Contents


```
long pow_mod(long x, long n, long p) {
  if (n == 0) return 1;
  if (n & 1)
    return (pow_mod(x, n-1, p) * x) % p;
  x = pow_mod(x, n/2, p);
  return (x * x) % p;
}

/* Takes as input an odd prime p and n < p and returns
    r
 * such that r * r = n [mod p]. */
long tonelli_shanks(long n, long p) {
  long s = 0;
  long q = p - 1;
  while ((q & 1) == 0) { q /= 2; ++s; }
  if (s == 1) {
    long r = pow_mod(n, (p+1)/4, p);
    if ((r * r) % p == n) return r;
    return 0;
  }
  // Find the first quadratic non-residue z by brute-
     force search
  long z = 1;
  while (pow_mod(++z, (p-1)/2, p) != p - 1);
  long c = pow_mod(z, q, p);
  long r = pow_mod(n, (q+1)/2, p);
  long t = pow_mod(n, q, p);
  long m = s;
  while (t != 1) {
    long tt = t;
    long i = 0;
    while (tt != 1) {
      tt = (tt * tt) % p;
      ++i;
      if (i == m) return 0;
    }
    long b = pow_mod(c, pow_mod(2, m-i-1, p-1), p);
    long b2 = (b * b) % p;
    r = (r * b) % p;
    t = (t * b2) % p;
    c = b2;
    m = i;
  }
  if ((r * r) % p == n) return r;
  return 0;
}
```

```
#include <iostream>
#include <unordered_map>
#include <cmath>

using namespace std;

/* a^k = b (mod p). k is the returning answer. */
class Discrete_Log {
public:
    long long pow(long long base, long long power, long
         long p) {
        long long ans = 1, temp = base;
        while(power != 0) {
            if(power % 2 == 1) ans = (ans * temp) % p;
            temp = (temp * temp) % p;
            power >>= 1;
        }
        return ans;
    }
    long long solve(long long a, long long b, long long
         p) {
        unordered_map<long long, long long> appear;
        long long m = sqrt(p) + 1, temp;
        for(long long i = 0;i <= m;i++) {
            temp = pow(a, m * i, p);
            temp = pow(temp, p - 2, p);
            temp = (temp * b) % p;
            appear[temp] = i;
        }
        for(long long i = 0;i <= m;i++) {
            temp = pow(a, i, p);
            if(appear.find(temp) != appear.end())
                return appear[temp] * m + i;
        }
        return -1;
    }
} Log;

int main() {
    long long a, b, p, ans;
    cin >> a >> b >> p;
    ans = Log.solve(a, b, p);
    cout << ans << endl;
    cout << Log.pow(a, ans, p) << " " << b << endl;
}
```

```
#include <iostream>
#include <memory.h>
#include <set>
#include <algorithm>
#include <assert.h>

using namespace std;
const int MAXN = 5e4 + 50 ,MAXM = 2e5 + 50 ,INF = (1LL
    << 31) - 1;
const int MAXV = MAXN + MAXM;
class LCT {
    int ch[MAXV][2] ,par[MAXV] ,rev[MAXV] ,mini[MAXV] ,
        val[MAXV];
    int Get_Child(int x) { return (ch[par[x]][1] == x ?
