ACM-ICPC Cheat Sheet

```
Orange Juice 情報
```

Table of Contents *generated with DocToc*

- 6. Number + Mathematics
  - 6.1 BigInteger + BigDecimal
    - 6.1.1 C++ Big Integer
    - 6.1.2 The Java Approach
  - 6.2 Matrix
  - 6.3 Number Theory
    - 6.3.1 欧拉函数？
    - 6.3.2 欧几里得算法 / gcd
    - 6.3.3 扩展欧几里得算法
    - 6.3.4 求解不定方程
    - 6.3.5 求解模线性方程（线性同余方程）
    - 6.3.6 求解模的逆元
    - 6.3.7 中国剩余定理
    - 6.3.8 最小公倍数
    - 6.3.9 分解质因数
    - 6.3.10 因数个数
    - 6.3.11 素数判定
      - 6.3.11.1 Miller Rabin Primality Test
    - 6.3.12 进制转换
    - 6.3.13 A / C
    - 6.3.14 质数表
    - 6.3.15 Fast Exponention
    - 6.3.16 Fast Fourier Transform FFT
  - 6.4 Game Theory 博弈论
    - 6.4.1 Impartial Combinatorial Game
      - 6.4.1.1 Nim Game
      - 6.4.1.1 Composite Games – Sprague-Grundy Theorem and Nim Value
- 8. Tricks + Miscellaneous
  - 8.1 Bit Manipulation
  - 8.1 Cantor Expansion / Reverse Cantor Expansion
  - 8.2 pass 2-D array
  - 8.3 Binary Display
  - 8.4 Fast Log
  - 8.5 Squre Root

6. Number + Mathematics

6.1 BigInteger + BigDecimal

6.1.1 C++ Big Integer

```
const int BASE_LENGTH = 2;
const int BASE = (int) pow(10, BASE_LENGTH);
const int MAX_LENGTH = 500;

string int_to_string(int i, int width, bool zero) {
    string res = "";
    while (width--) {
        if (!zero && i == 0) return res;
        res = (char)(i%10 + '0') + res;
        i /= 10;
    }
    return res;
}

struct bigint {
    int len, s[MAX_LENGTH];

    bigint() {
        memset(s, 0, sizeof(s));
        len = 1;
    }

    bigint(unsigned long long num) {
        len = 0;
        while (num >= BASE) {
            s[len] = num % BASE;
            num /= BASE;
            len ++;
        }
        s[len++] = num;
    }

    bigint(const char* num) {
        int l = strlen(num);
        len = l/BASE_LENGTH;
        if (l % BASE_LENGTH) len++;
        int index = 0;
        for (int i = l - 1; i >= 0; i -= BASE_LENGTH) {
            int tmp = 0;
            int k = i - BASE_LENGTH + 1;
            if (k < 0) k = 0;
            for (int j = k; j <= i; j++) {
                tmp = tmp*10 + num[j] - '0';
            }
            s[index++] = tmp;
        }
    }
```

```
    void clean() {
        while(len > 1 && !s[len-1]) len--;
    }

    string str() const {
        string ret = "";
        if (len == 1 && !s[0]) return "0";
        for(int i = 0; i < len; i++) {
            if (i == 0) {
                ret += int_to_string(s[len - i - 1], BASE_LENGTH, false);
            } else {
                ret += int_to_string(s[len - i - 1], BASE_LENGTH, true);
            }
        }
        return ret;
    }

    unsigned long long ll() const {
        unsigned long long ret = 0;
        for(int i = len-1; i >= 0; i--) {
            ret *= BASE;
            ret += s[i];
        }
        return ret;
    }

    bigint operator + (const bigint& b) const {
        bigint c = b;
        while (c.len < len) c.s[c.len++] = 0;
        c.s[c.len++] = 0;
        bool r = 0;
        for (int i = 0; i < len || r; i++) {
            c.s[i] += (i<len)*s[i] + r;
            r = c.s[i] >= BASE;
            if (r) c.s[i] -= BASE;
        }
        c.clean();
        return c;
    }

    bigint operator - (const bigint& b) const {
        if (operator < (b)) throw "cannot do subtract";
        bigint c = *this;
        bool r = 0;
        for (int i = 0; i < b.len || r; i++) {
            c.s[i] -= b.s[i];
            r = c.s[i] < 0;
            if (r) c.s[i] += BASE;
        }
        c.clean();
        return c;
    }

    bigint operator * (const bigint& b) const {
```

```
        bigint c;
        c.len = len + b.len;
        for(int i = 0; i < len; i++)
            for(int j = 0; j < b.len; j++)
                c.s[i+j] += s[i] * b.s[j];
        for(int i = 0; i < c.len-1; i++){
            c.s[i+1] += c.s[i] / BASE;
            c.s[i] %= BASE;
        }
        c.clean();
        return c;
    }

    bigint operator / (const int b) const {
        bigint ret;
        int down = 0;
        for (int i = len - 1; i >= 0; i--) {
            ret.s[i] = (s[i] + down * BASE) / b;
            down = s[i] + down * BASE - ret.s[i] * b;
        }
        ret.len = len;
        ret.clean();
        return ret;
    }

    bool operator < (const bigint& b) const {
        if (len < b.len) return true;
        else if (len > b.len) return false;
        for (int i = 0; i < len; i++)
            if (s[i] < b.s[i]) return true;
            else if (s[i] > b.s[i]) return false;
        return false;
    }

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

    bool operator > (const bigint& b) const {
        return b < *this;
    }
};
```

