**Binar daraja nima?**

Ma’no jihatidan bu **darajaga oshirishni O(log n)** vaqtda hisoblaydigan usul. Oddiy (“naive”) usul a^n ni n marta ko‘paytiradi — O(n). Binar daraja esa ko‘paytirishlar sonini keskin kamaytiradi: n ning ikkilik ko‘rinishidagi bitlarini ketma-ket ko‘rib chiqadi va har qadamda:

* agar bit 1 bo‘lsa — natijani hozirgi “asos” (base) ga ko‘paytiradi,
* keyin asosni kvadratlaydi (a = a \* a),
* darajani bitta bit o‘ngga suradi (n >>= 1).

**Intuitiv misol (3¹³)**

13 = 1101₂. Algoritm:

* start: res = 1, a = 3, n = 13
* bit1 (1): res = res \* a = 3, a = 9, n = 6
* bit2 (0): res = 3, a = 81, n = 3
* bit3 (1): res = 3\*81 = 243, a = 6561, n = 1
* bit4 (1): res = 243\*6561 = 1 594 323, n = 0 → tugadi.

**Iterativ C++ (butun sonlar, overflow nazoratisiz)**

long long **binpow\_ll**(long long a, long long n) {

    long long res = 1;

    while (n > 0) {

        if (n & 1) res = res \* a; *// bit = 1 bo'lsa, ko'paytiramiz*

        a = a \* a; *// asosni kvadratlaymiz*

        n >>= 1; *// bitta bit o'ngga suramiz*

    }

    return res;

}

**Eslatma**: katta qiymatlarda res \* a va a \* a overflow beradi.

**Modul bo‘yicha daraja (eng amaliy ko‘rinish)**

Ko‘pincha a^n mod m kerak bo‘ladi (kriptografiya, kombinatorika, sonlar nazariyasi). Bu yerda har bir ko‘paytirishdan keyin mod olish kerak. Katta oraliq ko‘paytmalarda ham overflow bo‘lmasligi uchun \_\_int128 dan foydalanish qulay.

#include <cstdint>

long long **modpow**(long long a, long long n, long long mod) {

    a %= mod;

    long long res = 1 % mod;

    while (n > 0) {

        if (n & 1) {

            \_\_int128 t = (\_\_int128)res \* a;

            res = (long long)(t % mod);

        }

        \_\_int128 sq = (\_\_int128)a \* a;

        a = (long long)(sq % mod);

        n >>= 1;

    }

    return res;

}

**Qisqa sinov**

// 3^13 = 1594323

cout << binpow\_ll(3, 13) << "\n"; // 1594323

cout << modpow(3, 13, 100) << "\n"; // 23

cout << modpow(2, 1000000000000000000LL, 1000000007) << "\n"; // tez va xavfsiz

**Rekursiv yozuv (o‘qish oson varianti)**

long long binpow\_rec(long long a, long long n) {

if (n == 0) return 1;

long long half = binpow\_rec(a, n / 2);

long long sq = half \* half; // ehtiyot: overflow mumkin

if (n % 2 == 0) return sq;

return sq \* a;

}

Iterativ usul O(1) xotira ishlatadi, rekursiv usul esa chaqiruv chuqurligi ~O(log n).

**Manfiy darajalar va suzuvchi nuqta (double)**

Butun sonlar uchun a^(-n) odatda aniqlanmaydi (kasr bo‘lib ketadi). Real sonlarda esa a^-n = 1 / a^n:

double binpow\_double(double a, long long n) {

bool neg = n < 0;

if (neg) n = -n;

double res = 1.0;

while (n > 0) {

if (n & 1) res \*= a;

a \*= a;

n >>= 1;

}

return neg ? 1.0 / res : res;

}

**Tez-tez uchraydigan xatolar va maslahatlar**

* **Overflow**: a\*a va res\*a 64-bitdan oshishi mumkin. Modli hisobda \_\_int128 yordam beradi. Juda katta modul (≥1e18) uchun ham ehtiyot bo‘ling; kerak bo‘lsa Karatsuba/FFT yoki ko‘paytmani modda xavfsiz olishning boshqa usullari qo‘llanadi.
* **0^0**: matematikada noaniq; ko‘pincha algoritmlarda n == 0 bo‘lsa 1 qaytariladi (masalan, bo‘sh ko‘paytma ta’rifi).
* **Manfiy a va mod**: a %= mod dan keyin a manfiy bo‘lib qolishi mumkin; if (a < 0) a += mod; bilan to‘g‘rilang.
* **Fermat inver­si**: mod tub bo‘lsa va gcd(a, mod)=1 bo‘lsa, a^(-1) ≡ a^(mod-2) (mod mod):
* long long modinv(long long a, long long mod) {
* return modpow(a, mod - 2, mod); // mod – tub bo'lsa
* }

**Qisqa yakun**

* Vaqt murakkabligi: **O(log n)** ko‘paytirish.
* Xotira: iterativ — **O(1)**.
* Amaliyoti: sonlar, polinomlar, matritsalar (masalan, Fibonachchini 2×2 matritsa bilan O(log n) da) va kriptografiyada juda ko‘p qo‘llanadi.

