Multiply two polynomials

Given two polynomials represented by two arrays, write a function that multiplies given two polynomials.

Example:

```
Input:  A[] = {5, 0, 10, 6}
        B[] = {1, 2, 4}
Output: prod[] = {5, 10, 30, 26, 52, 24}

The first input array represents "5 + 0x^1 + 10x^2 + 6x^3"
The second array represents "1 + 2x^1 + 4x^2"
And Output is "5 + 10x^1 + 30x^2 + 26x^3 + 52x^4 + 24x^5"
```

We strongly recommend that you click here and practice it, before moving on to the solution.

A simple solution is to one by one consider every term of first polynomial and multiply it with every term of second polynomial. Following is algorithm of this simple method.

```
multiply(A[0..m-1], B[0..n01])
1) Create a product array prod[] of size m+n-1.
2) Initialize all entries in prod[] as 0.
3) Travers array A[] and do following for every element A[i]
...(3.a) Traverse array B[] and do following for every element B[j]
          prod[i+j] = prod[i+j] + A[i] * B[j]
4) Return prod[].
```

The following is C++ implementation of above algorithm.

```
// Simple C++ program to multiply two polynomials
#include <iostream>
using namespace std;

// A[] represents coefficients of first polynomial
// B[] represents coefficients of second polynomial
// m and n are sizes of A[] and B[] respectively
int *multiply(int A[], int B[], int m, int n)
```

```
{
   int *prod = new int[m+n-1];

   // Initialize the porduct polynomial
   for (int i = 0; i<m+n-1; i++)
     prod[i] = 0;

   // Multiply two polynomials term by term

   // Take ever term of first polynomial
   for (int i=0; i<m; i++)
   {
     // Multiply the current term of first polynomial
     // with every term of second polynomial.
     for (int j=0; j<n; j++)
         prod[i+j] += A[i]*B[j];
   }

   return prod;
}

// A utility function to print a polynomial
void printPoly(int poly[], int n)
{
    for (int i=0; i<n; i++)
    {
       cout << poly[i];
       if (i != 0)
        cout << "x^" << i ;
       if (i != n-1)
       cout << " + ";
    }
}

// Driver program to test above functions
int main()
{
    // The following array represents polynomial 5 + 10x^2 + 6x^3
    int A[] = {5, 0, 10, 6};

    // The following array represents polynomial 1 + 2x + 4x^2
    int B[] = {1, 2, 4};
    int m = sizeof(A)/sizeof(A[0]);
    int n = sizeof(B)/sizeof(B[0]);

    cout << "First polynomial is \n";
    printPoly(A, m);
    cout << "\nSecond polynomial is \n";
    printPoly(B, n);

    int *prod = multiply(A, B, m, n);

    cout << "\nProduct polynomial is \n";
    printPoly(prod, m+n-1);

    return 0;
}
```

Run on IDE

Output

```
First polynomial is
5 + 0x^1 + 10x^2 + 6x^3
Second polynomial is
1 + 2x^1 + 4x^2
```

```
Product polynomial is
5 + 10x^1 + 30x^2 + 26x^3 + 52x^4 + 24x^5
```

Time complexity of the above solution is O(mn). If size of two polynomials same, then time complexity is $O(n^2)$.

Can we do better?

There are methods to do multiplication faster than $O(n^2)$ time. These methods are mainly based on divide and conquer. Following is one simple method that divides the given polynomial (of degree n) into two polynomials one containing lower degree terms(lower than n/2) and other containing higher degree terns (higher than or equal to n/2)

```
Let the two given polynomials be A and B.
For simplicity, Let us assume that the given two polynomials are of
same degree and have degree in powers of 2, i.e., n = 2^i

The polynomial 'A' can be written as A0 + A1*x^(n/2)
The polynomial 'B' can be written as B0 + B1*x^(n/2)

For example 1 + 10x + 6x^2 - 4x^3 + 5x^4 can be
written as (1 + 10x) + (6 - 4x + 5x^2)*x^2

A * B  = (A0 + A1*x^(n/2)) * (B0 + B1*x^(n/2))
       = A0*B0 + A0*B1*x^(n/2) + A1*B0*x^(n/2) + A1*B1*x^n
       = A0*B0 + (A0*B1 + A1*B0)x^(n/2) + A1*B1*x^n
```

So the above divide and conquer approach requires 4 multiplications and O(n) time to add all 4 results. Therefore the time complexity is T(n) = 4T(n/2) + O(n). The solution of the recurrence is $O(n^2)$ which is same as the above simple solution.

The idea is to reduce number of multiplications to 3 and make the recurrence as T(n) = 3T(n/2) + O(n)

How to reduce number of multiplications?

This requires a little trick similar to Strassen's Matrix Multiplication. We do following 3 multiplications.

```
X = (A0 + A1)*(B0 + B1) // First Multiplication
Y = A0B0  // Second
Z = A1B1  // Third

The missing middle term in above multiplication equation A0*B0 + (A0*B1 +
A1*B0)x^(n/2) + A1*B1*x^n can obtained using below.
A0B1 + A1B0 = X - Y - Z
```

So the time taken by this algorithm is T(n) = 3T(n/2) + O(n)
The solution of above recurrence is $O(n^{Lg3})$ which is better than $O(n^2)$.

We will soon be discussing implementation of above approach.

There is a O(nLogn) algorithm also that uses Fast Fourier Transform to multiply two polynomials (Refer this and this for details)

Sources:

http://www.cse.ust.hk/~dekai/271/notes/L03/L03.pdf