Advanced Application Programming

- Big Integer Library and DLP Calculator -

Ji Yong-Hyeon

Department of Information Security, Cryptology, and Mathematics
College of Science and Technology
Kookmin University

November 10, 2023

Contents

Chapter 1

Preliminaries

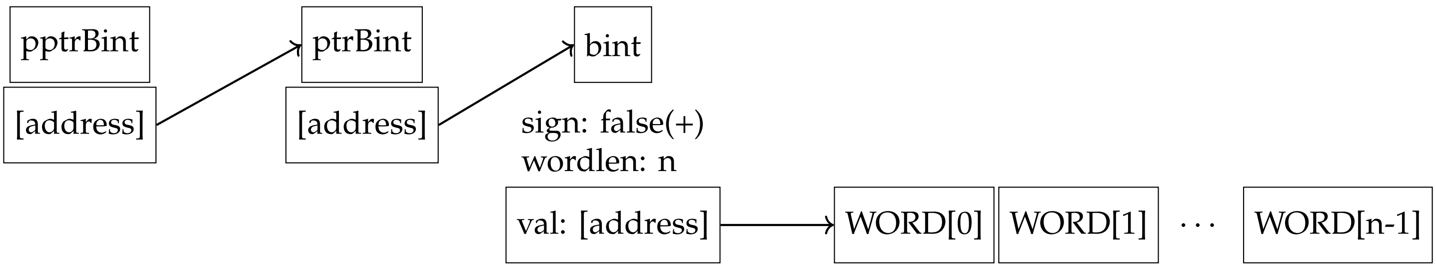
1.1 Set-up

```c
typedef unsigned char u8;
typedef unsigned int u32;
typedef unsigned long u64;

#define false 0
#define true !false
#define FLAG 5 // Karatsuba Depth
#define MAX(x1, x2) (x1 > x2 ? x1 : x2)
#define MIN(x1, x2) (x1 < x2 ? x1 : x2)

void exit_on_null_error(const void* ptr, const char* ptr_name, const char* function_name) {
    if(!ptr) {
        fprintf(stderr, "Error: '%s' is NULL in '%s'\n", ptr_name, function_name); exit(1);
    }
}

#define CHECK_PTR_AND_DEREF(ptr, name, func) \
do { \
    exit_on_null_error(ptr, name, func); \
    exit_on_null_error(*ptr, "*" name, func); \
} while(0)

#define CHECK_PTR_DEREF_AND_VAL(ptr, name, func) \
do { \
    exit_on_null_error(ptr, name, func); \
    exit_on_null_error(*ptr, "*" name, func); \
    if(*ptr) exit_on_null_error((*ptr)->val, "(*" name ")->val", func); \
} while(0)
```

1.2 Data Structure

```c
#define WORD_BITLEN 32

// Supports operations in 8-bit, 32-bit, and 64-bit units.
#if WORD_BITLEN == 8
typedef u8 WORD;
#elif WORD_BITLEN == 64
typedef u64 WORD:
#else
typedef u32 WORD;
#endif

/**
* Represents a large integer in binary format.
*/
typedef struct {
    bool sign;      // false if 0 or positive, true if negative.
    int wordlen;    // The number of WORDs
    WORD* val;      // Pointer to the array of WORDs
```

```c
BINT bint;
BINT* ptrBint = &bint;
BINT** pptrBint = &ptrBint;
```



1.3 Initialization and Delete BINT: init_bint, delete_bint

```
void init_bint(BINT** pptrBint, int wordlen) { // ptrBint = *pptrBint
    if((*pptrBint) != NULL)
        delete_bint(pptrBint);

    // Allocate memory for BINT structure
    *pptrBint = (BINT*)malloc(sizeof(BINT));
    if(!(*pptrBint)) {
        fprintf(stderr, "Error: Unable to allocate memory for BINT.\n");
        exit(1);
    }
    // Allocate memory for val (array of WORD)
    (*pptrBint)->val = (WORD*)calloc(wordlen,sizeof(WORD));
    if (!(*pptrBint)->val) {
        free(*pptrBint); // freeing the already allocated BINT memory before exiting
        fprintf(stderr, "Error: Unable to allocate memory for BINT val.\n");
        exit(1);
    }
    // Initialize structure members
    (*pptrBint)->sign = false;
    (*pptrBint)->wordlen = wordlen;
}
void delete_bint(BINT** pptrBint) {
    if(!(*pptrBint))
    return;
    free((*pptrBint)->val);
    free(*pptrBint);
    *pptrBint = NULL;
}
```

1.4 Copy BINT (contain init_bint): copyBINT

```
void copyBINT(BINT** pptrBint_dst, BINT** pptrBint_src) {
    CHECK_PTR_AND_DEREF(pptrBint_src, "pptrBint_src", "copyBINT");

    if(*pptrBint_dst != NULL)
    delete_bint(pptrBint_dst);

    init_bint(pptrBint_dst, (*pptrBint_src)->wordlen);
    for(int i = 0; i < (*pptrBint_src)->wordlen; i++)
        (*pptrBint_dst)->val[i] = (*pptrBint_src)->val[i];

    (*pptrBint_dst)->wordlen = (*pptrBint_src)->wordlen;
    (*pptrBint_dst)->sign = (*pptrBint_src)->sign;
}
```

Chapter 2

Addition and Subtraction

2.1 Addition

2.1.1 Memory for Addition

Remark 2.1. A positive integer $A \in \left[W^{n-1}, W^{n}\right)$ is a n-word string.

Upper and Lower Bound of Addition

Proposition 2.1. *Let A and B are n-word and m-word strings, respectively,* i.e.,

$$A \in [W^{n-1}, W^{n}-1], \quad B \in [W^{m-1}, W^{m}-1].$$

Then

$$W^{\max(n,m)-1} < A + B < W^{\max(m,n)+1}.$$

Proof. A and B can be expressed as follows: $\begin{cases} A = aW^{n-1} + A' \\ B = bW^{m-1} + B' \end{cases}$ where

$$a, b \in (0, W), \quad A' \in [0, W^{n-1}-1], \quad B' \in [0, W^{m-1}-1].$$

Suppose that $n \geq m$ then

$$\begin{aligned} W^{n-1} \leq \max(A, B) < A + B &= (aW^{n-1} + A') + (bW^{m-1} + B') \\ &< (a+b)W^{n-1} + (W^{n-1} + W^{n-1}) \\ &= (a+b+2)W^{n-1} \\ &\leq ((W-1) + (W-1) + 2)W^{n-1} \\ &= 2W^{n} \leq W^{n+1}. \end{aligned}$$

Thus $W^{n-1} < A + B < W^{n+1}$. Here, $n = \max(n, m)$. □

Corollary 2.1.1.

$$\mathsf{wordlen}(A) = n, \mathsf{wordlen}(B) = m \implies \mathsf{wordlen}(A+B) \leq \max(n, m) + 1.$$

2.1.2 Single-Word Addition: add_carry

Example 2.1.

```c
#include <stdio.h>
int main() {
    int x = 0x12345678; int y = 0xffffffff;
    printf("%08x + %08x = %08x\n", x, y, x+y);

    return 0;
}
/**
 * 12345678 + ffffffff = 12345677
*/
```

$$
\begin{aligned}
\texttt{0x12345678} \boxplus \texttt{0xffffffff} &= (\texttt{0x12345678} + \underbrace{\texttt{0xffffffff}}_{2^{32}-1}) \bmod 2^{32} \\
&= (\texttt{0x12345678} + (-\texttt{0x01})) \bmod 2^{32} \\
&= \texttt{0x12345677}.
\end{aligned}
$$

Note that $\texttt{0x1234568} + \texttt{0xffffffff} = \texttt{0x112345677}$

Single-Word Addition $A + B$

Proposition 2.2. *Let $A, B \in [0, W-1]$.*

(1)

$$A + B = \left\lfloor \frac{A+B}{W} \right\rfloor W + ((A+B) \bmod W) = \left\lfloor \frac{A+B}{W} \right\rfloor W + (A \boxplus B).$$

(2) *(carry)* $\left\lfloor \frac{A+B}{W} \right\rfloor \in \{0, 1\}$.

(3) $0 \leq A + B < W^2$, i.e., $\mathsf{wordlen}(A+B) \leq 2$.

(4) $W \leq A + B$, i.e., $\left\lfloor \frac{A+B}{W} \right\rfloor = 1 \iff (A \boxplus B < A) \vee (A \boxplus B < B)$.

Proof. (1) Use Division Algorithm.

(2) Clearly, $0 \leq A + B < W + W = 2W$. Then

$$0 \leq \left\lfloor \frac{A+B}{W} \right\rfloor < \lfloor 2 \rfloor \implies 0 \leq \left\lfloor \frac{A+B}{W} \right\rfloor \leq 1.$$

(3) $0 \leq A + B < 2W \leq W^2$ for $W \geq 2$.

(4) ($\Rightarrow$) Suppose that $A + B \geq W$. Since $A + B \in [W, 2W - 1]$, we have

$$\begin{aligned} A \boxplus B &= (A + B) - W = A - (W - B) \\ &< A \quad \because B < W, \text{i.e.}, W - B > 0. \end{aligned}$$

($\Leftarrow$) Let's prove it using contraposition:

$$A + B < W \implies A \boxplus B \geq A.$$

Assume that $A + B < W$ then

$$A \boxplus B = A + B \geq A.$$

$\square$

Algorithm 1: Single Word Addition

Input: Single-word strings $X, Y \in [0, W - 1]$
Output: $q \in \{0, 1\}$ and $r \in [0, W - 1]$ s.t. $X + Y = qW + r$

```
q ← 0;
r ← X ⊞ Y;                                   // K ← X + Y in C
if r < X then
  | q ← 1;                                   // r < X ∨ r < Y ⇒ q ≠ 0 ⇒ q = 1
end
return q, r;
```

Single-Word Addition with Carry $A + B + c$

Proposition 2.3. *Let $A, B \in [0, W - 1]$ and $c \in \{0, 1\}$.*

(1) (carry) $\left\lfloor \dfrac{A + B + c}{W} \right\rfloor \in \{0, 1\}$.

