КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп’ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Лабораторна робота №1

з предмету «Математичні основи захисту інформації»

Варіант №3

Підготували:
Курдельчук Тетяна, ІПС-42
Грищенко Юрій, ІПС-42

Київ – 2022

Ролі виконавців:

Курдельчук Тетяна	Написання звіту
	Теорія
	Оцінка арифметичної складності алгоритму
	Перелік літературних джерел.
Грищенко Юрій	Опис програми для для 3-стрічкової детермінованої машини Тьюрінга (з коментарями і поясненнями)
	Аналіз складності програми для ДМТ

Завдання:

1. Оцінити арифметичну складність цього алгоритму.
2. Написати програму для 3-стрічкової детермінованої машини Тьюрінга (ДМТ).
3. Оцінити складність програми ДМТ.
4. Виконану роботу описати у звіті.

Теорія

Алгоритм mult(x,y) – множення n-бітових чисел a та b.

Алгоритм $mult(a, b, n)$

Вхід: $a, b, n : integer$. (* a, b – цілі n-бітові числа зі знаком *)
Вихід: добуток $a \cdot b$.
Метод:
begin
$s := sign(a) \cdot sign(b)$;
$a := abs(a)$;
$b := abs(b)$; (* тепер $a \geq 0$ і $b \geq 0$ *)
if $n = 1$ then
  if $(a = 1 \wedge b = 1)$ then $rerurn(s)$ else $return(0)$ fi
else
  $c :=$ ліві $n/2$ біти числа a;
  $d :=$ праві $n/2$ біти числа a;
  $u :=$ ліві $n/2$ біти числа b;
  $v :=$ праві $n/2$ біти числа b;
  $m_1 := mult(c, u, n/2)$;
  $m_2 := mult(c - d, v - u, n/2)$;
  $m_3 := mult(d, v, n/2)$;
  $return(s \cdot (m_1 2^n + (m_1 + m_2 + m_3) 2^{n/2} + m_3))$;
fi
end

Алгоритм, що наведено вище, називається множенням Карацуби.

Множення Карацуби - метод швидкого множення, який дозволяє перемножувати два n-значних числа зі складністю обчислення:

$$M(n) = O(n^{\log_2 3}), \quad \log_2 3 \approx 1{,}5849\ldots$$

Цей підхід відкрив новий напрямок в обчислювальній математиці.

Множення двох n-значних цілих чисел звичайним (шкільним) методом «у стовпчик» зводиться, по суті, до додаванняn n-значних чисел. Тому для складності цього «шкільного» або «наївного» методу маємо оцінку зверху:

$$M(n) = O(n^2).$$

У 1956 р. А. М. Колмогоров сформулював гіпотезу, що нижня оцінка для *M(n)* при будь-якому методі множення є також величина порядку ❑❑ n^2 (так звана «гіпотеза n^2» Колмогорова). На правдоподібність гіпотези n^2 вказував той факт, що метод множення «в стовпчик» відомий не менше чотирьох тисячоліть (наприклад, цим методом користувалися шумери), і якби був швидший метод множення, то він, ймовірно, вже був би знайдений. Однак, 1960 року Анатолій Карацуба знайшов новий метод множення двох n-значних чисел з оцінкою складності:

$$M(n) = O(n^{\log_2 3})$$

і тим самим спростував «гіпотезу n^2».

Згодом метод Карацуби був узагальнений до парадигми «розділяй і володарюй», іншими важливими прикладами якої є метод двійкового розбиття, двійковий пошук, метод бісекції тощо.

Перший варіант множення Карацуби:

Цей варіант заснований на формулі

$$(a + bx)^2 = a^2 + ((a + b)^2 - a^2 - b^2)x + b^2x^2.$$

Оскільки $4ab = (a + b)^2 - (a - b)^2$, то множення двох чисел a і b еквівалентне за складністю виконання піднесенню до квадрата.

Нехай X є n-значним числом, тобто

$$X = e_0 + 2e_1 + \ldots + 2^{n-1}e_{n-1},$$

$$\text{де } e_j \in 0,\ 1,\ j = 0,\ 1,\ \ldots,\ n - 1.$$

Будемо вважати для простоти, що $n = 2^m,\ m \geqslant 1;\ n = 2k$. Представляючи X у вигляді

$$X = X_1 + 2^k X_2,$$

де

$$X_1 = e_0 + 2e_1 + \ldots + 2^{k-1}e_{k-1}$$

і

$$X_2 = e_k + 2e_{k+1} + \ldots + 2^{k-1}e_{n-1},$$

знаходимо:

$$X^2 = (X_1 + 2^k X_2)^2 = X_1^2 + ((X_1 + X_2)^2 - (X_1^2 + X_2^2))2^k + X_2^2 2^n. \qquad (1)$$

Числа X_1 і X_2 є k-значними. Число $X_1 + X_2$ може мати $k+1$ знаків. У цьому випадку представимо його у вигляді $2X_3 + X_4$, де X_3 є k-значне число, X_4- однозначне число. Тоді

$$(X_1 + X_2)^2 = 4X_3^2 + 4X_3X_4 + X_4^2.$$

Позначимо $\varphi(n)$ - кількість операцій, достатня для зведення n -значного числа в квадрат за формулою (1). З (1) випливає, що для $\varphi(n)$ справедливо нерівність:

$$\varphi(n) \leqslant 3\varphi(n2^{-1}) + cn, \qquad (2)$$

де $c>0$ є абсолютна константа. Справді, права частина (1) містить суму трьох квадратів k-значних чисел, $k = n2^{-1}$, які для свого обчислення вимагають

$3\varphi(m) = 3\varphi(n2^{-1})$ операцій. Усі інші обчислення в правій частині (1), а саме множення на 4, 2^k, 2^n, п'ять додавань і одне віднімання не більше ніж $2n$-значних чисел вимагають по порядку не більше n операцій. Звідси випливає (2).

