Segment Tree - Binary Search on Unsorted Array:

```
// 1 based indexing
struct SegTree {
  int n;
  vector<int> seg;
  SegTree(int _n) {
    n = _n;
    seg.assign(4 * n + 5, 0);
  }
  void build(int node, int l, int r) {
    if (l == r) {
      seg[node] = 1;
      return;
    }  // each position initially present
    int mid = (l + r) >> 1;
    build(node * 2, l, mid);
    build(node * 2 + 1, mid + 1, r);
    seg[node] = seg[node * 2] + seg[node * 2 + 1];
  }
  // set position pos to value val (0 or 1)
  void update(int node, int l, int r, int pos, int val) {
    if (l == r) {
      seg[node] = val;
      return;
    }
    int mid = (l + r) >> 1;
    if (pos <= mid)
      update(node * 2, l, mid, pos, val);
    else
      update(node * 2 + 1, mid + 1, r, pos, val);
    seg[node] = seg[node * 2] + seg[node * 2 + 1];
  }
  // find index of k-th "present" element (1-based k)
  int kth(int node, int l, int r, int k) {
    if (l == r) return l;
    int leftCnt = seg[node * 2];
    int mid = (l + r) >> 1;
    if (k <= leftCnt)
      return kth(node * 2, l, mid, k);
    else
      return kth(node * 2 + 1, mid + 1, r, k - leftCnt);
  }
};

void solve() {
  int n;
  cin >> n;
  vector<ll> a(n + 1), p(n + 1);
  for (int i = 1; i <= n; ++i) {
    cin >> a[i];
  }
  for (int i = 1; i <= n; ++i) {
    cin >> p[i];
  }
  SegTree st(n);
  st.build(1, 1, n);
  // For each removal request p[i], find the p[i]-th present
element,
  // print it and mark that position as removed (set to 0).
  for (int i = 1; i <= n; ++i) {
    int k = p[i];
    int idx = st.kth(1, 1, n, k);  // index in original array
    cout << a[idx] << (i == n ? '\n' : ' ');
    st.update(1, 1, n, idx, 0);
  }
}
```