(2) $0 \leq A + B + c < W^2$, i.e., $\mathsf{wordlen}(A + B + c) \leq 2$.

(3) $W \leq A + B \implies 0 \leq (A \boxplus B) + c < W$. *In other words,*

$$\left\lfloor \frac{A + B}{W} \right\rfloor = 1 \implies \left\lfloor \frac{A \boxplus B + c}{W} \right\rfloor = 0.$$

Proof. (1) $0 \leq A + B + c \leq (W - 1) + (W - 1) + 1 = 2W - 1 < 2W \implies 0 \leq \frac{A+B+c}{W} < 2$.

(2) $0 \leq A + B + c < 2W \leq W^2$.

(3) Suppose that $W \leq A + B$. By **Proposition 2.2**, we have

$$0 \leq (A \boxplus B) + c < A + c \leq (W - 1) + 1 = W.$$

$\square$

Algorithm 2: ADD$^{\text{carry}}(X, Y, k)$

Input: Single-word strings $X, Y \in [0, W-1]$ and carry $k \in \{0, 1\}$
Output: $q \in \{0, 1\}$ and $r \in [0, W-1]$ s.t. $X + Y + k = qW + r$

1 **Function** ADD$^{carry}(X, Y, k)$**:**
2   $q \leftarrow 0$;
3   $r \leftarrow X \boxplus Y$;
4   **if** $r < X$ **then**
5     $q \leftarrow 1$;   // $X \boxplus Y < X \iff q = \lfloor (X+Y)/2^w \rfloor = 1$
6   **end**
7   $r \leftarrow r \boxplus k$;   // $X \boxplus Y \boxplus k$
8   **if** $r < k$ **then**
    /* $q = \lfloor (X+Y)/2^w \rfloor = 1 \implies \lfloor (A \boxplus B + c)/2^w \rfloor = 0$ */
9     $q \leftarrow q + 1$;
10   **end**
   `/* By Prop. 2.3 - (3), if line 4 is Ture then line 8 must be False */`
11   **return** q, r;
12 **end**

```c
void add_carry(WORD x, WORD y, WORD k, WORD* ptrQ, WORD* ptrR) {
    WORD sum = x + y;
    *ptrQ = 0x00;
    if(sum < x) *ptrQ = 0x01;
    sum += k;
    *ptrR = sum;
    if(sum < k) *ptrQ += 0x01;
}
```

2.1.3 Muti-Precision Addition: add_core_xyz, ADD

Algorithm 3: $\mathsf{ADD}^{\mathsf{Core}}(X, Y)$

Input: n-word string $X = (-1)^{\mathsf{sign}} \sum_{i=0}^{n-1} x_i W^i$ and m-word string $Y = (-1)^{\mathsf{sign}} \sum_{i=0}^{m-1} y_i W^i$, where $a_i, b_i \in [0, W-1]$, $\mathsf{sign} \in \{0, 1\}$ and $n \geq m$.
Output: $Z = X + Y = (-1)^{\mathsf{sign}} \sum_{i=0}^{l} z_i W^i$, where $z_i \in [0, W-1]$ and $l \in \{n-1, n\}$

Function $\mathsf{ADD}^{\mathsf{Core}}(X, Y)$**:**
  for $i = m$ **to** $n - 1$ **do**
    $y_i \leftarrow 0$; // $0 \cdots 0 \| y_{m-1} \cdots y_0$
  end
  $k \leftarrow 0$;
  for $i = 0$ **to** $n - 1$ **do**
    $k, z_i \leftarrow \mathsf{ADD}^{\mathsf{carry}}(x_i, y_i, k)$;
  end
  $z_n \leftarrow k$;
  if $z_n == 1$ **then**
    return $(-1)^{\mathsf{sign}} \sum_{i=0}^{n} z_i W^i$;
  else
    return $(-1)^{\mathsf{sign}} \sum_{i=0}^{n-1} z_i W^i$;
  end
end

```c
void add_core_xyz(BINT** pptrX, BINT** pptrY, BINT** pptrZ) {
    CHECK_PTR_AND_DEREF(pptrX, "pptrX", "add_core_xyz");
    CHECK_PTR_AND_DEREF(pptrY, "pptrY", "add_core_xyz");
    if (!compare_abs_bint(pptrX,pptrY)) {
        add_core_xyz(pptrY, pptrX, pptrZ); return;
    }
    int n = (*pptrX)->wordlen; int m = (*pptrY)->wordlen;
    if (!pptrZ || !*pptrZ || !(*pptrZ)->val) init_bint(pptrZ, n+1);
    CHECK_PTR_AND_DEREF(pptrZ, "pptrZ", "add_core_xyz");

    WORD res = 0x00;
    WORD carry = 0x00;
    WORD k = 0x00;

    for (int i = 0; i < m; i++) {
        add_carry((*pptrX)->val[i],(*pptrY)->val[i], k, &carry, &res);
        (*pptrZ)->val[i] = res; k = carry;
    } for (int i = m; i < n; i++) {
        add_carry((*pptrX)->val[i],(WORD)0, k, &carry, &res);
        (*pptrZ)->val[i] = res; k = carry;
    }
    if(k) { (*pptrZ)->val[n] = k; }
    else { (*pptrZ)->wordlen = n; }
}
```

Algorithm 4: ADD(X, Y)

Input: $X, Y \in \mathbb{Z}$
Output: $X + Y \in \mathbb{Z}$

```
Function ADD(X, Y):
    if X > 0 && Y < 0 then
        return SUB(X, |Y|);
    else if X < 0 && Y > 00 then
        return SUB(Y, |X|);
    end
    if wordlen(X) ≥ wordlen(Y) then
        return ADD^Core(X, Y);
    else
        return ADD^Core(Y, X);
    end
end
```

```c
void ADD(BINT** pptrX, BINT** pptrY, BINT** pptrZ) {
    CHECK_PTR_AND_DEREF(pptrX, "pptrX", "ADD");
    CHECK_PTR_AND_DEREF(pptrY, "pptrY", "ADD");

    if (!compare_abs_bint(pptrX, pptrY)) {
        ADD(pptrY, pptrX, pptrZ); return;
    }

    int n = (*pptrX)->wordlen;

    init_bint(pptrZ, n+1);
    CHECK_PTR_AND_DEREF(pptrZ, "pptrZ", "ADD");

    if ((*pptrX)->sign == (*pptrY)->sign) {
        // If signs are the same, add the numbers
        add_core_xyz(pptrX, pptrY, pptrZ);
        if ((*pptrX)->sign) {
            // If both numbers are negative, then result is negative
            (*pptrZ)->sign = true;
        }
    } else {
        // If signs are different, subtract the numbers
        sub_core_xyz(pptrX, pptrY, pptrZ);
        if ((*pptrX)->sign) {
            // If X is negative and Y is positive, then result is
               negative
            (*pptrZ)->sign = true;
        }
    }
}
```

2.2 Subtraction

2.2.1 Memory for Subtraction

Upper and Lower Bound of Subtraction

Proposition 2.4. *Let A and B are n-word and m-word strings, respectively,* i.e.,

$$A \in [W^{n-1}, W^n - 1], \quad B \in [W^{m-1}, W^m - 1].$$

Then, for $A \geq B$,

$$0 \leq A - B < A < W^n.$$

2.2.2 Sing-Word Subtraction: sub_borrow

Example 2.2.

```c
#include <stdio.h>
typedef unsigned int u32;
int main() {
    u32 x = 0xffffffff; u32 y = 0x12345678;
    u32 z = y - x; //z1 <- y - x mod 2^(32)
    printf("%08x - %08x = %08x\n", y, x, z2);
    return 0;
}
/*
12345678 - ffffffff = 12345679
*/
```

$$\begin{aligned}
\texttt{0x12345678} \boxminus \texttt{0xffffffff} &= (\texttt{0x12345678} - \underbrace{\texttt{0xffffffff}}_{2^{32}-1}) \bmod 2^{32} \\
&= (\texttt{0x12345678} - (-\texttt{0x01})) \bmod 2^{32} \\
&= \texttt{0x12345679} \bmod 2^{32} \\
&= \texttt{0x12345679}.
\end{aligned}$$

Note that $-2^{32} + \texttt{0x12345679} = -\texttt{0xedcba987}$.

Single-Word Subtraction $A - B$

Proposition 2.5. *Let $A, B \in [0, W-1]$.*

(1) $A - B \in [-(W-1), W-1] \subseteq (-W, W)$

(2) $A - B = \begin{cases} A - B & : A \geq B, \\ -(B-A) = -W + \underbrace{(W - (B-A))}_{A \boxminus B \in [0, W-1]} & : A < B. \end{cases}$

Algorithm 5: Single Word Subtraction

Input: Single-word strings $X, Y \in [0, W-1]$
Output: $q \in \{0,1\}$ and $r \in [0, W-1]$ s.t. $X - Y = -qW + r$

$q \leftarrow 0$;
$r \leftarrow X \boxminus Y$; // $r \leftarrow X - Y$ in C
$q \leftarrow (X < Y)$;
return q, r;

Single-Word Subtraction with Borrow $A - b - B$

Proposition 2.6. *Let $A, B \in [0, W-1]$ and $b \in \{0,1\}$.*

(1) $-W \le A - b - B < W$

(2) $\left\lfloor \frac{A-b-B}{W} \right\rfloor \in \{-1, 0\}$.

(3) $-W \le A - b - B < 0 \implies A - b - B = -W + (A \boxminus b \boxminus B)$.

