

# Team reference document

The Gaussians at IfI:

Bakhodir Ashirmatov, Azat Khuziyakhmetov, Jerome Baum

Georg-August Universität Göttingen

November 15, 2014

## Contents

<b>Data structures</b>	<b>1, p. 2</b>
1.1 Union-find (2). 1.2 Fenwick tree (2). 1.3 Interval tree (3).	
<b>Graph theory</b>	<b>2, p. 4</b>
2.1 BFS (4). 2.2 DFS (5). 2.3 Bridges (5). 2.4 Cut points (5). 2.5 Strong connectivity (6). 2.6 Dijkstra (6). 2.7 Dijkstra heap (7). 2.8 Floyd (7). 2.9 Bellman (7). 2.10 Prim (8). 2.11 Kruskal (8).	
<b>Lowest common ancestor and maximal pair matching</b>	<b>3, p. 8</b>
3.1 Tarjan's algo (8). 3.2 Kuhn's algo (9). 3.3 Hungarian algo (10).	
<b>Game theory</b>	<b>4, p. 10</b>
4.1 Sprague-Grundy theory (10).	
<b>Number theory</b>	<b>5, p. 11</b>
5.1 Binary powering (11). 5.2 Euler's function (11). 5.3 Factorisation (11). 5.4 GCD extended (11). 5.5 Inverse by modulo (12). 5.6 Eratosthenes sieve (12). 5.7 Gray's code (12). 5.8 Submasks (12). 5.9 Garner's algo (12).	
<b>String algorithms</b>	<b>6, p. 13</b>
6.1 Z-function (number of different substrings) (13). 6.2 Prefix function (13). 6.3 Rabin-Carp (13). 6.4 Suffix array (14).	
<b>Numerical algorithms</b>	<b>7, p. 15</b>
7.1 Binary search (15). 7.2 Ternary search (15). 7.3 Newton integration (15). 7.4 Gauss algorithm (15).	

# 1 Data structures

## 1.1 Union-find

```
const int MAXN = 500;

struct UnionFind {
    int id;
    int rank;
    int val;
};

UnionFind uf[MAXN];

void combine(int a, int b) {
    uf[b].val = max(uf[b].val, uf[a].val);
}

void build(int n) {
    for (int i = 0; i < n; i++) {
        uf[i].id = i;
        uf[i].rank = 1;
        uf[i].val = 0;
    }
}

int find(int v) {
    if (v == uf[v].id)
        return v;
    return uf[v].id = find(uf[v].id);
}

bool unite(int a, int b) {
    a = find(a);
    b = find(b);
    if (a != b) {
        if (uf[a].rank < uf[b].rank)
            swap(a, b);
        uf[b].id = a;
        combine(a, b);
        if (uf[a].rank == uf[b].rank)
            uf[a].rank++;
        return true;
    }
    return false;
}
```

## 1.2 Fenwick tree

```
const int MAXN = 100000;
int fenwick[MAXN];

void build(int n) {
    for (int i = 0; i < n; i++) {
        fenwick[i] = 0;
    }
}

int sum(int r) {
    int result = 0;
    for (; r >= 0; r = (r & (r + 1)) - 1) {
        result += fenwick[r];
    }
    return result;
}

void inc(int i, int n, int delta) {
    for (; i < n; i = (i | (i + 1))) {
        fenwick[i] += delta;
    }
}

int sum(int l, int r) {
```

```

        return sum(r) - sum(l - 1);
}

```

## 1.3 Interval tree

```

const int MAXN = 100000;
const long long MAXVAL = 12345600000011;
const long long ZERO = -1;

long long a[MAXN];

struct Tree{
    //----- MIN / MAX
    long long maxa;
    long long mina;
    //----- COLOR / SUM
    long long color;
    long long sum;
    //----- ADD
    long long add;
    //-----
};

Tree tree[4 * MAXN];

Tree create(long long minval, long long maxval,
            long long colval, long long sumval) {
    Tree result;
    //----- MIN / MAX
    result.maxa = maxval;
    result.mina = minval;
    //----- COLOR / SUM
    result.color = colval;
    result.sum = sumval;
    //----- ADD
    result.add = 0;
    //-----
    return result;
}

Tree combine(Tree a, Tree b) {
    Tree result;
    //----- MIN / MAX
    result.maxa = max(a.maxa, b.maxa);
    result.mina = min(a.mina, b.mina);
    //----- COLOR / SUM
    result.sum = a.sum + b.sum;
    result.color = ZERO;
    //----- ADD
    result.add = 0;
    //-----
    return result;
}

void change(int v, int l, int r, int val) {
    //----- ADD
    tree[v].add += val;
    //----- COLOR / SUM
    tree[v].sum += val * (r - l + 1);
    //----- MIN / MAX
    tree[v].maxa += val;
    tree[v].mina += val;
    //-----
}

void push(int v, int tl, int tm, int tr) {
    //----- ADD
    change(v * 2 + 1, tl, tm, tree[v].add);
    change(v * 2 + 2, tm + 1, tr, tree[v].add);
    tree[v].add = 0;
    //----- COLOR / SUM
    if (tree[v].color != ZERO) {
        tree[v * 2 + 1] = create(tree[v].color, tree[v].color, tree[v].color, tree[v].color * (tm
            - tl + 1));
    }
}

