Homework 3

Mengxiang Jiang
CSEN 5303 Foundations of Computer Science

September 11, 2022

Problem 1. *Euclid's algorithm* (or the *Euclidean algorithm*) is an algorithm that computes the *greatest common divisor*, denoted by *gcd*, of two integers. Below are the original versions of Euclid's algorithm that uses *repeated subtraction* and another one that uses the *remainder*.

```
int gcd_sub(int a, int b)
{
    if (!a)
        return b;
    while (b)
        if (a > b)
            a = a - b;
        else
            b = b - a;
    return a;
}
```

```
int gcd_rem(int a, int b)
{
    int t;
    while (b)
    {
        t = b;
        b = a % b;
        a = t;
    }
    return a;
}
```

1. Trace each of the above aglorithm using specific values for a and b.

Iteration	a	b
1	119	544
2	119	425
3	119	306
4	119	187
5	119	68
6	51	68
7	51	17
8	34	17
9	17	17
10	17	0

Trace with $a = 119$ and $b = 544$ for *gcd_sub*.

Iteration	a	b
1	119	544
2	544	119
3	119	68
4	68	51
5	51	17
6	17	0

Trace with $a = 119$ and $b = 544$ for *gcd_rem*.

2. Compare both algorithms.

Algorithm *gcd_sub* is simpler by using only subtraction with no need for a temporary variable. However, it will take a lot more iterations when a and b greatly differ in size compared to *gcd_rem*. Algorithm *gcd_rem* is able to reduce the larger of a and b to be smaller than the smaller of a and b in one iteration by taking advantage of the *modulus*(%) operator, but this operator is more complicated than subtraction.

Problem 2. Given a fixed integer $B(B \geq 2)$, we demonstrate that any integer $N(N \geq 0)$ can be written in a unique way in the form of the sum $p + 1$ terms as follows:

$$N = a_0 + a_1 \times B + a^2 \times B^2 + \ldots + a_p \times B^p$$

where all a_i, for $0 \leq a_i \leq B - 1$.

The notation $a_p a_{p-1} \ldots a_0$ is called the *representation* of N in base B. Notice that a_0 is the remainder of the *Euclidean* division of N by B. If Q is the *quotient*, a_1 is the remainder of the *Euclidean* division of Q by B, etc.

1. Write an algorithm that generates the representation of N in base B.

```
function BaseB(n: int, b: int) : string;
    var
        rem_temp : int;
        char_temp : character;
        result : string;
    begin
        result := "";
        while n > 0 do
            begin
                rem_temp := n mod b;
                char_temp := CharInBaseB(rem_temp, b);
                result := char_temp + result;
                n := n / b;
            end;
        return result;
    end;
```

2. Compute the time complexity of your algorithm.
$T(n) = T(n/b) + \Theta(1)$, $a = 1$, $b = b$, $f(n) = \Theta(1)$
$\log_b a = \log_b 1 = 0$, $n^{\log_b a} = n^0 = 1 = \Theta(1)$
Since $f(n)$ is the same size as $n^{\log_b a}$, case 2 of the master theorem applies,
so $T(n) = \Theta(\log(n))$