Chih-Hsiang Wang, 933271081

1.

```c
// language: C
// reference: https://www.geeksforgeeks.org/multiply-two-polynomials-2/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
int main() {
    int max_1 = 0, max_2 = 0, max_3 = 0, coef = 0;
    int poly_A[100], poly_B[100], poly_C[100] = {0};

    printf("Enter the power of polynomial A: \n");
    scanf("%d", &max_1);

    printf("Enter each coefficient of polynomial A: (from x^0 to x^n)\n");
    for (int i = 0; i <= max_1; i++) {
        scanf("%d",&poly_A[i]);
    }

    printf("Enter the power of polynomial B: \n");
    scanf("%d",&max_2);

    printf("Enter each coefficient of polynomial B: (from x^0 to x^n)\n");
    for(int i = 0; i <= max_2; i++) {
        scanf("%d",&poly_B[i]);
    }

    // iterative multiplication
    for(int i = 0; i <= max_1; i++){
        for(int j = 0; j <= max_2; j++){
            poly_C[i+j] += poly_A[i] * poly_B[j];
        }
    }

    // print out the result
    max_3 = max_1 + max_2;
    coef = 0;
    for (int i = 0; i <=max_3; i++) {
        printf("a%d: %d\n", coef, poly_C[i]);
        coef++;
    }

    return 0;
}
```

2.
// language: C++
// reference: http://algorithm.cs.nthu.edu.tw/~course/Extra_Info/Divide%20and%20Conquer_supplement.pdf
// reference: https://github.com/nextco/cormen-fft/blob/master/pm-divide-and-conquer.cpp

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <chrono>
#include <time.h>

using namespace std;
typedef long long ll;

// polynomial coefficients are saved in increasing order of degree
// coefficient of x**i in polynomial p = p[i]

// multiply polynomials p and q, both of size sz,
// where sz is multiple of 2
void karatsuba(ll *res, const ll *p, const ll *q, int sz){
    ll t0[sz], t1[sz], r[sz<<1];

    memset(r, 0, (sz<<1) * sizeof(ll));

    if ( sz <= 4 ){ // base case, no recursion, do basic school multiplication
        for ( int i = 0 ; i < sz ; i++ )
        for ( int j = 0 ; j < sz ; j++ ){
            r[i + j] += p[i] * q[j];
        }
    } else {
        // let p = a*x**nSz + b
        //     q = c*x**nSz + d
        //     r = ac*x**sz + ((a+b)*(c+d) - ac - bd)*x**nSz + bd
        int nSz = (sz >> 1);

        for ( int i = 0 ; i < nSz ; i++ ){
            t0[i] = p[i] + p[nSz + i];   // t0 = a + b
            t1[i] = q[i] + q[nSz + i];   // t1 = c + d
            t0[i + nSz] = t1[i + nSz] = 0;  // initialize
        }

        karatsuba(r + nSz, t0, t1, nSz);      // r[nSz...sz] = (a+b) (c+d)
        karatsuba(t0, p, q, nSz);             // t0 = bd
        karatsuba(t1, p + nSz, q + nSz, nSz); // t1 = ac
```

```
        for ( int i = 0 ; i < sz ; i++ ){
            r[i] += t0[i];                    // bd
            r[i + nSz] -= t0[i] + t1[i];      // ((a+b)(c+d) - ac - bd) * x**nSz
            r[i + sz] += t1[i];               // ac * x**sz
        }
    }

    memcpy(res, r, (sz<<1) * sizeof(ll));
}

// multiply two polynomials p and q, both of size sz = degree + 1
// save the output in array r
// NOTE: the maximum capacity of p, q, r should be power of two
// NOTE: r should be at least double of p or q in size
void polyMult(ll *r, ll *p, ll *q, int sz){
    if ( sz & (sz - 1) ){ // if size is not power of two
        int k = 1;
        while ( k < sz ) k <<= 1;
        while ( ++sz <= k ) p[sz - 1] = q[sz - 1] = 0;
        sz--;
    }

    karatsuba(r, p, q, sz);
}

// print polynomial in descending order of degree
void polyPrint(ll *p, int sz){
    while ( --sz >= 0 ) cout << p[sz] <<" ";
    puts("");
}

int main(){
    ll p[4] = {4,3,2,1};
    ll q[4] = {4,3,2,1};
    ll r[8];
    int degree = 3;

    polyMult(r, p, q, degree + 1);
    polyPrint(r, degree * 2 + 1);
    return 0;
}
```

3.