```

```

        tree[v * 2 + 2] = create(tree[v].color, tree[v].color, tree[v].color, tree[v].color * (tr
        - tm));
        tree[v].color = ZERO;
    }
    //-----
}

void build(int v, int tl, int tr) {
    if (tl == tr) {
        tree[v] = create(a[tl], a[tl], ZERO, a[tl]);
    }
    else {
        int tm = (tl + tr) >> 1;
        build(v * 2 + 1, tl, tm);
        build(v * 2 + 2, tm + 1, tr);
        tree[v] = combine(tree[v * 2 + 1], tree[v * 2 + 2]);
    }
}

void update(int v, int tl, int tr, int l, int r, long long val){
    if (l == tl && r == tr) {
        tree[v] = create(val, val, val, val * (r - l + 1));
    }
    else{
        int tm = (tl + tr) >> 1;
        push(v, tl, tm, tr);
        if (l <= tm)
            update(v * 2 + 1, tl, tm, l, min(r, tm), val);
        if (r > tm)
            update(v * 2 + 2, tm + 1, tr, max(l, tm + 1), r, val);
        tree[v] = combine(tree[v * 2 + 1], tree[v * 2 + 2]);
    }
}

void add(int v, int tl, int tr, int l, int r, long long val){
    if (l == tl && r == tr) {
        change(v, l, r, val);
    }
    else{
        int tm = (tl + tr) >> 1;
        push(v, tl, tm, tr);
        if (l <= tm)
            update(v * 2 + 1, tl, tm, l, min(r, tm), val);
        if (r > tm)
            update(v * 2 + 2, tm + 1, tr, max(l, tm + 1), r, val);
        tree[v] = combine(tree[v * 2 + 1], tree[v * 2 + 2]);
    }
}

Tree get(int v, int tl, int tr, int l, int r) {
    if (l > r) {
        return create(MAXVAL, - MAXVAL, 0, 0);
    }
    if (l == tl && r == tr) {
        return tree[v];
    }
    else {
        int tm = (tl + tr) >> 1;
        push(v, tl, tm, tr);
        return combine( get(v * 2 + 1, tl, tm, l, min(r, tm)),
                        get(v * 2 + 2, tm + 1, tr, max(l, tm + 1), r));
    }
}

```

## 2 Graph theory

### 2.1 BFS

```

queue<int> q;
q.push(s);
used[s] = 1;
from[s] = s;

```

```

while (!q.empty()) {
    int cur = q.front();
    q.pop();
    for (int i = 0; i < edges[cur].size(); i++) {
        int next = edges[cur][i];
        if (!used[next]) {
            used[next] = used[cur] + 1;
            from[next] = cur;
            q.push(next);
        }
    }
}

```

## 2.2 DFS

```

bool dfs(int v, int h = 0) {
    used[v] = 1;
    p[h] = v;
    for (int i = 0; i < edges[v].size(); i++) {
        int next = edges[v][i];
        if (used[next] == 1) {
            printf("YES\n");
            int cur = h;
            while (p[cur] != next) {
                cur--;
            }
            while (cur <= h) {
                printf("%d_", p[cur] + 1);
                cur++;
            }
            return false;
        }
        else if (used[next] == 0) {
            if (!dfs(next, h + 1)) {
                return false;
            }
        }
    }
    used[v] = 2;
    return true;
}

