程式作業二

Group 6

107502501林廷真 107502502林欣蓓

107502503陸品潔 107502504歐亭昀

107502509王君安 107502511廖冠閔

107502540陳皇宇

1. **traditional multiplication**

**資料結構**：因為int 有大小上限，因此把input 用string 來存。

**演算法**：按照傳統乘法

**時間複雜度**：O()

**Pseudo code**：

string multiplication(string num1, string num2)

{

if num1.size() < num2.size()

swap(num1,num2)

int maxlen = max(num1.size(), num2.size());

paddingzero(&num1, maxlen - num1.size(), 1);

paddingzero(&num2, maxlen - num2.size(), 1);

for (int i = maxlen - 1; i >= 0; i--)

{

plus\_num = "";

c = 0;

for (int j = maxlen - 1; j >= 0; j--)

{

int mult = num1[i]\* num2[j];

mult = mult + c;

c = mult / 10;

mult = mult % 10;

plus\_num = to\_string(mult) + plus\_num;

}

paddingzero(&plus\_num, (maxlen - i - 1), 0);

final\_num = plusnum(final\_num, plus\_num);

}

return final\_num;

}

1. **Karatsuba algorithm**

**資料結構**：因為int 有大小上限，因此把input 用string 來存。

**演算法**：

設 A = a\*10+b , B = c\*10+d

r = (a+b)\*(c+d)-a\*c-b\*d

A\*B = (a\*c)\* + (r)\* + (b\*d) \*

**黃底**為乘法，共3次

可看出原本要 b\*d , b\*c , a\*d , a\*c 4次乘法降為3次

**時間複雜度**：

T(n) = 3T(n/2)+O(n) 根據 master theorem

a = 3 , b = 2 , f(n) = n

因為f(n) = O()

所以T(n) = Θ(

**Pseudo code**：

string karatsuba(string num1, string num2)

{

if num1.size() < num2.size()

swap(num1,num2)

if (min(num1.size(), num2.size()) <= 1)

return to\_string(stoi(num1) \* stoi(num2));

maxlen = max(num1.size(), num2.size());

paddingzero(&num1, maxlen - num1.size(), 1);

paddingzero(&num2, maxlen - num2.size(), 1);

maxlen\_2 = maxlen / 2;

x = num1.substr(0, maxlen - maxlen\_2);

y = num1.substr(maxlen - maxlen\_2, maxlen);

w = num2.substr(0, maxlen - maxlen\_2);

z = num2.substr(maxlen - maxlen\_2, maxlen);

xw = karatsuba(x, w);

yz = karatsuba(y, z);

plus\_xw\_yz = subnum(subnum(karatsuba(plusnum(x, y), plusnum(w, z)), xw), yz);

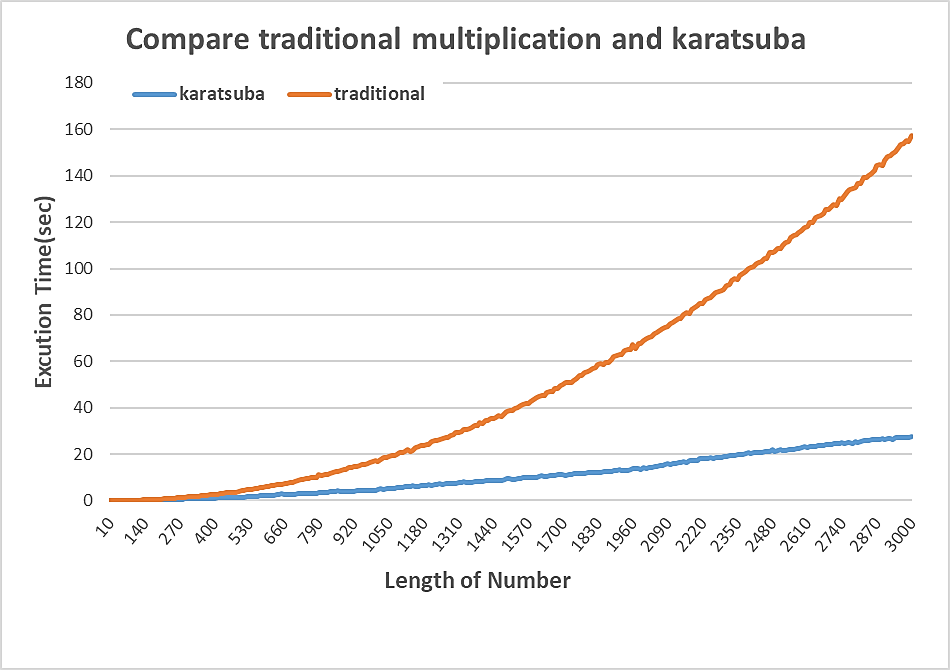
paddingzero(&xw, maxlen\_2 \* 2, 0);

paddingzero(&plus\_xw\_yz, maxlen\_2, 0);

string final\_num = plusnum(plusnum(xw, yz), plus\_xw\_yz);

return final\_num;

}



**n=95**

When n>95 karatsuba algorithm will faster then traditional method.

