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(n^2)

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;
}
```

2. 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)*10^2 + (r)*10^1 + (b*d) *10^0

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

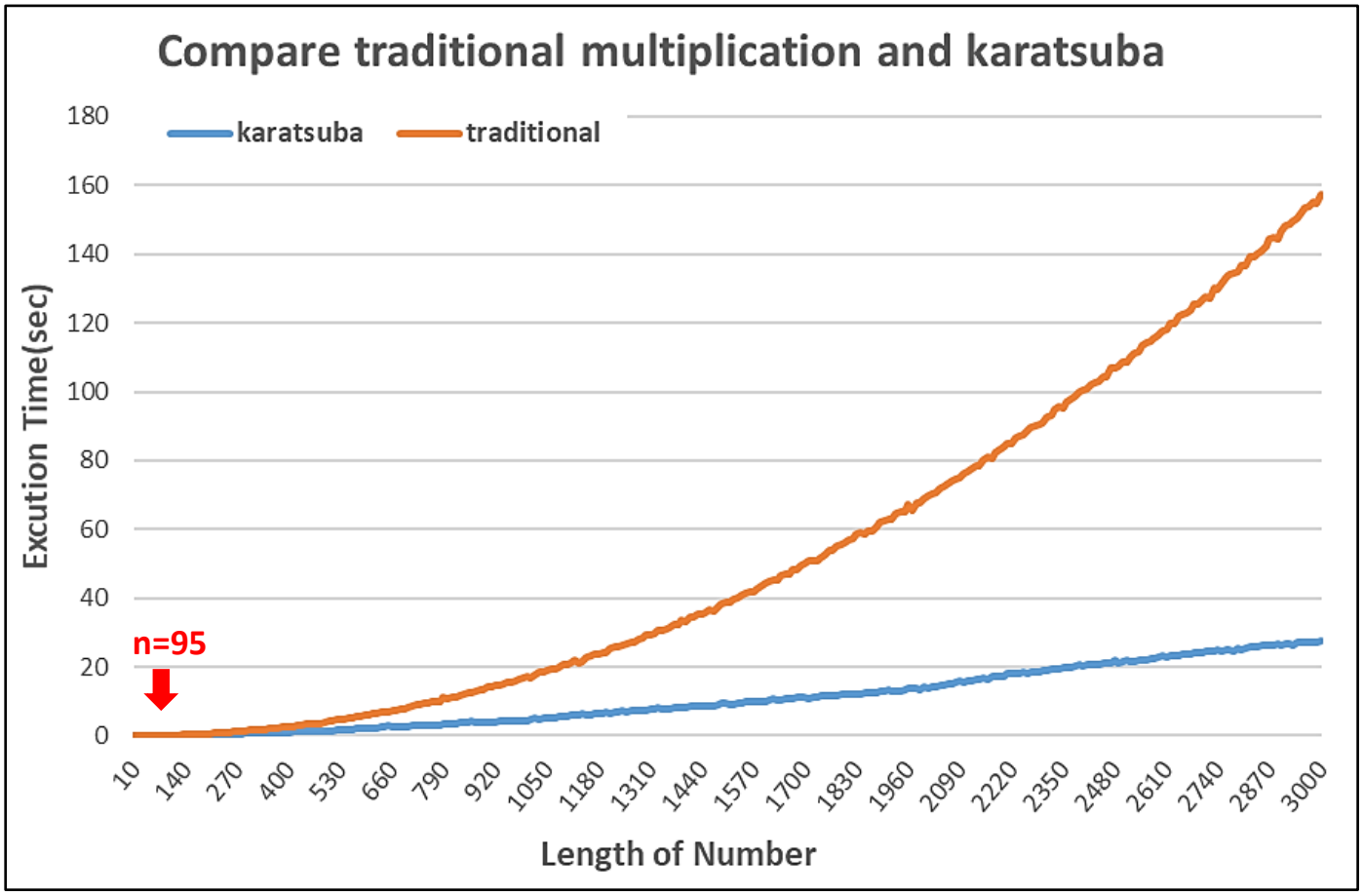
a = 3 , b = 2 , f(n) = n

因為 f(n) = O($n^{log_2 3-c}$)

所以 T(n) = Θ($n^{log_2 3}$)

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;
}
```



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

3. 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)

