সেগমেন্ট ট্রি (Segment tree)

① Array তে particular range এর মধ্যে যোগ, বড়, ছোট ইত্যাদি বের করে।

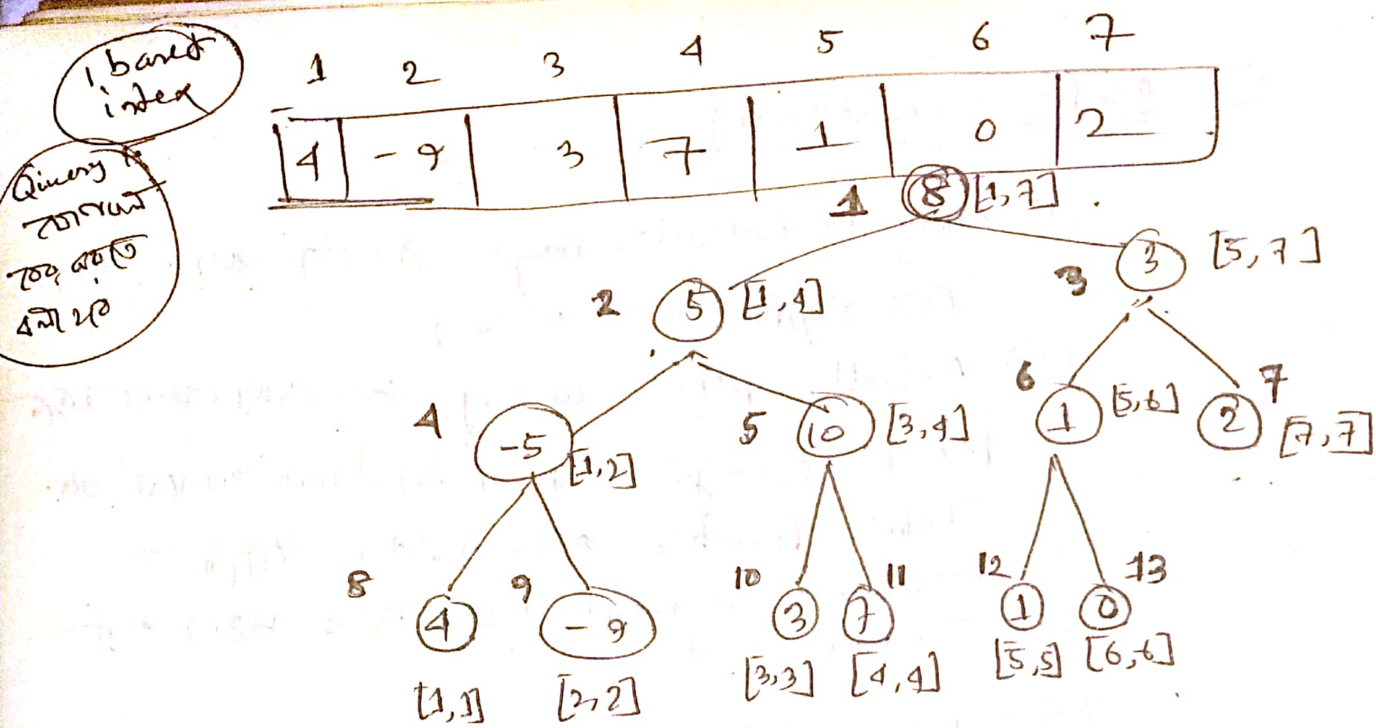
② Actually বিভিন্ন Query, যেমন দেওয়া থাকবে যদি বিভিন্ন range এ বিভিন্ন data আপডেট এবং অন্যান্য function করতে বলবে। তখন আমরা segment tree দিয়ে সহজে করা যায়।

③ Range Minimum Query type এর problem দেওয়া থাকে প্রতিযোগিতার সময়, এগুলো নরমাল loop চালিয়ে করতে গেলে অবশ্যই TLE খাবে। Cause যেহেতু 10^5 বা তারও বেশি data থাকে সেটা ঐ Time limit এর মধ্যে করা সম্ভব হয় না।

নিচে একটা নিয়ে বিস্তারিত আলোচনা করা হল :-

4 -9 3 7 1 0 2

এই একটা Array দেওয়া আছে। first step হচ্ছে tree টা তৈরি করা। তারপর ঐ tree কে use করে আমরা Query গুলোর ans খুঁজে বের করব।

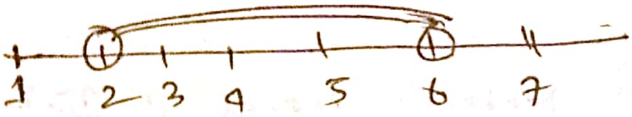


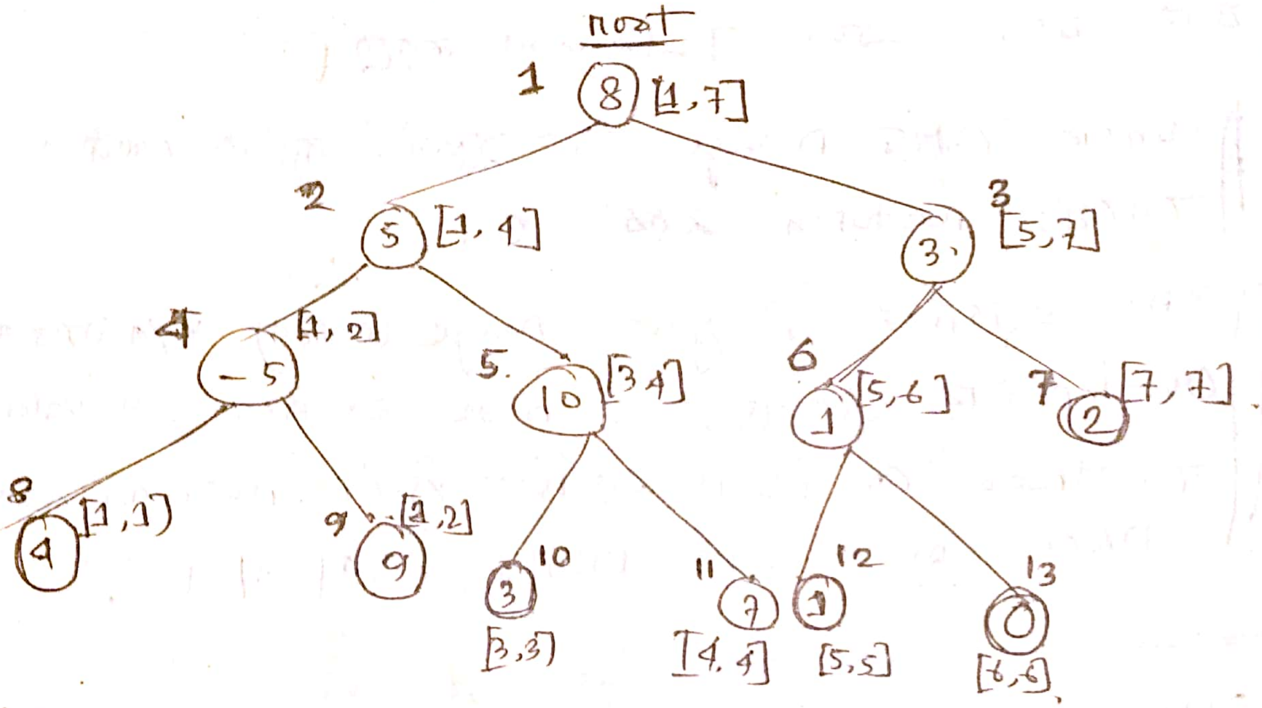
// Bold করা নাম্বার গুলো হচ্ছে সেগমেন্ট ট্রির indexing. অনুযায়ী এই রকম একটি Array তে করে রাখা হবে। অর্থাৎ, প্রত্যেকের দুইটি করে child থাকবে, একটি left child আর একটি right child.