(4) $A - b < 0 \implies A - b = -1 = -W + (W-1) \implies (A \boxminus b) - B \in [0, W-1]$.

Proof. (1)

(2) By (1), it holds.

(3) $A \boxminus b \boxminus B = A - b - B + W$

(4) $(A \boxminus b) - B = (W-1) - B \in [0, W-1]$

□

Algorithm 6: $\mathsf{SUB}^{\text{borrow}}(X, Y, b)$

Input: Single-word strings $X, Y \in [0, W-1]$ and borrow $b \in \{0,1\}$
Output: $q \in \{0,1\}$ and $r \in [0, W-1]$ s.t. $X - b - Y = -qW + r$

Function $\mathsf{SUB}^{borrow}(X, Y, b)$:
  $q \leftarrow 0$;
  $r \leftarrow X \boxminus b$;
  $q \leftarrow (X < b)$;
  $q \leftarrow q + (r < Y)$;
  $r \leftarrow r \boxminus Y$;
  return q, r;
end

```c
void sub_borrow(WORD x, WORD y, WORD b, WORD* ptrQ, WORD* ptrR) {
    WORD tmp = x - *ptrQ;
    *ptrQ = 0x00;
    if (x < tmp) *ptrQ = 0x01;
    if (tmp < y) *ptrQ += 0x01;
    tmp -= y; *ptrR = tmp;
}
```

2.2.3 Multi-Precision Subtraction

Algorithm 7: $\mathsf{SUB}^{\mathsf{Core}}(X, Y)$

Input: $X = \sum_{i=0}^{n-1} x_i W^i$ and $Y = \sum_{i=0}^{m-1} y_i W^i$, where $a_i, b_i \in [0, W-1]$ and $X \geq Y > 0$.
Output: $Z = X - Y = \sum_{i=0}^{l-1} z_i W^i$

Function $\mathsf{SUB}^{\mathsf{Core}}(X, Y)$**:**
  for $i = m$ **to** $n-1$ **do**
    $y_i \leftarrow 0$; // $0 \cdots 0 \parallel y_{m-1} \cdots y_0$
  end
  $b \leftarrow 0$;
  for $i = 0$ **to** $n-1$ **do**
    $b, z_i \leftarrow \mathsf{SUB}^{\mathsf{borrow}}(x_i, y_i, b)$;
  end
  $l \leftarrow \min\{i : z_{n-1} = z_{n-2} = \cdots = z_i = 0\}$;
  return $\sum_{i=0}^{l-1} z_i W^i$;
end

```c
void sub_core_xyz(BINT** pptrX, BINT** pptrY, BINT** pptrZ) {
    CHECK_PTR_AND_DEREF(pptrX, "pptrX", "sub_core_xyz");
    CHECK_PTR_AND_DEREF(pptrY, "pptrY", "sub_core_xyz");
    if(!compare_abs_bint(pptrX, pptrY)) swapBINT(pptrX,pptrY);

    int n = (*pptrX)->wordlen;
    int m = (*pptrY)->wordlen;

    init_bint(pptrZ, n);
    CHECK_PTR_AND_DEREF(pptrZ, "pptrZ", "sub_core_xyz");

    WORD res = 0x00;
    WORD borrow = 0x00;

    matchSize(*pptrX,*pptrY);
    for(int i = 0; i < m; i++) {
        sub_borrow((*pptrX)->val[i], (*pptrY)->val[i], &borrow, &res);
        (*pptrZ)->val[i] = res;
    }
    for(int i = m; i < n; i++) {
        sub_borrow((*pptrX)->val[i], (WORD)0, &borrow, &res);
        (*pptrZ)->val[i] = res;
    }
    refine_BINT(*pptrX); refine_BINT(*pptrY);
}
```

Algorithm 8: SUB(X, Y)

Input: $X, Y \in \mathbb{Z}$
Output: $X - Y \in \mathbb{Z}$

Function SUB(X, Y)**:**
  if $0 < Y \leq X$ **then**
    return SUB$^{\mathsf{XY}}(X, Y)$
  else if $0 < X < Y$ **then**
    return $-$SUB$^{\mathsf{XY}}(Y, X)$;
  else if $0 > X \geq Y$ **then**
    return SUB$^{\mathsf{XY}}(|Y|, |X|)$;
  else if $0 > Y > X$ **then**
    return $-$SUB$^{\mathsf{XY}}(|X|, |Y|)$;
  else if $X > 0$ && $Y < 0$ **then**
    return ADD$(X, |Y|)$;
  else
    return $-$ADD$(|X|, Y)$
  end
end

```c
void SUB(BINT** pptrX, BINT** pptrY, BINT** pptrZ) {
    CHECK_PTR_AND_DEREF(pptrX, "pptrX", "SUB");
    CHECK_PTR_AND_DEREF(pptrY, "pptrY", "SUB");
    int n = (*pptrX)->wordlen; int m = (*pptrY)->wordlen;
    bool sgnX = (*pptrX)->sign; bool sgnY = (*pptrY)->sign;
    init_bint(pptrZ, MAX(n,m));
    CHECK_PTR_AND_DEREF(pptrZ, "pptrZ", "SUB");

    if ((*pptrY)->sign == false && compare_bint(pptrX,pptrY)) {
        sub_core_xyz(pptrX,pptrY,pptrZ);
    } else if ((*pptrX)->sign == false && !compare_bint(pptrX,pptrY)) {
        sub_core_xyz(pptrY,pptrX,pptrZ); (*pptrZ)->sign = true;
    } else if ((*pptrX)->sign == true && compare_bint(pptrX,pptrY)) {
        (*pptrX)->sign = false; (*pptrY)->sign = false;
        sub_core_xyz(pptrY,pptrX,pptrZ);
    } else if ((*pptrY)->sign == true && !compare_bint(pptrX,pptrY)) {
        (*pptrX)->sign = false; (*pptrY)->sign = false;
        sub_core_xyz(pptrX,pptrY,pptrZ); (*pptrZ)->sign = true;
    } else if ((*pptrX)->sign == false && (*pptrY)->sign == true) {
        (*pptrY)->sign = false; ADD(pptrX,pptrY,pptrZ);
    } else {
        (*pptrX)->sign = false; ADD(pptrX,pptrY,pptrZ);
        (*pptrZ)->sign = true;
    }
    (*pptrX)->sign = sgnX; (*pptrY)->sign = sgnY;
}
```

Chapter 3

Multiplication

Note (Memory for Addition).

$$\begin{cases} A \in \left[W^{n-1}, W^{n}\right) \\ B \in \left[W^{n-1}, W^{n}\right) \end{cases} \implies \mathsf{wordlen}(AB) \in \{n+m-1, n+m\}$$

Proof. Since

$$\begin{aligned} W^{n-1} \cdot W^{m-1} &\leq AB < W^{n} \cdot W^{m}, \\ W^{n+m-2} &\leq AB < W^{n+m}, \end{aligned}$$

we have $AB \in \left[W^{n+m-2}, W^{n+m}\right) = \left[W^{n+m-2}, W^{n+m-1}\right) \cup \left[W^{n+m-1}, W^{n+m}\right)$. Thus, either

$$\mathsf{wordlen}(AB) = n+m-1 \quad \text{or} \quad \mathsf{wordlen}(AB) = n+m.$$

□

3.1 Single-Word Multiplication: `mul_xyz`

Note that

$$P, Q \in \left[0, W^{1/2}\right) \implies PQ \in [0, W).$$

Let A, B satisfy the following:

$$\begin{cases} A = A_1 W^{1/2} + A_0 \\ B = B_1 W^{1/2} + B_0 \end{cases}, \text{ where } A_i, B_i \in \left[0, W^{1/2}\right) \text{ for } i = 0, 1.$$

The product AB can be calculated using four $w/2$-bit integer multiplication operations:

$$\begin{array}{r} A_1 \,\|\, A_0 \\ \times \qquad B_1 \,\|\, B_0 \end{array}$$

		A_0B_0	
	A_1B_0		
	A_0B_1		
A_1B_1			

$$\begin{aligned} AB &= (A_1 W^{1/2} + A_0)(B_1 W^{1/2} + B_0) \\ &= (A_1B_1)W + A_0B_0 + (A_1B_0 + A_0B_1)W^{1/2} \\ &= ((A_1B_1 \ll w) + A_0B_0) + ((A_1B_0 + A_0B_1) \ll w/2). \end{aligned}$$

Algorithm 9: Single-Word Multiplication

Input: $X, Y \in [0, W)$

Output: $Z = XY \in \left[0, W^2\right)$

1 **Function** $\mathsf{MUL}^{XY}(X, Y)$**:**

2 $X_1, X_0 \leftarrow X_{[w:w/2]}, X_{[w/2:0]}$;

3 $Y_1, Y_0 \leftarrow Y_{[w:w/2]}, Y_{[w/2:0]}$;

4 $T_1, T_0 \leftarrow X_1 Y_0, X_0 Y_1$; // $T_1, T_0 \in [0, W)$

5 $T_0 \leftarrow T_1 \boxplus T_0$;

6 $T_1 \leftarrow T_0 < T_1$; // $T_1 W + T_0 = X_1 Y_0 + X_0 Y_1$, where $T_1 \in \{0, 1\}$ is carry

7 $Z_1, Z_0 \leftarrow X_1 Y_1, X_0 Y_0$; // $Z_1, Z_0 \in [0, W)$

8 $T \leftarrow Z_0$;

9 $Z_0 \leftarrow Z_0 \boxplus (T_0 \ll w/2)$;

/* $Z_0 = \left[X_0 Y_0 + (X_1 Y_0 + X_0 Y_1) 2^{w/2}\right] \bmod 2^w$ */

10 $Z_1 \leftarrow Z_1 + (T_1 \ll w/2) + (T_0 \gg w/2) + (Z_0 < T)$; // $Z_1 \in [0, W)$