Застосовуючи (2) послідовно до

$$\varphi(n2^{-1}),\ \varphi(n2^{-2}),\ \ldots,\ \varphi(n2^{-m+1})$$

і беручи до уваги, що

$$\varphi(n2^{-m}) = \varphi(1) = 1,$$

отримуємо

$$\begin{aligned}\varphi(n) &\leqslant 3(3\varphi(n2^{-2}) + cn2^{-1}) + cn = 3^2\varphi(n2^{-2}) + 3c(n2^{-1}) + cn \leqslant \ldots \leqslant \\ &\leqslant 3^m\varphi(n2^{-m}) + 3^{m-1}c(n2^{-m+1}) + 3^{m-2}c(n2^{-m+2}) + \ldots + 3c(n2^{-1}) + cn = \\ &= 3^m + cn\left(\left(\frac{3}{2}\right)^{m-1} + \left(\frac{3}{2}\right)^{m-2} + \ldots + \left(\frac{3}{2}\right) + 1\right) = \\ &= 3^m + 2cn\left(\left(\frac{3}{2}\right)^m - 1\right) < 3^m(2c+1) = (2c+1)n^{\log_2 3}.\end{aligned}$$

Тим самим для кількості $\varphi(n)$ операцій, достатнього для зведення n-значного числа в квадрат за формулою (1) виконується оцінка:

$$\varphi(n) < (2c+1)n^{\log_2 3}, \quad \log_2 3 = 1{,}5849\ldots$$

Якщо ж n не є ступенем двох, то визначаючи ціле число m нерівностями

$2^{m-1} < n \leqslant 2^m$, представимо X як 2^m-значне число, тобто вважаємо останні $2^m n$ знаків рівними нулю:

$$E_n = \ldots = e_{2^m - 1} = 0.$$

Всі інші міркування залишаються в силі і для $\varphi(n)$ виходить така ж верхня оцінка за порядком величини n.

Другий варіант множення Карацуби:

Це безпосереднє множення двох n -значних чисел, засноване на формулі

$$(a+bx)(c+dx) = ac + ((a+b)(c+d) - ac - bd)x + bdx^2.$$

Нехай, як і раніше $n=2^m$, $n=2k$, A і B - два n -значних числа. Представляючи A і B у вигляді

$$A = A_1 + 2^k A_2,\ quad B = B_1 + 2^k B_2,$$

де $A_1,\ A_2,\ B_1,\ B_2$ - k-значні числа, знаходимо:

$$AB = A_1 B_1 + 2^k((A_1 + A_2)(B_1 + B_2) - (A_1 B_1 + A_2 B_2)) + 2^n A_2 B_2. \quad (3)$$

Таким чином, у цьому випадку формула (1) замінюється формулою (3). Якщо тепер позначити символом $\varphi(n)$ кількість операцій, достатню для множення двох n -значних чисел за формулою (3), то для $\varphi(n)$ виконується нерівність (2), і, отже, справедливою є нерівність:

$$\varphi(n) < (2c+1)n^{\log_2 3}, \quad \log_2 3 = 1{,}5849\ldots$$

Оцінка арифметичної складності алгоритму

Нехай у нас є два поліноми *a(x)* та *b(x)* рівні довжини *n=2k* і ми хочемо їх перемножити. Розділимо їх коефіцієнти на дві рівні частини та представимо як:

$$a(x) = a_1(x) + x^k a_2(x) b(x) = b_1(x) + x^k b_2(x)$$

Тепер рекурсивно обчислимо поліном-добутку p_1 і p_2:

$$p_1(x) = a_1(x) \cdot b_1(x) p_2(x) = a_2(x) \cdot b_2(x)$$

А також поліном *t*:

$$t(x) = (a_1(x) + a_2(x)) \cdot (b_1(x) + b_2(x))$$

Результат множення вихідних поліномів тепер можна порахувати за такою формулою:

$$c(x) = a(x) \cdot b(x) = p_1(x) + x^k \cdot (t(x) - p_1(x) - p_2(x)) + x^{2k} \cdot p_2(x)$$

Результат множення – поліном розміру *2n*.

Якщо порахувати необхідні операції, то з'ясовується, що для перемноження двох багаточленів розміру *n* потрібно порахувати три добутки - p_1, p_2 та *t* – розміру *n/2* та константна кількість додавань, віднімань та зрушень (домножень на x^k), які сумарно можна виконати за *O(n).*

Складність всього алгоритму буде $O(n^{\log_2 3})$: в даному випадку наше завдання розбивається на 3 частини в 2 рази меншого розміру, а об'єднання відбувається за *O(n)*.

Опис програми для для 3-стрічкової детермінованої машини Тьюрінга (з коментарями і поясненнями)

Загальна ідея програми:

Алгоритм явно рекурсивний, на машині Тюринга його можна виконати за допомогою “стеку викликів”.

Матимемо стан X, який символізує “вхід у функцію” mult() з заданим виразом. Виконавши необхідні операції, повернемося у “вихідний стан” W.

З W будемо переходити до інших дій в залежності від контексту (наприклад, виконавши mult() для числа m_1, перейдемо до задання виразу для обчислення m_2). Якщо ж контекст є початковим символом (позначимо його як “>”), то вважаємо, що цей виклик функції mult() був коренем рекурсії, отже в такому випадку результат скопіюємо на вивідну стрічку.

Для простоти припустимо, що кількість бітів завжди дорівнює 2^n.

Алфавіт:
{0,1,z (“маркер zero”), n (“маркер non-zero”), +, -, x, a)

Вхідна стрічка:

кількість бітів, знак A, число A, “x”, знак B, число B

1111+0101x+1100

Копіюємо у “робочу” стрічку:

\> - початок стрічки
I — стан init

```
Start, (>,>,#) -> I, (#,>,>), (r,r,s)
I, (+,#,#) -> I, (#,+,#), (r,r,s)
I, (-,#,#) -> I, (#,-,#), (r,r,s)
I, (0,#,#) -> I, (#,0,#), (r,r,s)
I, (1,#,#) -> I, (#,1,#), (r,r,s)
I, (x,#,#) -> I, (#,x,#), (r,r,s)

I, (#,#,#) -> I2, (#,#,#), (l,l,s)
```

Повертаємося на початок:

```
I2, (+,+,#) -> I2, (#,+,#), (l,l,s)
I2, (-,-,#) -> I2, (#,-,#), (l,l,s)
I2, (0,0,#) -> I2, (#,0,#), (l,l,s)
I2, (1,1,#) -> I2, (#,1,#), (l,l,s)
I2, (x,x,#) -> I2, (#,x,#), (l,l,s)
I2, (>,>,#) -> X, (#,>,>), (s,r,s)
```

Якщо кількість бітів дорівнює 1, то маємо крайній випадок:
1+1x-1 -> 1-1

Інакше, ставимо роздільний символ A, виносимо ліву половину першого множника та праву половину другого множника:
11+01x-10 -> 11+01x-10A1+0x+1

Надалі для простоти запису програми будемо ігнорувати читання з вхідної стрічки та запис на вихідну стрічку.

```
1111+0101x+1100
^
X
```

