LU_dAREdevils
Leading University

```
pair<ll, ll> extendedEuclid(ll a, ll b) // returns x, y;
ax + by = gcd(a,b)
{
   if (b == 0) return {1, 0};
   else
   {
      pair<ll, ll> d = extendedEuclid(b, a % b);
      return {d.y, d.x - d.y * (a / b)};
   }
}
ll modularInverseEE(ll a, ll Mod)
{
   pair<ll, ll> ret = extendedEuclid(a, Mod);
   return ((ret.x % Mod) + Mod) % Mod;
}

ll modularInverseFL(ll A, ll B) // (A / B) % mod
{
   ll inverse = ((A % mod) * (Pow(B, mod - 2) %
mod)) % mod; // (A * B^-1) % mod
   return (inverse + mod) % mod;
}
```

Segment Tree:

```
const int N = 3e5 + 9; // start
int a[N];
int tree[4 * N];
void build(int node, int st, int en)    //=> O(N)
{
   if (st == en)
   {
      tree[node] = a[st];
      return;
   }
   int mid = (st + en) / 2;
   build(2 * node, st, mid);
   build(2 * node + 1, mid + 1, en);
   tree[node] = tree[2 * node] + tree[2 * node + 1];
}
int query(int node, int st, int en, int l, int r)
 //=> O(logn)
{
   if (st> r || en< l) {
      return 0;
   }
   if (l <= st && en<= r) {
      return tree[node];
   }
   int mid = (st + en) / 2;
   int q1 = query(2 * node, st, mid, l, r);
   int q2 = query(2 * node + 1, mid + 1, en, l, r);
   return q1 + q2;
}
```

```
void update(int node, int st, int en, int idx, int val)
//=> O(logn)
{
   if (st == en) {
      a[st] = val;
      tree[node] = val;
      return;
   }
   int mid = (st + en) / 2;
   int left = 2 * node, right = 2 * node + 1;
   if (idx<= mid) update(left, st, mid, idx, val);
   else update(right, mid + 1, en, idx, val);
   tree[node] = tree[left] + tree[right];
} // end
```

Lazy segment tree: **// start**

```
const int N = 5e5 + 9;
int a[N];
struct ST {
   #define lc (n << 1)
   #define rc ((n << 1) | 1)
   ll t[4 * N], lazy[4 * N];
   ST() {
      for (int i = 0; i< 4 * N; i++)
         t[i] = lazy[i] = 0;
   }
   inline void push(int n, int st, int en)
   {
      if (lazy[n] == 0) return;
      t[n] = t[n] + lazy[n] * (en - st + 1);
      if (st != en) {
         lazy[lc] = lazy[lc] + lazy[n];
         lazy[rc] = lazy[rc] + lazy[n];
      }
      lazy[n] = 0;
   }
   inline void pull(int n) {
      t[n] = t[lc] + t[rc];
   }
   void build(int n, int st, int en) {
      lazy[n] = 0;
      if (st == en) {
         t[n] = a[st];
         return;
      }
      int mid = (st + en) >> 1;
      build(lc, st, mid);
      build(rc, mid + 1, en);
      pull(n);
   }
   void update(int n, int st, int en, int l, int r, ll v)
   {
      push(n, st, en);  // push the value left and
                           right child
      if (r <st || en< l) return;
```