```

## 2.3 Bridges

```

int cnt = 0;

void dfs(int v, int pr = - 1){
    used[v] = 1;
    fout[v] = tin[v] = cnt++;
    for (int i = 0; i < edges[v].size(); i++){
        int to = edges[v][i];
        if (to == pr)
            continue;
        if (!used[to]){
            dfs(to, v);
            fout[v] = min(fout[v], fout[to]);
            if (tin[v] < fout[to])
                ans.push_back(num[v][i]);
        }
        else{
            fout[v] = min(fout[v], tin[to]);
        }
    }
}

```

## 2.4 Cur points

```

int cnt = 0;

void dfs(int v, bool root = false){
    used[v] = 1;
    fout[v] = tin[v] = cnt++;
}

```

```

    bool is = false;
    int sons = 0;
    for (int i = 0; i < edges[v].size(); i++){
        int to = edges[v][i];
        if (!used[to]){
            dfs(to);
            fout[v] = min(fout[v], fout[to]);
            if (tin[v] <= fout[to] && !root)
                is = true;
            sons++;
        }
        else{
            fout[v] = min(fout[v], tin[to]);
        }
    }
    if (root && sons >= 2)
        is = true;
    if (is){
        ans.push_back(v);
    }
}

```

## 2.5 Strong connectivity

```

int cnt = 0;

void dfs1(int v){
    used[v] = 1;
    for (int i = 0; i < edges[v].size(); i++){
        int to = edges[v][i];
        if (!used[to]){
            dfs1(to);
        }
    }
    topsort.push_back(v);
}

void dfs2(int v){
    color[v] = cnt;
    for (int i = 0; i < redges[v].size(); i++){
        int to = redges[v][i];
        if (!color[to]){
            color[to] = cnt;
            dfs2(to);
        }
    }
}

for (int i = 0; i < n; i++)
    if (!used[i])
        dfs1(i);

reverse(topsort.begin(), topsort.end());

for (int i = 0; i < n; i++){
    if (!color[topsort[i]]){
        cnt++;
        dfs2(topsort[i]);
    }
}

```

## 2.6 Dijkstra

```

used[s] = 1;
for (int i = 0; i < n; i++) {
    int mini = -1;
    for (int j = 0; j < n; j++) {
        if (used[j] != 1)
            continue;
        if (mini == -1 || deik[mini] > deik[j])
            mini = j;
    }
    if (mini == -1)

```

```

        break;
    used[mini] = 2;
    for (int j = 0; j < n; j++) {
        if (d[mini][j] == -1)
            continue;
        if (used[j] == 0 || deik[j] > deik[mini] + d[mini][j]) {
            used[j] = 1;
            deik[j] = deik[mini] + d[mini][j];
        }
    }
}
}

```

## 2.7 Dijkstra heap

```

set<pair<int, int> > deikst;
deik[0] = 0;
used[0] = 1;
from[0] = -1;
deikst.insert(make_pair(deik[0], 0));

while (!deikst.empty()) {
    int cur = deikst.begin()->second;
    deikst.erase(deikst.begin());
    for (int i = 0; i < edges[cur].size(); i++) {
        int next = edges[cur][i];
        int cost = costs[cur][i];
        if (used[next] == 0 || deik[next] > deik[cur] + cost) {
            used[next] = 1;
            deikst.erase(make_pair(deik[next], next));
            deik[next] = deik[cur] + cost;
            from[next] = cur;
            deikst.insert(make_pair(deik[next], next));
        }
    }
}
}

```

## 2.8 Floyd

```

for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
        }
    }
}
}

```

## 2.9 Bellman

```

for (int i = 0; i < n; i++) {
    d[i] = INF;
}
d[s] = 0;

for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n; j++) {
        if (d[j] == INF) {
            continue;
        }
        for (int k = 0; k < edges[j].size(); k++) {
            int next = edges[j][k];
            long long cost = costs[j][k];
            if (d[next] > d[j] + cost) {
                d[next] = d[j] + cost;
            }
        }
    }
}

for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n; j++) {
        if (d[j] == INF) {
            continue;
        }
    }
}
}

```

```

    }
    for (int k = 0; k < edges[j].size(); k++) {
        int next = edges[j][k];
        long long cost = costs[j][k];
        if (d[next] > d[j] + cost) {
            d[next] = - INF;
        }
    }
}
}
}