Переносимо к-сть байтів:

```
X, 1 -> XA, a, r
XA, + -> Xs, +, l — крайній випадок, розглянемо пізніше
XA, - -> Xs, -, l — крайній випадок, розглянемо пізніше
XA, 1 -> XB, a, r
XB, 1 -> XB, 1, r
XB, 0 -> XB, 0, r
XB, + -> XB, +, r
XB, - -> XB, -, r
XB, x -> XB, x, r
XB, # -> XC, A, r
XB, A -> XC, A, r
XC, 1 -> XC, 1, r
XC, # -> XD, 1, l
XD, 0 -> XD, 0, l
XD, 1 -> XD, 1, l
XD, + -> XD, +, l
XD, - -> XD, -, l
XD, x -> XD, x, l
XD, a -> X, a, r
X, + -> XE, +, l
X, - -> XE, -, l
XE, a -> XE, 1, l — “чистимо” допоміжні символи a
XE, > -> Y, >, r
XE, A -> Y, A, r
XE, B -> Y, B, r
XE, C -> Y, C, r
```

```
1111+0101x+1100    ->
aa11+0101x+1100A1  ->
aaaa+0101x+1100A11 ->
1111+0101x+1100A11 ->
^
Y
```

Переносимо ліву половину лівого множника:

```
Y, 0 -> Y, 0, r
```

```
Y, 1 -> Y, 1, r
Y, + -> Y, +, r
Y, - -> Y, -, r
Y, x -> Y, x, r
Y, A -> YA, A, r
YA, a -> YA, a, r
YA, 1 -> YB, a, l — переносимо один біт з лівого множника
YA, + -> YZ, +, l — перенесли всі біти
YB, a -> YB, a, l
YB, A -> YB, A, l
YB, 0 -> YB, 0, l
YB, 1 -> YB, 1, l
YB, + -> YB, +, l
YB, - -> YB, -, l
YB, x -> YC, x, l
YC, 0 -> YC, 0, l
YC, 1 -> YC, 1, l
YC, + -> YD, +, r
YC, - -> YD, -, r
YC, z -> YD, z, r
YC, n -> YD, n, r
YD, 0 -> YE0, z, r — тимчасово замінюємо 0 на z та переносимо
YD, 1 -> YE1, n, r — тимчасово замінюємо 1 на n та переносимо

YE0, 0 -> YE0, 0, r
YE0, 1 -> YE0, 1, r
YE0, + -> YE0, +, r
YE0, - -> YE0, -, r
YE0, x -> YE0, x, r
YE0, A -> YE0A, A, r
YE0A, 0 -> YE0A, 0, r
YE0A, 1 -> YE0A, 1, r
YE0A, + -> YE0B, +, r
YE0A, # -> YE0B, +, r
YE0B, 0 -> YE0B, 0, r
YE0B, 1 -> YE0B, 1, r
YE0B, # -> YF, 0, l — перенесли 0, повертаємося

YE1, 0 -> YE1, 0, r
YE1, 1 -> YE1, 1, r
YE1, + -> YE1, +, r
YE1, - -> YE1, -, r
YE1, x -> YE1, x, r
YE1, A -> YE1A, A, r
YE1A, 0 -> YE1A, 0, r
YE1A, 1 -> YE1A, 1, r
YE1A, + -> YE1B, +, r
YE1A, # -> YE1B, +, r
YE1B, 0 -> YE1B, 0, r
YE1B, 1 -> YE1B, 1, r
YE1B, # -> YF, 1, l — перенесли 1, повертаємося
```

```
YF, 0 -> YF, 0, l
YF, 1 -> YF, 1, l
YF, + -> YF, +, l
YF, - -> YF, -, l
YF, a -> YF, a, l
YF, x -> YF, x, l
YF, z -> Y, z, r
YF, n -> Y, n, r
```

Коли перенесли потрібні біти лівого множника:

```
1111+0101x+1100A11         ->         1111+0101x+1100Aaa+01
                                                       ^
                                                       YZ
```

“Чистимо” допоміжні символи a, потім переносимо ліву половину правого множника:

```
YZ, a -> YZ, 1, l
YZ, A -> z, A, l
```

(Далі програма схожа на стани Y, не будемо розписувати правила для станів z, ZA, ZB...)

...

Коли перенесли потрібні біти правого множника:

```
1111+0101x+1100A11         ->         1111+0101x+1100Aaa+01x+11
                                                       ^
                                                       ZZ
```

Чистимо допоміжні символи a, далі повертаємося в стан X, тобто множимо утсворений вираз:

```
ZZ, a -> ZZ, 1, l
ZZ, A -> X, A, r
```

```
...A11+01x+11
    ^
    X
```

Розглянемо крайній випадок (множення 1-бітних чисел):

```
...1+1x-1                     ->     ...-1
   ^                                   ^
   Xs                                  W
```

або

```
...1-1x+0                     ->     ...+0
   ^                                   ^
   Xs                                  W
```

```
Xs, 1 -> Xs, 1, r
Xs, + -> XsP, +, r
Xs, - -> XsN, -, r
XsP, 1 -> XsP1, 1, r
XsN, 1 -> XsN1, 1, r
XsP, 0 -> Xs0, 0, r — лівий множник 0, отже результат +00
XsN, 0 -> Xs0, 0, r
XsP1, x -> XsP1, x, r
XsP1, + -> XsP1, x, r
XsN1, + -> XsN1, x, r
XsP1, - -> XsN1, x, r
XsN1, - -> XsP1, x, r
XsP1, 1 -> XsP1A, 1, r — результат +01
XsN1, 1 -> XsN1A, 1, r — результат -01
XsP1, 0 -> Xs0, 0, r — результат +00
XsN1, 0 -> Xs0, 0, r — результат +00
```

Ідемо до кінця стрічки, потім з кінця все стираємо, і записуємо результат:

```
Xs0, 0 -> Xs0, 0, r
Xs0, 1 -> Xs0, 1, r
Xs0, + -> Xs0, +, r
Xs0, - -> Xs0, -, r
Xs0, x -> Xs0, x, r
Xs0, # -> Xs0B, #, l
Xs0B, 0 -> Xs0B, #, l
Xs0B, 1 -> Xs0B, #, l
Xs0B, + -> Xs0B, #, l
Xs0B, - -> Xs0B, #, l
Xs0B, x -> Xs0B, #, l
Xs0B, > -> Xs0C, >, r
Xs0B, A -> Xs0C, A, r
Xs0B, B -> Xs0C, B, r
Xs0B, C -> Xs0C, C, r
Xs0C, # -> Xs0D, +, r
Xs0D, # -> Xs0D, 0, r
Xs0D, # -> Xs0E, 0, l
Xs0E, 0 -> Xs0F, +, l
XD0F, + -> W, +, l
```

Маємо аналогічні правила для XsP1A (записуємо +01) та XsN1A (записуємо -01)

...

