

```

302 Dynamic segment tree with lazy:
303 #define left l , (l+r)>>1
304 #define right ((l+r)>>1)+1 , r
305
306 struct DynamicSegTree{
307     struct Vertex {
308         int value , lazy , l , r;
309         Vertex *lChild, *rChild;
310
311         Vertex(int value , int l , int r) :
312             lChild(nullptr), rChild(nullptr),
313             value(value), lazy(0) , l(l) , r(r){}
314
315         void extend() {
316             if(lChild==nullptr){
317                 lChild = new Vertex(0 , left);
318                 rChild = new Vertex(0 , right);
319             }
320         }
321     };
322     Vertex *root;
323     int n;
324
325     DynamicSegTree(int n) : n(n){
326         root = new Vertex(0 , 0 , n);
327     }
328
329     void push(Vertex *p){
330         if(!(p->lazy)) return;
331         //product in size of segment if you want all element to inc
332         p->lChild->value+=p->lazy; p->rChild->value+=p->lazy;
333         p->lChild->lazy+=p->lazy; p->rChild->lazy+=p->lazy;
334         p->lazy = 0;
335     }
336
337     void update(int i , int j , int val){
338         update(i , j , val , root);
339     }
340
341     int update(int i , int j , int val , Vertex *p){
342         if(j<p->l || p->r<i) return p->value;
343         if(i<=p->l && p->r<=j){
344             p->lazy+=val; //to change
345             p->value+=val; //to change
346             return p->value;
347         }
348         p->extend(); push(p);
349         return p->value = (update(i , j , val , p->lChild) +
350                         update(i , j , val , p->rChild)); //to change
351     }
352
353     int query(int i , int j){
354         return query(i , j , root);
355     }
356
357     int query(int i , int j , Vertex *p){
358         if(j<p->l || p->r<i) return 0; //to change
359         if(i<=p->l && p->r<=j) return p->value;
360         p->extend(); push(p);
361         return (query(i , j , p->lChild) + query(i , j , p->rChild)); //to change
362     }
363 };

```