```
    if (l <= st && en<= r) {
      lazy[n] = v; // set lazy
      push(n, st, en);
      return;
    }
    int mid = (st + en) >> 1;
    update(lc, st, mid, l, r, v);
    update(rc, mid + 1, en, l, r, v);
    pull(n);
  }
  inline ll combine(ll a, ll b) {
    return a + b;
  }
  ll query(int n, int st, int en, int l, int r) {
    push(n, st, en);
    if (l >en || st> r) return 0; // return null
    if (l <= st && en<= r) return t[n];
    int mid = (st + en) >> 1;

    return combine(query(lc, st, mid, l, r),
                   query(rc, mid + 1, en, l, r));
  }
} st; // end lazy
```

Binary Indexed tree(BIT):

```
/* 1'base indexing */ start
const int N = 3e5 + 9;
ll bit1[N];
ll bit2[N];
ll n;
void update(lli, ll x, ll *bit)          // O(logn)
{
while (i < N) {
   bit[i] += x;
   i += (i & (-i));
}
}
ll query(ll i, ll *bit)          // O(logn)
{
  ll sum = 0;
  while (i> 0) {
    sum += bit[i];
    i -= (i & (-i));
  }
  return sum;
}
void rupdate(ll l, ll r, ll val) {
   update(l, val, bit1);
   update(r + 1, -val, bit1);

   update(l, val * (l - 1), bit2);
   update(r + 1, -val * r, bit2);
}
ll rquery(ll l, ll r) {
ll sum1 = query(r, bit1) * r - query(r, bit2);
ll sum2 = query(l-1, bit1) * (l-1) - query(l-1, bit2);
return sum1 - sum2;
} // end
```

Mo's offline Query:

```
//=> O((N+Q)*sqrt(N))

const int N = 1e6 + 10;
int rootN;
struct Q {
  int l, r, idx;
};
Q q[N];
bool comp(Q q1, Q q2) {
  if (q1.l / rootN == q2.l / rootN) return q1.r >
q2.r;
  return q1.l / rootN< q2.l / rootN;
}
int main() {
  int n;
  cin>> n;
  int a[n];
  for (int i = 0; i< n; ++i) cin>> a[i];
  int query;
  cin>> query;
  rootN = sqrtl(n) + 1;
  for (int i = 0; i< query; ++i) {
    int l, r;
    cin>> l >> r;
    q[i].l = l;
    q[i].r = r;
    q[i].idx = i;
  }
  sort(q, q + query, comp);
  int curr_l = 0, curr_r = -1, l, r;
  llcurr_ans = 0;
  vector<ll>ans(query);
  for (int i = 0; i< query; i++) {
    l = q[i].l, r = q[i].r;
    --l, --r;
    while (curr_r< r) {
      ++curr_r;
      curr_ans += a[curr_r];
    }
    while (curr_l> l) {
      --curr_l;
      curr_ans += a[curr_l];
    }


    while (curr_l< l) {
      ++curr_l;
      curr_ans -= a[curr_l];
```