Examples

```
ASSERT((a+b).str()=="10001")
ASSERT((a+b)==bigint(10001))
ASSERT((b/2)==4999)
ASSERT(c == 12345)
ASSERT(c < 123456)
ASSERT(c > 123)
ASSERT(!(c > 123456))
```

```
ASSERT(!(c < 123))
ASSERT(!(c == 12346))
ASSERT(!(c == 12344))
ASSERT(c.str() == "12345")
ASSERT((b-1)==9998)
ASSERT(a.ll() == 2)
ASSERT(b.ll() == 9999)
```

6.1.2 The Java Approach

BigInteger & BigDecimal

6.2 Matrix

```
operator+
operator*
```

> Square matrix

```
struct Matrix {
    // int height;
    // int width;

    long long value[32][32];

    Matrix operator* (const Matrix& that);
    Matrix operator+ (const Matrix& that);
    Matrix mirror();
    void show() {
        cout << endl;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++)
                cout << this->value[i][j] << " ";
            cout << endl;
        }
    }
};

void mod_it(Matrix& temp) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            temp.value[i][j] %= m;
}

Matrix Matrix::operator* (const Matrix& that) {
    Matrix temp;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
```

```
                temp.value[i][j] = 0;
                for (int k = 0; k < n; k++)
                    temp.value[i][j] += this->value[i][k] * that.value[k][j];
            }
        }
        mod_it(temp);
        return temp;
    }

    Matrix Matrix::operator+ (const Matrix& that) {
        Matrix temp;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                temp.value[i][j] = this->value[i][j] + that.value[i][j];
        mod_it(temp);
        return temp;
    }

    Matrix Matrix::mirror() {
        Matrix temp;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                temp.value[i][j] = this->value[i][j];
        return temp;
    }
```

6.3 Number Theory

6.3.1 欧拉函数？

6.3.2 欧几里得算法 / gcd

```
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}

int lcm(int a, int b) {
   if(gcd(a,b)>0){
    return (a / gcd(a, b)) * b;
   }
   return 0;

}
```

6.3.3 扩展欧几里得算法

http://www.cnblogs.com/frog112111/archive/2012/08/19/2646012.html

> 对于不完全为 0 的非负整数 a, b, 必然存在整数对 (x, y), 使得 gcd(a, b) = ax + by

> suppose: a > b, we want to get (x1, y1)

