

Sum queries (everything is 1 indexed)

Compute the greatest common divisor / least common multiple

(change sum with lcm or gcd or min but change the base case for example for min use the base case biggest possible number etc)

```
int n, t[4*MAXN];
```

```
void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = t[v*2] + t[v*2+1];
    }
}
```

```
int sum(int v, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && r == tr) {
        return t[v];
    }
    int tm = (tl + tr) / 2;
    return sum(v*2, tl, tm, l, min(r, tm))
        + sum(v*2+1, tm+1, tr, max(l, tm+1), r);
}
```

```
void update(int v, int tl, int tr, int pos, int new_val) {
    if (tl == tr) {
        t[v] = new_val;
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos, new_val);
        else
            update(v*2+1, tm+1, tr, pos, new_val);
        t[v] = t[v*2] + t[v*2+1];
    }
}
```

Finding the maximum and the number of times it appears

```
pair<int, int> t[4*MAXN];

pair<int, int> combine(pair<int, int> a, pair<int, int> b) {
    if (a.first > b.first)
        return a;
    if (b.first > a.first)
        return b;
    return make_pair(a.first, a.second + b.second);
}

void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = make_pair(a[tl], 1);
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = combine(t[v*2], t[v*2+1]);
    }
}

pair<int, int> get_max(int v, int tl, int tr, int l, int r) {
    if (l > r)
        return make_pair(-INF, 0);
    if (l == tl && r == tr)
        return t[v];
    int tm = (tl + tr) / 2;
    return combine(get_max(v*2, tl, tm, l, min(r, tm)),
                  get_max(v*2+1, tm+1, tr, max(l, tm+1), r));
}

void update(int v, int tl, int tr, int pos, int new_val) {
    if (tl == tr) {
        t[v] = make_pair(new_val, 1);
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos, new_val);
        else
            update(v*2+1, tm+1, tr, pos, new_val);
        t[v] = combine(t[v*2], t[v*2+1]);
    }
}
```

Counting the number of zeros, searching for the k-th zero

Same build update and cout_zero functions as sum

In the implementation we can handle the special case, a[] containing less than k zeros, by returning -1.

```
int find_kth(int v, int tl, int tr, int k) {
    if (k > t[v])
        return -1;
    if (tl == tr)
        return tl;
    int tm = (tl + tr) / 2;
    if (t[v*2] >= k)
        return find_kth(v*2, tl, tm, k);
    else
        return find_kth(v*2+1, tm+1, tr, k - t[v*2]);
}
```

Searching for the first element greater than a given amount

```
int get_first(int v, int lv, int rv, int l, int r, int x) {
    if(lv > r || rv < l) return -1;
    if(l <= lv && rv <= r) {
        if(t[v] <= x) return -1;
        while(lv != rv) {
            int mid = lv + (rv-lv)/2;
            if(t[2*v] > x) {
                v = 2*v;
                rv = mid;
            }else {
                v = 2*v+1;
                lv = mid+1;
            }
        }
        return lv;
    }
}

int mid = lv + (rv-lv)/2;
int rs = get_first(2*v, lv, mid, l, r, x);
if(rs != -1) return rs;
return get_first(2*v+1, mid+1, rv, l, r, x);
}
```

Range updates (Lazy Propagation)

Addition on segments

```
void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = 0;
    }
}

void update(int v, int tl, int tr, int l, int r, int add) {
    if (l > r)
        return;
    if (l == tl && r == tr) {
        t[v] += add;
    } else {
        int tm = (tl + tr) / 2;
        update(v*2, tl, tm, l, min(r, tm), add);
        update(v*2+1, tm+1, tr, max(l, tm+1), r, add);
    }
}

int get(int v, int tl, int tr, int pos) {
    if (tl == tr)
        return t[v];
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return t[v] + get(v*2, tl, tm, pos);
    else
        return t[v] + get(v*2+1, tm+1, tr, pos);
}
```

Assignment on segments

```
void push(int v) {
    if (marked[v]) {
        t[v*2] = t[v*2+1] = t[v];
        marked[v*2] = marked[v*2+1] = true;
        marked[v] = false;
    }
}

void update(int v, int tl, int tr, int l, int r, int new_val) {
    if (l > r)
        return;
    if (l == tl && tr == r) {
        t[v] = new_val;
        marked[v] = true;
    } else {
        push(v);
        int tm = (tl + tr) / 2;
        update(v*2, tl, tm, l, min(r, tm), new_val);
        update(v*2+1, tm+1, tr, max(l, tm+1), r, new_val);
    }
}

int get(int v, int tl, int tr, int pos) {
    if (tl == tr) {
        return t[v];
    }
    push(v);
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return get(v*2, tl, tm, pos);
    else
        return get(v*2+1, tm+1, tr, pos);
}
```

Adding on segments, querying for maximum

```
void push(int v) {
    t[v*2] += lazy[v];
    lazy[v*2] += lazy[v];
    t[v*2+1] += lazy[v];
    lazy[v*2+1] += lazy[v];
    lazy[v] = 0;
}

void update(int v, int tl, int tr, int l, int r, int addend) {
    if (l > r)
        return;
    if (l == tl && tr == r) {
        t[v] += addend;
        lazy[v] += addend;
    } else {
        push(v);
        int tm = (tl + tr) / 2;
        update(v*2, tl, tm, l, min(r, tm), addend);
        update(v*2+1, tm+1, tr, max(l, tm+1), r, addend);
        t[v] = max(t[v*2], t[v*2+1]);
    }
}

int query(int v, int tl, int tr, int l, int r) {
    if (l > r)
        return -INF;
    if (l == tl && tr == r)
        return t[v];
    push(v);
    int tm = (tl + tr) / 2;
    return max(query(v*2, tl, tm, l, min(r, tm)),
               query(v*2+1, tm+1, tr, max(l, tm+1), r));
}
```