left = node*2;

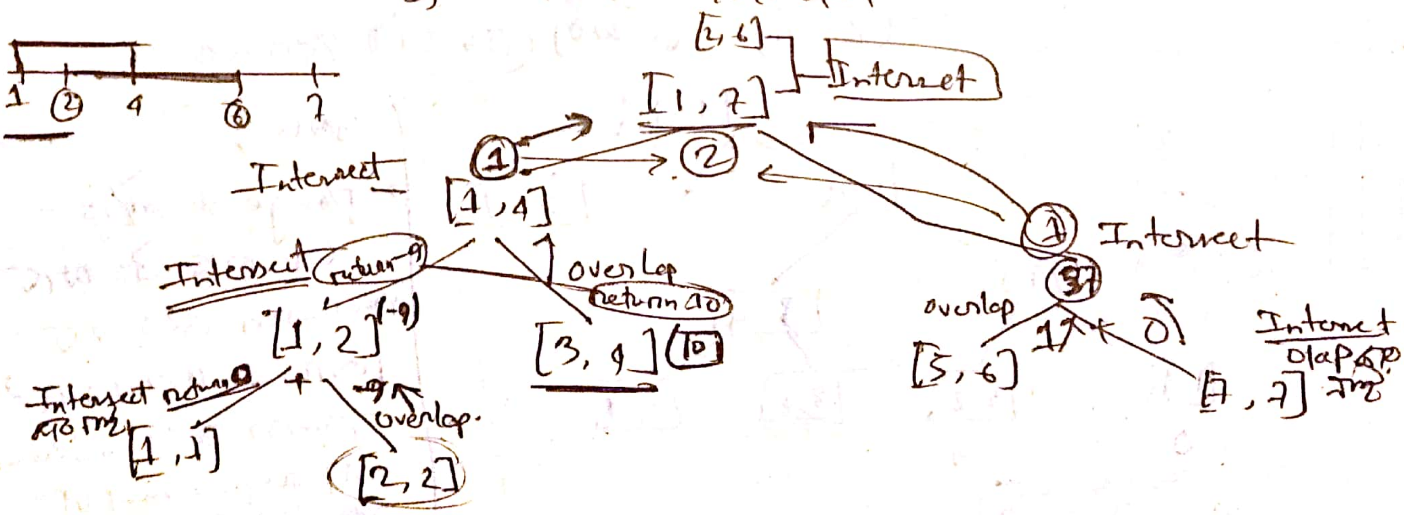
right = node*2+1.

প্রশ্ন 2 থেকে 6 নাম্বার Index এর যোগ ফল বের করার উপায় দেখাও ?





2 থেকে 6 নাম্বার Index এর ভেলুগুলো যোগ করো। আমরা শুরু করি 1 No node থেকে অর্থাৎ root থেকে। root এর range (1,7)। এখানে Intersect করছে (2,6) কে। সেজন্য বামে ও ডানে দুই জায়গায় যেতে হবে।



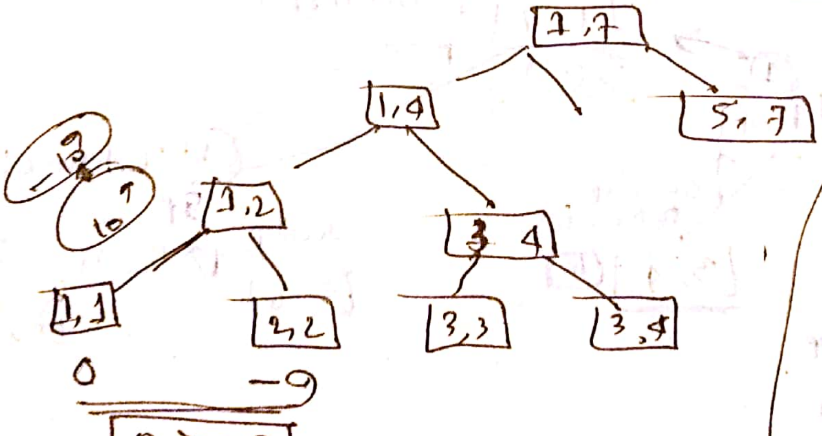
এক্ষেত্রে পুরোটাই Node এর range এর দুইদিকে থাকে যায় অথবা একটা Intersect করবে।

যেখানে কোনো range টা পুরোটাই বাইরে গেলে সেখানে return হবে 0।

আর পুরোটাই ঐ given range এর মধ্যে থাকে তাহলে আমি Overlap করে সেক্ষেত্রে ঐ node এর মধ্যে যে 'value' টা tree-তে fixed করেছি সেটা return করবো। আমরা একটার আর Divide করবো না।

যখন Query র মধ্যে বড়/ছোট বের করার কথা বলা হবে সেক্ষেত্রে একই নিয়ম-ই থাকবে। তবে শুধু Return করার সময় মান পাল্টাতে হবে। যেমন একটা leaf level এর Example দেখা যাক। আমরা:

(2-6) এর মধ্যে (বড় টা) বের করব



0   -9

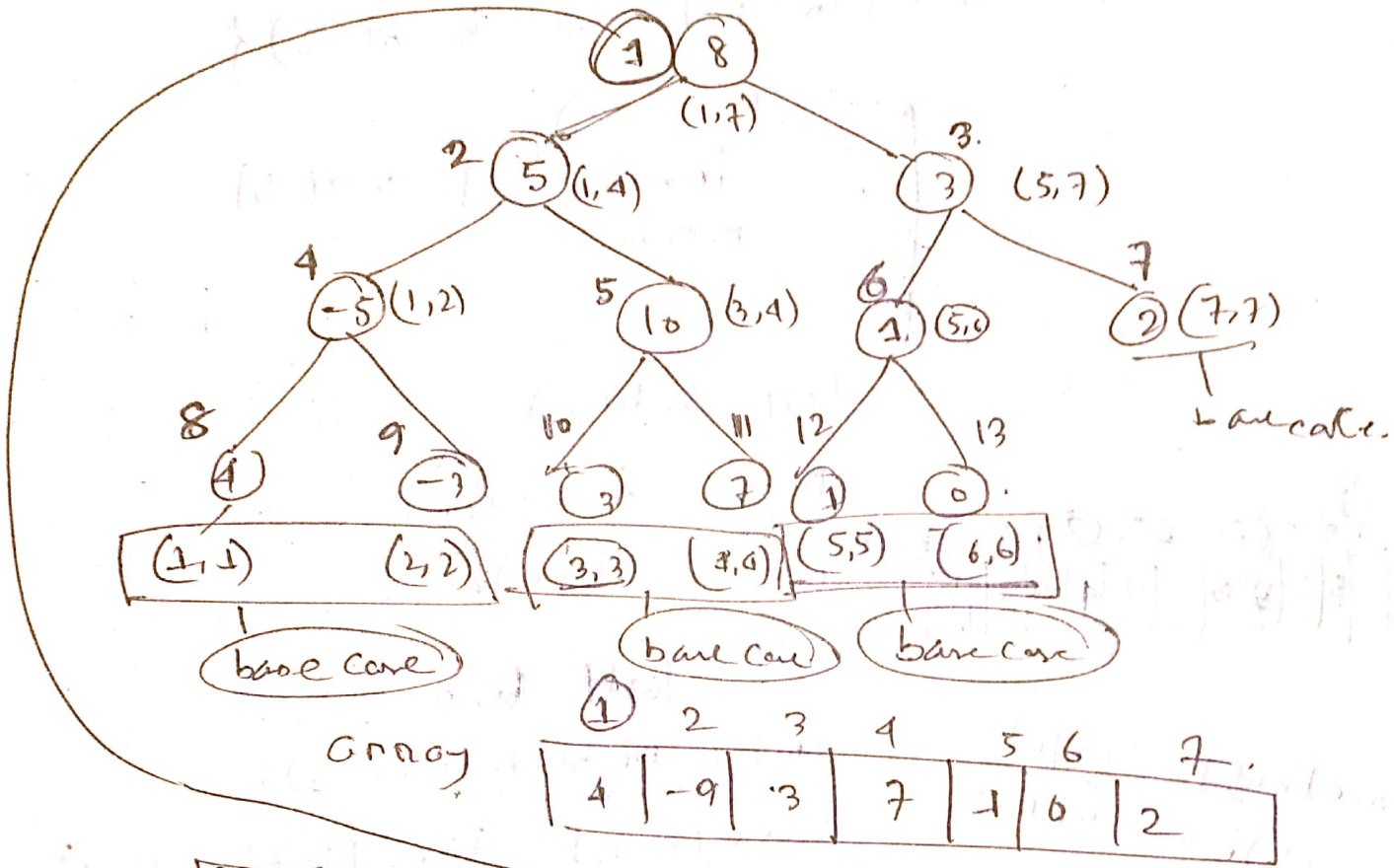
0 > -9

এখানে, range এর বাইরে হলে তাকে এমন মান return করব যে মানটা বড়/ছোট হবে না। এমন একটা value এটা requirement এর উপর নির্ভর করবে।

তাই বড় টা return করব এখানে। তখন বড় হচ্ছে 0। for 0 range এর বাইরে থাকলে কি করব?

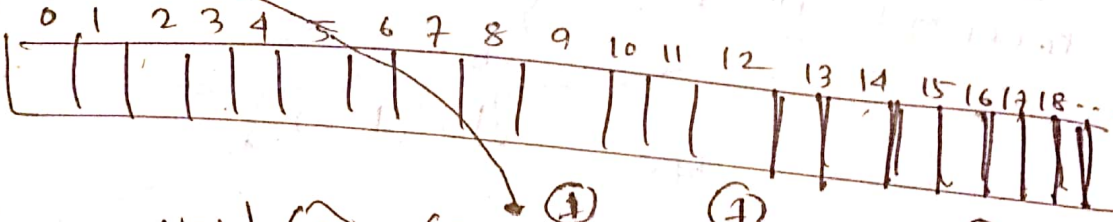
Implementation of segment tree

base
left = right



2*3=24

tree[arr*3] =



void func (int node, int left, int right)

if (b == e)

{ return ;

left = node*2 =

right = node*2+1 =

mid = (left+right)/2.

arr =

1	2	3	4	5	6	7
4	-9	3	7	-7	0	2

tree = [arr × 3] =

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

```
void init (int node, int b, int e){
    if (b == e){
        tree[node] = arr['b]
        return
    }
    left = (node * 2) =
    right = (node * 2) + 1
    mid = (b + e) / 2;
    init (left, b, mid);
    init ( right, mid+1, e);
    tree[node] = tree[left] + tree[right]
}
```