Стан W назвемо “прикінцевим випадком”, тобто при ньому завершена одна ітерація множення. Подальші дії визначаються наявністю початкового символа > або символа з підмножини separators.

Якщо голівка робочої стрічки у стані W знаходиться на початковому символі, то рекурсивний алгоритм завершено: виводимо результат
W, (>,>,#) -> W, (>,>,>), (r,r,r)

```
W, (>,+,#) -> W, (>,#,+), (r,r,r)
W, (>,-,#) -> W, (>,#,-), (r,r,r)
W, (>,0,#) -> W, (>,#,0), (r,r,r)
W, (>,1,#) -> W, (>,#,1), (r,r,r)
W, (>,#,#) -> End, (>,#,#)
```

Якщо голівка робочої стрічки у стані W знаходиться на символі A, то переходмо до наступного кроку “стандартного випадку” алгоритму множення: переносимо кількість байтів поділених на 2, d-c та u-v.

W, A -> V, A, r

(Правила переносу к-сті байтів схожі на правила для станів XA, XB, XC, ..., не будемо розписувати їх, вважаємо, що використовуються стани VA, VB, VC... і потім переходимо в U)

Вважаємо, що поставили роздільний символ B, та перенесли к-сть байтів, поділену на 2:

1111+0101x+1100A+0011B11

Переносимо біти з лівої половини першого множника зі знаком -, у вигляді доповняльного коду (two’s complement)

-01 (001) -> інвертуємо -> 110 (-10)

(насправді для two’s complement необхідно після інвертування додати одиницю, але ми це зробимо в наступному етапі)

```
1111+0101x+1100A+0011B11  ->   1111+ZN01x+1100A+0011B11-10
^                                                      ^
U                                                      T1
```

Перенесення одного біта виглядає приблизно так:

```
1111+0101x+1100A+0011B11     ->
1111+0101x+1100A+0011Ba1     ->
1111+Z101x+1100A+0011Ba1     ->
1111+Z101x+1100A+0011Ba1-1   -> ...
```

```
U, 0 -> U, 0, r
U, 1 -> U, 1, r
U, + -> U, +, r
U, - -> U, -, r
U, x -> U, x, r
U, A -> U, A, r
U, B -> UA, B, r
UA, a -> UA, a, r
UA, - -> Us, -, l — перенесли потрібні біти
UA, 1 -> UB, a, l — повертаємося, аби перенести один біт
UB, a -> UB, a, l
UB, 0 -> UB, 0, l
UB, 1 -> UB, 1, l
```

```
UB, + -> UB, +, l
UB, - -> UB, -, l
UB, A -> UB, A, l
UB, B -> UB, B, l
UB, x -> UC, x, l
UC, 0 -> UC, 0, l
UC, 1 -> UC, 1, l
UC, + -> UD, +, r
UC, - -> UD, -, r
UC, z -> UD, z, r
UC, n -> UD, n, r


UD, 0 -> UE0, z, r — переносимо 0, потім його інвертуємо
UE0, 0 -> UE0, 0, r
UE0, 1 -> UE0, 1, r
UE0, + -> UE0, +, r
UE0, - -> UE0, -, r
UE0, x -> UE0, x, r
UE0, A -> UE0, A, r
UE0, B -> UF0, B, r
UF0, a -> UF0, a, r
UF0, 1 -> UF0, 1, r
UF0, # -> UG0, -, r
UF0, - -> UG0, -, r
UG0, 0 -> UG0, 0, r
UG0, 1 -> UG0, 1, r
UG0, # -> UF, 1, r - перенесли 0 (інвертований: 1), повертаємося до
B
UF, 0 -> UF, 0, l
UF, 1 -> UF, 1, l
UF, - -> UF, -, l
UF, a -> UF, a, l
UF, B -> UA, B, r — переносимо наступний біт, якщо це необхідно

UD, 1 -> UE1, n, r — переносимо 1, потім його інвертуємо
UE1, 0 -> UE1, 0, r
UE1, 1 -> UE1, 1, r
UE1, + -> UE1, +, r
UE1, - -> UE1, -, r
UE1, x -> UE1, x, r
UE1, A -> UE1, A, r
UE1, B -> UF1, B, r
UF1, a -> UF1, a, r
UF1, 1 -> UF1, 1, r
UF1, # -> UG1, -, r
UF1, - -> UG1, -, r
UG1, 0 -> UG1, 0, r
UG1, 1 -> UG1, 1, r
UG1, # -> UF, 0, r - перенесли 1 (інвертований: 0), повертаємося до
B
```

Us, a -> Us, 1, l — чистимо допоміжні символи a
Us, B -> T1, B, r

Переносимо біти з правої половини першого множника, одночасно додаємо їх до наявного -c на правому кінці стрічки, також додаємо 1 для утворення two’s complement

-10 (110) + 1 = 111 (-11); -11 + 01 = +00

```
1111+ZN01x+1100A+0011B11-10  -> 1111+0101x+1100A+0011B11+00
                      ^              ^
                      T              S
```

Перенесення одного біта виглядає приблизно так:
1111+ZN01x+1100A+0011B11-10 ->
1111+ZN01x+1100A+0011Ba1-10 ->
1111+ZN0Nx+1100A+0011B11-10 ->
1111+ZN0Nx+1100A+0011B11-1Z -> ...

