Problem Statement: This technosearch we are interested in searching for the beginning. And hence we are searching the beginning of everything that is we are looking for zeroes.

For a given array of size N, you will be given Q queries.
Each query can of three type:
1.] For given indices: l and r : Find all zeroes in the range: [l,r]

```
2.] For given K:  Find the index of Kth zero
                  If not as many zeroes then print:
                  “Not Enough Zeroes” without quotes
```

3.] For a given index p and value v: Update value of arr[p] to v

```
Constraints:
      1 <= N <= 100000
      1 <= Q <= 100000
      1 <= l,r <= N
      1 <= v <= 10^7
```

CODE:

```
// Author: thecodekaiser
// This code finds the number of 0s in a segment and also finds the position of
the k th zero in the array
#define _CRT_SECURE_NO_DEPRECATE
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <cmath>

using namespace std;
typedef long long LL;
#define INF 1000007

template <typename T> T gcd(T a, T b)
{
      if(a % b == 0) return b;
      else         return gcd(b, a%b);
}

template <typename T> T lcm(T a, T b)
{     return a * b / gcd(a,b);     }


int combine(int A,int B)
{
      return A + B;
}


template <typename T> class SEG
{

private:
      T * copyArr;
      int * tree;
      int len;

      // Function : To build the tree
      void build(int Node, int l, int r)
      {
        if(l == r)
        {
              tree[Node] = ((copyArr[l] == 0)? 1 : 0);
        }
        else
        {
              int left = 2 * Node, right = 2 * Node + 1, mid = (l+r)/2;
              build(left, l, mid);
              build(right, mid+1, r);

              tree[Node] = combine(tree[left],tree[right]);
        }
  }

  // Query operation
  int query1(int Node, int l, int r, int i, int j)
  {
        if(i > r || j < l)      return 0;

        if(i <= l and j >= r)   return tree[Node];

        else
        {
              int left = 2 * Node, right = 2 * Node + 1, mid = (l+r)/2;
              int p1 = query1(left, l, mid, i, j);
              int p2 = query1(right, mid+1, r, i, j);

              return combine(p1,p2);
        }
  }

  // Query 2
  int find_kth(int Node, int l, int r, int k)
  {
        if(tree[Node] < k)                  // Not as many zeroes
              return -1;

        if(l == r)  return l;
        else
        {
              int left = 2 * Node, right = 2 * Node + 1, mid = (l+r)/2;

              if(tree[left] >= k)
                    return find_kth(left, l, mid, k);
              else
                    return find_kth(right, mid+1, r, k - tree[left]);
        }
  }

  // Update operation
  void update(int Node, int l, int r, int i, int new_val)
  {
        if(i > r || i < l)            return ;

        if(l == r) {if(new_val == 0) tree[Node] = 1; else tree[Node] = 0;}

        else
        {
              int left = 2 * Node , right = 2 * Node + 1, mid = (l+r)/ 2;

              if(i <= mid)
                    update(left, l, mid, i, new_val);
              else
                    update(right, mid+1, r, i, new_val);
```

```
				tree[Node] = combine(tree[left], tree[right]);
			}

		}

	public:
		// Constructor
		SEG(T * arr, int N)
		{
			len = N;
			copyArr = (T *) malloc (len * sizeof(T));
			tree  = (int *) malloc (4 * len * sizeof(int));

			for(int i = 0; i < len; i++)
				copyArr[i] = arr[i];

			build(1, 0, len - 1);
		}

		// Query 1 -> No of zeroes in segment [i,j]
		void query1(int i, int j)
		{
			int result = query1(1, 0, len-1, i, j);
			cout << result << endl;
		}

		// Query 2 -> Kth zero in the array
		void query2(int k)
		{
			int result = find_kth(1, 0, len-1, k);
			if(result != -1)
				cout << k << "th zero occurs at: " << result + 1 << "
position.\n";
			else
				cout << "There are not as many zeroes." << endl;
		}

		// Update
		void update(int i, int val)
		{
			copyArr[i] = val;
			update(1, 0, len-1, i, val);
		}

};


int main()
{
	int N, Q, l, r, c, val;
	cin >> N;
	int ptr[N];

	for(int i = 0; i < N; i++)
		cin >> ptr[i];

	SEG<int> st = SEG<int>(ptr,N);

	cin >> Q;
	for(int i = 0; i < Q; i++)
	{
		cin >> c;
		if (c == 0)
```

```
            {
                  cin >> l >> val;
                  st.update(l-1, val);
            }
            else if(c == 1)
            {
                  cin >> l >> r;
                  st.query1(l-1, r-1);
            }
            else if(c == 2)
            {
                  cin >> l;
                  st.query2(l);
            }
      }
      return 0;
}
```