         1 : 0); }
    bool Is_Root(int x) { return par[x] == -1 || (ch[
        par[x]][0] != x && ch[par[x]][1] != x); }
    void Pull_Up(int x) {
        mini[x] = val[x];
        if(ch[x][0] != -1) mini[x] = min(mini[x] ,mini[
            ch[x][0]]);
        if(ch[x][1] != -1) mini[x] = min(mini[x] ,mini[
            ch[x][1]]);
    }
    void Send(int x) { swap(ch[x][0] ,ch[x][1]); rev[x]
         ^= 1; }
    void Push_Down(int x) {
        if(rev[x]) {
            if(ch[x][0] != -1) Send(ch[x][0]);
            if(ch[x][1] != -1) Send(ch[x][1]);
            rev[x] = 0;
        }
    }
    void Rotate(int x) {
        int p = par[x] ,pp = par[p] ,style = Get_Child(
            x) ,subtree = ch[x][style ^ 1];
        par[x] = pp;
        if(!Is_Root(p)) ch[pp][Get_Child(p)] = x;
        if(subtree != -1) par[subtree] = p;
        ch[p][style] = subtree;
        par[p] = x;
        ch[x][style ^ 1] = p;
        Pull_Up(p); Pull_Up(x);
        if(pp != -1) Pull_Up(pp);
    }
    void Update(int x) {
        if(!Is_Root(x)) Update(par[x]);
```

```
        Push_Down(x);
    }
    void Splay(int x) {
        Update(x);
        for(int p;!Is_Root(x);Rotate(x)) {
            p = par[x];
            if(!Is_Root(p)) Rotate(Get_Child(x) ==
                Get_Child(p) ? p : x);
        }
    }
    int Access(int x) {
        int pre = -1;
        while(x != -1) { Splay(x); ch[x][1] = pre;
            Pull_Up(x); pre = x; x = par[x];  }
        return pre;
    }
    void Make_Root(int x) { Access(x); Splay(x); swap(
        ch[x][0] ,ch[x][1]); rev[x] ^= 1; }
    int Get_Root(int x) {
        Access(x); Splay(x);
        while(ch[x][0] != -1) x = ch[x][0];
        Splay(x);
        return x;
    }
public:
    bool Same_Boss(int a ,int b) { return Get_Root(a)
        == Get_Root(b); }
    int Find_Min(int a ,int b) { Make_Root(a); return
        mini[Access(b)]; }
    void Cut(int a ,int b) { Make_Root(a); Access(b);
        Splay(b); par[a] = ch[b][0] = -1; Pull_Up(b); }
    void Link(int a ,int b) { Make_Root(a); par[a] = b;
         }
    LCT() {
        memset(ch ,-1 ,sizeof(ch)); memset(par ,-1 ,
            sizeof(par)); memset(rev ,0 ,sizeof(rev));
        for(int i = 0;i < MAXV;i++) mini[i] = val[i] =
            (i >= MAXN ? i - MAXN : INF);
    }
} tree;

struct E { int src ,dst ,val; } edges[MAXM];
bool cmp(E a ,E b) { return a.val < b.val; }

void Cut(int e) { tree.Cut(edges[e].src ,MAXN + e);
    tree.Cut(edges[e].dst ,MAXN + e); }
void Link(int e) { tree.Link(edges[e].src ,MAXN + e);
    tree.Link(edges[e].dst ,MAXN + e); }

int main() {
    ios::sync_with_stdio(0) ,cin.tie(0);
    int N ,M ,temp ,ans = INF;
    int i ,lptr ,rptr;
    set<int> ids;
    cin >> N >> M;
    for(i = 0;i < M;i++) {
        cin >> edges[i].src >> edges[i].dst >> edges[i
            ].val;
        if(edges[i].src == edges[i].dst) { i -= 1; M -=
             1; }
    }
    sort(edges ,edges + M ,cmp);
    for(lptr = rptr = 0;lptr < M;lptr++) {
        while(!ids.empty() && *ids.begin() < lptr) {
            Cut(temp = *ids.begin());
            ids.erase(temp);
        }
        for(;ids.size() < N - 1 && rptr < M;rptr++) {
            if(tree.Same_Boss(edges[rptr].src ,edges[
                rptr].dst)) {
                Cut(temp = tree.Find_Min(edges[rptr].