T0, a -> T0, a, r
T0, - -> Ts, -, r — перенесли та додали всі біти, перенесемо їх вліво до B
T0, + -> Ts, +, r
T0, 1 -> T0A, a, l — йдемо вліво, шукаємо біт для перенесення
T0A, 0 -> T0A, 0, l
T0A, 1 -> T0A, 1, l
T0A, + -> T0A, +, l
T0A, - -> T0A, -, l
T0A, A -> T0A, A, l
T0A, B -> T0A, B, l
T0A, x -> T0B, x, l
T0B, n -> T0B, n, l
T0B, z -> T0B, z, l
T0B, 0 -> TB0, n, r — додаємо 0

TB0, 0 -> TB0, 0, r
TB0, 1 -> TB0, 1, r
TB0, n -> TB0, n, r
TB0, z -> TB0, z, r
TB0, + -> TB0, +, r
TB0, - -> TB0, -, r
TB0, x -> TB0, x, r
TB0, A -> TB0, A, r
TB0, B -> TC0, B, r
TC0, 0 -> TC0, 0, r
TC0, 1 -> TC0, 1, r

TC0, + -> TC0, +, r
TC0, - -> TC0, -, r
TC0, n -> TD0, n, l — дійшли до потрібного розряду
TC0, z -> TD0, z, l
TC0, # -> TD0, #, l

TD0, 0 -> TE0, z, l —   0 + 0 = 0, закодуємо як z, повертаємося
TE0, 0 -> TE0, 0, l
TE0, 1 -> TE0, 1, l
TE0, - -> TE0, -, l
TE0, + -> TE0, +, l
TE0, B -> T0, B, r

TD0, 1 -> TE0, n, l —   1 + 0 = 1, закодуємо як n, повертаємося

T0B, 1 -> TB1, n, r — додаємо одиницю

TB1, 0 -> TB1, 0, r
TB1, 1 -> TB1, 1, r
TB1, n -> TB1, n, r
TB1, z -> TB1, z, r
TB1, + -> TB1, +, r
TB1, - -> TB1, -, r
TB1, x -> TB1, x, r
TB1, A -> TB1, A, r
TB1, B -> TC1, B, r
TC1, 0 -> TC1, 0, r
TC1, 1 -> TC1, 1, r
TC1, + -> TC1, +, r
TC1, - -> TC1, -, r
TC1, n -> TD1, n, l — дійшли до потрібного розряду
TC1, z -> TD1, z, l
TC1, # -> TD1, #, l

TD1, 0 -> TE0, n, l —   0 + 1 = 1, закодуємо як n, повертаємося

TD1, 1 -> TE1, z, l —   1 + 1 = 10, закодуємо 0 як z та “запам’ятаємо” перенесення одиниці на наступний розряд
TE1, 0 -> TE1, 0, l
TE1, 1 -> TE1, 1, l
TE1, - -> TE1, +, l
TE1, + -> TE1, -, l
TE1, B -> T1, B, r

T1, a -> T1, a, r — аналогічно до T0, тільки з перенесенням 1 на наст. розряд
T1, - -> Ts, +, r
T1, + -> Ts, +, r

```
T1, 1 -> T1A, a, l — йдемо вліво, шукаємо біт для перенесення
T1A, 0 -> T1A, 0, l
T1A, 1 -> T1A, 1, l
T1A, + -> T1A, +, l
T1A, - -> T1A, -, l
T1A, A -> T1A, A, l
T1A, B -> T1A, B, l
T1A, x -> T1B, x, l
T1B, n -> T1B, n, l
T1B, z -> T1B, z, l
T1B, 0 -> TB1, n, r — додаємо одиницю, як і раніше
T1B, 1 -> TB2, n, r — додаємо 2

TB2, 0 -> TB2, 0, r
TB2, 1 -> TB2, 1, r
TB2, n -> TB2, n, r
TB2, z -> TB2, z, r
TB2, + -> TB2, +, r
TB2, - -> TB2, -, r
TB2, x -> TB2, x, r
TB2, A -> TB2, A, r
TB2, B -> TC2, B, r
TC2, 0 -> TC2, 0, r
TC2, 1 -> TC2, 1, r
TC2, + -> TC2, +, r
TC2, - -> TC2, -, r
TC2, n -> TD2, n, l — дійшли до потрібного розряду
TC2, z -> TD2, z, l
TC2, # -> TD2, #, l

TD2, 0 -> TE1, z, l —   10 + 0 = 10, закодуємо як z, перенесемо 1
на наступний розряд
TD2, 1 -> TE1, n, l —   10 + 1 = 11, закодуємо як n, перенесемо 1
на наступний розряд

Ts, z -> Ts, 0, r — перенесли та додали всі біти, “чистимо” n та z
Ts, n -> Ts, 1, r
Ts, # -> TT, #, l — повертаємося наліво та “чистимо” там
TT, 0 -> TT, 0, l
TT, 1 -> TT, 1, l
TT, + -> TT, -, l
TT, x -> TT, x, l
TT, A -> TT, A, l
TT, B -> TT, B, l
TT, z -> TU, 0, l
TT, n -> TU, 1, l
TU, z -> TU, 0, l
TU, n -> TU, 1, l
TU, - -> S, -, r
TU, + -> S, +, r
```

Наступний множник для m_2 обчислюється та записується схожим чином, тому не будемо розписувати правила для станів S, SA, SB, ..., R, RA, RB, ...

```
1111+0101x+1100A+0011B11+00                                      ->
1111+0101x+NN00A+0011B11+00x+11
       ^                                                     ^
       s                                                     R
-00 (000) -> інвертуємо -> 111 (-11); -11 + 1 = +00; +00 + 11 = +11
1111+0101x+NN00A+0011B11+00x+11                                  ->
1111+0101x+1100A+0011B11+00x+11
                        ^                      ^
                        R                      Q
```

Виконаємо множення для виразу, записаного після B:
Q, 0 -> Q, 0, r
Q, 1 -> Q, 1, r
Q, + -> Q, +, r
Q, - -> Q, -, r
Q, A -> Q, A, r
Q, B -> X, B, r — стан X рекурсивно множить

Після виконання множення переходимо у стан W, знаходячись у цей раз над символом B:

```
1111+0101x+1100A+0011B11+00x+11   ->   1111+0101x+1100A+0011B+0000
                      ^                                     ^
                      X                                     W
```

W, B -> QA, B, l

Далі обчислюємо та записуємо значення m_3:

Перенесли кількість байтів, поділену на 2, аналогічно до минулих випадків, не будемо розписувати правила для QA, QB, ...:

```
1111+0101x+1100A+0011B+0000   ->   1111+0101x+1100A+0011B+0000C11
                    ^              ^
                    QA             P
```

Перенесемо праву половину першого множника, оскільки зчитуємо справа, потрібно “зарезервувати місце”:

```
1111+0101x+1100A+0011B+0000C11    ->
1111+0101x+1100A+0011B+0000Ca1+0  ->
1111+0101x+1100A+0011B+0000Caa+00 ->
1111+010Nx+1100A+0011B+0000Caa+0N ->
1111+01ZNx+1100A+0011B+0000Caa+ZN ->
1111+0101x+1100A+0011B+0000C11+01
```

