CSC110 Fall 2022 Assignment 4: Number Theory, Cryptography, and Algorithm Running Time Analysis

Shivesh Prakash

November 21, 2022

Part 1: Proofs

1. Statement to prove: $\forall a, b, n \in \mathbb{Z},\ \big(n \neq 0 \land a \equiv b \pmod{n}\big) \Rightarrow \big(\forall m \in \mathbb{Z},\ a \equiv b + mn \pmod{n}\big)$

   Definition of divisibility: $d \mid n \iff \exists k \in \mathbb{Z}, n = dk \ where \ n, d \in \mathbb{Z}$
   Definition of modular equivalence: $a \equiv b \pmod{n} \iff n \mid a - b$

   Proof. Fix $a, b, n \in \mathbb{Z}$.
   Let us assume that $n \neq 0$ and $a \equiv b \pmod{n}$.
   Fix $m \in \mathbb{Z}$.
   Since $a \equiv b \pmod{n}$, using definition of modular equivalence and divisibility,
   there exists an integer $k = k_1$ such that $a - b = nk_1$.
   Subtracting mn on both sides of this equation gives:

$$a - b - mn = nk_1 - mn$$
$$\implies a - b - mn = n(k_1 - m)$$

   Since $k_1, m \in \mathbb{Z}$, $k_1 - m \in \mathbb{Z}$. Let $k_2 = k_1 - m$, observe that $k_2, m \in \mathbb{Z}$.
   Thus the equation becomes:

$$a - b - mn = nk_2$$
$$\implies a - (b + mn) = nk_2$$
$$\implies n \mid a - (b + mn)$$
$$\implies a \equiv b + mn \pmod{n}$$

$\square$

2. Statement to prove: $\forall f, g : \mathbb{N} \to \mathbb{R}^{\geq 0},\ \Big(g \in \mathcal{O}(f) \wedge \big(\forall m \in \mathbb{N},\ f(m) \geq 1)\Big) \Rightarrow g \in \mathcal{O}(\lfloor f \rfloor)$

Definition of $g \in \mathcal{O}(f)$: $\exists c, n_0 \in \mathbb{R}^+ \ s.t\ \forall n \in \mathbb{N}, n \geq n_0 \implies g(n) \leq c \cdot f(n)$

Proof. Fix $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$.
Let us assume that $g \in \mathcal{O}(f)$ and $\forall m \in \mathbb{N},\ f(m) \geq 1$.
That means: $\exists c_1, n_{01} \in \mathbb{R}^+ \ s.t\ \forall n \in \mathbb{N}, n \geq n_{01} \implies g(n) \leq c_1 \cdot f(n)$.
Also $\forall m \in \mathbb{N},\ f(m) \geq 1 \implies \lfloor f(m \rfloor \geq 1$, from the definition of floor function.
A property of floor function states that: $\forall x \in \mathbb{R}, \lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$.
Substituting $f(n)$ for x, this gives the inequality: $\lfloor f(n) \rfloor \leq f(n) < \lfloor f(n) \rfloor + 1$.
Observe that c_1 is positive, multiplying c_1 on all parts of the above inequality gives us:

$$
\begin{aligned}
c_1 \cdot \lfloor f(n) \rfloor \leq c_1 \cdot f(n) < c_1 \cdot (\lfloor f(n) \rfloor + 1) \\
\implies c_1 \cdot f(n) < c_1 \cdot \lfloor f(n) \rfloor + c_1 \\
\implies c_1 \cdot f(n) < (c_1 + \frac{c_1}{\lfloor f(n) \rfloor}) \cdot \lfloor f(n) \rfloor
\end{aligned}
$$

Let $c_1 + \frac{c_1}{\lfloor f(n) \rfloor}$ be represented by c_2. Since $c_1 \in \mathbb{R}^+$ and $n \in \mathbb{N}$, $c_2 \in \mathbb{R}^+$.
Modifying the definition statement of $g \in \mathcal{O}(f)$ with the inequality:

$$
\begin{aligned}
\exists c_1, n_{01} \in \mathbb{R}^+ \ s.t\ \forall n \in \mathbb{N}, n \geq n_{01} \implies g(n) \leq c_1 \cdot f(n) < (c_1 + \frac{c_1}{\lfloor f(n) \rfloor}) \cdot \lfloor f(n) \rfloor \\
\implies \exists c_2, n_{01} \in \mathbb{R}^+ \ s.t\ \forall n \in \mathbb{N}, n \geq n_{01} \implies g(n) \leq c_2 \cdot \lfloor f(n) \rfloor \\
\implies g \in \mathcal{O}(\lfloor f \rfloor)
\end{aligned}
$$

$\square$

Part 2: Running-Time Analysis

1. Function to analyse:

```
def f1(n: int) -> int:
    """Precondition: n >= 0"""
    total = 0

    for i in range(0, n):  # Loop 1
        total += i ** 2

    for j in range(0, total):  # Loop 2
        print(j)

    return total
```

The statement assigning `total` to `0` takes 1 step. The statement under `Loop 1` take 1 step and the loop runs `n` times, thus `Loop 1` takes a total of n steps. After `Loop 1` the value of `total` is sum of squares of first `(n-1)` integers. From Appendix C.1, this is equivalent to $\frac{(n-1)(n)(2n-1)}{6}$. The statement under `Loop 2` take 1 step and the loop runs `total =` $\frac{(n-1)(n)(2n-1)}{6}$ times, thus `Loop 2` takes a total of $\frac{(n-1)(n)(2n-1)}{6}$ steps. Hence the function runs a total of $1 + n + \frac{(n-1)(n)(2n-1)}{6}$ times. This expression is equivalent to $\frac{n^3}{3} - \frac{n^2}{2} + 7n + 6$, which $\in \Theta(n^3)$. Thus from our running-time analysis, the exact expression for $RT_{func}(n)$ is $\frac{n^3}{3} - \frac{n^2}{2} + 7n + 6$, which $\in \Theta(n^3)$.

