset$ if $v$ does not have a left child.
- RightChild($T$, $v$) — Returns the node that is the right child of $v$ in $T$; or $\emptyset$ if $v$ does not have a right child.

## 3 Making change [10 marks]

Prove or disprove the following statements regarding the greedy coin changing algorithm GreedyChange seen in class. In the case of statements that are true, you should include a complete proof by induction. And for statements that are false, you should include a counter-example.

(a) The GreedyChange algorithm always returns an optimal solution to coin changing problem when the set of denominations $d_1 > d_2 > ... > d_n = 1$ are such that $d_i$ divides $d_{i-1}$ for every $i = 2, 3, ..., n$.

(b) The GreedyChange algorithm always returns an optimal solution to coin changing problem when the set of denominations $d_1 > d_2 > ... > d_n = 1$ are such that $d_{i-1} \geq 2d_i$ for every $i = 2, 3, ..., n$.

## 4 Making the deadline [10 marks]

In the Deadline problem, an instance is a set of $n$ tasks that each take 1 unit of time to complete and a set of deadlines $d_1, \ldots, d_n$. When a task is not completed by its deadline, a fixed penalty of 100$ is applied. A valid solution to the Deadline problem is an ordering of the tasks that minimizes the total penalty incurred when the tasks are completed one after another in the order provided starting at time 0.

Design a greedy algorithm that solves the Deadline problem.

## 5 Efficient polynomial multiplication [20 marks]

For this problem, you will design, analyze, and implement an algorithm that solves the following problem. The description, pseudocode, proof of correctness, and time complexity analysis will be submitted through Crowdmark. Your implementation will be submitted through Marmoset.

**The problem.** Given two polynomials with integer coefficients

$$P = \sum_{i=0}^{n} p_i x^i = p_n x^n + p_{n-1} x^{n-1} + \cdots + p_1 x + p_0,$$

$$Q = \sum_{i=0}^{m} q_i x^i = q_m x^m + q_{m-1} x^{m-1} + \cdots + q_1 x + q_0$$

and a nonzero integer number $A$, determine if the equality

$$P^2 = A \cdot Q$$

is true or false.

An algorithm that solves this problem must implement algorithms that perform arithmetic operations on polynomials. You can do this by representing a polynomial as an array of coefficients and standard definitions, such as the following: given polynomials $P$ and $Q$ with the degree $n \geq m$, and an integer $A$

$$P \pm Q = \sum_{i=m+1}^{n} p_i x^i + \sum_{i=0}^{m} (p_i \pm q_i) x^i;$$

$$A \cdot P = \sum_{i=0}^{n} (A p_i) x^i;$$

$$P \cdot Q = \sum_{k=0}^{n+m} \left( \sum_{i,j \geq 0, i+j=k} p_i q_j \right) x^k = \sum_{k=0}^{n+m} \left( \sum_{i=0}^{k} p_i q_{k-i} \right) x^k.$$

However, the straightforward implementation of polynomial multiplication will have worst-case time complexity $\Theta(nm)$ (or $\Theta(n^2)$ when $m \in \Theta(n)$).

Your task is to implement an efficient Divide & Conquer algorithm for polynomial multiplication based on the same idea as Karatsuba's integer multiplication described in lectures. The time complexity of your algorithm has to be $O(n^{\log_2 3})$.

**Input and output.** For the implementation, the input consists of several lines. The first line contains $n$ (the upper bound for the degree of $P$), $m$ (the upper bound for the degree of $Q$), and $A$.

The following $n+1$ lines contain the coefficients of polynomial $P$, in order from $p_0$ to $p_n$.
Another $m+1$ lines contain the coefficients of polynomial $Q$, in order from $q_0$ to $q_m$.

The output of your algorithm must have 1 line that contains `TRUE` or `FALSE`.

**Sample input 1**

```
2 4 4
6
2
2
9
6
7
2
1
```

**Sample output 1**

```
TRUE
```

Note, that the input represents polynomials $P = 2x^2 + 2x + 6$ and $Q = x^4 + 2x^3 + 7x^2 + 6x + 9$ with $P^2 = 4Q$.

**Sample input 2**

```
2 5 4
6
2
2
9
6
7
2
1
0
```

**Sample output 2**

```
TRUE
```

Note, that the input represents polynomials $P = 2x^2 + 2x + 6$ and $Q = x^4 + 2x^3 + 7x^2 + 6x + 9$ with $P^2 = 4Q$.

**Sample input 3**

```
1 2 5
0
2
0
0
1
```

**Sample output 3**

```
FALSE
```

Note, that the input represents polynomials $P = 2x$ and $Q = x^2$ with $P^2 \neq 5Q$.

**Remarks, useful facts and hints.** Given two polynomials $P$ and $Q$ as described above,

- The degree of the polynomial $P$ is the largest value of $i$ such that $0 \leq i \leq n$ & $p_i \neq 0$. Thus, the values of $n$ and $m$ given in the input are upper bounds for the degrees of given polynomials (see Sample input 2).
- $\deg(P \cdot Q) = \deg P + \deg Q$. This can be used to quickly recognize some `FALSE` instances of the given problem, but the whole input must be processed to do so, taking into account the previous note.
- $\deg(P \pm Q) \leq \max(\deg P, \deg Q)$ (watch for potential cancellation of the leading coefficients of the equal degree polynomials).
- If $n$ is not a power of 2, padding the given polynomial $P$ with several leading zero terms in order to make $n$ to be the closest power of 2 increases the time time complexity by a constant factor only.
- You can assume that the input we provide will not lead to an overflow during the computations, as long as you use 64 bit integers (`long` in Java and `long long int` in C) to represent the coefficients.