segTree

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

void InOutFast() {
    ios::sync_with_stdio(false);
    cout.tie(nullptr);
    cin.tie(nullptr);
}

#define bye return 0
#define int long long
#define all(a) ((a).begin()), ((a).end())

struct Node {
    int sum;  // ⚠ Core stored value: change this name and meaning depending on the operation (e.g., max, min, gcd)
    Node() {
        sum = 0;  // ⚠ Neutral value: 0 for sum, LLONG_MIN for max, LLONG_MAX for min, 0 or specific for gcd
    }
    Node(int x) {
        sum = x;  // ⚠ Base value from array
    }
    void change(int x) {
        sum = x;  // ⚠ Used in point update (you can add here if it’s += or something else)
    }
};

struct segTree {
    int treeSize;
    vector<Node> segData;

    segTree(int n) {
        treeSize = 1;
        while (treeSize < n) treeSize *= 2;  // find smallest power of 2 ≥ n
        segData.assign(2 * treeSize, Node());  // initialize all nodes with neutral value
    }

    Node merge(Node &li, Node &ri) {
        Node ans = Node();
        ans.sum = li.sum + ri.sum;  // ⚠ Main operation: change this to max(), min(), __gcd(), etc.
        return ans;
    }

    void set(int idx, int val, int ni, int lx, int rx) {
        int mid = (lx + rx) / 2;
        if (rx - lx == 1) {
            segData[ni].change(val);  // ⚠ Base case: set value at leaf
            return;
        }
        if (idx < mid) {
            set(idx, val, 2 * ni + 1, lx, mid);
        } else {
            set(idx, val, 2 * ni + 2, mid, rx);
        }
        segData[ni] = merge(segData[2 * ni + 1], segData[2 * ni + 2]);  // ⚠ Update internal node after recursion
    }

    void set(int idx, int val) {
        set(idx, val, 0, 0, treeSize);  // public function to call set
    }

    Node get(int l, int r, int ni, int lx, int rx) {
        if (l <= lx && r >= rx) {
            return segData[ni];  // ⚠ Complete overlap
        }
        if (lx >= r || rx <= l) {
            return Node();  // ⚠ No overlap: return neutral value
```

```
        }
        int mid = (lx + rx) / 2;
        Node lf = get(l, r, 2 * ni + 1, lx, mid);
        Node ri = get(l, r, 2 * ni + 2, mid, rx);

        return merge(lf, ri);  // ⚠️ Partial overlap: combine answers
    }

    int get(int l, int r) {
        return get(l, r, 0, 0, treeSize).sum;  // public function to call get
    }
};

int32_t main() {
    InOutFast();
    int n, m;
    cin >> n >> m;
    segTree st = segTree(n);
    vector<int> arr(n);
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
        st.set(i, arr[i]);
    }
    while (m--) {
        int op;
        cin >> op;
        if (op == 1) {
            int idx, val;
            cin >> idx >> val;
            st.set(idx, val);  // ⚠️ Point update
        } else {
            int l, r;
            cin >> l >> r;
            cout << st.get(l, r) << '\n';  // ⚠️ Range query on [l, r)
        }
    }
    bye;
}
```

📊 Table of Common Segment Tree Variants:

Operation	`Node` variable	Neutral value	`merge()` logic	`change(x)` logic
Sum	`sum`	`0`	`a + b`	`sum = x`
Max	`maxVal`	`LLONG_MIN`	`max(a, b)`	`maxVal = x`
Min	`minVal`	`LLONG_MAX`	`min(a, b)`	`minVal = x`
GCD	`g`	`0`	`__gcd(a, b)`	`g = x`
Product	`product`	`1`	`a * b`	`product = x`
XOR	`xorVal`	`0`	`a ^ b`	`xorVal = x`
AND	`andVal`	`ALL_ONES`	`a & b`	`andVal = x`

> 🔷 *`ALL_ONES` = `~0LL` if you're using `long long`.*

> *"أطول subarray داخل [L, R] بحيث GCD > 1"*
> *وده معناه:*

- بدل ما نستخدم `sum` في الـ `Node`، نستخدم `gcd`.
- ونغير `merge` بحيث يبقى `gcd(left, right)`.
- ولما ييجي استعلام [L, R]، هنمشي على كل نقطة داخلها، ونستخدم binary search على أقصى `r' ≥ i` بحيث `GCD(i, r') > 1`.

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

void InOutFast() {
    ios::sync_with_stdio(false);
    cout.tie(nullptr);
    cin.tie(nullptr);
}
#define bye return 0
#define int long long
#define all(a) ((a).begin()), ((a).end())

// ---------- Node now stores gcd ----------
struct Node{
    int gcd;
    Node(){
        gcd = 0;
    }
    Node(int x){
        gcd = x;
    }
    void change(int x){
        gcd = x;
    }
};

// ---------- Segment Tree ----------
struct segTree{
    int treeSize;
    vector<Node> segData;

    segTree(int n){
        treeSize = 1;
        while(treeSize < n) treeSize *= 2;
        segData.assign(2 * treeSize, Node());
    }

    Node merge(Node &li , Node &ri){
        Node ans;
        ans.gcd = __gcd(li.gcd, ri.gcd);
        return ans;
    }

    void set(int idx, int val, int ni, int lx, int rx){
        if(rx - lx == 1){
            segData[ni].change(val);
            return;
        }
        int mid = (lx + rx)/2;
        if(idx < mid){
            set(idx, val, 2*ni+1, lx, mid);
        } else {
            set(idx, val, 2*ni+2, mid, rx);
        }
        segData[ni] = merge(segData[2*ni+1], segData[2*ni+2]);
    }

    void set(int idx, int val){
        set(idx, val, 0, 0, treeSize);
    }

    Node get(int l, int r, int ni, int lx, int rx){
        if(l >= rx || lx >= r) return Node();
        if(l <= lx && rx <= r) return segData[ni];
        int mid = (lx + rx)/2;
        Node lf = get(l, r, 2*ni+1, lx, mid);
        Node ri = get(l, r, 2*ni+2, mid, rx);
        return merge(lf, ri);
    }

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

```
        return get(l, r, 0, 0, treeSize).gcd;
    }
};

int32_t main() {
    InOutFast();
    int n, m;
    cin >> n >> m;
    segTree st(n);
    vector<int> arr(n);
    for(int i = 0; i < n; i++){
        cin >> arr[i];
        st.set(i, arr[i]);
    }

    while (m--){
        int op;
        cin >> op;
        if(op == 1){
            int idx, val;
            cin >> idx >> val;
            st.set(idx, val);
        }
        else{
            int l, r;
            cin >> l >> r;
            int ans = 0;
            for(int i = l; i < r; i++){
                int low = i, high = r - 1, best = i - 1;
                while(low <= high){
                    int mid = (low + high) / 2;
                    int g = st.getGCD(i, mid + 1);
                    if(g > 1){
                        best = mid;
                        low = mid + 1;
                    } else {
                        high = mid - 1;
                    }
                }
                ans = max(ans, best - i + 1);
            }
            cout << ans << '\n';
        }
    }

    bye;
}
```

```
for (int i = l; i < r; i++) {
    binary search on j in [i, r) such that GCD(i..j) > 1
}
```

- نحاول نمد subarray من i بأقصى قدر ممكن طالما GCD > 1.
- لو وصلنا لـ GCD = 1، نوقف.