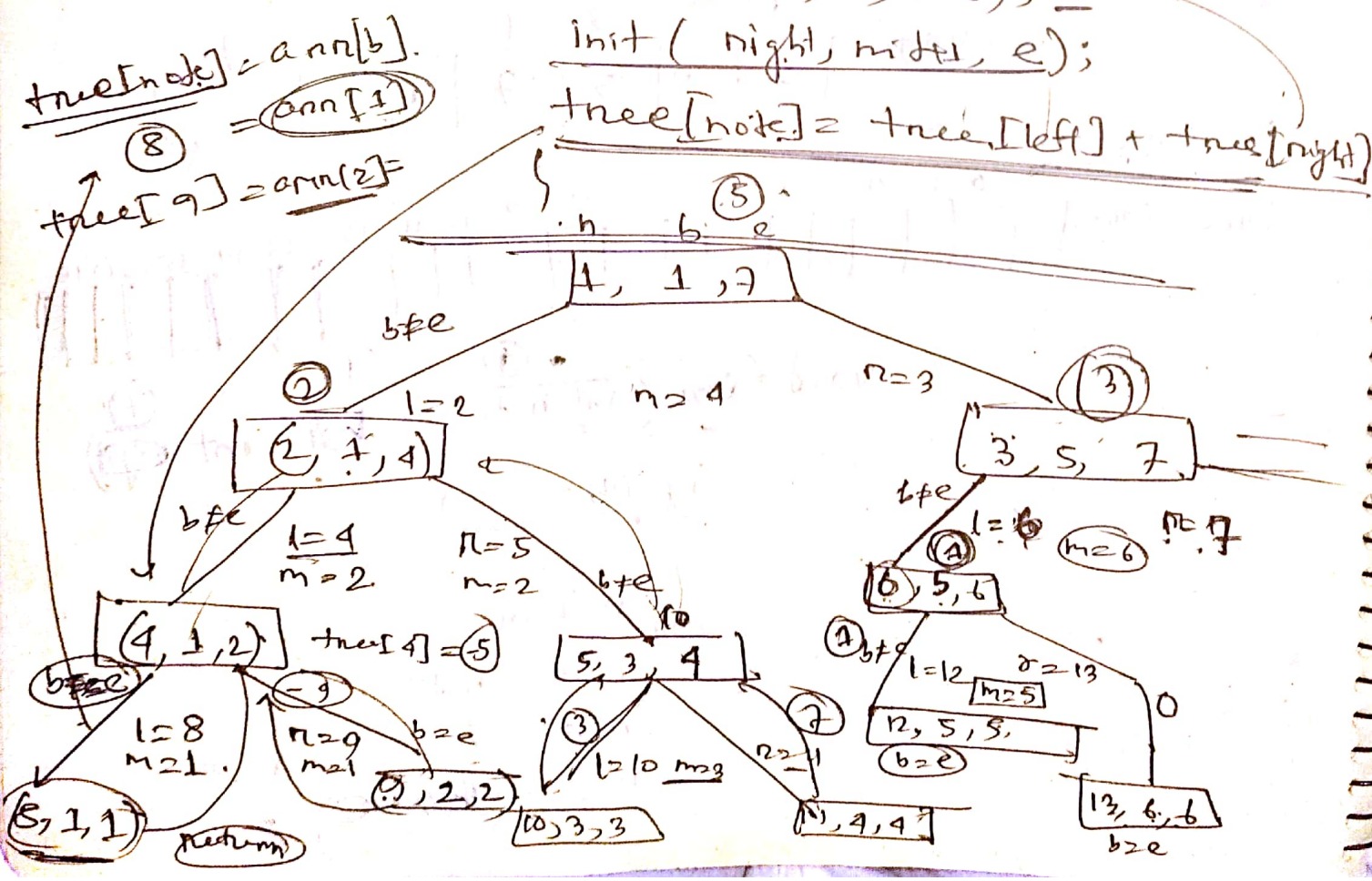
tree

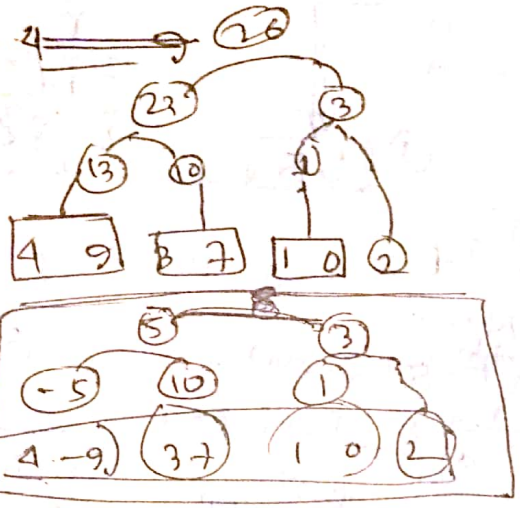
1	2	3	4	5	6	7	8	9	10	11	12	13
5	3	5	10	2	4	-9	3	7	1	0		

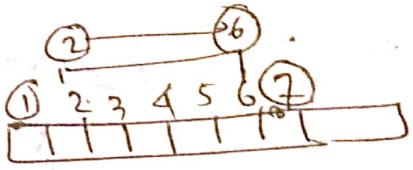
tree[node] = arr[b]

⑧ = arr[1]

tree[9] = arr[2]





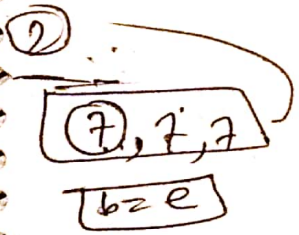


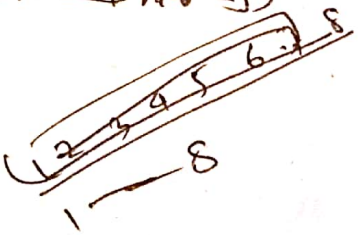
এখন আমরা tree টি তৈরি করে নেব, এই tree টি তৈরি করে এখন আমরা এই tree তে Query করব। এখানে যোগফলের সীমা ① এবং ① এর মধ্যে value (range) যত এবার তার সংখ্যাগুলো যোগফলটি হবে। সেখানে আমরা Query function টি লেখব।