```
// language: C++
// reference: https://www.geeksforgeeks.org/fast-fourier-transformation-poynomial-multiplication/
// reference: https://hk.saowen.com/a/b6e9f0ca70a669575a8b8e56c746ba63780958c4c12159aec6bfb05a7ff2e409
// http://www.voidcn.com/article/p-rzrkhina-boa.html

#include <cstdio>
#include <cmath>
#include <iostream>
#include <time.h>

const int MAXN = 4 * 1e5 + 10;
double Pi = acos(-1);

struct Complex {
    double r, i;
    Complex() {}
    Complex(double _r, double _i) { r = _r; i = _i; }
    Complex operator + (const Complex &y) { return Complex(r + y.r, i + y.i); }
    Complex operator - (const Complex &y) { return Complex(r - y.r, i - y.i); }
    Complex operator * (const Complex &y) { return Complex(r*y.r - i * y.i, r*y.i + i * y.r); }
    Complex operator *= (const Complex &y) {
        double t = r;
        return Complex(r = r * y.r - i * y.i, i = t * y.i + i * y.r); }
} a[MAXN], b[MAXN];

void FFT(Complex* a, long length, int op){
    if(length == 1)
    return;
    Complex * a0 = new Complex[length/2];
    Complex * a1 = new Complex[length/2];
    for(long i = 0; i < length; i += 2){
        a0[i/2] = a[i];
        a1[i/2] = a[i+1];
    }

    FFT(a0, length/2, op);
    FFT(a1, length/2, op);

    Complex wn(cos(2*Pi/length),op * sin(2*Pi/length));
    Complex w(1, 0);

    for(long i = 0; i < (length/2); i++){
        a[i] = a0[i]+w*a1[i];
```

```
        a[i+length/2] = a0[i]-w*a1[i];
        w = w*wn;
    }

    delete[] a0;
    delete[] a1;
}

int main(int argc, const char * argv[]) {
    clock_t start, finish;
    float duration;

    int ordP1,ordP2,length;
    double innc;
    Complex poly_A[100],poly_B[100],poly_C[100];

    printf("Enter the power of polynomial A: \n");
    scanf("%d",&ordP1);
    printf("Enter each coefficient of polynomial A: (from x^0 to x^n)\n");
    for(int i=0;i<=ordP1;i++){
        scanf("%lf",&innc);
        poly_A[i].r=innc;
        poly_A[i].i=0;
    }

    printf("Enter the power of polynomial B: \n");
    scanf("%d",&ordP2);
    printf("Enter each coefficient of polynomial B: (from x^0 to x^n)\n");
    for(int i=0;i<=ordP2;i++){
        scanf("%lf",&innc);
        poly_B[i].r=innc;
        poly_B[i].i=0.0;
    }

    start = clock();
//    FFT ALG
    for(int i=1; i<=abs(ordP1-ordP2) && (ordP1-ordP2!=0) ;i++){
        if(ordP1-ordP2<0){
            poly_A[i+ordP1]*=Complex(0.0,0.0);
        }
        else{
            poly_B[i+ordP2]*=Complex(0.0,0.0);
        }
    }

    length = (ordP1-ordP2)?(ordP1+1):(ordP2+1);
```

```
    for(length = 2; length < ordP1+ordP2+2;length*=2);

    for(int i =ordP1+1;i<length;i++){
        poly_A[i] = poly_A[i] * Complex(0.0,0.0);
        poly_B[i] = poly_B[i] * Complex(0.0,0.0);
    }

