Math

Number Theory

GCD: $O(lg(n))$

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

Extended GCD for Bezout coefficients $a \cdot x + b \cdot y = GCD(a, b)$

```
int gcdE(int a, int b, int &x, int &y)
{
    if (b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }
    int ans = gcdE(b, a % b, y, x);
    y -= x * (a / b);
    return ans;
}
```

Solve $a \cdot x + b \cdot y = c$
$x = x_0 \cdot c/GCD(a, b)$, $y = y_0 \cdot c/GCD(a, b)$, where x_0, y_0 are Bezout coefficients.
$(x + k \cdot b', y + k \cdot a')$, where $a' = a/GCD(a, b)$, $b' = b/GCD(a, b)$

LCM

```
int lcm(int a, int b)
{
    return a / gcd(a, b) * b;
}
```

Eratosthenes: $O(\frac{n}{ln(n)})$

```
int n;
vector<bool> prime(n + 1, true);
for (int i = 2; i <= sqrt(n + 0.5); ++i)
    if (prime[i])
        for (int j = i * i; j <= n; j += i)
            prime[j] = false;
```

Modulo of a big integer

```
int m;
string bigint;
long long ans = 0;
for (char c : bigint)
    ans = (ans * 10 + c - '0') % m;
```

Modulo of power (divide and conquer)

```
1  int pow_mod(int a, int n, int m)
2  {
3      if (n == 0)
4          return 1;
5      long long ans = pow_mod(a, n / 2, m);
6      ans = ans * ans % m;
7      if (n % 2)
8          ans = ans * a % m;
9      return ans;
10 }
```

Solve $a \cdot x \equiv b (mod\ n)$
$\iff a \cdot x - n \cdot k = b$

Computational Geometry

Co-linear: cross = 0
Parallel: dot = 0
Line: $P_0 + t \cdot v$ or $P_A + t \cdot (P_B - P_A)$

```
#include <bits/stdc++.h>
using namespace std;

struct Vec
{
    double x, y;
    Vec(double x = 0, double y = 0) : x(x), y(y) { }

    double len() const
    {
        return sqrt(x * x + y * y);
    }

    Vec normalize() const
    {
        return Vec(-y / len(), x / len());
    }

    double angle() const
    {
        return atan2(y, x);
    }
};

using Pt = Vec;

struct Line
{
    Pt pt;
    Vec vec;
    Line(Pt pt = 0, Vec vec = 0) : pt(pt), vec(vec) { }
};

struct Seg
{
    Pt a, b;
    Vec ab;
    Seg(Pt a = 0, Pt b = 0) : a(a), b(b)
    {
        ab = b - a;
    }
};

const double EPS = 1e-10;
int dcmp(double x)
{
    return abs(x) < EPS ? 0 : (x < 0 ? -1 : 1);
}

```

```
50 bool operator==(const Pt &l, const Pt &r)
51 {
52     return abs(l.x - r.x) <= EPS && abs(l.y - r.y) <= EPS;
53 }
54
55 bool operator<(const Pt &l, const Pt &r)
56 {
57     return l.x < r.x || (l.x == r.x && l.y < r.y);
58 }
59
60 Vec operator-(const Vec &l, const Vec &r)
61 {
62     return Vec(l.x - r.x, l.y - r.y);
63 }
64
65 Vec operator+(const Vec &l, const Vec &r)
66 {
67     return Vec(l.x + r.x, l.y + r.y);
68 }
69
70 Vec operator*(double k, const Vec &vec)
71 {
72     return Vec(k * vec.x, k * vec.y);
73 }
74
75 Vec operator/(const Vec &vec, double k)
76 {
77     return Vec(vec.x / k, vec.y / k);
78 }
79
80 double dot(const Vec &l, const Vec &r)
81 {
82     return l.x * r.x + l.y * r.y;
83 }
84
85 double angle(const Vec &l, const Vec &r)
86 {
87     return acos(dot(l, r) / l.len() / r.len());
88 }
89
90 double cross(const Vec &l, const Vec &r)
91 {
92     return l.x * r.y - l.y * r.x;
93 }
94
95 double area(const Pt &a, const Pt &b, const Pt &c)
96 {
97     return cross(b - a, c - a);
98 }
99
100
101
102
103
```

```
104 // Make sure !pts.empty()
105 double area(const vector<Pt> &pts)
106 {
107     double ans = 0;
108     for (int i = 1; i < pts.size() - 1; ++i)
109         ans += cross(pts[i] - pts[0], pts[i + 1] - pts[0]);
110     return ans / 2;
111 }
112
113 Vec rotate(const Vec &vec, double a)
114 {
115     return Vec(vec.x * cos(a) - vec.y * sin(a),
116                vec.x * sin(a) + vec.y * cos(a));
117 }
118
119 // Make sure cross(vp, vq) != 0
120 Pt intersect(const Pt &p, const Vec &vp, const Pt &q, const Vec &vq)
121 {
122     return p + cross(vq, p - q) / cross(vp, vq) * vp;
123 }
124
125 double dist(const Pt &p, const Line &line)
126 {
127     return abs(cross(line.vec, p - line.pt)) / line.vec.len();
128 }
129
130 double dist(const Pt &p, const Seg &seg)
131 {
132     if (seg.a == seg.b)
133         return (p - seg.a).len();
134     Vec ap = p - seg.a, bp = p - seg.b;
135     if (dot(seg.ab, ap) < -EPS)
136         return ap.len();
137     else if (dot(seg.ab, bp) > EPS)
138         return bp.len();
139     return abs(cross(seg.ab, ap)) / seg.ab.len();
140 }
141
142 Pt projection(const Pt &p, const Line &line)
143 {
144     return line.pt +
145         dot(line.vec, p - line.pt) / dot(line.vec, line.vec) * line.vec;
146 }
147
148 bool hasIntersection(const Seg &l, const Seg &r)
149 {
150     double la = cross(l.ab, r.a - l.a), lb = cross(l.ab, r.b - l.a),
151            ra = cross(r.ab, l.a - r.a), rb = cross(r.ab, l.b - r.a);
152     return dcmp(la) * dcmp(lb) < 0 && dcmp(ra) * dcmp(rb) < 0;
153 }
154
155
156 bool on(const Pt &p, const Seg &seg)
157 {
```

```
158     return dcmp(cross(seg.a - p, seg.b - p)) == 0 &&
159            dcmp(dot(seg.a - p, seg.b - p)) < 0;
160 }
```