```cpp
/*
  ##########################################################################
  ######################   THE   BIG   INT   ###############################
*/
const int base = 1000000000;
const int base_digits = 9;
struct bigint {
    vector<int> a;
    int sign;
    /*<arpa>*/
    int size(){
    if(a.empty())return 0;
    int ans=(a.size()-1)*base_digits;
    int ca=a.back();
    while(ca)
        ans++,ca/=10;
    return ans;
    }
    bigint operator ^(const bigint &v){
    bigint ans=1,a=*this,b=v;
    while(!b.isZero()){
        if(b%2)
        ans*=a;
        a*=a,b/=2;
    }
    return ans;
    }
    string to_string(){
    stringstream ss;
    ss << *this;
    string s;
    ss >> s;
    return s;
    }
    int sumof(){
    string s = to_string();
    int ans = 0;
    for(auto c : s)  ans += c - '0';
    return ans;
    }
    /*</arpa>*/
    bigint() :
    sign(1) {
    }

    bigint(long long v) {
    *this = v;
    }

    bigint(const string &s) {
    read(s);
```

```
}

void operator=(const bigint &v) {
sign = v.sign;
a = v.a;
}

void operator=(long long v) {
sign = 1;
a.clear();
if (v < 0)
    sign = -1, v = -v;
for (; v > 0; v = v / base)
    a.push_back(v % base);
}

bigint operator+(const bigint &v) const {
if (sign == v.sign) {
    bigint res = v;

    for (int i = 0, carry = 0; i < (int) max(a.size(), v.a.size()) ||
     carry; ++i) {
    if (i == (int) res.a.size())
        res.a.push_back(0);
    res.a[i] += carry + (i < (int) a.size() ? a[i] : 0);
    carry = res.a[i] >= base;
    if (carry)
        res.a[i] -= base;
    }
    return res;
}
return *this - (-v);
}

bigint operator-(const bigint &v) const {
if (sign == v.sign) {
    if (abs() >= v.abs()) {
    bigint res = *this;
    for (int i = 0, carry = 0; i < (int) v.a.size() || carry; ++i) {
        res.a[i] -= carry + (i < (int) v.a.size() ? v.a[i] : 0);
        carry = res.a[i] < 0;
        if (carry)
        res.a[i] += base;
    }
    res.trim();
    return res;
    }
    return -(v - *this);
}
return *this + (-v);
}
```

```
void operator*=(int v) {
if (v < 0)
    sign = -sign, v = -v;
for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i) {
    if (i == (int) a.size())
    a.push_back(0);
    long long cur = a[i] * (long long) v + carry;
    carry = (int) (cur / base);
    a[i] = (int) (cur % base);
    //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"(cur), "c"(base));
}
trim();
}

bigint operator*(int v) const {
bigint res = *this;
res *= v;
return res;
}

void operator*=(long long v) {
if (v < 0)
    sign = -sign, v = -v;
for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i) {
    if (i == (int) a.size())
    a.push_back(0);
    long long cur = a[i] * (long long) v + carry;
    carry = (int) (cur / base);
    a[i] = (int) (cur % base);
    //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"(cur), "c"(base));
}
trim();
}

bigint operator*(long long v) const {
bigint res = *this;
res *= v;
return res;
}

friend pair<bigint, bigint> divmod(const bigint &a1, const bigint &b1) {
int norm = base / (b1.a.back() + 1);
bigint a = a1.abs() * norm;
bigint b = b1.abs() * norm;
bigint q, r;
q.a.resize(a.a.size());

for (int i = a.a.size() - 1; i >= 0; i--) {
    r *= base;
    r += a.a[i];
```

```
    int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
    int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() - 1];
    int d = ((long long) base * s1 + s2) / b.a.back();
    r -= b * d;
    while (r < 0)
    r += b, --d;
    q.a[i] = d;
}

q.sign = a1.sign * b1.sign;
r.sign = a1.sign;
q.trim();
r.trim();
return make_pair(q, r / norm);
}

bigint operator/(const bigint &v) const {
return divmod(*this, v).first;
}

bigint operator%(const bigint &v) const {
return divmod(*this, v).second;
}

void operator/=(int v) {
if (v < 0)
    sign = -sign, v = -v;
for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i) {
    long long cur = a[i] + rem * (long long) base;
    a[i] = (int) (cur / v);
    rem = (int) (cur % v);
}
trim();
}

bigint operator/(int v) const {
bigint res = *this;
res /= v;
return res;
}

int operator%(int v) const {
if (v < 0)
    v = -v;
int m = 0;
for (int i = a.size() - 1; i >= 0; --i)
    m = (a[i] + m * (long long) base) % v;
return m * sign;
}

void operator+=(const bigint &v) {
```

```
*this = *this + v;
}
void operator-=(const bigint &v) {
*this = *this - v;
}
void operator*=(const bigint &v) {
*this = *this * v;
}
void operator/=(const bigint &v) {
*this = *this / v;
}

bool operator<(const bigint &v) const {
if (sign != v.sign)
    return sign < v.sign;