P, 0 -> P, 0, r
P, 1 -> P, 1, r

P, + -> P, +, r
P, - -> P, -, r
P, x -> P, x, r
P, A -> P, A, r
P, B -> P, B, r
P, C -> PA, C, r — починаємо резервувати місце після C
PA, a -> PA, a, r
PA, + -> Ps, +, l — зарезервували всі біти, починаємо переносити (розгл. Цей випадок пізніше)
PA, 1 -> PB, a, r — йдемо направо, щоб там додати 0
PB, 1 -> PB, 1, r
PB, # -> PC, +, r
PB, + -> PC, +, r
PC, 0 -> PC, 0, r
PC, # -> PD, 0, l — записали 0, повертаємося назад
PD, 0 -> PD, 0, l
PD, 1 -> PD, 1, l
PD, + -> PD, +, l
PD, a -> PA, a, r

Зарезервували місце:

```
1111+0101x+1100A+0011B+0000C11 -> 1111+0101x+1100A+0011B+0000Caa+00
^                                                               ^
P                                                               Ps
```

Тепер переносимо біти:
Ps, a -> Ps, a, l
Ps, 0 -> Ps, 0, l
Ps, 1 -> Ps, 1, l
Ps, - -> Ps, -, l
Ps, + -> Ps, +, l
Ps, A -> Ps, A, l
Ps, B -> Ps, B, l
Ps, C -> Ps, C, l
Ps, x -> PT, x, l — дійшли до лівого множника
PT, z -> PT, z, l
PT, n -> PT, n, l
PT, 0 -> PT0, 0, r — переносимо 0
PT, 1 -> PT1, 1, r — переносимо 1

PT0, 0 -> PT0, 0, r
PT0, 1 -> PT0, 1, r
PT0, + -> PT0, +, r
PT0, - -> PT0, -, r
PT0, x -> PT0, x, r
PT0, A -> PT0, A, r
PT0, B -> PT0, B, r
PT0, C -> PT0A, C, r
PT0A, a -> PT0A, a, r
PT0A, + -> PT0A, +, r
PT0A, 0 -> PT0A, 0, r

```
PT0A, # -> PT0B, #, l — дійшли до потрібного розряду
PT0A, z -> PT0B, z, l — дійшли до потрібного розряду
PT0A, n -> PT0B, n, l — дійшли до потрібного розряду
PT0B, 0 -> PU, z, l — записали 0 (закодовали як z)

PT1, 0 -> PT1, 0, r
PT1, 1 -> PT1, 1, r
PT1, + -> PT1, +, r
PT1, - -> PT1, -, r
PT1, x -> PT1, x, r
PT1, A -> PT1, A, r
PT1, B -> PT1, B, r
PT1, C -> PT1A, C, r
PT1A, a -> PT1A, a, r
PT1A, + -> PT1A, +, r
PT1A, 0 -> PT1A, 0, r
PT1A, # -> PT1B, #, l — дійшли до потрібного розряду
PT1A, z -> PT1B, z, l — дійшли до потрібного розряду
PT1A, n -> PT1B, n, l — дійшли до потрібного розряду
PT1B, 0 -> PU, n, l — записали 1 (закодовали як n)

PU, 0 -> Ps, 0, l — далі копіюємо біти
PU, + -> PX, +, r — заповнили всі біти, робитимемо “чистку” n, z, a
PX, z -> PX, 0, r
PX, n -> PX, 1, r
PX, # -> PY, #, l
PY, 0 -> PY, 0, l
PY, 1 -> PY, 1, l
PY, a -> PY, 1, l — прибираємо a
PY, + -> PY, +, l
PY, - -> PY, -, l
PY, x -> PY, x, l
PY, A -> PY, A, l
PY, B -> PY, B, l
PY, C -> PY, C, l
PY, z -> PZ, 0, l
PY, n -> PZ, 1, l
PZ, z -> PZ, 0, l
PZ, n -> PZ, 1, l
PZ, 0 -> O, 0, r — прибрали всі n, z
PZ, 1 -> O, 1, r
```

Перенесли потрібні біти та зробили очистку:

```
1111+0101x+1100A+0011B+0000Caa+00
                              ^   ->
                              Ps
1111+0101x+1100A+0011B+0000C11+01
       ^
       O
```

(аналогічно переносимо молодші біти правого множника, не будемо розписувати правила для станів O, OA, OB, ...)

```
1111+0101x+1100A+0011B+0000C11+01
       ^                           ->
       O
1111+0101x+1100A+0011B+0000C11+01x+00
             ^
             N
```

Виконаємо множення для утвореного виразу після C:
N, 0 -> N, 0, r
N, 1 -> N, 1, r
N, + -> N, +, r
N, - -> N, -, r
N, x -> N, x, r
N, A -> N, A, r
N, B -> N, B, r
N, C -> N, C, r — стан X рекурсивно множить
W, C -> NA, C, l

```
1111+0101x+1100A+0011B+0000C11+01x+00
             ^                         ->
             N
1111+0101x+1100A+0011B+0000C+0000
                          ^
                          NA
```

Тепер скопіюємо число m_2 (що знаходиться після B), додавши зліва n/2 нулів, та за потреби інвертувавши (це число може бути від'ємним, тому знову використаємо two's complement):

```
1111+0101x+1100A+0011B+0000C+0000
                          ^       ->
                          NA
aaaa+0101x+1100A+0011B+ZZZZC+0000D+000000
                          ^
                          M
```