/* $Z_1 = X_1 Y_1 + (T_0 < T_1) 2^{w/2} + \left\lfloor T_0 / 2^{w/2} \right\rfloor +$ (carry in line 9) */

11 **return** $(Z_1 \ll w) + Z_0$; // $Z \leftarrow Z_1 \parallel Z_0 \in \left[0, W^2\right)$

12 **end**

```c
void mul_xyz(WORD valX, WORD valY, BINT** pptrZ) {
    if (!pptrZ || !*pptrZ || !(*pptrZ)->val) { return; }

    int half_w = WORD_BITLEN / 2; // if w=32, half_w = 16 = 2^4
    WORD MASK = (1 << half_w) - 1;

    // Split the WORDs into halves
    WORD X0 = valX & MASK; WORD X1 = valX >> half_w;
    WORD Y0 = valY & MASK; WORD Y1 = valY >> half_w;
    // Cross multiplication
    WORD T0 = X0 * Y1; WORD T1 = X1 * Y0;
    T0 = T0 + T1;
    T1 = T0 < T1; // overflow
    // Direct multiplication
    WORD Z0 = X0 * Y0; WORD Z1 = X1 * Y1;
    // Adjust for overflows
    WORD T = Z0;
    Z0 += (T0 << half_w);
    Z1 += (T1 << half_w) + (T0 >> half_w) + (Z0 < T);
    // Set results
    (*pptrZ)->val[0] = Z0;
    (*pptrZ)->val[1] = Z1;
}
```

3.2 Multi-Precision Multiplication

$\mathsf{MUL}^{\mathsf{Core}}$	Computational Complexity	Year
TextBook	$O(n^2)$	-
Karatsuba	$O(n^{\log_2 3}) = O(n^{1.585})$	1960
Toom-Cook	$O(n^{\log_3 5}) = O(n^{1.465})$	1963

3.2.1 Textbook: `mul_core_TxtBk_xyz`

Let $\begin{cases} A = A_{n-1} \parallel \cdots \parallel A_0 = \sum_{i=0}^{n-1} A_i W^i \\ B = B_{m-1} \parallel \cdots \parallel B_0 = \sum_{j=0}^{m-1} B_j W^j \end{cases}$ with $A_i, B_i \in [0, W)$. Then

$$\begin{array}{lr} & A_2 \parallel A_1 \parallel A_0 \\ \times & B_2 \parallel B_1 \parallel B_0 \end{array}$$

<table>
<tr><td></td><td></td><td></td><td></td><td colspan="2">A_0B_0</td></tr>
<tr><td></td><td></td><td></td><td colspan="2">A_1B_0</td><td></td></tr>
<tr><td></td><td></td><td colspan="2">A_2B_0</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td colspan="2">A_0B_1</td><td></td></tr>
<tr><td></td><td></td><td colspan="2">A_1B_1</td><td></td><td></td></tr>
<tr><td></td><td colspan="2">A_2B_1</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td colspan="2">A_0B_2</td><td></td><td></td></tr>
<tr><td></td><td colspan="2">A_1B_2</td><td></td><td></td><td></td></tr>
<tr><td colspan="2">A_2B_2</td><td></td><td></td><td></td><td></td></tr>
</table>

$$\begin{aligned} C = AB &= \left(\sum_{i=0}^{n-1} A_i W^i\right)\left(\sum_{j=0}^{m-1} B_j W^j\right) \\ &= \sum_{j=0}^{m-1}\left(\sum_{i=0}^{n-1}(A_iB_j)W^{i+j}\right) \in \left[0, W^{n+m}\right). \end{aligned}$$

Algorithm 10: Textbook Multiplication

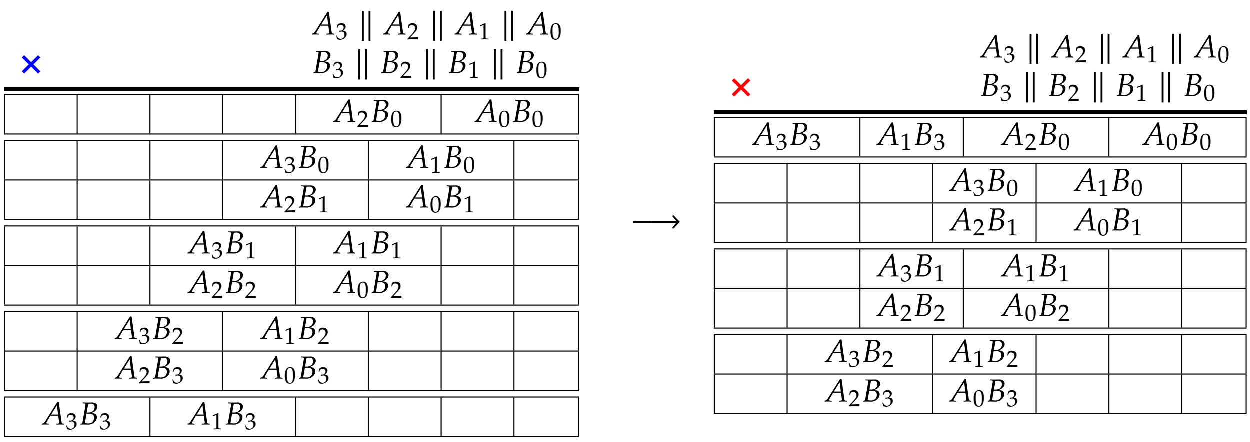
Input: $X = \sum_{i=0}^{n-1} x_i W^i, Y = \sum_{j=0}^{m-1} y_j W^j$, where $x_i, y_i \in [0, W)$
Output: $Z = XY \in \left[0, W^{n+m}\right)$

Function $\mathsf{MUL}^{\mathsf{Core-TxtBk}}(X, Y)$:
  $Z \leftarrow 0$;
  for $i = 0$ **to** $n-1$ **do**
    for $j = 0$ **to** $m-1$ **do**
      $T \leftarrow x_i y_j$;
      $T \leftarrow T \ll w(i+j)$;
      $Z \leftarrow \mathsf{ADD}^{\mathsf{Core}}(Z, T)$;
    end
  end
  return Z;
end

```c
void mul_core_TxtBk_xyz(BINT** pptrX, BINT** pptrY, BINT** pptrZ) {
    CHECK_PTR_AND_DEREF(pptrX, "pptrX", "mul_core_TxtBk_xyz");
    CHECK_PTR_AND_DEREF(pptrY, "pptrY", "mul_core_TxtBk_xyz");

    int n = (*pptrX)->wordlen;
    int m = (*pptrY)->wordlen;

    init_bint(pptrZ, n+m);
    CHECK_PTR_AND_DEREF(pptrZ, "pptrZ", "mul_core_TxtBk_xyz");

    BINT* ptrWordMul = NULL;
    BINT* ptrTemp = NULL;
    init_bint(&ptrTemp, m+n);;

    for(int i = 0; i < n; i++) {
        for(int j = 0; j < m; j++) {
            init_bint(&ptrWordMul, 2);
            mul_xyz((*pptrX)->val[i], (*pptrY)->val[j], &ptrWordMul);

            left_shift_word(&ptrWordMul, (i+j));

            add_core_xyz(pptrZ, &ptrWordMul , &ptrTemp);
            copyBINT(pptrZ,&ptrTemp);
        }
    }
    delete_bint(&ptrWordMul);

    if((*pptrX)->sign != (*pptrY)->sign)
        (*pptrZ)->sign = true;
}
```

3.2.2 Improved Textbook: `MUL_Core_ImpTxtBk_xyz`

Let $n = 2p$ and $m = 2q$, and let $\begin{cases} A = A_{2p-1} \parallel \cdots \parallel A_0 = \sum_{i=0}^{2p-1} A_i W^i \\ B = B_{2q-1} \parallel \cdots \parallel B_0 = \sum_{j=0}^{2q-1} B_j W^j \end{cases}$ with $A_i, B_i \in [0, W)$. Then



$$
\begin{aligned}
AB = \left(\sum_{i=0}^{2p-1} A_i W^i\right)\left(\sum_{j=0}^{2q-1} B_j W^j\right) &= \sum_{j=0}^{2q-1}\left(\sum_{i=0}^{2p-1} (A_i B_j) W^{i+j}\right) \\
&= \sum_{j=0}^{2q-1}\left(\sum_{k=0}^{p-1} (A_{2k}B_j)W^{2k+j} + \sum_{k=0}^{p-1} (A_{2k+1}B_j)W^{2k+1+j}\right) \\
&= \sum_{j=0}^{2q-1}\left(\left(\sum_{k=0}^{p-1} (A_{2k}B_j)W^{2k}\right)W^j + \left(\sum_{k=0}^{p-1} (A_{2k+1}B_j)W^{2k}\right)W^{j+1}\right) \\
&= \sum_{j=0}^{2q-1}\left(\left(\sum_{k=0}^{p-1} (A_{2k}B_j)W^{2k}\right) + \left(\sum_{k=0}^{p-1} (A_{2k+1}B_j)W^{2k}\right)W\right)W^j
\end{aligned}
$$

Algorithm 11: Improved Textbook Multiplication

Input: $X = \sum_{i=0}^{n-1} x_i W^i, Y = \sum_{j=0}^{m-1} y_j W^j$, where $n = 2p$, $m = 2q$, and $x_i, y_i \in [0, W)$