if (a.size() != v.a.size())
    return a.size() * sign < v.a.size() * v.sign;
for (int i = a.size() - 1; i >= 0; i--)
    if (a[i] != v.a[i])
    return a[i] * sign < v.a[i] * sign;
return false;
}

bool operator>(const bigint &v) const {
return v < *this;
}
bool operator<=(const bigint &v) const {
return !(v < *this);
}
bool operator>=(const bigint &v) const {
return !(*this < v);
}
bool operator==(const bigint &v) const {
return !(*this < v) && !(v < *this);
}
bool operator!=(const bigint &v) const {
return *this < v || v < *this;
}

void trim() {
while (!a.empty() && !a.back())
    a.pop_back();
if (a.empty())
    sign = 1;
}

bool isZero() const {
return a.empty() || (a.size() == 1 && !a[0]);
}

bigint operator-() const {
```

```
bigint res = *this;
res.sign = -sign;
return res;
}

bigint abs() const {
bigint res = *this;
res.sign *= res.sign;
return res;
}

long long longValue() const {
long long res = 0;
for (int i = a.size() - 1; i >= 0; i--)
    res = res * base + a[i];
return res * sign;
}

friend bigint gcd(const bigint &a, const bigint &b) {
return b.isZero() ? a : gcd(b, a % b);
}
friend bigint lcm(const bigint &a, const bigint &b) {
return a / gcd(a, b) * b;
}

void read(const string &s) {
sign = 1;
a.clear();
int pos = 0;
while (pos < (int) s.size() && (s[pos] == '-' || s[pos] == '+')) {
    if (s[pos] == '-')
    sign = -sign;
    ++pos;
}
for (int i = s.size() - 1; i >= pos; i -= base_digits) {
    int x = 0;
    for (int j = max(pos, i - base_digits + 1); j <= i; j++)
    x = x * 10 + s[j] - '0';
    a.push_back(x);
}
trim();
}

friend istream& operator>>(istream &stream, bigint &v) {
string s;
stream >> s;
v.read(s);
return stream;
}

friend ostream& operator<<(ostream &stream, const bigint &v) {
```

```
if (v.sign == -1)
    stream << '-';
stream << (v.a.empty() ? 0 : v.a.back());
for (int i = (int) v.a.size() - 2; i >= 0; --i)
    stream << setw(base_digits) << setfill('0') << v.a[i];
return stream;
}

static vector<int> convert_base(const vector<int> &a, int old_digits, int
 new_digits) {
vector<long long> p(max(old_digits, new_digits) + 1);
p[0] = 1;
for (int i = 1; i < (int) p.size(); i++)
    p[i] = p[i - 1] * 10;
vector<int> res;
long long cur = 0;
int cur_digits = 0;
for (int i = 0; i < (int) a.size(); i++) {
    cur += a[i] * p[cur_digits];
    cur_digits += old_digits;
    while (cur_digits >= new_digits) {
    res.push_back(int(cur % p[new_digits]));
    cur /= p[new_digits];
    cur_digits -= new_digits;
    }
}
res.push_back((int) cur);
while (!res.empty() && !res.back())
    res.pop_back();
return res;
}

typedef vector<long long> vll;

static vll karatsubaMultiply(const vll &a, const vll &b) {
int n = a.size();
vll res(n + n);
if (n <= 32) {
    for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        res[i + j] += a[i] * b[j];
    return res;
}

int k = n >> 1;
vll a1(a.begin(), a.begin() + k);
vll a2(a.begin() + k, a.end());
vll b1(b.begin(), b.begin() + k);
vll b2(b.begin() + k, b.end());

vll a1b1 = karatsubaMultiply(a1, b1);
```

```
    vll a2b2 = karatsubaMultiply(a2, b2);

    for (int i = 0; i < k; i++)
        a2[i] += a1[i];
    for (int i = 0; i < k; i++)
        b2[i] += b1[i];

    vll r = karatsubaMultiply(a2, b2);
    for (int i = 0; i < (int) a1b1.size(); i++)
        r[i] -= a1b1[i];
    for (int i = 0; i < (int) a2b2.size(); i++)
        r[i] -= a2b2[i];

    for (int i = 0; i < (int) r.size(); i++)
        res[i + k] += r[i];
    for (int i = 0; i < (int) a1b1.size(); i++)
        res[i] += a1b1[i];
    for (int i = 0; i < (int) a2b2.size(); i++)
        res[i + n] += a2b2[i];
    return res;
    }

    bigint operator*(const bigint &v) const {
    vector<int> a6 = convert_base(this->a, base_digits, 6);
    vector<int> b6 = convert_base(v.a, base_digits, 6);
    vll a(a6.begin(), a6.end());
    vll b(b6.begin(), b6.end());
    while (a.size() < b.size())
        a.push_back(0);
    while (b.size() < a.size())
        b.push_back(0);
    while (a.size() & (a.size() - 1))
        a.push_back(0), b.push_back(0);
    vll c = karatsubaMultiply(a, b);
    bigint res;
    res.sign = sign * v.sign;
    for (int i = 0, carry = 0; i < (int) c.size(); i++) {
        long long cur = c[i] + carry;
        res.a.push_back((int) (cur % 1000000));
        carry = (int) (cur / 1000000);
    }
    res.a = convert_base(res.a, 6, base_digits);
    res.trim();
    return res;
    }
};
/*
  ########################   THE   BIG   INT   ############################
  ########################################################################
*/
```