> (i) if b == 0, then gcd(a, b) = a = ax + 0, then x1 = 1, y1 = 0

> (ii) if b != 0:

> (1): a _ x1 + b _ y1 = gcd(a, b)

> (2): b _ x2 + (a % b) _ y2 = gcd(b, a % b)

> (1) == (2)

> so: a _ x1 + b _ y1 = b _ x2 + (a % b) _ y2

> so: a _ x1 + b _ y1 = b _ x2 + (a - (int)(a / b) _ b) * y2

> so: a _ x1 + b _ y1 = a _ y2 + b _ (x2 - (int)(a / b) * y2)

> so: x1 = y2, y1 = x2 - (int)(a / b) * y2, can get (x1, y1) from (x2, y2)

> next:

```
(1): b * x2 + (a % b) * y2 = gcd(b, a % b)
```

```
(2): (a % b) * x3 + b % (a % b) * y3 = gcd(a % b, b % (a % b))
```

```
so: can get (x2, y2) from (x3, y3)
```

```
next: ... until in gcd(a, b), b == 0, then xi = 1, yi = 0, go back
...
```

```cpp
long long ansx, ansy, ansd;

void euclidean(long long a, long long b) {
    if (b == 0) {
        ansx = 1;
        ansy = 0;
        ansd = a;
    } else {
        euclidean(b, a % b);
        long long temp = ansx;
        ansx = ansy;
        ansy = temp - a / b * ansy;
    }
```

```
}

int main(int argc, char const *argv[]) {
    long long a, b, c;
    cin >> a >> b >> c;

    ansx = 0;
    ansy = 0;
    ansd = 0;
    euclidean(a, b);

    // now (ansx, ansy) is the answer (x, y) for a * x1 + b * y1 = gcd(a,
b)
    // ansd is the a when b == 0, which is just gcd(a, b)
}
```

6.3.4 求解不定方程

> for: p _ a + q _ b = c

> if c % gcd(a, b) == 0, then 有整数解 (p, q), else NO

> if we get (p0, q0) for p0 _ a + q0 _ b = gcd(a, b)

> then: for p _ a + q _ b = gcd(a, b) (k is any integer)

> p = p0 + b / gcd(a, b) * k

> q = q0 - a / gcd(a, b) * k

> then: for p _ a + q _ b = c = c / gcd(a, b) * gcd(a, b) (k is any integer)

> p = (p0 + b / gcd(a, b) _ k) _ c / gcd(a, b)

> q = (q0 - a / gcd(a, b) _ k) _ c / gcd(a, b)

```
// after get ansx, ansy, ansd
// test if c % ansd == 0
// ansx = (ansx + b / gcd(a, b) * k) * c / gcd(a, b)
// ansy = (ansy - a / gcd(a, b) * k) * c / gcd(a, b)
// smallest: ansx % (b / gcd(a, b) + b / gcd(a, b)) % (b / gcd(a, b))
```

6.3.5 求解模线性方程（线性同余方程）

> (a * x) % n = b % n, x = ?

> same as: a _ x + n _ y= b

> so: one answer for a _ x + n _ y= b is: x * b / gcd(a, n)

> so: one answer for (a _ x) % n = b % n is: x0 = (x _ b / gcd(a, n)) % n

> other answer xi = (x0 + i * (n / gcd(a, n))) % n, i = 0...gcd(a, n)-1

> smallest answer is x0 % (n / gcd(a, n) + gcd(a, n)) % gcd(a, n)

```

```

6.3.6 求解模的逆元

> (a * x) % n = 1, x = ?