1. dynamic karatsuba

我們試著把只能分兩段的karatsuba改成能夠自由分段的karatsuba

但是發現我們的算法分越多個，就會執行越多次的乘法

演算法：(以divide 3為例)

A = a\*+b\*+c

B=d\*+e\*+f

A\*B = a\*d\* +

((a+b)\*(d+e)-a\*d-b\*e)\* +

b\*e\* +

((a+c)\*(d+f)-a\*d-c\*f)\* +

((b+c)\*(e+f)-b\*e-c\*f)\* +

c\*f\*

共 6 次乘法

Trandition 9次乘法

Divide 2 karatsuba 5次乘法

**Pseudo code**：

string dynamic\_karatsuba(int divide, string num1, string num2)

{

if num1.size() < num2.size()

swap(num1,num2)

int maxlen = max(num1.size(), num2.size());

if (min(num1.size(), num2.size()) <= 1)

return to\_string(stoi(num1) \* stoi(num2));

else if (maxlen < divide)

divide = 2;

paddingzero(&num1, maxlen - num1.size(), 1);

paddingzero(&num2, maxlen - num2.size(), 1);

maxlen\_divide = maxlen / divide;

last\_len = (maxlen - maxlen\_divide \* divide);

for (int i = 0; i < divide; i++)

if (i == 0 && last\_len != 0)

{

tmp1 = num1.substr(0, last\_len + maxlen\_divide);

tmp2 = num2.substr(0, last\_len + maxlen\_divide);

sub\_num1.append (tmp1);

sub\_num2. append (tmp2);

}

else

{

tmp1 = num1.substr(last\_len + maxlen\_divide \* i, maxlen\_divide);

tmp2 = num2.substr(last\_len + maxlen\_divide \* i, maxlen\_divide);

sub\_num1.append(tmp1);

sub\_num2.append(tmp2);

}

for (int i = 0; i < terms; i++)

tmp = dynamic\_karatsuba(divide, sub\_num1[i], sub\_num2[i]);

part\_multiplication.append (tmp);

for (int i = 0; i < terms - 1; i++)

for (int j = i + 1; j < terms; j++)

plus\_num1 = sub\_num1[i];

plus\_num2 = sub\_num2[i];

plus\_num1 = plusnum(sub\_num1[i], sub\_num1[j]);

plus\_num2 = plusnum(sub\_num2[i], sub\_num2[j]);

multiplication\_plus = dynamic\_karatsuba(divide, plus\_num1, plus\_num2);

multiplication\_plus = subnum(subnum(multiplication\_plus, part\_multiplication[i]), part\_multiplication[j]);

padd = (part\_multiplication.size() - i - 1);

padd = padd + (part\_multiplication.size() - j - 1);

paddingzero(&multiplication\_plus, ((terms - i - 1) + (terms - j - 1)) \* maxlen\_divide, 0);

part\_plus.append(multiplication\_plus);

for (int i = 0; i < terms; i++)

paddingzero(&part\_multiplication[i], maxlen\_divide \* (terms - i - 1) \* 2, 0);

final\_num = part\_multiplication[0];

for (int i = 1; i < terms; i++)

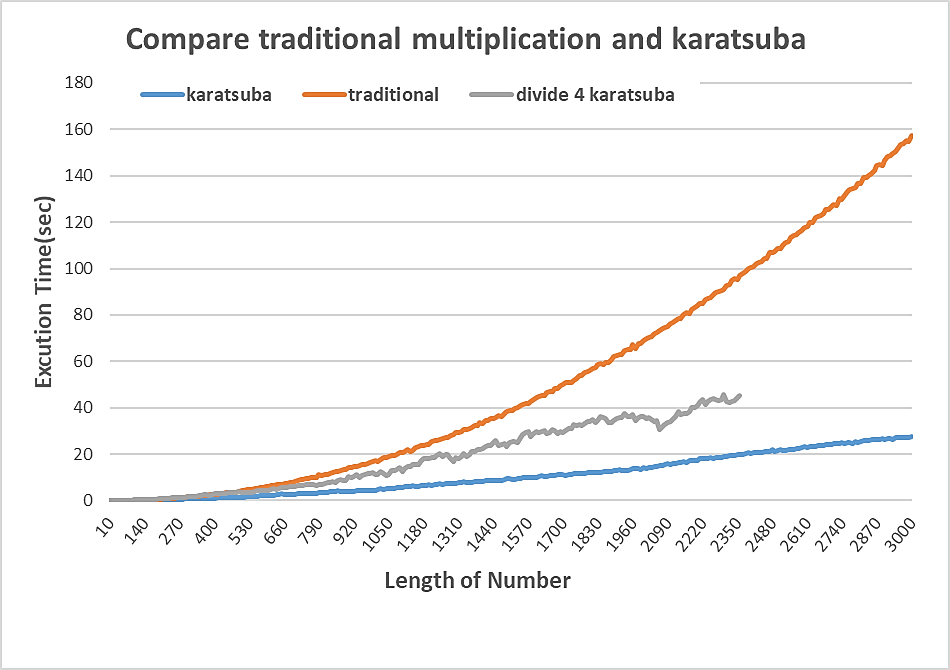
final\_num = plusnum(final\_num, part\_multiplication[i]);

for (int i = 0; i < part\_plus.size(); i++)

final\_num = plusnum(final\_num, part\_plus[i]);

return final\_num;

}



**n=400**

When n>400 karatsuba algorithm will faster then traditional method.