$A = a*10^2+b*10^1+c$

$B=d*10^2+e*10^1+f$

$A*B = a*d*10^4 +$

$((a+b)*(d+e)-a*d-b*e)* 10^3 +$

$b*e*10^2 +$

$((a+c)*(d+f)-a*d-c*f)* 10^2 +$

$((b+c)*(e+f)-b*e-c*f)* 10^1 +$

$c*f*10^0$

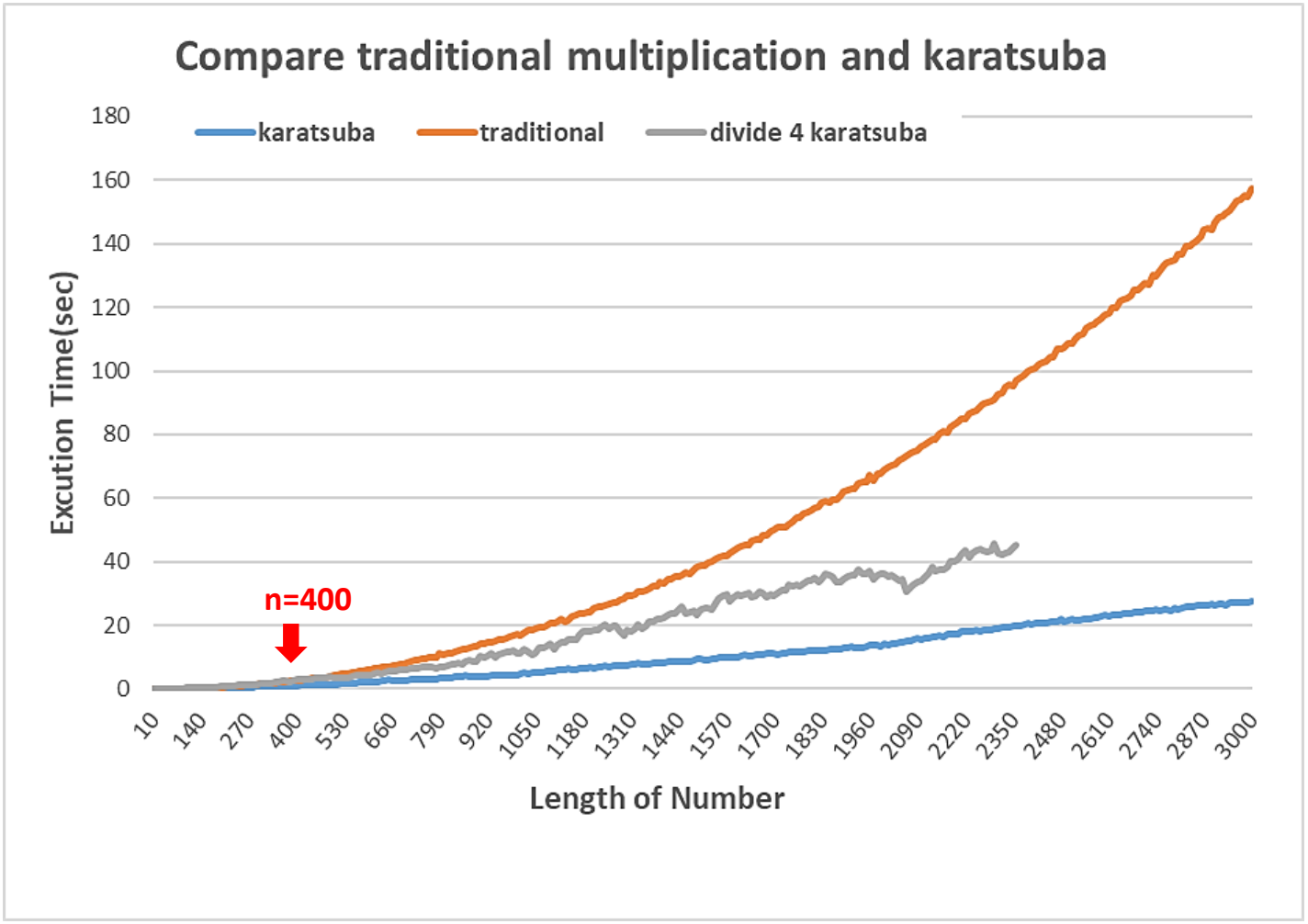
共 6 次乘法

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;
}
```



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