                    src ,edges[rptr].dst));
                ids.erase(temp);
            }
            Link(rptr); ids.insert(rptr);
        }
        if(ids.size() == N - 1) ans = min(ans ,edges[*
            ids.rbegin()].val - edges[*ids.begin()].val
            );
        else break;
    }
    cout << ans << endl;
}
```

2.2 Treap

```
#include <iostream>

#define P 13
int rgen = 1;

using namespace std;
class treap {
    public:
    treap *l ,*r ,*p;
    int key ,elements = 0 ,pri ,value;

    treap(int key ,int value = 0) {
        this->value = value;
        this->key = key;
        this->elements = 1;
        this->pri = rgen;
        rgen = (rgen << 1) % P;
        this->l = this->r = 0;
    }

    treap* merge(treap* tr) {
        if(!tr) return this;
        if(!this) return tr;
        if(this->pri < tr->pri) {
            this->r = this->r->merge(tr);
            this->pull();
            return this;
        } else {
            tr->l = this->merge(tr->l);
            tr->pull();
            return tr;
        }
    }

    void split(int key ,treap*& a, treap*& b) {
        if(!this) {
            a = b = 0;
        } else if(this->key <= key) {
            a = this;
            this->r->split(key ,a->r ,b);
            a->pull() ,b->pull();
        } else if(this->key > key) {
            b = this;
            this->l->split(key ,a ,b->l);
            a->pull() ,b->pull();
        }
    }

    void pull() {
        if(!this) return;
        this->elements = (this->l ? this->l->elements :
             0) + (this->r ? this->r->elements : 0) +
            1;
    }

    void dfs(int pkey = -1) {
        if(!this) return;
        this->l->dfs(this->key);
        printf("k:%d\tv:%d\tpri:%d\tcount:%d\tpvalue:%d
            \n" ,this->key ,this->value ,this->pri ,
            this->elements ,pkey);
        this->r->dfs(this->key);
    }

    treap* kth(int rank) {
        int lcount = (this->l ? this->l->elements : 0);
        if(rank <= lcount) {
            return this->l->kth(rank);
        } else if(lcount + 1 == rank) {
            return this;
        } else {
            return this->r->kth(rank - lcount - 1);
        }
    }
};
```

```
inline void insert(treap* root ,int data) {
    treap *a ,*b ;
    root->split(data ,a ,b);
    root = a->merge(new treap(data));
    root = root->merge(b);
}

int main()
{
    int N ,Q;
    int i ,eax ,cmd;
    while(cin >> N) {
        treap *root = new treap(0);
        for(i = 1;i <= N;i++) {
            cin >> eax;
            insert(root ,eax);
        }
        root->dfs();

        for(cin >> Q;Q--;) {
            cin >> cmd;
            if(cmd == 1) {
                cin >> eax;
                printf("The %d-th minimum number is %d\
                    n" ,eax ,root->kth(eax)->key);
            } else {
                cin >> eax;
                printf("Insert %d to the set\n", eax);
                insert(root ,eax);
            }
        }
    }
    return 0;
}
```

2.3 Zkw

```
#pragma GCC optimize("Ofast,no-stack-protector")
#include <iostream>

using namespace std;
const int MAXN = 2e6 + 50, INF = 1e9;
class zkw {
    int mini[MAXN * 5], tag[MAXN * 5], leaf;
public:
    zkw() {}
    zkw(int N) {
        for(leaf = 1;leaf < N + 2;leaf <<= 1);
        for(int i = leaf * 2;i >= 1;i--) mini[i] = tag[
            i] = 0;
    }
    void pull(int x) { mini[x] = min(mini[x * 2], mini[
        x * 2 + 1]) + tag[x]; }
    void modify(int l, int r, int delta) {
        if(l >= r) return;
        int pos;
        for(l += leaf, r += leaf + 1;l ^ r ^ 1;l >>= 1,
             r >>= 1) {
            if(l % 2 == 0) { pos = l ^ 1; tag[pos] +=
                delta; pull(pos); }
            if(r % 2 == 1) { pos = r ^ 1; tag[pos] +=
                delta; pull(pos); }
            pull(l >> 1); pull(r >> 1);
        }
        for(pos = (l >> 1);pos != 0;pos >>= 1) pull(pos
            );
    }
    int minimum(int l, int r) {
        int ans = INF;
        for(l += leaf, r += leaf + 1;l ^ r ^ 1;l >>= 1,