```

## 2.10 Prim

```

used[0] = 1;

double sum = 0;

for (int i = 0; i < n; i++) {
    int mini = -1;
    for (int j = 0; j < n; j++) {
        if (used[j] != 1) {
            continue;
        }
        if (mini == -1 || prim[mini] > prim[j]) {
            mini = j;
        }
    }
    if (mini == -1) {
        break;
    }
    double add = prim[mini];
    sum += sqrt(add);
    used[mini] = 2;
    for (int j = 0; j < n; j++) {
        int len = sqrt(x[j] - x[mini]) + sqrt(y[j] - y[mini]);
        if (used[j] == 0 || (used[j] == 1 && prim[j] > len)) {
            used[j] = 1;
            prim[j] = len;
        }
    }
}
}

```

## 2.11 Kruskal

```

sort(v.begin(), v.end());

double sum = 0;

vector<pair<int, int>> ans;

for (int i = 0; i < v.size(); i++) {
    int a = v[i].second.first, b = v[i].second.second;
    double c = v[i].first;
    if (unite(a, b)) {
        ans.push_back(make_pair(a + 1, b + 1));
        sum += sqrt(c);
    }
}

```

# 3 Lowest common ancestor and maximal pair matching

## 3.1 Tarjan's algo

```

const int MAXN = 500;
vector<int> edges[MAXN], q[MAXN];
int id[MAXN], ancestor[MAXN];
bool used[MAXN];

int get(int v) {
    return v == id[v] ? v : id[v] = get(id[v]);
}

```



```

void unite(int a, int b, int new_ancestor) {
    a = get(a), b = get(b);
    if (rand() & 1) {
        swap (a, b);
    }
    id[a] = b, ancestor[b] = new_ancestor;
}

void dfs(int v) {
    id[v] = v, ancestor[v] = v;
    used[v] = true;
    for (int i = 0; i < edges[v].size(); i++)
        if (!u[edges[v][i]]) {
            dfs(edges[v][i]);
            unite(v, edges[v][i], v);
        }
    for (int i = 0; i < q[v].size(); i++)
        if (u[q[v][i]]) {
            printf ("%d_%d->%d\n", v + 1, q[v][i] + 1,
                ancestor[ get(q[v][i]) ] + 1);
        }
}

int main() {

    for (;;) {
        int a, b = ...;
        --a, --b;
        q[a].push_back (b);
        q[b].push_back (a);
    }

    dfs (0);
}

```

## 3.2 Kuhn's algo

```

const int MAXN = 500;

vector <int> edges[MAXN];
int match[MAXN];
bool used[MAXN];

bool try_kuhn (int v) {
    if (used[v]) {
        return false;
    }
    used[v] = true;
    for (int i = 0; i < edges[v].size; i++) {
        int to = edges[v][i];
        if (match[to] == -1 || try_kuhn(match[to])) {
            match[to] = v;
            return true;
        }
    }
    return false;
}

int main() {

    for (int i = 0; i < k; i++) {
        match[i] = - 1;
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            used[j] = false;
        }
        try_kuhn(i);
    }

    for (int i = 0; i < k; i++) {
        if (match[i] != -1) {
            printf ("%d_%d'n", match[i] + 1, i + 1);

```

```

    }
}

```

### 3.3 Hungarian algo

```

vector<int> u (n+1), v (m+1), p (m+1), way (m+1);
for (int i=1; i<=n; ++i) {
    p[0] = i;
    int j0 = 0;
    vector<int> minv (m+1, INF);
    vector<char> used (m+1, false);
    do {
        used[j0] = true;
        int i0 = p[j0], delta = INF, j1;
        for (int j=1; j<=m; ++j)
            if (!used[j]) {
                int cur = a[i0][j]-u[i0]-v[j];
                if (cur < minv[j])
                    minv[j] = cur, way[j] = j0;
            }
        if (minv[j] < delta)
            delta = minv[j], j1 = j;
    }
    for (int j=0; j<=m; ++j)
        if (used[j])
            u[p[j]] += delta, v[j] -= delta;
    else
        minv[j] -= delta;
    j0 = j1;
    } while (p[j0] != 0);
    do {
        int j1 = way[j0];
        p[j0] = p[j1];
        j0 = j1;
    } while (j0);
}

vector<int> ans (n+1);
for (int j=1; j<=m; ++j)
    ans[p[j]] = j;

int cost = -v[0];