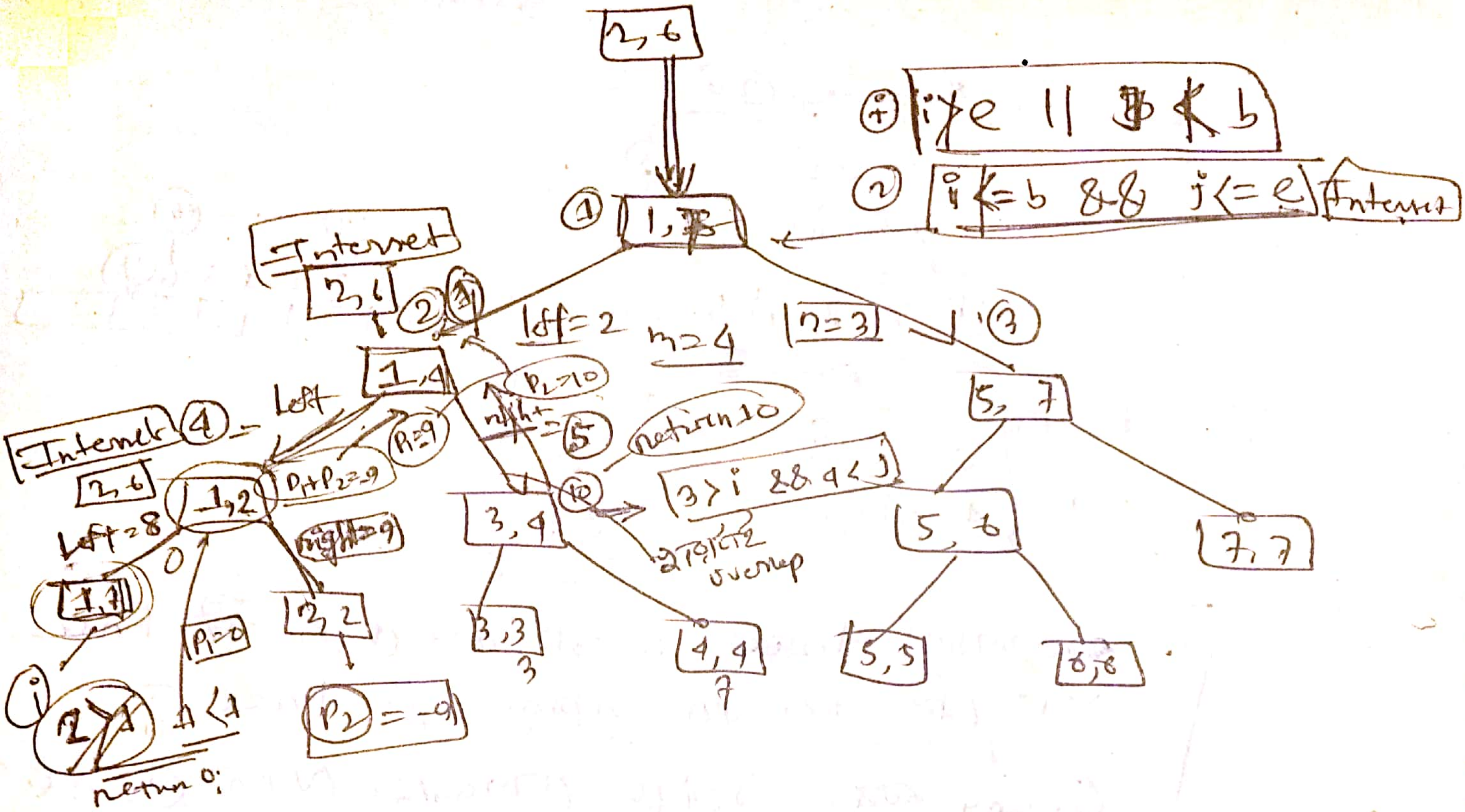
① ⑦ ② — ⑥

```
int Query(int node, int b, int e, int i, int j)
{
    if(i>e || j<b)
        return 0;
    if(b>=i && e<=j)
        return tree[node];
    left = node*2;
    right = node*2+1;
    mid = (b+e)/2;
    p1 = Query(left, b, mid, i, j)
    p2 = Query(right, mid+1, e, i, j);
    return p1+p2;
}
```

যোগফল সীমা

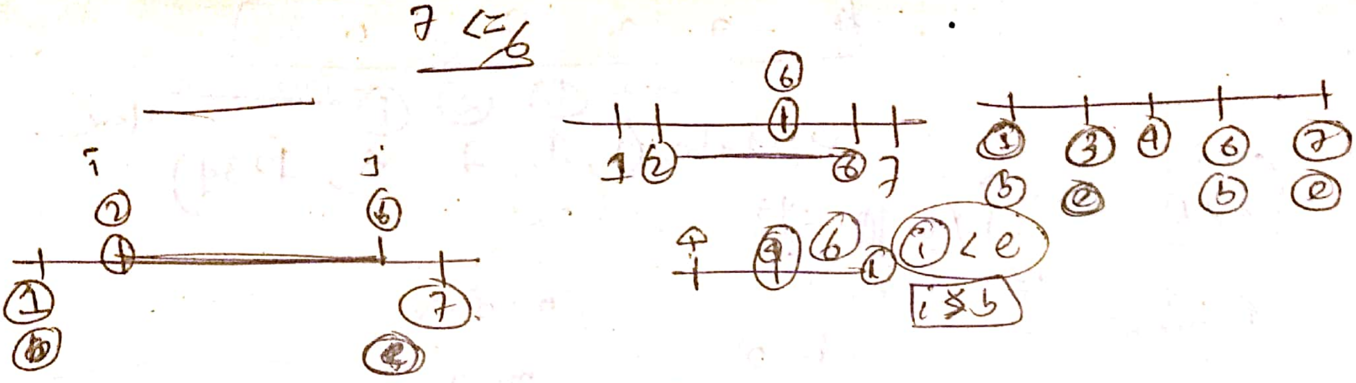






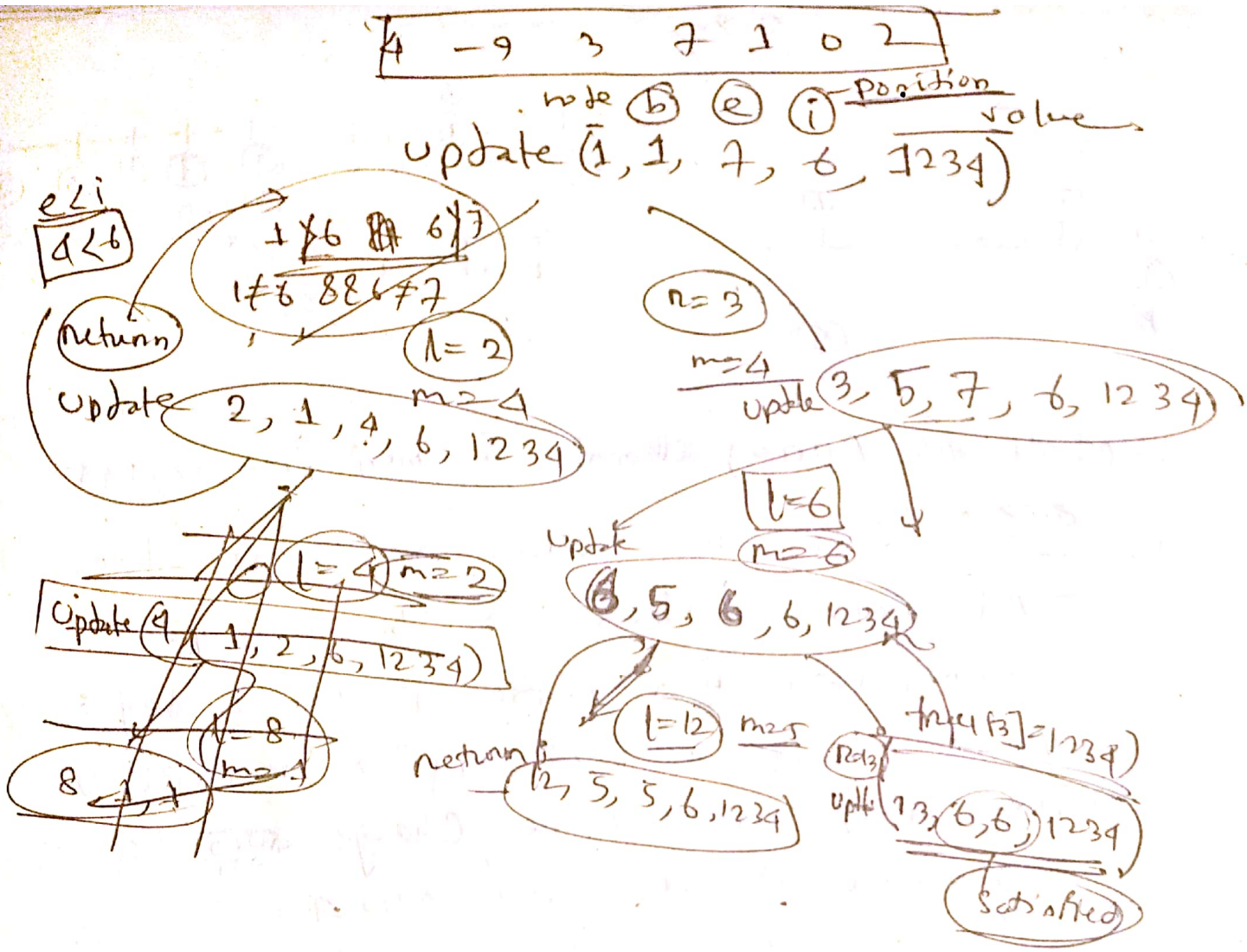
। এভাবে recursive করলে আমরা segmented tree এ-