Output: $Z = XY \in \left[0, W^{n+m}\right) = \left[0, W^{2(p+q)}\right)$

Function $\mathsf{MUL}^{\mathsf{Core-ImpTxtBk}}(X, Y)$**:**
    $Z \leftarrow 0$;
    for $j = 0$ **to** $2q - 1$ **do**
        $T_0, T_1 \leftarrow \mathtt{NULL}, \mathtt{0}^w$;
        for $k = 0$ **to** $p - 1$ **do**
            $T_0 \leftarrow x_{2k} y_j \parallel T_0$;
            $T_1 \leftarrow x_{2k+1} y_j \parallel T_1$;
        end
        $T \leftarrow \mathsf{ADD}^{\mathsf{Core}}(T_1, T_0)$; // $\mathsf{wordlen}(T_1) \geq \mathsf{wordlen}(T_0)$
        $T \leftarrow T \ll wj$;
        $Z \leftarrow \mathsf{ADD}^{\mathsf{Core}}(Z, T)$; // $\mathsf{wordlen}(Z) = \mathsf{wordlen}(T)$
    end
    return Z;
end

j	k	T_0	T_1	wordlen(T_0)	wordlen(T_1)
0	0	$x_0y_0 \parallel \mathtt{NULL}$	$x_1y_0 \parallel \mathtt{0}^w$	W^2	W^2W
	1	$x_2y_0 \parallel x_0y_0$	$x_3y_0 \parallel x_1y_0 \parallel \mathtt{0}^w$	W^2W^2	W^2W^2W
	2	$x_4y_0 \parallel x_2y_0 \parallel x_0y_0$	$x_5y_0 \parallel x_3y_0 \parallel x_1y_0 \parallel \mathtt{0}^w$	$W^{2\cdot3}$	$W^{2\cdot3+1}$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$p-1$	$x_{2(p-1)}y_0 \parallel \cdots \parallel x_0y_0$	$x_{2p-1}y_0 \parallel \cdots \parallel \mathtt{0}^w$	W^{2p}	W^{2p+1}
1	0	$x_0y_1 \parallel \mathtt{NULL}$	$x_1y_1 \parallel \mathtt{0}^w$	W^2	W^2W
	1	$x_2y_1 \parallel x_1y_1$	$x_3y_1 \parallel x_1y_1 \parallel \mathtt{0}^w$	W^2W^2	W^2W^2W
	2	$x_4y_1 \parallel x_2y_1 \parallel x_0y_1$	$x_5y_1 \parallel x_3y_1 \parallel x_1y_1 \parallel \mathtt{0}^w$	$W^{2\cdot3}$	$W^{2\cdot3+1}$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$p-1$	$x_{2(p-1)}y_1 \parallel \cdots \parallel x_0y_1$	$x_{2p-1}y_1 \parallel \cdots \parallel \mathtt{0}^w$	W^{2p}	W^{2p+1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2q-1$	0	$x_0y_{2q-1} \parallel \mathtt{NULL}$	$x_1y_{2q-1} \parallel \mathtt{0}^w$	W^2	W^2W
	1	$x_2y_{2q-1} \parallel x_1y_{2q-1}$	$x_3y_{2q-1} \parallel x_1y_{2q-1} \parallel \mathtt{0}^w$	W^2W^2	W^2W^2W
	2	$x_4y_{2q-1} \parallel x_2y_{2q-1} \parallel x_1y_{2q-1}$	$x_5y_{2q-1} \parallel \cdots \parallel \mathtt{0}^w$	W^2W^2	W^2W^2W
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$p-1$	$x_{2(p-1)}y_{2q-1} \parallel \cdots \parallel x_0y_{2q-1}$	$x_{2p-1}y_{2q-1} \parallel \cdots \parallel \mathtt{0}^w$	W^{2p}	W^{2p+1}

Table 3.1: Values of T_0 and T_1 for different j and k

```
void MUL_Core_ImpTxtBk_xyz(BINT** pptrX, BINT** pptrY, BINT** pptrZ) {
    CHECK_PTR_AND_DEREF(pptrX, "pptrX", "mul_core_ImpTxtBk_test");
    CHECK_PTR_AND_DEREF(pptrY, "pptrY", "mul_core_ImpTxtBk_test");
    if(!compare_abs_bint(pptrX,pptrY)) swapBINT(pptrX,pptrY);
    matchSize(*pptrX, *pptrY); makeEven(*pptrX); makeEven(*pptrY);
    int n = (*pptrX)->wordlen; int m = (*pptrX)->wordlen;
    init_bint(pptrZ, n+m);
    CHECK_PTR_AND_DEREF(pptrZ, "pptrZ", "mul_core_ImpTxtBk_test");
    int p = n / 2; int q = n / 2;
    BINT* ptrT = NULL; init_bint(&ptrT, n+m);
    BINT* ptrT0 = NULL; init_bint(&ptrT0, 2*p);
    BINT* ptrT1 = NULL; init_bint(&ptrT1, 2*p+1);
    BINT* ptrTmp0 = NULL; init_bint(&ptrTmp0, 2*p);
    BINT* ptrTmp1 = NULL; init_bint(&ptrTmp1, 2*p+1);
    BINT* ptrTmpZ = NULL; init_bint(&ptrTmpZ, (*pptrZ)->wordlen);

    for(int j = 0; j < 2 * q; j++) {
        for(int k = 0; k < p; k++) {
            reset_bint(ptrTmp0); reset_bint(ptrTmp1);
            mul_xyz((*pptrX)->val[2*k], (*pptrY)->val[j], &ptrTmp0);
            mul_xyz((*pptrX)->val[2*k+1], (*pptrY)->val[j], &ptrTmp1);
            if (!k) {
                copyBINT(&ptrT0, &ptrTmp0);
                copyBINT(&ptrT1, &ptrTmp1);
            } else {
                left_shift_word(&ptrTmp0, 2*k);
                refine_BINT_word(ptrTmp0, 2*k);
                OR_BINT(ptrTmp0, ptrT0, &ptrT0);
                left_shift_word(&ptrTmp1, 2*k);
                refine_BINT_word(ptrTmp1, 2*k);
                OR_BINT(ptrTmp1, ptrT1, &ptrT1);
            }
        }
        left_shift_word(&ptrT1, 1);

        add_core_xyz(&ptrT1, &ptrT0, &ptrT);
        left_shift_word(&ptrT, j);

        copyBINT(&ptrTmpZ, pptrZ);
        add_core_xyz(&ptrTmpZ, &ptrT, pptrZ);
    }
    delete_bint(&ptrT); delete_bint(&ptrT0); delete_bint(&ptrT1);
    delete_bint(&ptrTmp0); delete_bint(&ptrTmp1);
    delete_bint(&ptrTmpZ);
    refine_BINT(*pptrX); refine_BINT(*pptrY);
    if((*pptrX)->sign != (*pptrY)->sign) (*pptrZ)->sign = true;
}
```

3.2.3 Karatsuba (★) : `MUL_Core_Krtsb_xyz`

Note (Divide and Conquer). Let $A \in \left[W^{n-1}, W^n\right)$ and $B \in \left[W^{m-1}, W^m\right)$ then

$$A = A_1 W^l + A_0, \quad B = B_1 W^l + B_0, \quad A_i, B_i \in \left[0, W^l\right), \quad l = \left\lfloor \frac{\max(n,m)+1}{2} \right\rfloor.$$

And so

$$\begin{aligned} AB = (A_1W^l + A_0)(B_1W^l + B_0) &= \left((A_1B_1)W^{2l} + (A_0B_0)\right) + (A_0B_1 + A_1B_0)W^l \\ &= \left((A_1B_1)W^{2l} + (A_0B_0)\right) + \left((A_0 - A_1)(B_1 - B_0) + A_0B_0 + A_1B_1\right) W^l. \end{aligned}$$

Note. $A_0B_1 + A_1B_0 = (A_0 + A_1)(B_1 + B_0) - A_0B_0 - A_1B_1$ but $(A_0 + A_1)(B_1 + B_0) \in \left[0, W^{l+1}W^{l+1}\right)$.

Time Complexity of Karatsuba

Proposition 3.1. *The Karatsuba multiplication of n-word integers has a computational complexity of $O(n^{\log_2 3})$ based on a 1-word multiplication operation.*

Algorithm 12: Karatsuba Multiplication

Input: flag, $X = \sum_{i=0}^{n-1} x_i W^i \in [0, W^n)$, $Y = \sum_{j=0}^{m-1} y_j W^j \in [0, W^m)$, where $x_i, y_i \in [0, W)$