2. Function to analyse:

```
def f2(n: int) -> int:
    """Precondition: n >= 0"""
    sum_so_far = 0

    for i in range(0, n):  # Loop 1
        sum_so_far += i

        if sum_so_far >= n:
            return sum_so_far

    return 0
```

The statement assigning `sum_so_far` to `0` takes 1 step. The innermost `if` loop early returns when `sum_so_far` exceeds `n`. After `k + 1` iterations of `for Loop 1`, `sum_so_far` is $\frac{(k)(k+1)}{2}$.
For this to be $\geq$ `n` $\implies$

$$\begin{aligned}
\frac{(k)(k+1)}{2} &\geq n \\
\implies k^2 + k &\geq 2n \\
\implies k^2 + k - 2n &\geq 0 \\
\implies k \geq \frac{-1+\sqrt{1+8n}}{2} &\ \textit{or}\ k \leq \frac{-1-\sqrt{1+8n}}{2} \\
\implies k \geq \sqrt{\frac{1}{4}+2n} - \frac{1}{2} &\ \textit{Since number of iterations must be positive}
\end{aligned}$$

This is calculated using quadratic formula while ensuring that `k` is positive and less than `n`.
To maintain `k` as an integer and taking into account $\geq$ `n` for the expression, finally we take `k` as $\lceil\sqrt{\frac{1}{4}+2n}-\frac{1}{2}\rceil$. Hence the function runs for a total of $1+(k+1)+1$ times. This expression is equivalent to $3+\lceil\sqrt{\frac{1}{4}+2n}-\frac{1}{2}\rceil$, which $\in \Theta(\sqrt{n})$. Thus from our running-time analysis, the exact expression for $RT_{func}(n)$ is $3+\lceil\sqrt{\frac{1}{4}+2n}-\frac{1}{2}\rceil$, which $\in \Theta(\sqrt{n})$.

Part 3: Extending RSA

Complete this part in the provided `a4_part3.py` starter file. Do **not** include your solutions in this file.

Part 4: Digital Signatures

Part (a): Introduction

Complete this part in the provided `a4_part4.py` starter file. Do **not** include your solutions in this file.

Part (b): Generalizing the message digests

Complete most of this part in the provided `a4_part4.py` starter file. Do **not** include your solutions in this file, *except* for the following two questions:

3b. `def find_collision_len_times_sum(message: str) -> str:`

```
    """Return a new message, not equal to the given message, that can be verified
    using the same signature when using the RSA digital signature scheme with
    the len_times_sum message digest.

    Preconditions:
    - len(message) >= 2
    - any({ord(c) < 1114111 for c in message})
    """
    m = [ord(c) for c in message]
    m[0] = m[0] + 1
    m[-1] = m[-1] - 1
    characters = [chr(c) for c in m]
    return ''.join(characters)
```

The idea here is to return a string of the same length and same sum of `ord()` values for each character. I have achieved this by adding and subtracting `1` to the first and last character of the message respectively. I assigned variable `m` to the list of `ord()` values for each character in `message` by iterating over it. I reassigned the elements of `m` to modify some `ord()` values while maintaining their sum as a constant. I assigned variable `m` to the list of characters to be merged and returned. Lastly, my function returns the string by joining the list of characters using `str.join()`.

4b.
```
def find_collision_ascii_to_int(public_key: tuple[int, int], message: str) -> str:

    """Return a new message, distinct from the given message, that can be verified
    using the same signature, when using the RSA digital signature scheme with the
    ascii_to_int message digest and the given public_key.

    The returned message must contain only ASCII characters, and cannot contain any
    leading chr(0) characters.

    Preconditions:
    - signature was generated from message using the algorithm in rsa_sign and
    digest len_times_sum, with a valid RSA private key
    - len(message) >= 2
    - ord(message[0]) > 0

    NOTES:
        - Unlike the other two "find_collision" functions, this function takes in the
        public key used to generate signatures. Use it!
        - You may NOT simply add leading chr(0) characters to the message string.
          (While this does correctly produces a collision, we want you to think
          a bit harder to come up with a different approach.)
        - You may find it useful to review Part 1, Question 1.
    """
    n = public_key[0]
    modified_int = ascii_to_int(message) + n
    num_in_base = a4_part3.int_to_base128(modified_int)
    characters = [chr(c) for c in num_in_base]
    return ''.join(characters)
```

The idea here is to return a string whose signature matches that of `message`, determined using `rsa_sign()`. On inspection of the `rsa_sign()` function, it is evident that the `sign` produced will be same when the `digest` computed using `digest = compute_digest(message) % n` is the same. For our case, `compute_digest() is ascii_to_int()`. If `ascii_to_int(returned_string)` is of the form `ascii_to_int(message) + mn` where `m` is an integer and `n` is the first element of `public_key`, then using Proof 1 from Part 1, `digest` will be the same form `message` and `returned_message`.

In my function, I added `n` to `ascii_to_int(message)` to implement the concept explained above. Then I used this `modified_int` to reverse it into a `string` using `a4_part3.int_to_base128()` and `chr()` along with `str.join()`.