NA, 0 -> NA, 0, l
NA, 1 -> NA, 1, l
NA, + -> NA, +, l
NA, - -> NA, -, l
NA, A -> NA, A, l
NA, B -> NA, B, l
NA, C -> NA, C, l
NA, D -> NA, D, l
NA, x -> NB, x, l
NB, 0 -> NB, 0, l
NB, 1 -> NB, 1, l
NB, + -> NC, +, l
NB, - -> NC, -, l — дійшли до кількості бітів

```
NC, a -> NC, a, l
NC, > -> Ns, >, r — дійшли до початку виразу (розгланемо потім)
NC, A -> Ns, A, r
NC, B -> Ns, B, r
NC, C -> Ns, C, r

NC, 1 -> ND, a, l — не дійшли до початку виразу, помічаємо два біти
ND, 1 -> NE, a, r — помітили два біти, один нуль запишемо справа
NE, a -> NE, a, r
NE, 0 -> NE, 0, r
NE, 1 -> NE, 1, r
NE, + -> NE, +, r
NE, - -> NE, -, r
NE, x -> NE, x, r
NE, A -> NE, A, r
NE, B -> NF, B, r
NF, + -> NGP, +, r — число додатнє, записуватимемо +000...
NF, - -> NGN, -, r — число від’ємне, записуватимемо -000...

NGP, 0 -> NGP, 0, r
NGP, 1 -> NGP, 1, r
NGP, + -> NGP, +, r
NGP, - -> NGP, -, r
NGP, C -> NGP, C, r
NGP, # -> NHP, D, r
NGP, D -> NHP, D, r
NHP, # -> NIP, +, r — записали +, якщо це необхідно
NHP, + -> NIP, +, r
NIP, # -> NA, 0, l — записали 0, повертаємося наліво

NGN, 0 -> NGN, 0, r
NGN, 1 -> NGN, 1, r
NGN, + -> NGN, +, r
NGN, - -> NGN, -, r
NGN, C -> NGN, C, r
NGN, # -> NHN, D, r
NGN, D -> NHN, D, r
NHN, # -> NIN, -, r — записали -, якщо це необхідно
NHN, - -> NIN, -, r
NIN, # -> NA, 1, l — записали 1, повертаємося наліво

Ns, 0 -> Ns, 0, r — йдемо направо до B
Ns, 1 -> Ns, 1, r
Ns, + -> Ns, +, r
Ns, - -> Ns, -, r
Ns, x -> Ns, x, r
Ns, A -> Ns, A, r
Ns, B -> NT, B, r
NT, + -> NUP, +, r — додатнє число, просто копіюємо біти
NT, - -> NUN, -, r — від’ємне число, інвертуємо копійовані біти
```

```
NUP, z -> NUP, z, r
NUP, n -> NUP, n, r
NUP, C -> M, C, l — скопіювали всі біти після B
NUP, 0 -> NV0, z, r — запишемо 0
NUP, 1 -> NV1, n, r — запишемо 1

NUN, z -> NUN, z, r
NUN, n -> NUN, n, r
NUN, C -> M, C, l — скопіювали всі біти після B
NUN, 0 -> NV1, z, r — запишемо 1
NUN, 1 -> NV0, n, r — запишемо 0

NV0, 0 -> NV0, 0, r
NV0, 1 -> NV0, 1, r
NV0, + -> NV0, +, r
NV0, - -> NV0, -, r
NV0, C -> NV0, C, r
NV0, D -> NV0, D, r
NV0, # -> NW, 0, l — записали 0, повертаємося до B

NV1, 0 -> NV1, 0, r
NV1, 1 -> NV1, 1, r
NV1, + -> NV1, +, r
NV1, - -> NV1, -, r
NV1, C -> NV1, C, r
NV1, D -> NV1, D, r
NV1, # -> NW, 1, l — записали 0, повертаємося до B

NW, 0 -> NW, 0, l
NW, 1 -> NW, 1, l
NW, + -> NW, +, l
NW, - -> NW, -, l
NW, z -> NT, z, r
NW, n -> NT, n, r
NW, D -> NW, D, l
NW, C -> NW, C, l
NW, B -> NT, B, r — копіюємо наступний біт

M, z -> M, 0, l
M, n -> M, 1, l — робимо “чистку” помічених бітів після B
```

Потім виконуватимемо додавання m_b (A) до числа після D. *Не будемо розписувати правила для M, M0, M1, ..., бо вони схожі на правила для T, T0, T1, ... (проте зауважимо, що число після D необов’язково від’ємне, тому “початковий” стан M0 та M1 визначається знаком + або — біля B)*

```
aaaa+0101x+1100A+0011B+0000C+0000D+000000
                      ^                   ->
                      M
```

```
aaaa+0101x+1100A+0011B+0000C+0000D+000011
               ^
               L
```

Аналогічно для додавання m_3 (числа після C)

```
aaaa+0101x+1100A+0011B+0000C+0000D+000011
               ^                          ->
               L

aaaa+0101x+1100A+0011B+0000C+0000D+000011
                           ^              ->
                           K
```

Тепер допишемо n/2 молодших бітів в кінець виразу:

```
aaaa+0101x+1100A+0011B+0000C+0000D+000011
                           ^              ->
                           K
1111+0101x+1100A+0011B+0000C+0000D+00001100
^
J
```

```
K, 0 -> K, 0, l
K, 1 -> K, 1, l
K, + -> K, +, l
K, - -> K, -, l
K, A -> K, A, l
K, B -> K, B, l
K, C -> K, C, l
K, D -> K, D, l
K, x -> KA, x, l
KA, 0 -> KA, 0, l
KA, 1 -> KA, 1, l
KA, + -> KB, +, l — дійшли до кількості бітів
KA, - -> KB, -, l

KB, 1 -> KB, 1, l
KB, a -> KC, 1, l
KC, a -> KD, 1, r — зняли помітки з двох бітів, допишемо один 0 чи
1 в кінець
KC, > -> J, >, r — зняли всі помітки
KC, A -> J, A, r
KC, B -> J, B, r
KC, C -> J, C, r

KD, a -> KD, a, r
KD, 0 -> KD, 0, r
KD, 1 -> KD, 1, r
KD, + -> KD, +, r
```

```
KD, - -> KD, -, r
KD, A -> KD, A, r
KD, B -> KD, B, r
KD, C -> KD, C, r
KD, D -> KE, D, r — дійшли до числа після D
KE, + -> KFP, +, r — число додатнє, ставитимемо 0
KE, - -> KFN, -, r — число від’ємне, ставитимемо 1

KFP, 0 -> KFP, 0, r
KFP, 1 -> KFP, 1, r
KFP, # -> K, 0, l

KFN, 0 -> KFN, 0, r
KFN, 1 -> KFN, 1, r
KFN, # -> K, 1, l
```

Повторимо додавання числа m_3, на цей раз змістившись на n/2 розрядів. Знову правила для J, J0, J1, ... аналогічні на попередні для додавання, за вийнятком того, що ми не виконуватимемо “чистку” після знаходження суми:

```
1111+0101x+1100A+0011B+0000C+0000D+00001100
^                                           ->
J
1111+0101x+1100A+0011B+0000C+ZZZZD+0000NNZZ
                            ^               ->
                            I
```

Аналогічно для додавання числа m_1 до старших розрядів числа після D:

```
1111+0101x+1100A+0011B+0000C+ZZZZD+0000NNZZ
                            ^               ->
                            I
1111+0101x+1100A+ZZNNB+0000C+ZZZZD+0011NNZZ
              ^
              H
```

