Week 34: Advanced Math & Number Theory – FFT, NTT, and Polynomial Operations

Topics: - Fast Fourier Transform (FFT) for Polynomial Multiplication - Number Theoretic Transform (NTT) for Modular Polynomials - Polynomial Inversion and Division - Convolution Applications in Combinatorics and String Matching - Multipoint Evaluation and Interpolation - Modular Arithmetic and Primitive Roots for NTT

Weekly Tips: - FFT multiplies polynomials in O(n log n) over real/complex numbers. - NTT works over integers modulo a prime suitable for primitive roots. - Polynomial inversion allows solving series expansions and recurrences. - Convolution can solve subset sum, pattern matching, and coefficient extraction problems. - Careful modular arithmetic is essential for precision and correctness.

Problem 1: Polynomial Multiplication using FFT Link: [Codeforces Example]() **Difficulty:** Advanced

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
using cd = complex<double>;
const double PI=acos(-1);
void fft(vector<cd> &a,bool invert){
    int n=a.size();
    for(int i=1,j=0;i<n;i++){
        int bit=n>>1;
        for(;j&bit;bit>>=1) j^=bit;
        j^=bit;
        if(i<j) swap(a[i],a[j]);
    }
    for(int len=2;len<=n;len<<=1){
        double ang=2*PI/len*(invert?-1:1);
        cd wlen(cos(ang),sin(ang));
        for(int i=0;i<n;i+=len){
            cd w(1);
            for(int j=0;j<len/2;j++){
                cd u=a[i+j],v=a[i+j+len/2]*w;
                a[i+j]=u+v; a[i+j+len/2]=u-v;
                w*=wlen;
            }
        }
    }
    if(invert) for(cd &x:a) x/=n;
}
vector<long long> multiply(vector<long long> const& a, vector<long long> const&
b){
    vector<cd> fa(a.begin(),a.end()), fb(b.begin(),b.end());
    int n=1; while(n<a.size()+b.size()) n<<=1;
    fa.resize(n); fb.resize(n);
```

```
    fft(fa,false); fft(fb,false);
    for(int i=0;i<n;i++) fa[i]*=fb[i];
    fft(fa,true);
    vector<long long> result(n);
    for(int i=0;i<n;i++) result[i]=round(fa[i].real());
    return result;
}
int main(){
    int n,m; cin>>n>>m;
    vector<long long> a(n),b(m);
    for(int i=0;i<n;i++) cin>>a[i];
    for(int i=0;i<m;i++) cin>>b[i];
    vector<long long> res=multiply(a,b);
    for(int x:res) cout<<x<<' '; cout<<endl;
}
```

Explanation Comments: - FFT converts polynomials to frequency domain for multiplication. - Inverse FFT retrieves coefficients. - Reduces naive O(n^2) multiplication to O(n log n). - Useful for combinatorial convolution and string pattern problems.

Problem 2: Number Theoretic Transform (NTT) Example Link: [CP-Algorithms NTT]() **Difficulty:** Advanced

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
const int MOD=998244353, root=15311432, root_1=469870224, root_pw=1<<23;
void ntt(vector<int> & a, bool invert){
    int n=a.size();
    for(int i=1,j=0;i<n;i++){
        int bit=n>>1; for(;j&bit;bit>>=1) j^=bit; j^=bit;
        if(i<j) swap(a[i],a[j]);
    }
    for(int len=2;len<=n;len<<=1){
        int wlen = invert ? root_1 : root;
        for(int i=len;i<root_pw;i<<=1) wlen = (int)(1LL*wlen*wlen%MOD);
        for(int i=0;i<n;i+=len){
            int w=1;
            for(int j=0;j<len/2;j++){
                int u=a[i+j], v=(int)(1LL*a[i+j+len/2]*w%MOD);
                a[i+j]=(u+v<MOD? u+v: u+v-MOD);
                a[i+j+len/2]=(u-v>=0? u-v: u-v+MOD);
                w=(int)(1LL*w*wlen%MOD);
            }
        }
    }
```

```
    if(invert){
        int n_1=1; for(int i=0;i<MOD-2;i++) n_1=(int)(1LL*n_1* n%MOD);
        for(int & x: a) x=(int)(1LL*x*n_1%MOD);
    }
}
```

Explanation Comments: - NTT performs polynomial multiplication modulo prime efficiently. - Avoids precision errors inherent in floating-point FFT. - Essential for modular arithmetic problems and large coefficient multiplications.

End of Week 34 - FFT and NTT enable high-performance polynomial operations. - Practice convolutions, multipoint evaluation, and modular polynomial arithmetic for ACM-ICPC contests.