Agar xohlasangiz, shu kodlarni bitta main() bilan to‘liq sinov dasturi ko‘rinishida ham berib qo‘yaman.

Zo‘r savol 👍. Bu juda ko‘p boshlang‘ich va o‘rta darajadagi dasturchilarga qiziq tuyuladi. Batafsil tushuntirib beraman:

## 1. Nega natijalarni tub songa (odatda ****1e9+7****, ****998244353**** va hokazo) bo‘lib chiqarishadi?

Dasturlashda ko‘p algoritmlar katta sonlarni hisoblaydi: kombinatsiyalar, darajalar, yo‘llar soni va hokazo. Bu sonlar tezda **64-bit integer** (long long) sig‘imidan chiqib ketadi. Shuning uchun odatda javobni biror **mod m** da hisoblaymiz. Endi savol: nima uchun aynan **tub son** tanlanadi?

### Asosiy sabablar:

1. **Matematik qulaylik (invers mavjudligi)**
   * Agar modul **tub son** bo‘lsa, a ≠ 0 (mod p) uchun modul bo‘yicha teskari son (invers) har doim mavjud.
   * Fermatning kichik teoremasi ishlaydi:

ap−1≡1 (mod p)a^{p-1} \equiv 1 \ (\text{mod } p)

va

a−1≡ap−2 (mod p).a^{-1} \equiv a^{p-2} \ (\text{mod } p).

* + Masalan, kombinatorikada (nk)=n!k!(n−k)!\binom{n}{k} = \frac{n!}{k!(n-k)!}. Buni modda hisoblash uchun bo‘lish kerak. Oddiy modda bo‘lish yo‘q, lekin tub modulda invers yordamida bemalol ishlaymiz.

1. **Matritsa va polinom algebra sodda ishlaydi**  
   Tub modul bo‘lsa, “ring” emas, balki “field” hosil bo‘ladi → ko‘plab algebraik teoremalar to‘liq ishlaydi (masalan, polinomlarni unikal faktorlanishi, Gauss eliminatsiyasi).
2. **Yaxshi chegaralar va mos keladigan son**
   * 1e9+7 (1000000007) — tub va 32-bitdan katta emas, shuning uchun long longda ko‘paytirish paytida overflow bo‘lmaydi (\_\_int128 ishlatmasdan ham ko‘p hollarda yetadi).
   * 998244353 ham tub va juda “maxsus”: bu 119⋅223+1119 \cdot 2^{23} + 1. Bu ko‘rinish NTT (Fast Fourier Transform modda) uchun juda qulay.

## 2. Agar modul ****murakkab son**** bo‘lsa (tub bo‘lmasa) qanday kamchiliklar bor?

Agar modul tub bo‘lmasa, quyidagi muammolar yuzaga keladi:

1. **Invers har doim mavjud emas**
   * Masalan, mod m=12m = 12.  
     8 ning teskari sonini topmoqchi bo‘lsak: 8x≡1(mod12)8x \equiv 1 (mod 12).  
     Hech qanday x chiqmaydi, chunki gcd(8,12) ≠ 1.
   * Demak, ko‘p hollarda bo‘lish operatsiyasini qilolmaymiz.
2. **Fermat teoremasi ishlamaydi**  
   Tub bo‘lmagan modulda am−1≡1a^{m-1} \equiv 1 majburiy emas. Euler teoremasi bor, lekin u murakkabroq va ko‘p hollarda algoritmlarda ishlash qiyinlashadi.
3. **Kombinatorika formulasida xatoliklar**  
   (nk)\binom{n}{k} ni hisoblash uchun faktoriallarning inverslari kerak bo‘ladi. Agar modul tub bo‘lmasa, ba’zi sonlarning inversi yo‘q → natija yo‘q yoki noto‘g‘ri chiqadi.
4. **Amaliy tajriba**  
   Dasturlash musobaqalari va kutubxonalar odatda tub modlarga asoslanadi. Shuning uchun tubdan boshqa mod ishlatish — odatiy emas, ko‘plab tayyor algoritmlar ishlamay qoladi.

## 3. Xulosa

* **Tub modul**: teskari element doim bor (0 dan boshqa barcha sonlar uchun), Fermat teoremasi ishlaydi, kombinatorika, kriptografiya va algebraik amallar to‘g‘ri ketadi.
* **Tub bo‘lmagan modul**: bo‘lish har doim ishlamaydi, ko‘plab teorema va algoritmlar yaroqsiz bo‘lib qoladi.

Shu sababli odatda **1e9+7** yoki **998244353** kabi tub modlar tanlanadi.

Xohlaysizmi, men sizga **mod 12** kabi tub bo‘lmagan modda kombinatsiyalarni hisoblashda qanday muammo chiqishini **C++ kod** bilan ko‘rsatib beray?