> if gcd(a, n) != 1, then NO answer

> else:

> same as: a _ x + n _ y = 1

> can get only one answer x

```
// after get ansx, ansy, ansd
// if ansd != 1, then NO answer
// smallest ansx = (ansx % (n / gcd(a, n)) + (n / gcd(a, n))) % (n /
gcd(a, n))
```

6.3.7 中国剩余定理

6.3.8 最小公倍数

```
a / gcd(a, b) * b
```

6.3.9 分解质因数

```
long long x;
cin >> x;
for (long long factor = 2; x != 1; factor++) {
    if (x % factor == 0)
        cout << factor << " is a prime factor" << endl;
    while (x % factor == 0)
        x = x / factor;
}
```

6.3.10 因数个数

```

```

```
n = p1 ^ x1 * p2 ^ x2 * ... * pn ^ xn
total = (x1 + 1) * (x2 + 1) * ... * (xn + 1)
```

6.3.11 素数判定

> 大于 3 的质数可以被表示为 6n - 1 或 6n + 1

```
// A prime number greater than 3 can be written in the form 6n − 1 or 6n +
1
// This is of the order O(sqrt(n))  with reduced leading constant
bool is_prime(int n) {
    if (n == 1 || n % 2 == 0){
        return false;
    }

    if ((n == 2)||(n==3)){
        return true;
    }
    int t = sqrt(n);
    int k = t / 6;
    for(int i = 1; i <=k; i++){
        if((n%(6*t − 1)==0)||(n%(6*t + 1)==0)){
        return false;
    }
    }
    return true;
}
```

6.3.11.1 Miller Rabin Primality Test

> O(k(logN)^3)

```
class MillerRabin { // O(k(logX)^3)
    long long mulmod(long long a, long long b, long long c) {
        if (a < b)
            swap(a, b);
        long long res = 0, x = a;
        while (b > 0) {
            if (b & 1) {
                res = res + x;
                if (res >= c)
                    res −= c;
            }
            x = x * 2;
            if (x >= c)
                x −= c;
            b >>= 1;
        }
```

```
        return res % c;
    }

    long long bigmod(long long a, long long p, long long mod) {
        long long x = a, res = 1;
        while (p) {
            if (p & 1)
                res = mulmod(res, x, mod);
            x = mulmod(x, x, mod);
            p >>= 1;
        }
        return res;
    }

    bool witness(long long a, long long d, long long s, long long n) {
        long long r = bigmod(a, d, n);
        if (r == 1 || r == n - 1)
            return false;
        for (int i = 0; i < s - 1; i++) {
            r = mulmod(r, r, n);
            if (r == 1)
                return true;
            if (r == n - 1)
                return false;
        }
        return true;
    }

public:
    const vector<long long> aaa{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
37}; // enough for N < 2^64

    bool test(long long n) {
        if (n <= 1)
            return false;

        long long p = n - 1;
        long long s = 0;
        while (!(p & 1)) {
            p /= 2;
            s++;
        }

        for (int i = 0; i < aaa.size() && aaa[i] < n; i++)
            if (witness(aaa[i], p, s, n))
                return false;
        return true;
    }
};
```

6.3.12.1 进制转换

```
void convert_dec_to_base(int n, const int base) {
    if (n == 0)
        cout << 0 << endl;
    while (n != 0) {
        int e = n % base;
        cout << e << " "; // printing in reverse order
        n /= base;
    } cout << endl;
}

int convert_base_to_dec(const int s[], const int len, const int base) {
    int result = 0;
    for (int i = 0; i < len; i++)
        result = result * base + s[i];
    return result;
}
```

6.3.12.2

6.3.13 A / C

> C(n, k) = C(n-1, k) + C(n-1, k-1)
>
> C(n, k) = C(n, n-k)

```
#define MAXN 1002
#define MOD 1000000007
long long choose[MAXN][MAXN];

void init_choose_n_k() {
    for (int i = 1; i < MAXN; i++) {
        choose[i][i] = choose[i][0] = 1;
        for (int j = 1; j < i; j++)
            choose[i][j] = (choose[i-1][j-1] + choose[i-1][j]) % MOD;
    }
}
```