Output: $Z = XY \in \left[0, W^{n+m}\right)$

Function $\mathsf{MUL}^{\mathsf{Core-Krtsb}}(X, Y)$**:**
  /* n = wordlen(X) and m = wordlen(Y) */
  if $flag \geq \min(n, m)$ **then**
    return $\mathsf{MUL}^{\mathsf{Core-ImpTxtBk}}(X, Y)$; // Improved Textbook Multiplication
  end
  $l \leftarrow \max(n, m+1) \gg 1$;
  $X_1, X_0 \leftarrow X \gg lw, X \bmod 2^{lw}$; // $X_i \in \left[0, W^l\right)$
  $Y_1, Y_0 \leftarrow X \gg lw, Y \bmod 2^{lw}$; // $Y_i \in \left[0, W^l\right)$
  $T_1, T_0 \leftarrow \mathsf{MUL}^{\mathsf{Core-Krtsb}}(X_1, Y_1), \mathsf{MUL}^{\mathsf{Core-Krtsb}}(X_0, Y_0)$; // $T_i \in \left[0, W^{2l}\right)$
  $Z \leftarrow (T_1 \ll 2lw) + T_0$; // $Z = T_1 \parallel T_0 \in \left[0, W^{4l}\right)$
  $S_1, S_0 \leftarrow \mathsf{SUB}(X_0, X_1), \mathsf{SUB}(Y_1, Y_0)$;
  $S \leftarrow (-1)^{\mathsf{sgn}(S_1) \oplus \mathsf{sgn}(S_2)} \mathsf{MUL}^{\mathsf{Core-Krtsb}}(|S_1|, |S_0|)$;
  $S \leftarrow \mathsf{ADD}(S, T_1)$;
  $S \leftarrow \mathsf{ADD}(S, T_0)$; // $S \geq 0$
  $S \leftarrow S \ll lw$;
  $Z \leftarrow \mathsf{ADD}(Z, S)$;
  return Z;
end

```
void MUL_Core_Krtsb_xyz(BINT** pptrX, BINT** pptrY, BINT** pptrZ) {
    CHECK_PTR_AND_DEREF(pptrX, "pptrX", "MUL_Core_Krtsb_xyz");
    CHECK_PTR_AND_DEREF(pptrY, "pptrY", "MUL_Core_Krtsb_xyz");

    int n = (*pptrX)->wordlen; int m = (*pptrX)->wordlen;
    static int lenZ = -1; // Declare lenZ as a static variable
    if (lenZ == -1) {
        lenZ = n + m; // Calculate lenZ only if it hasn't been initialized yet
        init_bint(pptrZ, lenZ);
        CHECK_PTR_AND_DEREF(pptrZ, "pptrZ", "MUL_Core_Krtsb_xyz");
    }
    if (FLAG >= MIN(n,m)) {
        BINT* tmpTxtBk_X = NULL; BINT* tmpTxtBk_Y = NULL;
        copyBINT(&tmpTxtBk_X, pptrX);
        copyBINT(&tmpTxtBk_Y, pptrY);
        MUL_Core_ImpTxtBk_xyz(&tmpTxtBk_X,&tmpTxtBk_Y,pptrZ);
        delete_bint(&tmpTxtBk_X); delete_bint(&tmpTxtBk_Y);
        return;
    }
    init_bint(pptrZ, n+m);
    CHECK_PTR_AND_DEREF(pptrZ, "pptrZ", "MUL_Core_Krtsb_xyz");
    matchSize(*pptrX,*pptrY);
    int l = (MAX(n,m) + 1) >> 1;

    BINT* ptrX0 = NULL; BINT* ptrX1 = NULL;
    BINT* ptrY0 = NULL; BINT* ptrY1 = NULL;
    BINT* ptrT0 = NULL; BINT* ptrT1 = NULL;
    BINT* ptrShiftT1 = NULL;
    BINT* ptrS0 = NULL; BINT* ptrS1 = NULL;
    BINT* ptrS = NULL; init_bint(&ptrS, 2*l);

    BINT* ptrR = NULL; init_bint(&ptrR, (*pptrZ)->wordlen);
    BINT* ptrTmpR = NULL;
    BINT* ptrTmpST1 = NULL; BINT* ptrTmpST0 = NULL;

    copyBINT(&ptrX1, pptrX); right_shift_word(&ptrX1, l);
    copyBINT(&ptrX0, pptrX); reduction(&ptrX0, l * WORD_BITLEN);

    copyBINT(&ptrY1, pptrY); right_shift_word(&ptrY1, l);
    copyBINT(&ptrY0, pptrY); reduction(&ptrY0, l * WORD_BITLEN);

    MUL_Core_Krtsb_xyz(&ptrX1, &ptrY1, &ptrT1);
    MUL_Core_Krtsb_xyz(&ptrX0, &ptrY0, &ptrT0);

    copyBINT(&ptrShiftT1, &ptrT1);
    left_shift_word(&ptrShiftT1, 2*l);
```

```
    ADD(&ptrShiftT1, &ptrT0, &ptrR);

    SUB(&ptrX0, &ptrX1, &ptrS1);
    SUB(&ptrY1, &ptrY0, &ptrS0);

    bool sgn_S = ((ptrS0)->sign) ^ ((ptrS1)->sign);
    ptrS0->sign = false;
    ptrS1->sign = false;
    MUL_Core_Krtsb_xyz(&ptrS1, &ptrS0, &ptrS);
    ptrS->sign = sgn_S;

    ADD(&ptrS,&ptrT1,&ptrTmpST1);
    copyBINT(&ptrS,&ptrTmpST1);

    ADD(&ptrS,&ptrT0,&ptrTmpST0);
    copyBINT(&ptrS, &ptrTmpST0);

    left_shift_word(&ptrS, 1);

    ADD(&ptrR, &ptrS, pptrZ);

    delete_bint(&ptrX0); delete_bint(&ptrX1);
    delete_bint(&ptrY0); delete_bint(&ptrY1);
    delete_bint(&ptrT0); delete_bint(&ptrT1);
    delete_bint(&ptrShiftT1);
    delete_bint(&ptrS0); delete_bint(&ptrS1);
    delete_bint(&ptrS);
    delete_bint(&ptrR); delete_bint(&ptrTmpR);
    delete_bint(&ptrTmpST0); delete_bint(&ptrTmpST1);
}
```

3.3 Measuring Performance

3.3.1 Hardware Environment

The experiments were conducted on the following hardware setup:

- **Processor:** AMD Ryzen 7 5800X3D, 3400 MHz, 8-Core
- **Memory:** 32 GB, DDR4-3200 (16GB) PC4-25600 ×2
- **Storage:** 1 TB, Samsung 980 PRO M.2 NVMe SSD
- **Operating System:** Linux Mint 21.1 Vera x86_64
- **Compiler Version:** gcc (Ubuntu 11.4.0-1ubuntu1 22.04) 11.4.0