             r >>= 1) {
            if(l % 2 == 0) ans = min(ans, mini[l ^ 1]);
            if(r % 2 == 1) ans = min(ans, mini[r ^ 1]);
        }
        for(int pos = (l >> 1);pos != 0;pos >>= 1) ans
            += tag[pos];
        return ans;
    }
} seg;
```

```
int raw[MAXN];
int main() {
    ios::sync_with_stdio(0); cin.tie(0);
    int T, N, K, eax, ebx, l, r; int i;
    for(cin >> T;T--;) {
        cin >> N >> K;
        if(T < 4) seg = zkw(K * 2 + 1);
        for(i = 0;i < N;i++) cin >> raw[i];
        for(i = 0;i < N / 2;i++) {
            eax = min(raw[i], raw[N - i - 1]);
            ebx = max(raw[i], raw[N - i - 1]);
            if(T < 4) {
                l = eax + 1; r = ebx + K + 1;
                seg.modify(l, eax + ebx, 1);
                seg.modify(eax + ebx + 1, r ,1);
                seg.modify(2, l, 2);
                seg.modify(r, K * 2 + 1, 2);
            }
        }
        if(T < 4) cout << seg.minimum(2, K * 2 + 1) <<
            '\n';
    }
}
```

2.4 Fibonacci Heap

```
#include <iostream>
#include <cmath>
#include <memory.h>

using namespace std;
const int MAXN = 5e5 ,MAXA = 1e3;

struct Node { int parent ,kid ,left ,right ,deg ,value;
     } data[MAXN]; int used = 0;
class FibHeap {
    int maxi ,siz ,A[MAXA];
    void Link(int x ,int y) {
        int xl = data[x].left ,yl = data[y].left;
        data[xl].right = y; data[y].left = xl; data[x].
            left = yl; data[yl].right = x;
    }
    void Isolate(int x) {
        int l = data[x].left ,r = data[x].right;
        data[x].left = data[x].right = x; data[l].right
             = r; data[r].left = l;
    }
    void Link_Heap(int father ,int kid) {
        Isolate(kid);
        if(data[father].kid == -1) data[father].kid =
            kid;
        else Link(data[father].kid ,kid);
        data[kid].parent = father; data[father].deg +=
            1; data[father].kid = kid;
    }
    void Consolindate() {
        int arr_size = (log(siz) / log(2)) + 1; memset(
            A ,-1 ,sizeof(A));
        int last = data[maxi].right ,iter = maxi ,deg ,
            old ,head;
        bool final_round;
        do {
            final_round = (iter == last); head = iter;
                deg = data[iter].deg; iter = data[iter
                ].left;
            while(A[deg] != -1) {
                old = A[deg];
                if(data[head].value < data[old].value)
                    swap(head ,old);
                Link_Heap(head ,old); A[deg] = -1; deg
                    += 1;
            }
            A[deg] = head;
        } while(!final_round);
        for(maxi = -1 ,iter = 0;iter < arr_size;iter++)
            if(A[iter] != -1) if(maxi == -1 || data[
                maxi].value < data[A[iter]].value) maxi
                 = A[iter];
    }
public:
```

```
    int Extract() {
        int ans_value = data[maxi].value ,kid = data[
            maxi].kid;
        int left_shift = (data[maxi].left == maxi ? -1
            : data[maxi].left);
        if(kid != -1) {
            for(int i = data[kid].left;i != kid;i =
                data[i].left) data[i].parent = -1;
            Link(kid ,maxi);
        }
        Isolate(maxi); maxi = (kid == -1 ? left_shift :
             kid);
        Consolindate(); siz -= 1;
        return ans_value;
    }
    void Meld(FibHeap* H) {
        Link(this->maxi ,H->maxi); siz += H->siz;
        if(data[H->maxi].value > data[this->maxi].value
            ) this->maxi = H->maxi;
    }
    int Peek() { return data[maxi].value; }
    FibHeap(int value):maxi(used) ,siz(1) { data[used]
        = Node{-1 ,-1 ,used ,used ,0 ,value}; used +=
        1; }
};

int main() {
    int cmd ,value;
    FibHeap* heap = new FibHeap(0);
    while(cin >> cmd) {
        if(cmd == 1) { cin >> value; heap->Meld(new
            FibHeap(value)); }
        if(cmd == 2) { cout << "Extracted: " << heap->
            Extract() << endl; }
        if(cmd == 3) { cout << "Value: " << heap->Peek
            () << endl; }
    }
}
/*
1 13
1 13
1 13
1 13
2
2
2
2
2
2
2


*/
```