```

## 4 Game theory

### 4.1 Sprague-Grundy theory

```

int mex(vector<int> a) {
    set<int> b(a.begin(), a.end());
    for (int i = 0; ; i++)
        if (!b.count(i))
            return i;
}

int g[1000];

int grundy(int n) {
    vector<int> v;
    if (n >= 2) {
        v.push_back(g[n - 2]);
    }
    for (int i = 2; i <= n - 1; i++) {
        v.push_back(g[i - 2] ^ g[n - i - 1]);
    }
    return mex(v);
}

const int D = 10;
static bool used[D + 1] = { 0 };

int mex (vector<int> a) {
    int c = (int) a.size();

```

```

    for (int i = 0; i < c; i++)
        if (a[i] <= D)
            used[a[i]] = true;

    int result;
    for (int i = 0; i <= D; i++)
        if (!used[i]) {
            result = i;
            break;
        }

    for (int i = 0; i < c; i++)
        if (a[i] <= D)
            used[a[i]] = false;

    return result;
}

```

## 5 Number theory

### 5.1 Binary powering

```

long long bincpow(long long a, long long n, long long mod) {
    if (n == 0)
        return 1;
    long long res = bincpow(a, n / 2, mod);
    res = (res * res) % mod;
    if (n % 2 == 1)
        return res = (res * a) % mod;
    return res;
}

```

### 5.2 Euler's function

```

int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1) {
        result -= result / n;
    }
    return result;
}

```

### 5.3 Factorisation

```

void factorization(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            int cnt = 0;
            while (n % i == 0) {
                n /= i;
                cnt++;
            }
            printf("%d^%d", i, cnt);
        }
    }
    if (n > 1) {
        printf("%d^1", n);
    }
}

```

### 5.4 GCD extended

```
int gcdex(int a, int b, int &x, int &y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }
    int x1, y1;
    int d = gcd(b % a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return d;
}
```

## 5.5 Inverse by modulo

```
long long inverse(long long a, long long mod) {
    return binpow(a, mod - 2, mod);
}

void inverse(int a, int b) {
    int x, y;
    int g = gcdex(a, m, x, y);
    if (g != 1) {
        cout << "no solution";
    }
    else {
        x = (x % m + m) % m;
        cout << x;
    }
}
```

## 5.6 Eratosthenes sieve

```
int n;
vector<char> prime (n+1, true);
prime[0] = prime[1] = false;
for (int i = 2; i <= n; i++)
    if (prime[i])
        if (i * 1ll * i <= n)
            for (int j = i * i; j <= n; j += i)
                prime[j] = false;
```

## 5.7 Gray's code

```
int g (int n) {
    return n ^ (n >> 1);
}

int rev_g (int g) {
    int n = 0;
    for (; g; g>>=1)
        n ^= g;
    return n;
}
```

## 5.8 Submasks

```
for (int m = 0; m < (1 << n); m++) {
    for (int s = m; s; s = (s - 1) & m) {
    }
}
```

## 5.9 Garner's algo

```
for (int i=0; i<k; ++i) {
    x[i] = a[i];
    for (int j=0; j<i; ++j) {
        x[i] = r[j][i] * (x[i] - x[j]);
    }
}
```

```

        x[i] = x[i] % p[i];
        if (x[i] < 0)
            x[i] += p[i];
    }
}

```

## 6 String algorithms

### 6.1 Z-function (number of different substrings)

```

string s;
int len = (int) s.length();

string t = "";
int tlen = 0;
int cnt = 0;

for (int si = 0; si < len; si++) {
    t.push_back(s[si]);
    tlen++;

    reverse(t.begin(), t.end());

    int j = 0, k = 0;
    int zmax = 0;
    for (int i = 1; i < tlen; i++) {
        z[i] = 0;
        if (i < j + k) {
            z[i] = min(z[i - j], j + k - i);
        }
        while (i + z[i] < len && t[z[i]] == t[i + z[i]]) {
            z[i]++;
        }
        if (i + z[i] > j + k) {
            j = i;
            k = z[i];
        }
        zmax = max(zmax, z[i]);
    }

    cnt += tlen - zmax;

    reverse(t.begin(), t.end());
}

```

### 6.2 Prefix function

```

vector<int> prefix_function (string s) {
    int n = (int) s.length();
    vector<int> pi (n);
    for (int i=1; i<n; ++i) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j]) ++j;
        pi[i] = j;
    }
    return pi;
}

```

### 6.3 Rabin-Carp

```

string s, t;
const int p = 31;
vector<long long> p_pow (max (s.length(), t.length()));
p_pow[0] = 1;
for (size_t i=1; i<p_pow.size(); ++i)
    p_pow[i] = p_pow[i-1] * p;

vector<long long> h (t.length());
for (size_t i=0; i<t.length(); ++i)