    FFT(poly_A,length,1);
    FFT(poly_B,length,1);

    for (int i =0;i<length;i++){
        poly_C[i]=poly_A[i]*poly_B[i];
    }

    FFT(poly_C,length,-1);
    for (int i =0;i<length;i++){
        poly_C[i].r /= length;
    }
    finish = clock();
    duration = (double)(finish - start) / CLOCKS_PER_SEC;
    //    printf( "%f seconds\n", duration );
    int coef = 0;
    for(int i = 0;i< ordP1+ordP2+1 ;i++) {
        printf("a%d: %lf \n", coef, poly_C[i].r);
        coef++;
    }

    printf("\n");
    return 0;
}
```

4.
// classical iterative method
// reference: (Cull, Classnote, p.45)
// reference: http://web.cs.iastate.edu/~cs577/handouts/polymultiply.pdf

$P(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1}$
$Q(x) = b_0 + b_1x + b_2x^2 + \ldots + b_{n-1}x^{n-1}$

$P(x)Q(x) = c_0 + c_1x + \ldots + c_{2n-2}x^{2n-2}$, where $c_i = \sum_{\max\{0,i-(n-1)\} \le k \le \min\{i,n-1\}} a_k b_{i-k}.$

because every a_i is multiply with every b_j, for $0 \le i, j \le n-1$
$\Longrightarrow \Theta(n^2)$

// divide and conquer method
// reference: http://algorithm.cs.nthu.edu.tw/~course/Extra_Info/Divide%20and%20Conquer_supplement.pdf

$P(x) = P_0(x) + P_1(x)x^{n/2}$
$Q(x) = Q_0(x) + Q_1(x)x^{n/2}$
$P(x)Q(x) = P_0(x)Q_0(x) + (P_0(x)\, Q_1(x) + P_1(x)Q_0(x))x^{n/2} + P_1(x)Q_1(x)x^n$
where $P_0(x)$, $P_1(x)$, $Q_0(X)$, $Q_1(x)$ are all polynomials of degree n/2 - 1

$M(n) = 3M(n/2) + cn$
$\quad = 3^k(n/2^k) + (1+2+4+\ldots+2^{k-1})cn$
when $n/2^k = 1 \Longrightarrow M(1) \Longrightarrow k = \log_2 n$
$\Longrightarrow M(n) = 3^{\log n} M(1) + cn(2^{\log n} - 1) = 3^{\log n} M(1) + cn(n - 1)$
$\Longrightarrow \Theta(3^{\log n})$
$\Longrightarrow \Theta(n^{\log 3})$

// FFT method
// reference: http://web.cs.iastate.edu/~cs577/handouts/polymultiply.pdf
// reference: (Cull, Classnote, p.50~54)

$\textsc{Recursive-DFT}(\boldsymbol{a}, n)$

if $n = 1$
  then return $\boldsymbol{a}$
$w_n \leftarrow e^{i\frac{2\pi}{n}}$
$w \leftarrow 1$
$\boldsymbol{a}^{[0]} \leftarrow (a_0, a_2, \ldots, a_{n-2})$
$\boldsymbol{a}^{[1]} \leftarrow (a_1, a_3, \ldots, a_{n-1})$
$\hat{\boldsymbol{a}}^{[0]} \leftarrow \textsc{Recursive-DFT}(\boldsymbol{a}^{[0]}, \frac{n}{2})$
$\hat{\boldsymbol{a}}^{[1]} \leftarrow \textsc{Recursive-DFT}(\boldsymbol{a}^{[1]}, \frac{n}{2})$
for $k = 0$ **to** $\frac{n}{2} - 1$ **do**
  $\hat{a}_k \leftarrow \hat{a}_k^{[0]} + \omega \hat{a}_k^{[1]}$
  $\hat{a}_{k+\frac{n}{2}} \leftarrow \hat{a}_k^{[0]} - \omega \hat{a}_k^{[1]}$
  $\omega \leftarrow \omega \omega_n$
return $(\hat{a}_0, \hat{a}_1, \ldots, \hat{a}_{n-1})$

$\Longrightarrow$ T(n, |x|) = 2*T(n/2, |x|) + O(n + |x|)

if T(n/2, |x|/2)
$\Longrightarrow$ T(n) = 2T(n/2) + O(n)
by the formula of Master Theorem
$\Longrightarrow$ Θ(nlogn)

By the big theta of three algorithms, I predict that FFT method will be the fastest for large degree polynomials. Because nlogn will be smaller than n^2 if n is large.

5.

case 1: choose $P(x) = 1 + 2x^2 + 3x^3 + 4x^4$, $Q(x) = 1 + 2x^2 + 3x^3 + 4x^4$

// classical iterative method