থেকে নির্দিষ্ট ইন্টারভ্যাল এর sum পেতে পারি।



Update করা Array element Segment tree ব্যবহার করে। এক্ষেত্রে আমাদের same - ই। অনুযায়ী tree তৈরি করতে হবে। then -ই- tree তে update array তথ্য দিয়ে ই Position -এ tree[node] এর value change করে দিতে হবে। এর প্রত্যেকটি ওই অভ্যন্তর node গুলোর value Change করতে হবে নতুন value এর মান সেট করতে হবে।

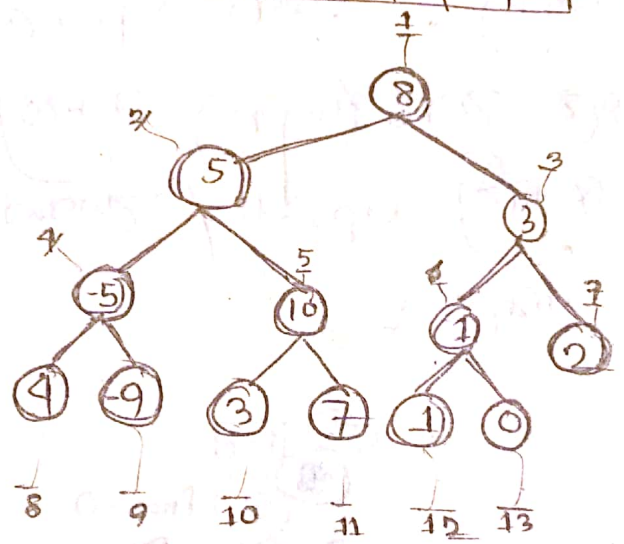
```
Update (node, b, e, i, val)
    if (b>i || e<i) return;
    if (b==i && e==i) {
            tree[node] = val;
        }. return;
    left = node*2
    right= node*2+1;
    mid= (b+e)/2;
    update (left, b, mid, i, val);
    update( right, mid+1, e, i, val);
    tree[node]=tree[left] + tree[right];
    }
}.
```



Segment tree with lazy propagation:

Let's have a segment tree:

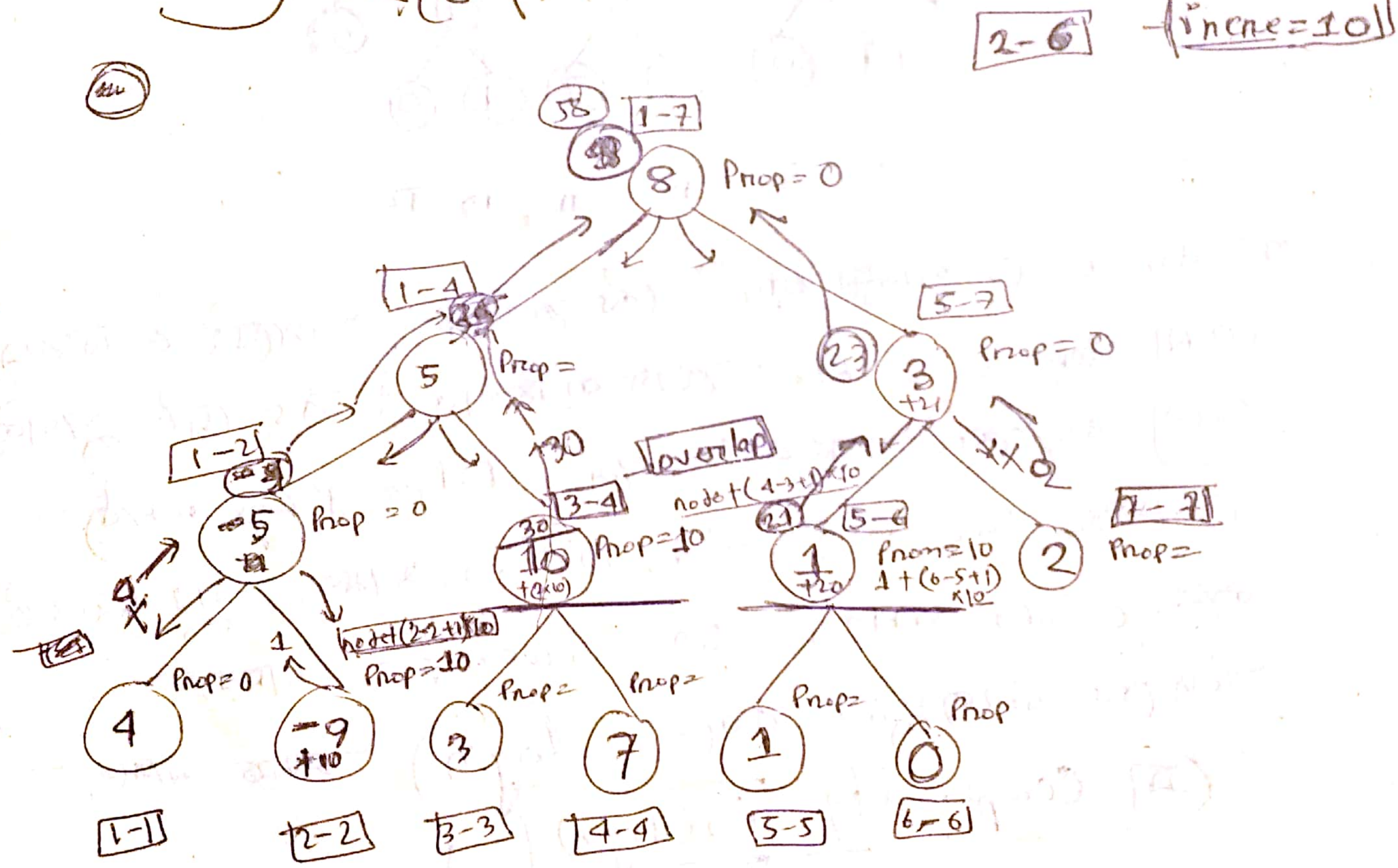
4	-9	3	7	1	0	2



এই tree তে আমাদের যদি কেউ কোনো কিছু আপডেট করতে চায়, কোনো element এর মান যোগ বা বিয়োগ, বড় ছোট, কোনটি সবচেয়ে বড়/ছোট বের করতে বলে, নির্দিষ্ট Range এর মধ্যে যোগফল বের করতে বলে, সেখানে আমরা খুব সহজেই করে ফেলতে পারি। এই tree তে আমরা উপরের কাজগুলো করতে (log n) সময় লাগে, যার Complexity O(log n)।