В кінці маємо число +0011NNZZ, якому відповідає 00111100. Дійсно: 0101 (5) x 1100 (12) = 111100 (60)
Необхідно визначити знак добутку, прибрати помітки Z/N, та врешті-решт вивести результат.

```
1111+0101x+1100A+ZZNNB+0000C+ZZZZD+0011NNZZ
              ^                             ->
              H

1111+0101x+1100A+ZZNNB+0000C+ZZZZD+00111100
                                           ^
                                           HE
```

```
H, 0 -> H, 0, l
H, 1 -> H, 1, l
H, + -> HP, +, l — знак “+”
H, - -> HN, -, l — знак “-”

HP, x -> HAP, x, l
HAP, 0 -> HAP, 0, l
HAP, 1 -> HAP, 1, l
HAP, + -> HBP, +, r — результат додатній
HAP, - -> HBN, -, r — результат від’ємний

HN, x -> HAN, x, l
HAN, 0 -> HAN, 0, l
HAN, 1 -> HAN, 1, l
HAN, + -> HBN, +, r — результат від’ємний
HAN, - -> HBP, -, r — результат додатній

HBP, 0 -> HBP, 0, r
HBP, 1 -> HBP, 1, r
HBP, + -> HBP, +, r
HBP, - -> HBP, -, r
HBP, x -> HBP, x, r
HBP, z -> HBP, z, r
HBP, n -> HBP, n, r
HBP, A -> HBP, A, r
HBP, B -> HBP, B, r
HBP, C -> HBP, C, r
HBP, D -> HCP, D, r
HCP, + -> HD, +, r — знак “+” залишається

HBN, 0 -> HBN, 0, r
HBN, 1 -> HBN, 1, r
HBN, + -> HBN, +, r
HBN, - -> HBN, -, r
HBN, x -> HBN, x, r
HBN, z -> HBN, z, r
HBN, n -> HBN, n, r
HBN, A -> HBN, A, r
HBN, B -> HBN, B, r
HBN, C -> HBN, C, r
HBN, D -> HCN, D, r
HCN, + -> HD, -, r — записали знак “-” в результат

HD, 0 -> HD, 0, r
HD, 1 -> HD, 1, r
HD, z -> HD, 0, r
HD, n -> HD, 1, r
HD, # -> HE, #, l — прибрали помітки z/n
```

Перенесемо число після D на початок виразу, для цього заповнимо все між початком та D допоміжними символами a.

```
>1111+0101x+1100A+ZZNNB+0000C+ZZZZD+00111100
                                            ^  ->
                                            HE
>+00111100##################################
^
W
```

```
HE, 0 -> HE, 0, l — повертаємося до D
HE, 1 -> HE, 1, l
HE, + -> HF, +, l
HE, - -> HF, -, l
HF, D -> HF, a, l — заповнимо всі символи до D маркерами a
HF, 0 -> HF, a, l
HF, 1 -> HF, a, l
HF, + -> HF, a, l
HF, - -> HF, a, l
HF, A -> HF, a, l
HF, B -> HF, a, l
HF, C -> HF, a, l
HF, z -> HF, a, l
HF, n -> HF, a, l
HF, x -> HG, a, l
HG, 0 -> HG, a, l
HG, 1 -> HG, a, l
HG, + -> HG, a, l
HG, - -> HG, a, l
HG, > -> HH, >, l — дійшли до початку виразу, тепер переносимо результат
HG, A -> HH, A, l
HG, B -> HH, B, l
HG, C -> HH, C, l

HH, a -> HH, a, r
HH, # -> HI, #, l — перенесли всі символи, прибираємо допоміжні a
HI, a -> HI, #, l
HI, 0 -> HI, 0, l
HI, 1 -> HI, 1, l
HI, + -> W, +, l
HI, - -> W, -, l — завершили ітерацію
```

(вивід результату на вивідну стрічку при стані W було показано раніше)

Аналіз складності програми для ДМТ

Просторова складність.

Кількість використаних комірок залежить лише від кількості бітів n. Розглянемо один етап рекурсії:

```
1111+0101x+1100A+0011B+0000C+0000D+00111100
n    n     n     n     n     n     2n
```

Умовно розділимо стани програми на етапи: (етап X: стани XA, XB, XC, ...; етап Y: стани YA, YB, YC...; тощо), тоді побачимо, що кожен етап може:

- не збільшувати кількість використаних комірок
- використати додаткові n/2+c клітинок (копіюючи в кінець стрічки кількість байтів, поділену на 2, або ж копіюючи половину множника, що займає n/2 клітинок)
  - єдиний виняток: “етап N”, який додає 3n/2+c клітинок.
  - c — константа, не більше 2 (зазвичай літера та знак + чи -)
- рекурсивно запустити алгоритм mult(), проте зменшивши кількість бітів вдвічі. Кінцевий результат алгоритму буде мати розмір O(n), проте може займати більше клітинок під час виконання.

Якщо не враховувати рекурсивні спуски, то очевидно, що один прохід по всім “етапам” використовує O(n) клітинок, оскільки етапи впорядковані, завжди виконуються по порядку, їх кількість скінченна та задана алгоритмом.

Всього є три етапи, які призводять до рекурсії: при цьому кожен раз розмір задачі зменшується вдвічі. Як зазначалося в попередньому розділі, з цього випливає **просторова складність програми $O(n^{\log_2 3})$.**

Часова складність.

Знову скористаємося поняттям “етапів” програму. На кожному етапі відбувається копіювання або переміщення n або n/2 символів. Ми знаємо, що просторова складність в межах одного рівня рекурсії становить O(n), отже переміщення n символів в обмежену просторі займає час $O(n^2)$. Етапи виконуються впорядковано та їх кількість скінченна та задана алгоритмом, отже, не враховуючи рекурсії, час виконання всіх етапів оцінюється як $O(n^2)$.

Маємо таке рекурентне співвідношення:

$T(n) = 3T(n/2) + n^2$

Спробуємо застосувати майстер-метод (основу теорему про рекурентні співвідношення):

$a=3$, $b=2$, $f(n) = O(n^c)$, $c=2$

$\log_b a = \log_2 3 < 2$

Перевіряємо другу умову: $3f(n/2) <= kf(n)$ для деякого $k < 1$

Підставимо f: $3/4n^2 <= kn^2$, ця умова виконується при $k=3/4$

Отже, отримуємо: $T(n) = \Theta(n^2)$

Перелік літературних джерел:

1. Карацуба Є. А. Швидкі алгоритми і метод БВЕ, 2008.
2. Алексєєв В. Б. Від методу Карацуба для швидкого множення чисел до швидких алгоритмах для дискретних функцій // Тр. МІАН. — 1997. — С. 20-27.
3. Карацуба А. А. Складність обчислень // Тр. МІАН. — 1995. — С. 186-202.
4. https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)