```

```

{
    h[i] = (t[i] - 'a' + 1) * p_pow[i];
    if (i) h[i] += h[i-1];
}

long long h_s = 0;
for (size_t i=0; i<s.length(); ++i)
    h_s += (s[i] - 'a' + 1) * p_pow[i];

for (size_t i = 0; i + s.length() - 1 < t.length(); ++i)
{
    long long cur_h = h[i+s.length()-1];
    if (i) cur_h -= h[i-1];
    if (cur_h == h_s * p_pow[i])
        cout << i << '␣';
}

```

## 6.4 Suffix array

```

const int SIZE = 100000;
char s[SIZE + 1];

int c[SIZE] = {0};
int cnt[SIZE] = {0};
int p[SIZE] = {0};

int newp[SIZE] = {0};
int newc[SIZE] = {0};

const int MAXA = 128;

void build(int n) {
    for (int i = 0; i < n; i++) {
        cnt[s[i]]++;
    }
    for (int i = 1; i < MAXA; i++) {
        cnt[i] += cnt[i - 1];
    }
    for (int i = 0; i < n; i++) {
        cnt[s[i]]--;
        p[cnt[s[i]]] = i;
    }
    c[p[0]] = 0;
    for (int i = 1; i < n; i++) {
        c[p[i]] = c[p[i - 1]];
        if (s[p[i]] != s[p[i - 1]]) {
            c[p[i]]++;
        }
    }
    for (int len = 1; len <= n; len *= 2) {
        for (int i = 0; i < n; i++) {
            cnt[i] = 0;
        }
        for (int i = 0; i < n; i++) {
            cnt[c[i]]++;
        }
        for (int i = 1; i < n; i++) {
            cnt[i] += cnt[i - 1];
        }
        for (int i = n - 1; i >= 0; i--) {
            int j = (p[i] - len + n) % n;
            cnt[c[j]]--;
            newp[cnt[c[j]]] = j;
        }
        p[0] = newp[0];
        newc[p[0]] = 0;
        for (int i = 1; i < n; i++) {
            p[i] = newp[i];
            newc[p[i]] = newc[p[i - 1]];
            if (c[p[i]] != c[p[i - 1]] || c[(p[i] + len) % n] != c[(p[i - 1] + len) % n]) {
                newc[p[i]]++;
            }
        }
        for (int i = 0; i < n; i++) {

```

```

        c[i] = newc[i];
    }

}

```

## 7 Numerical algorithms

### 7.1 Binary search

```

int l = 0, r = 1000000000;
int best = 0;

while (l <= r) {
    int mid = (l + r) >> 1;
    if (true) {
        best = mid;
        l = mid + 1;
    }
    else {
        r = mid - 1;
    }
}

```

### 7.2 Ternary search

```

long double l = 0, r = 1;

while (r - l > eps) {
    long double x1 = l + (r - l) / 3.0;
    long double x2 = l + 2.0 * (r - l) / 3.0;

    long double f1 = f(x1);
    long double f2 = f(x2);
    if (f1 > f2) {
        l = x1;
    }
    else {
        r = x2;
    }
}

```

### 7.3 Newton integration

```

double a, b;
const int N = 1000*1000;
double s = 0;
double h = (b - a) / N;
for (int i=0; i<=N; ++i) {
    double x = a + h * i;
    s += f(x) * ((i==0 || i==N) ? 1 : ((i&1)==0) ? 2 : 4);
}
s *= h / 3;

```

### 7.4 Gauss algorithm

```

int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
    }
}

```

```
    where[col] = row;

    for (int i=0; i<n; ++i)
        if (i != row) {
            double c = a[i][col] / a[row][col];
            for (int j=col; j<=m; ++j)
                a[i][j] -= a[row][j] * c;
        }
    ++row;
}

ans.assign (m, 0);
for (int i=0; i<m; ++i)
    if (where[i] != -1)
        ans[i] = a[where[i]][m] / a[where[i]][i];
for (int i=0; i<n; ++i) {
    double sum = 0;
    for (int j=0; j<m; ++j)
        sum += ans[j] * a[i][j];
    if (abs (sum - a[i][m]) > EPS)
        return 0;
}

for (int i=0; i<m; ++i)
    if (where[i] == -1)
        return INF;
return 1;
}
```