এখন এক্ষেত্রে যদি আমাদের কোনো একটা (নির্দিষ্ট রেঞ্জ) দেওয়া থাকে এবং এই range এ সব আপডেট করতে বলা হয় সেখানে আমরা যদি এই প্রক্রিয়া অনুসরণ করি

তবে (ঐ নোডের ভ্যালুর জন্য) একবার করে কাজ করতে হয়।
আমরা জানি একেকে একবারের জন্য Complexity O(logn).
worst case → যখন Range এর জন্য Complexity
দাঁড়াবে O(nlogn). এখানে আমাদের অন্য Algorithm
কাজ করবে যেটি O(logn) এর মধ্যে যেকোন অন্য
Range এর মধ্যে update/ query and others এর করতে
সাহায্য করবে।

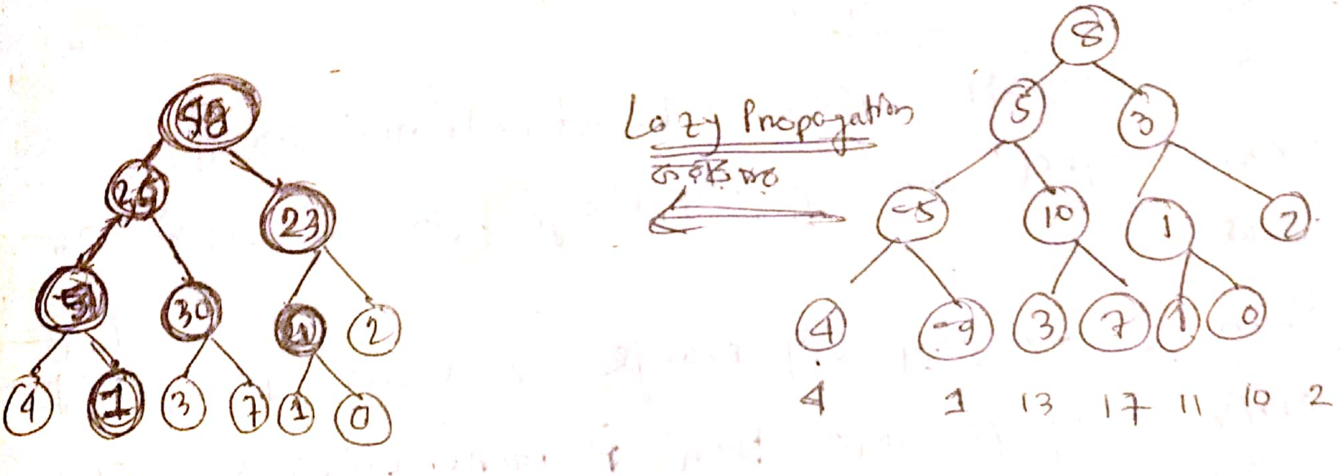


এটা আমাদের Basic tree. এখন বলা হল [2-5] এই Range এর মধ্যে যারা আছে তাদের প্রত্যেকের মান 10 করে বাড়ানো হোক।

সেক্ষেত্রে আমরা কি করব?

আমরা এক্ষেত্রে আগের মত - Leaf node পর্যন্ত যাব না। এক্ষেত্রে আমরা overlap করা Range পেলে সেখানেই থেমে যাব।

then ঐ overlap করা range এর under-এ যতগুলো leaf node আছে সেটা বের করে, then ঐ increment কে গুণ করে ঐ জায়গায়/node এ update করব। এবং আমরা যত Increment করলাম তাকে ঐ জায়গায়/node এর propagate value হিসেবে রেখে আসব। এক্ষেত্রে সবচেয়ে গুরুত্বপূর্ণ কথা হল: আমরা যেই node overlapped range টাকে এভাবে update এর জন্য পেলাম তার under এ যে leaf গুলো আছে (যেগুলো আমাদের Query তে update এর জন্য বলা হয়েছে), আমি কিন্তু তাদের এই loop গুলোর value increment করি নি। কারণ এরা যে overlap range -এ belong করেছে তাদের ঐ range node কে update করেছি ও যত update করলাম তার জন্য data prop এর মধ্যে রেখে আসলাম।



আমাদের (2-6) range এ 10 Increment করতে বলা হয়েছে। কিন্তু আমরা কোন না করেই update করে ফেললাম।

4 -9 3 7 1 0 2 ⇒ total = 8

4 1 13 17 11 10 2 ⇒ total = 58

এক্ষেত্রে খুব কম সহজেই update করলাম কিন্তু ঐ যেই Range কে update করতে বলা হয় সেগুলো update করলাম তো না কিন্তু। কিন্তু আমাদের Root node এ পুরো Array টা নতুন updated sum আছে। এক্ষেত্রে আমরা একটা struct use করব যেটা tree এর Array বানাবো।

```
struct info{
    int sum;
    int prop;
} tree[mx*4];

int array[mx];
```