3.3.2 Textbook vs Improved Textbook vs Karatsuba

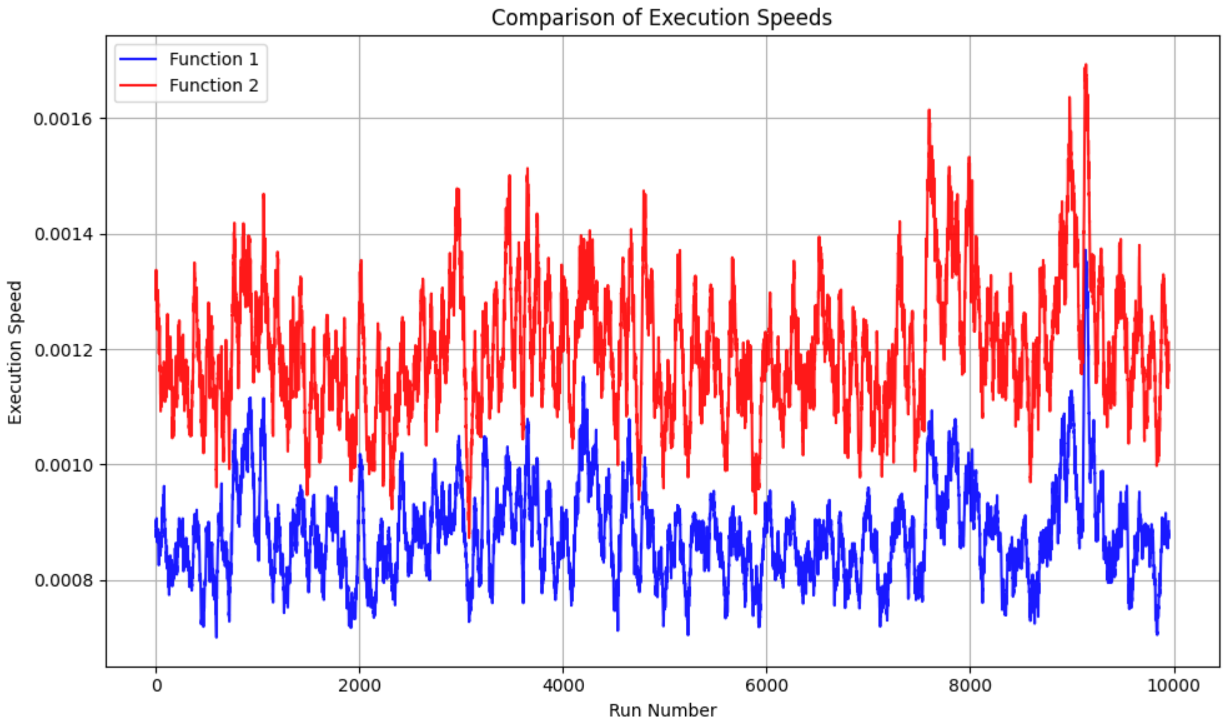


Figure 3.1: `MUL_Core_ImpTxtBk_xyz` and `mul_core_TxtBk_xyz` in 1024 ~ 2048 bits

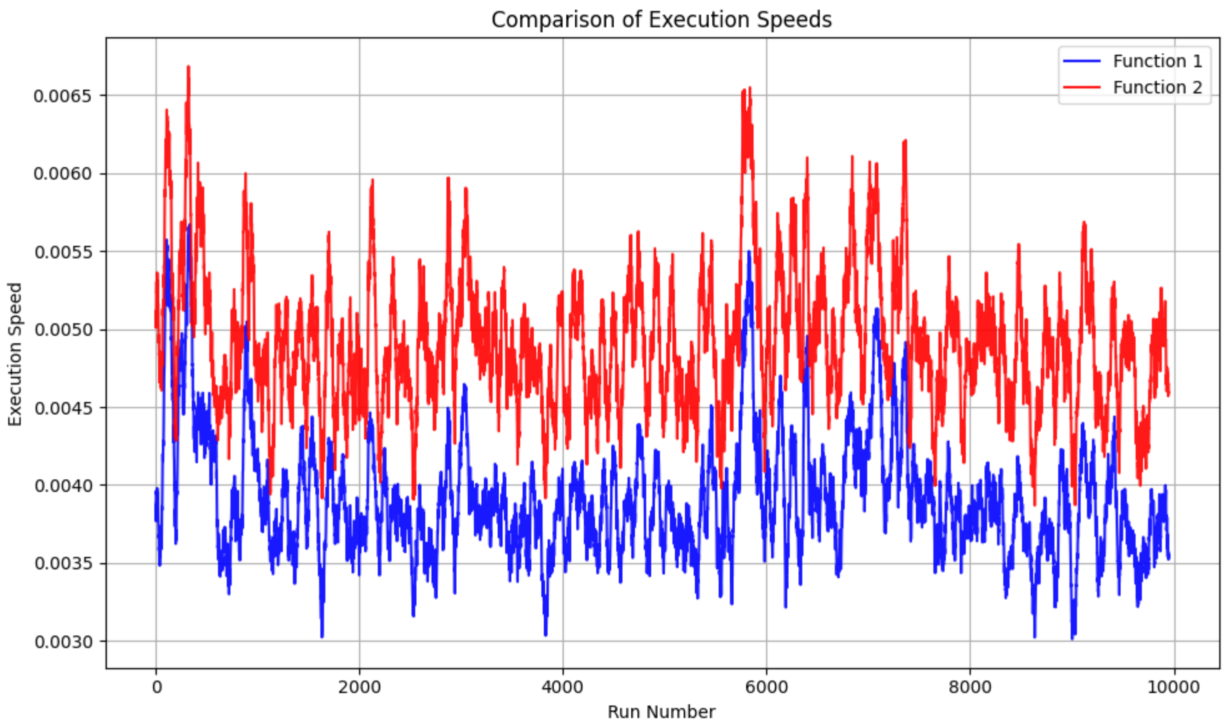


Figure 3.2: `MUL_Core_ImpTxtBk_xyz` and `mul_core_TxtBk_xyz` in 2048 ~ 3072 bits

Figure 3.3: `MUL_Core_ImpTxtBk_xyz` and `mul_core_TxtBk_xyz` in 3072 ∼ 7680 bits

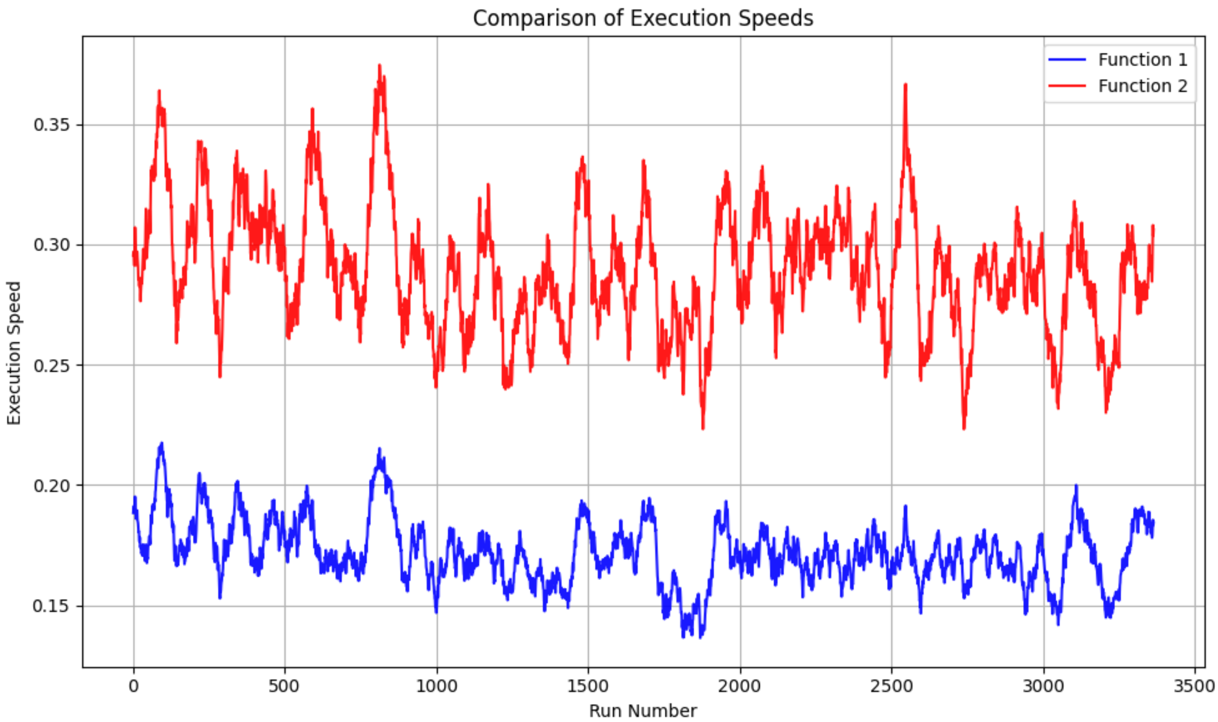


Figure 3.4: `MUL_Core_ImpTxtBk_xyz` and `mul_core_TxtBk_xyz` in 7680 ∼ 15360 bits

Figure 3.5: **`TxtBk`** v.s. **`ImpTxtBk`** v.s. **`Krtsb`** in 1024-bit with 10,000 experiments

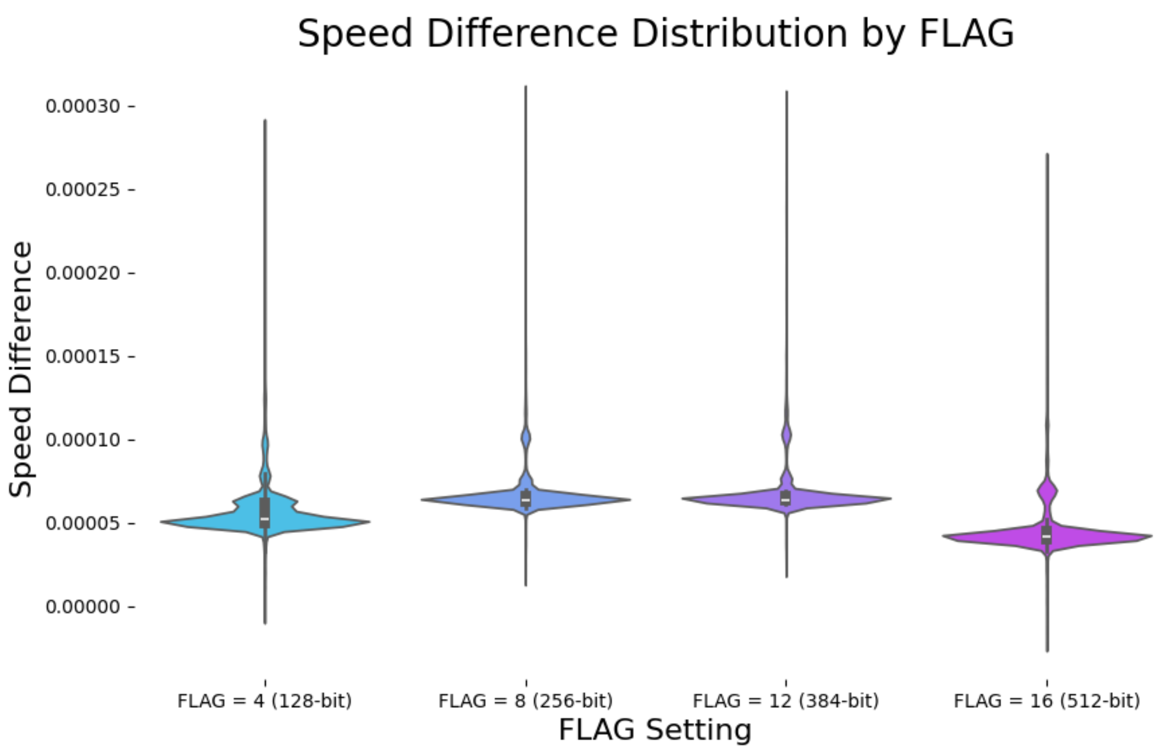


Figure 3.6: FLAGs in 1024-bit **`Krtsb`** with 10,000 experiments

Figure 3.7: **TxtBk** v.s. **ImpTxtBk** v.s. **Krtsb** in 3072-bit with 10,000 experiments

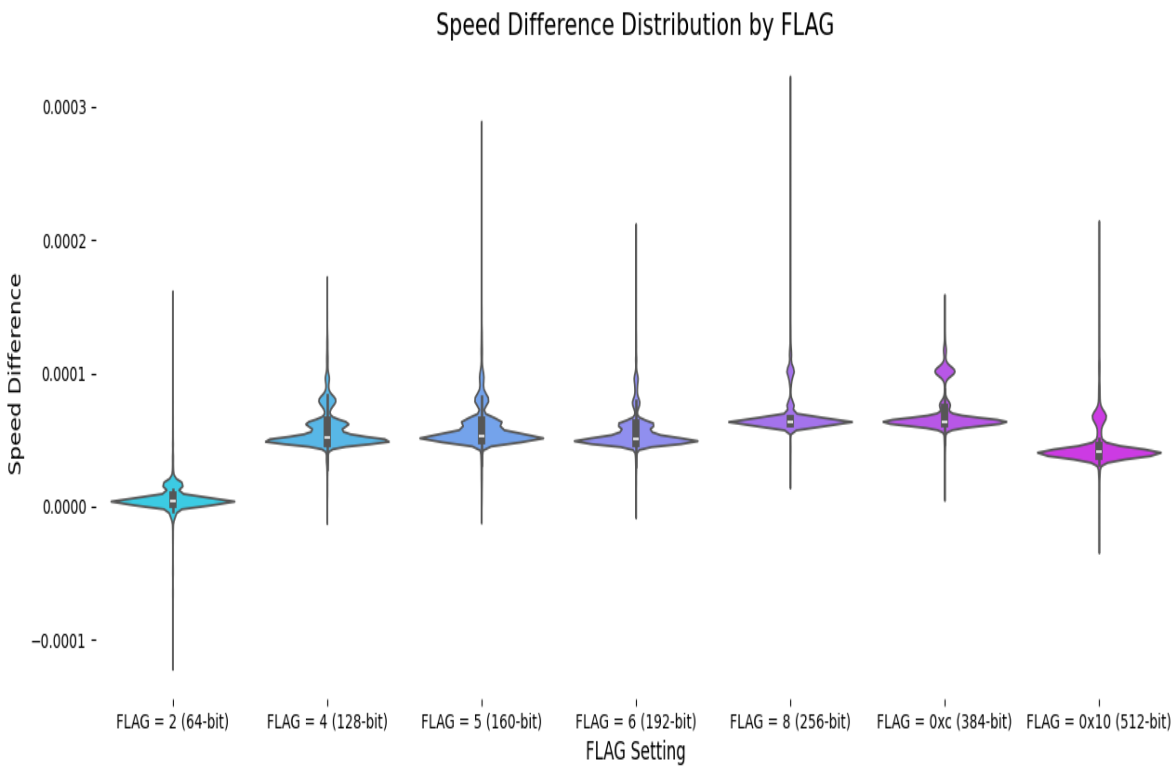


Figure 3.8: FLAGs in 3072-bit **Krtsb** with 10,000 experiments

Chapter 4

Division

Note (*General Dividend*)**.** For two non-negative integers $A \in \mathbb{Z}$ and $B \in \mathbb{Z} \setminus 0$, the Division Algorithm returns a quotient (Q) and a remainder (R) that satisfy the following:

$$A = BQ + R, \quad 0 \leq R < B$$

If $B = 0$, then the operation is undefined, and when $A < B$, then $Q = 0$ and $R = A$. Therefore, we only need to consider the division operation for the case where $A \geq B > 0$.