6.3.14 质数表

```
int is_prime[UP_LIMIT + 1];
for (int i = 1; i <= UP_LIMIT; i++) // init to 1
    is_prime[i] = 1;
for (int i = 4; i <= UP_LIMIT; i += 2) // even number is not
    is_prime[i] = 0;
for (int k = 3; k*k <= UP_LIMIT; k++) // start from 9, end at sqrt
    if (is_prime[k])
        for(int i = k*k; i <= UP_LIMIT; i += 2*k) // every two is not
            is_prime[i] = 0;
```

6.3.15 Fast Exponention

> To calculate n ^ p % M

```
int power_modulo(int n, int p, int M) {
    int result = 1;
    while (p > 0) {
        if (p % 2 == 1)
            result = (result*n) % M;
        p /= 2;
        n = (n*n) % M;
    }
    return result;
}
```

6.3.16 Fast Fourier Transform FFT

> Reference 1
>
> Reference 2
>
> Example: calculate two number C = A * B

```
#include <iostream>
#include <vector>
#include <complex>

#ifndef M_PI
const double M_PI = acos(-1);
#endif

using namespace std;

typedef complex<long double> Complex;

void FFT(vector<Complex> &A, int s) {
    int n = A.size();
    int p = __builtin_ctz(n);

    vector<Complex> a = A;

    for (int i = 0; i < n; ++i) {
        int rev = 0;
        for (int j = 0; j < p; ++j) {
            rev <<= 1;
            rev |= ((i >> j) & 1);
        }
        A[i] = a[rev];
```

```
    }

    Complex w, wn;


    for (int i = 1; i <= p; ++i) {
        int M = (1<<i), K = (M>>1);
        wn = Complex(cos(s*2.0*M_PI/(double)M),
sin(s*2.0*M_PI/(double)M));

        for (int j = 0; j < n; j += M) {
            w = Complex(1.0, 0.0);
            for (int l = j; l < K + j; ++l) {
                Complex t = w;
                t *= A[l + K];
                Complex u = A[l];
                A[l] += t;
                u -= t;
                A[l + K] = u;
                w *= wn;
            }
        }
    }

    if (s == -1) {
        for (int i = 0; i < n; ++i)
            A[i] /= (double)n;
    }
}

vector<Complex> FFT_Multiply(vector<Complex> &P, vector<Complex> &Q) {
    int n = P.size() + Q.size();

#define lowbit(x) (((x) ^ (x-1)) & (x))
    while (n != lowbit(n)) n += lowbit(n);
#undef lowbit

    P.resize(n, 0);
    Q.resize(n, 0);

    FFT(P, 1);
    FFT(Q, 1);

    vector<Complex> R;
    for (int i = 0; i < n; i++) R.push_back(P[i] * Q[i]);

    FFT(R, -1);

    return R;
}

int main() { // multiply two number
    string sa; // [a1 * 10^(?)] + [a2 * 10^(?)] + ... + [an * 10^(?)]
    string sb; // [b1 * 10^(?)] + [b2 * 10^(?)] + ... + [bm * 10^(?)]
```

```
    while (cin >> sa >> sb) {
        vector<Complex> a;
        vector<Complex> b;

        for (int i = 0; i < sa.length(); i++)
            a.push_back(Complex(sa[sa.length() - i - 1] - '0', 0)); // add
[a_ * (x)^(?)]
        for (int i = 0; i < sb.length(); i++)
            b.push_back(Complex(sb[sb.length() - i - 1] - '0', 0)); // add
[b_ * (x)^(?)]

        // [c1 * 10^(?)] + [c2 * 10^(?)] + ... + [cn * 10^(?)]
        // =
        // [a1 * 10^(?)] + [a2 * 10^(?)] + ... + [an * 10^(?)]
        // *
        // [b1 * 10^(?)] + [b2 * 10^(?)] + ... + [bm * 10^(?)]
        vector<Complex> c = FFT_Multiply(a, b);

        vector<int> ans(c.size());
        for (int i = 0; i < c.size(); i++)
            ans[i] = c[i].real() + 0.5; // extract [c_ * (x)^(?)] //
equivalent to [c_ * (x=10)^(?)]
        for (int i = 0; i < ans.size() - 1; i++) { // process carry
            ans[i + 1] += ans[i] / 10;
            ans[i] %= 10;
        }

        bool flag = false;
        for (int i = ans.size() - 1; i >= 0; i--) { // print from most
significant digit
            if (ans[i])
                cout << ans[i], flag = true;
            else if (flag || i == 0)
                cout << 0;
        }
        cout << endl;
    }
}
```