```
Enter the power of polynomial A:
3
Enter each coefficient of polynomial A: (from x^0 to x^n)
1
2
3
4
Enter the power of polynomial B:
3
Enter each coefficient of polynomial B: (from x^0 to x^n)
1
2
3
4
a0: 1
a1: 4
a2: 10
a3: 20
a4: 25
a5: 24
a6: 16
Program ended with exit code: 0
```

// divide and conquer method

from x_n -> constant

```
16 24 25 20 10 4 1
Program ended with exit code: 0
```

// FFT method

```
Enter the power of polynomial A:
3
Enter each coefficient of polynomial A: (from x^0 to x^n)
1
2
3
4
Enter the power of polynomial B:
3
Enter each coefficient of polynomial B: (from x^0 to x^n)
1
2
3
4
a0: 1.000000
a1: 4.000000
a2: 10.000000
a3: 20.000000
a4: 25.000000
a5: 24.000000
a6: 16.000000

Program ended with exit code: 0
```

case 2: choose $P(x) = 4 + 3x^1 + 2x^2 + 1x^3$,
$Q(x) = 4 + 3x^1 + 2x^2 + 1x^3$

// classical iterative method

```
Enter the power of polynomial A:
3
Enter each coefficient of polynomial A: (from x^0 to x^n)
4
3
2
1
Enter the power of polynomial B:
3
Enter each coefficient of polynomial B: (from x^0 to x^n)
4
3
2
1
a0: 16
a1: 24
a2: 25
a3: 20
a4: 10
a5: 4
a6: 1
Program ended with exit code: 0
```

// divide and conquer method
from x_n -> constant

```
1 4 10 20 25 24 16
Program ended with exit code: 0
```

// FFT method

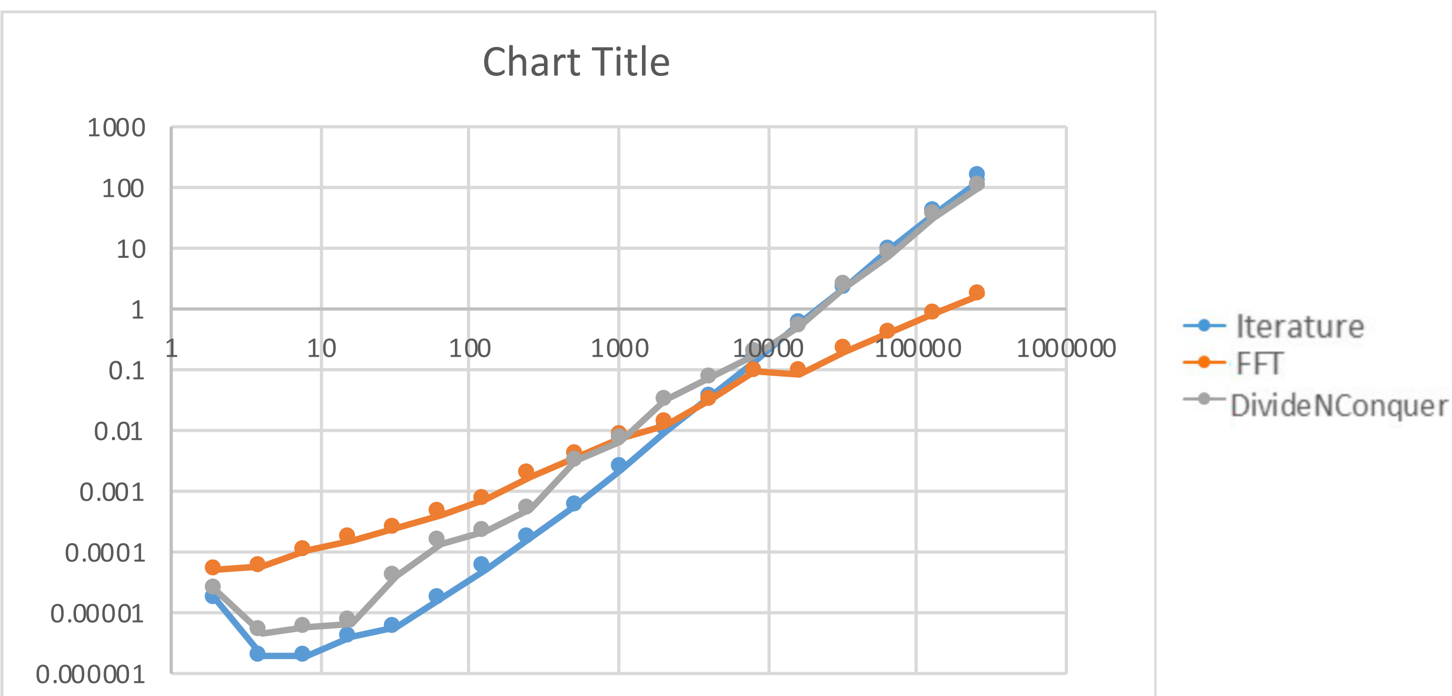
```
Enter the power of polynomial A:
3
Enter each coefficient of polynomial A: (from x^0 to x^n)
4
3
2
1
Enter the power of polynomial B:
3
Enter each coefficient of polynomial B: (from x^0 to x^n)
4
3
2
1
a0: 16.000000
a1: 24.000000
a2: 25.000000
a3: 20.000000
a4: 10.000000
a5: 4.000000
a6: 1.000000