Note (*Negative Dividend*)**.** In the case where $A < 0$, if we compute the quotient Q_0 and the remainder R_0 for $|A|$ then $-Q_0 - 1$ is the quotient when dividing A by B, and $B - R$ is the remainder.

$$\begin{aligned} |A| = BQ_0 + R_0 &\implies -|A| = -BQ_0 - R_0 \\ &\implies -|A| = -BQ_0 - B + B - R_0 \\ &\implies A = -|A| = (-Q_0 - 1)B + (B - R_0). \end{aligned}$$

Note (Memory for Division). For a non-negative n-word integer $A \in \left[W^{n-1}, W^n\right)$ and an m-word integer $B \in \left[W^{m-1}, W^m\right)$, the quotient Q is at most an $(n - m + 1)$-word integer, and the remainder R is at most an m-word integer.

Proof. (1) $0 \leq R < B \in \left[W^{m-1}, W^m\right) \implies R \in \left[W^{m-1}, W^m\right)$.

(2) Since $B \neq 0$,

$$A = BQ + R \implies Q = \frac{A - R}{B} \leq \frac{A}{B} < \frac{W^n}{W^{m-1}} = W^{n-m+1}.$$

□

4.1 Naive Division

Algorithm 13: Naive Division

```
Input: X, Y ∈ ℤ≥0
Output: Invalid or Q, R s.t. X = YQ + R with (0 ≤ R < B)
Function DIV^naive(X, Y):
    if Y ≤ 0 then
        return Invalid;
    end
    if X < Y then
        return (0, X);
    end
    if Y = 1 then
        return (X, 0);
    end
    (Q, R) ← (0, X);
    while R ≥ Y do
        (Q, R) ← (Q + 1, R − Y)
    end
    return (Q, R);
end
```

4.2 Long Division

Example 4.1. Let $A = \sum_{i=0}^{6} a_i 10^i$ with $a_5 \parallel a_4 \parallel a_3 \parallel a_2 \parallel a_1 \parallel a_0 := 1 \parallel 3 \parallel 2 \parallel 4 \parallel 3 \parallel 2$ and $B = 32$.

$$
\begin{array}{r r}
 & 0\;0\;4\;1\;3\;8 \\ \cline{2-2}
32 \big) & 1\;3\;2\;4\;3\;2 \\
 & -1\;2\;8\;\vdots\;\vdots\;\vdots \\ \cline{2-2}
 & 4\;4\;\vdots\;\vdots \\
 & -\;3\;2\;\vdots\;\vdots \\ \cline{2-2}
 & 1\;2\;3\;\vdots \\
 & -\quad 9\;6\;\vdots \\ \cline{2-2}
 & 2\;7\;2 \\
 & -\,2\;5\;6 \\ \cline{2-2}
 & 1\;6
\end{array}
$$

Index (i)	**Quotient Digit** (q_i)	**Remainder** (R_i)
`initial`		$R_6 = 0$
5	$q_5 = \left\lfloor \frac{0\cdot 10+1}{32} \right\rfloor = 0$	$R_5 = 0 \cdot 10 + 1 \bmod 32 = 1$
4	$q_4 = \left\lfloor \frac{1\cdot 10+3}{32} \right\rfloor = 0$	$R_4 = 1 \cdot 10 + 3 \bmod 32 = 13$
3	$q_3 = \left\lfloor \frac{13\cdot 10+2}{32} \right\rfloor = 4$	$R_3 = 13 \cdot 10 + 2 \bmod 32 = 4$
2	$q_2 = \left\lfloor \frac{4\cdot 10+4}{32} \right\rfloor = 1$	$R_2 = 4 \cdot 10 + 4 \bmod 32 = 12$
1	$q_1 = \left\lfloor \frac{12\cdot 10+3}{32} \right\rfloor = 3$	$R_1 = 12 \cdot 10 + 1 \bmod 32 = 27$
0	$q_0 = \left\lfloor \frac{27\cdot 10+2}{32} \right\rfloor = 8$	$R_0 = 27 \cdot 10 + 2 \bmod 32 = 16$

Thus, $Q = \sum_{i=0}^{6} q_i 10^i = 4138$ and $R_0 = 16$.

Remark 4.1. Let $X = \sum_{i=0}^{n-1} x_i W^i$ with $x_i \in [0, W)$ and $Y \neq 0$. Then

Index (i)	**Quotient Digit** (q_i)	**Remainder** (R_i)
`initial`		$R_n = 0$
$n-1$	$q_{n-1} = \left\lfloor \frac{R_n \cdot W + x_{n-1}}{Y} \right\rfloor$	$R_{n-1} = R_n \cdot W + x_{n-1} \bmod Y$
$n-2$	$q_{n-2} = \left\lfloor \frac{R_{n-1} \cdot W + x_{n-2}}{Y} \right\rfloor$	$R_{n-2} = R_{n-1} \cdot W + x_{n-2} \bmod Y$
$\vdots$	$\vdots$	$\vdots$
1	$q_1 = \left\lfloor \frac{R_2 \cdot W + x_1}{Y} \right\rfloor$	$R_1 = R_2 \cdot W + x_1 \bmod Y$
0	$q_0 = \left\lfloor \frac{R_1 \cdot W + x_0}{Y} \right\rfloor$	$R_0 = R_1 \cdot W + x_0 \bmod Y$

Algorithm 14: Multi-precision Long Divison

Input: $X = \sum_{i=0}^{n-1} x_i W^i$ with $x_i \in [0, W)$ and $Y \neq 0$
Output: $Q = \sum_{i=0}^{n-1} q_i W^i$ with $q_i \in [0, W)$ and R s.t. $X = QY + R$ $(0 \leq R < B)$

Function $\mathsf{DIV}^{long}(X, Y)$**:**
  if $X < Y$ **then**
    return $(0, X)$;
  end
  $R_n \leftarrow 0$;
  for $i = n-1$ **downto** 0 **do**
    $(q_i, R_i) = \left(\left\lfloor \frac{R_{i+1} \cdot W + x_i}{Y} \right\rfloor, R_{i+1}W + x_i \bmod Y \right) \leftarrow \mathsf{DIV}^{\mathsf{Core}}(R_{i+1}W + x_i, Y)$;
  end
  $Q \leftarrow \sum_{i=0}^{n-1} q_i W^i$
end

Proposition 4.1. *Let* $i \in \{0, 1, \ldots, n-1\}$ *then*

(1) $R_i \in [0, Y)$

(2) $R_{i+1}W + x_i \in [0, YW)$

(3) $q_i \in [0, W)$

Proof. (1) It is trivial.

(2)
$$0 \leq R_{i+1}W + x_i \leq (Y-1)W + (W-1) = YW - 1 < YW.$$

(3) By (2),
$$q_i = \left\lfloor \frac{R_{i+1} \cdot W + x_i}{Y} \right\rfloor \leq \frac{R_{i+1}W + x_i}{Y} < W.$$

□

Remark 4.2.
$$\mathsf{DIV}(X, Y) = \begin{cases} (0, X) & : 0 \leq X < Y \\ (1, X - Y) & : Y \leq X < 2Y \end{cases}$$

4.2.1 Binary Long Division

Let $W = 2^1$ then

$$2R_{i+1} + x_i \in [0, 2Y) \implies \mathsf{DIV}^{\mathsf{Core}}(2R_{i+1} + x_i, Y) = \begin{cases} (0, 2R_{i+1} + x_i) & 2R_{i+1} + x_i \in [0, Y)\,, \\ (1, 2R_{i+1} + x_i - Y) & 2R_{i+1} + x_i \in [Y, 2Y)\,. \end{cases}$$

Algorithm 15: Binary Long Divison

Input: $X = \sum_{i=0}^{n-1} x_i 2^i$ with $x_i \in [0, 2) = \{0, 1\}$ and $X \geq Y > 0$
Output: $Q = \sum_{i=0}^{n-1} q_i 2^i$ with $q_i \in [0, 2)$ and R s.t. $X = QY + R$ $(0 \leq R < B)$

Function $\mathsf{DIV}^{\mathsf{binary}-long}(X, Y)$**:**
    if $X < Y$ **then**
        return $(0, X)$;
    end
    $(Q, R) \leftarrow (0, 0)$;
    for $i = n - 1$ **downto** *0* **do**
        $R \leftarrow 2R + x_i$; // $R \leftarrow (R \ll 1) \oplus x_i$
        if $R \geq Y$ **then**
            $(Q, R) \leftarrow (Q + 2^i, R - Y)$; // $Q \leftarrow Q \oplus (1 \ll i)$
        end
    end
    return (Q, R);
end

4.2.2 General Long Division (★)