6.4 Game Theory 博弈论

6.4.1 Impartial Combinatorial Game

> In combinatorial game theory, an impartial game is a game in which the allowable moves depend only on the position and not on which of the two players is currently moving, and where the payoffs are symmetric. In other words, the only difference between player 1 and player 2 is that player 1 goes first.

> Impartial games can be analyzed using the Sprague–Grundy theorem.

> Impartial games include Nim, Sprouts, Kayles, Quarto, Cram, Chomp, and poset games. Go and chess are not impartial, as each player can only place or move pieces of their own color. Games like

> ZÈRTZ and Chameleon are also not impartial, since although they are played with shared pieces, the payoffs are not necessarily symmetric for any given position.

> A game that is not impartial is called a partisan game.
>
> source: wiki
>
> tutorial: topcoder

6.4.1.1 Nim Game

> One good choice: brute force to find some pattern.
>
> We will not be able to play many of the games without decomposing them to smaller parts (sub-games), pre-computing some values for them, and then obtaining the result by combining these values.
>
> Positions have the following properties:
>
> - All terminal positions are losing.
> - If a player is able to move to a losing position then he is in a winning position.
> - If a player is able to move only to the winning positions then he is in a losing position.
>
> Rules of the Game of Nim: There are n piles of coins. When it is a player's turn he chooses one pile and takes at least one coin from it. If someone is unable to move he loses (so the one who removes the last coin is the winner).
>
> Let n1, n2, ..., nk, be the sizes of the piles. It is a losing position for the player whose turn it is if and only if n1 xor n2 xor ... xor nk = 0.

6.4.1.1 Composite Games – Sprague-Grundy Theorem and Nim Value

> - Break composite game into subgames
> - Assign grundy number to every subgame according to which size of the pile in the Game of Nim it is equivalent to.
> - Now we have some subgames (piles), each subgame has its grundy number (size of pile)
> - xor all subgames

> - Losing position of subgame has grundy number = 0.
> - A position has grundy number = smallest number among its next positions don't have.

> If the table of grundy number is too large, we can precompute and find the pattern.

```
// hihocoer 1173
const int MAXSTATE = 2e4 + 2;

bool appear[MAXSTATE];
int sg[MAXSTATE]; // Sprague-Grundy // Nim Value
void init_sg() {
    sg[state] = 0;
    for (int state = 1; state < MAXSTATE; state++) { // sg(x) = mex{sg(y)
```

```
| y是x的后继局面} // mex{a[i]}表示a中未出现的最小非负整数
        fill_n(appear, MAXSTATE, false);
        for (int nx = 0; nx < state; nx++)
            appear[sg[nx]] = true;
        int pile_a = 1;
        int pile_b = state - pile_a;
        while (pile_a <= pile_b) {
            appear[sg[pile_a] ^ sg[pile_b]] = true;
            pile_a++;
            pile_b--;
        }
        while (appear[sg[state]])
            sg[state]++;
    }
}

int main() {

    init_sg();

    // --- start of finding pattern ---
    //
    // --- end of finding pattern ---

    int n;
    cin >> n;
    int result = 0;
    for (int i = 0; i < n; i++) {
        int a;
        cin >> a;

        // by grundy number
        // result ^= sg[a];

        // by pattern
        // if (a % 4 == 0)
        //  a--;
        // else if (a % 4 == 3)
        //  a++;
        // result ^= a;
    }
    if (result)
        cout << "Alice" << endl;
    else
        cout << "Bob" << endl;
}
```