Program ended with exit code: 0
```

The results are the same.

6.

n	iterative (sec)	divide and conquer (sec)	FFT (sec)
2	0.000016	0.000024	0.000051
4	0.000002	0.000005	0.000058
8	0.000002	0.000006	0.00011
16	0.000004	0.000007	0.000166
32	0.000006	0.000041	0.000247
64	0.000017	0.000141	0.000421
128	0.000055	0.000219	0.000746
256	0.000175	0.000517	0.001839
512	0.000595	0.003132	0.003755
1024	0.002367	0.007071	0.007828
2048	0.009903	0.031962	0.013254
4096	0.036968	0.07321	0.031336
8192	0.14393	0.187103	0.09269
16384	0.574414	0.530683	0.091832
32768	2.25881	2.353215	0.205122
65536	9.286633	7.879021	0.406042
131072	37.946583	33.402908	0.829161
262144	148.39357	110.21917	1.71501



7.
For a small number of coefficient, classical iterative method is faster than divide and conquer and FFT. However, FFT will be the fastest, and divide and conquer will be faster than iterative.

// classical iterative method
$c * n^2$
$c * (16384)^2 = 0.574414 \Longrightarrow c \approx 2.10 * 10^{-9}$
$c * (65536)^2 = 9.286633 \Longrightarrow c \approx 2.30 * 10^{-9}$
$c * (131072)^2 = 37.946583 \Longrightarrow c \approx 2.21 * 10^{-9}$
$c \approx 2.20 * 10^{-9}$
$\Longrightarrow 2.20 * 10^{-9} * n^2$

// divide and conquer method
$c * n^2$
$c * (16384)^2 = 0.530683 \Longrightarrow c \approx 1.98 * 10^{-9}$
$c * (65536)^2 = 9.286633 \Longrightarrow c \approx 2.17 * 10^{-9}$
$c * (131072)^2 = 37.946583 \Longrightarrow c \approx 2.21 * 10^{-9}$
$c \approx 2.12 * 10^{-9}$
$\Longrightarrow 2.12 * 10^{-9} * n^2$

// FFT method
$c * n(\log n)$
$c * 16384(\log 16384) = 0.091832 \Longrightarrow c \approx 4.00 * 10^{-7}$
$c * 32768(\log 32768) = 2.353215 \Longrightarrow c \approx 4.17 * 10^{-7}$
$c * 65536(\log 65536) = 7.879021 \Longrightarrow c \approx 3.87 * 10^{-7}$
$c \approx 4.01 * 10^{-7}$
$\Longrightarrow 4.01 * 10^{-7} * n(\log n)$

by the plot, the crossover point of iterative method and divide and conquer method is approximately at n = 8192

by the plot, the crossover point of iterative method and FFT method is approximately at n = 4096