→ Array-র size হতে 4 গুন বড় হবে।

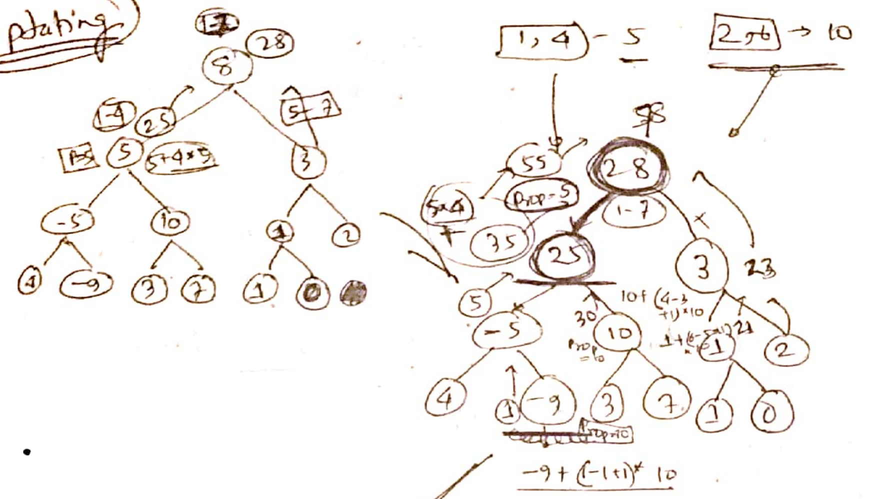
```
void update( int node, int b, int e, int x, int y, int val ){
    if ( b>y || e<x) return;   [out of (x-y)]
    if ( b>=x && e<=y){
        tree[node].sum = tree[node].sum + (e-b+1) * val;
        tree[node].prop += val;
        return; }
    int left, right, mid;
    left = node*2;
    right = node*2+1;
    mid = (b+e)/2;
    update( left, b, mid, x, y, val);
    update( right, mid+1, e, x, y, val);

    tree[node].sum = tree[left].sum + tree[right].sum +
                     (tree[node].prop * (e-b+1));
}
```

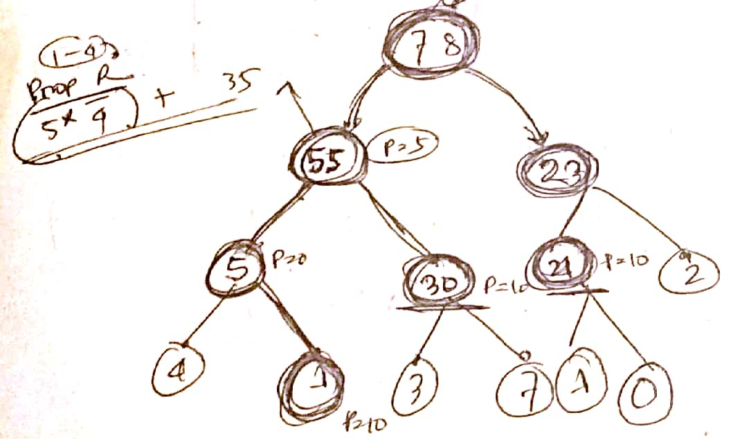
∴ এইতো গেল update করার ব্যাপার।

এবার প্রশ্নটা আসছে Query প্রশ্ন?

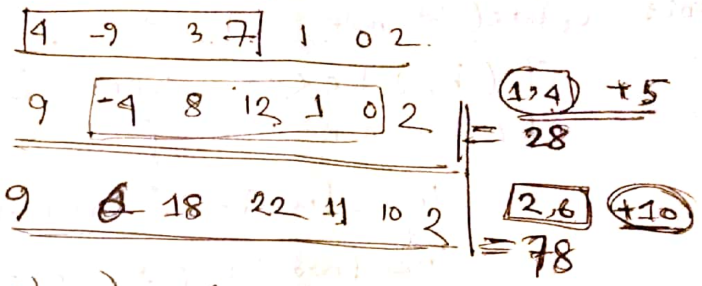
এখন আমাকে প্রশ্ন করা হল — (2-5) Range এর After update হবে value গুলো আছে তাদের যোগফল জিজ্ঞাসা করা হলে—

Updating





একেই বলা হয় Segment tree Lazy propagation.



এই দুই update শেষে আমরা যেভাবে বলেছি Array-র অবস্থা (প্রতিটি ক্ষেত্রে যে সংখ্যাগুলো সহ) seg tree এর অবস্থা দেখতে ঠিক এমন হবে। Tree এর অবস্থা দেখছো আগের page-এ।

এখন যদি বলা হয় (2—5) এর মধ্যে যে element গুলো আছে, সেগুলোর যোগফল দেখাও। তাহলে কি সমস্যা? তবে আমি যে (3 থেকে 4, 5) এর element গুলোর জন্য update-ই করি নাই। tree তে ওরা update করার আগেই চলে গেছি। এখন কি করা যায়?

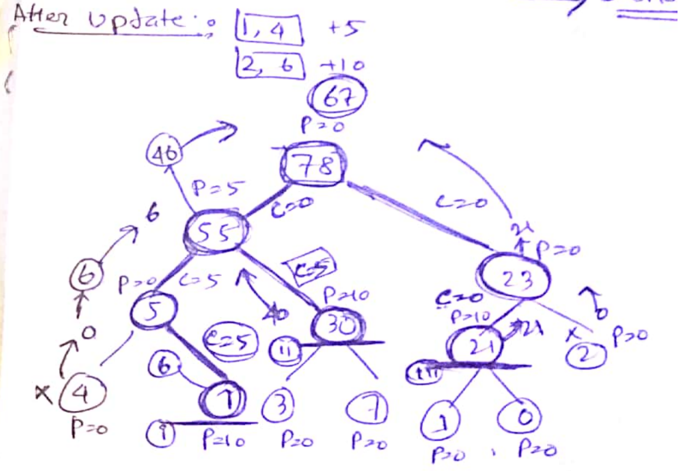
এবার Query করার সময় আমাদের Carry use করতে হবে। Carry-র কাজ কি?

Carry-র কাজ হবে ইন্ডেক্স এর element কে কতবার Increment করা হয়েছে সেই Info দিয়ে রাখা। কিন্তু Carry কিভাবে?

বাহ!! prop নামে একটাকে ব্যবহার করেছিলাম মনে আছে? তার কাজ ছিল কত Increment করলাম তার proof রাখা। এখন carry তে আমরা কোনো element/ leaf এ পৌঁছাতে যতগুলো prop পাব সব যোগ করে রাখব। আর ঐ leaf কে নিচে যত Increment করা হয় carry তে থাকবে। Carry += prop; আর এইটাকে ঐ leaf value র সাথে যোগ করলেই কাজ শেষ। এরপর বাকিটা আগের মতোই Return করে দিব। শেষ (সমাপ্তি) :) .

```
int Query(int node, int b, int e, int x, int y, int carry){
    if(b>y || e<x) return 0;
    if(b>=x && e<=y){
        return tree[node].sum+(e-b+1)*carry;
    }
    int left, right, mid;
    left = node*2;
    right = node*2+1;
    mid = (b+e)/2;

    int p1 = (left, b, mid, x, y, carry+tree[node].prop);
    int p2 = (right, mid+1, e, x, y, carry+tree[node].prop);
    return p1+p2;
}
```

Query Process



(i) $1+(c*(e-b+1))$

$1+(5*1) = 6$

(ii) $30+(c*(e-b+1))$

$30+(5*2) = 40.$

(iii) $21+(c*(e-b+1))$

$21+(0*2) = 21$