8. Tricks + Miscellaneous

8.1 Bit Manipulation

```
#define GET_BIT(n, i) (((n) & (1LL << ((i)-1))) >> ((i)-1)) // i start
from 1
#define SET_BIT(n, i) ((n) | (1LL << ((i)-1)))
#define CLR_BIT(n, i) ((n) & ~(1LL << ((i)-1)))
```

8.1 Cantor Expansion / Reverse Cantor Expansion

> for hashing, or ...

```
long long factorial(int n) {
    if (n == 0)
        return 1;

    long long ans = n;
    for (int i = 1; i < n; i++)
        ans = ans * i;
    return ans;
}

long long cantor_expansion(int permutation[], int n) {
    // input: (m-th permutation of n numbers, n)
    // return: m
    int used[n + 1];
    memset(used, n, sizeof(used));

    long long ans = 0;
    for (int i = 0; i < n; i++) {
        int temp = 0;
        used[permutation[i]] = 1;
        for (int j = 1; j < permutation[i]; j++)
            if (used[j] != 1)
                temp += 1;
        ans += factorial(n - 1 - i) * temp;
    }

    return ans + 1;
}

void reverse_cantor_expansion(int n, long long m) {
    // m-th permutation of n numbers
    int ans[n + 1], used[n + 1];
    memset(ans, -1, sizeof (ans));
    memset(used, 0, sizeof (used));

    m = m - 1;
    for (int i = n - 1; i >= 0; i--) {
        long long fac = factorial(i);
        int temp = m / fac + 1;
        m = m - (temp - 1) * fac;
        for (int j = 1; j <= temp; j++)
            if (used[j] == 1)
```

```
                temp++;

            ans[n - i] = temp;
            used[temp] = 1;
        }

        for (int i = 1; i < n + 1; i++)
            cout << ans[i] << " ";
        cout << "\n";
    }
```

8.2 pass 2-D array

```
// The parameter is a 2D array
int array[10][10];
void passFunc(int a[][10]) {
    // ...
}
passFunc(array);

// The parameter is an array containing pointers
int *array[10];
for(int i = 0; i < 10; i++)
    array[i] = new int[10];
void passFunc(int *a[10]) {
    // ...
}
passFunc(array);

// The parameter is a pointer to a pointer
int **array;
array = new int *[10];
for(int i = 0; i <10; i++)
    array[i] = new int[10];
void passFunc(int **a) {
    // ...
}
passFunc(array);
```

8.3 Binary Display

```
#include <bitset>
void show_binary(unsigned long long x) {
    printf("%s\n", bitset<64>(x).to_string().c_str());
}
```

8.4 Fast Log

> Built-in `log(double)` is not accurate for integer.
>
> Should `(int)(log(double)+0.000....001)`

```
int fastlog(unsigned long long x, unsigned long long base) {
    // ERROR VALUE IF X == BASE == ULLONG_MAX

    const unsigned long long HALF = 1ULL << 32;
    unsigned long long cache[7];
#define INIT(i) { cache[i] = base; if (base < HALF) base *= base; else
base = ULLONG_MAX; }
    INIT(0); INIT(1); INIT(2); INIT(3); INIT(4); INIT(5); INIT(6);
#undef INIT

    int ret = -(x == 0);
#define S(i, k) if (x >= cache[i]) ret += k, x /= cache[i]; else return
ret;
    S(6, 64); S(5, 32); S(4, 16); S(3, 8); S(2, 4); S(1, 2); S(0, 1);
#undef S
}
```

8.5 Squre Root

```
long long sq(long long a) {
    long long l = 1;
    long long r = a + 1;
    while (l + 1 < r) {
        long long m = (l + r) / 2;
        if (a / m < m)
            r = m;
        else
            l = m;
    }
    return l;
}
```