Secuencias extremas

```
#include <iostream>
#include <vector>
using namespace std;

// Pre: v.size() > 0; todos los elementos de v son enteros no negativos
// Post: retorna true si v es extrema (i, j son esas posiciones);
//               false, en caso contrario (i, j tienen cualquier valor)
bool es_extrema(const vector <int>& v, int& i, int& j) {
    i = 0;
    j = v.size() - 1;
    int suma = v[i] - v[j];
    while (suma != 0 and i < j) {
        if (suma > 0) {
            --j;
            suma = suma - v[j];
        } else {
            ++i;
            suma = suma + v[i];
        }
    }
    return suma == 0;
}

// Pre: en la entrada hay una secuencia de n enteros no negativos
// Post: se retorna un vector con los elementos de la secuencia de entrada
//       en el orden de aparición
vector <int> leer_secuencia(int n) {
    vector <int> v(n);
    for (int i = 0; i < n; ++i) cin >> v[i];
    return v;
}

int main() {
    int n;
    while (cin >> n) {
        vector <int> v = leer_secuencia(n);
        int i, j;
        if (es_extrema(v, i, j)) cout << i << " " << j << endl;
        else cout << "no" << endl;
    }
}
```

Robot simple de limpieza

```
#include <iostream>
#include <vector>
using namespace std;

typedef vector <vector <int> > Habitacion;

// Pre: en la entrada hay una secuencia de n*m enteros
// Post: se retorna una matriz de n*m con los elementos de la entrada
//       en el orden de aparición
Habitacion leer_matriz(int n, int m) {
    Habitacion h(n, vector <int>(m));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) cin >> h[i][j];
    }
    return h;
}

// Pre: j es la columna dentro de h; 0 <= from < v.size() es la fila de inicio dentro de h;
//      -1 <= to <= v.size() es la fila final; inc es -1 o +1
// Post: retorna el total de la basura de h en la columna f desde la fila from (incluída)
//      hasta la fila to (excluída) haciendo incremento inc. Escribe la fila y columna de
//      las casillas de h con valor 0 según el recorrido que realiza
int recoge(const Habitacion& h, int j, int from, int to, int inc) {
    int basura = 0;
    while (from != to) {
        if (h[from][j] == 0) cout << from << ' ' << j << endl;
        else basura = basura + h[from][j];
        from = from + inc;
    }
    return basura;
}

int main() {
    int n, m;
    cin >> n >> m;
    Habitacion hab = leer_matriz(n, m);
    int total = 0;
    for (int j = 0; j < m; ++j) {
        if (j%2 == 0) total += recoge(hab, j, 0, n, 1);
        else total += recoge(hab, j, n - 1, -1, -1);
        cout << "Total en columna " << j << ": " << total << endl;
    }
}
```

Destinos

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct Tren {
    string id_tren;
    string destino;
    string hora;
};

struct InfDest {
    string destino;
    string primer_id;
    int freq;
};

vector<Tren> lee_info_trenes(int n) {
    vector<Tren> v(n);
    for (int i = 0; i < n; ++i) cin >> v[i].id_tren >> v[i].destino >> v[i].hora;
    return v;
}

void escribir_resultados(const vector<InfDest>& v) {
    int m = v.size();
    for (int i = 0; i < m; ++i)
        cout << v[i].destino << ' ' << v[i].freq << ' ' << v[i].primer_id << endl;
}

// Pre: --
// Post: retorna true si a es más pequeño que b según criterios:
//           (1) destino; (2) hora; (3) identificador tren
//       false en caso contrario
bool cmp_tren(const Tren& a, const Tren& b) {
    if (a.destino != b.destino) return a.destino < b.destino;
    if (a.hora != b. hora) return a.hora < b.hora;
    return a.id_tren < b.id_tren;
}

// Pre: --
// Post: retorna true si a es más pequeño que b según criterios:
//           (1) frecuencia; (2) destino
//       false en caso contrario
bool cmp_inf_dest(const InfDest&  a, const InfDest& b) {
    if (a.freq == b.freq) return a.destino < b.destino;
    return a.freq > b.freq;
}
```

```
// Pre: m > 0 es el número de destinos; v es el vector de trenes con la info de cada Tren.
//      El vector v no es vacio y está ordenado por:
//         (1) criterio principal: destino
//         (2) criterio secundario: hora
//         (3) último criterio: identificador de tren.
// Post: retorna un vector de InfDest con la info de cada destino, frecuencia y primer tren
vector<InfDest> crea_v_inf_dest(const vector<Tren>& v, int m) {
    int n = v.size();
    vector<InfDest> w(m);    // resultado
    w[0].destino = v[0].destino;
    w[0].freq = 1;
    w[0].primer_id = v[0].id_tren;
    int j = 0;               // índice para w
    for (int i = 1; i < n; ++i) {
        if (v[i].destino == w[j].destino)  ++w[j].freq;
        else {
            ++j;
            w[j].destino = v[i].destino;
            w[j].primer_id = v[i].id_tren;
            w[j].freq = 1;
        }
    }
    return w;
}

int main() {
    int n;
    cin >> n;
    int m;
    cin >> m;
    vector<Tren> v_tren = lee_info_trenes(n);
    sort(v_tren.begin(), v_tren.end(), cmp_tren);
    vector<InfDest> v_inf_dest = crea_v_inf_dest(v_tren, m);
    sort(v_inf_dest.begin(), v_inf_dest.end(), cmp_inf_dest);
    escribir_resultados(v_inf_dest);
}
```

Última posición

```
#include <iostream>
#include <vector>
using namespace std;

// Pre: v.size() > 1, v is ordered and v[0] <= z and z is less than the last value of v
// Post: last position i such that v[i] <= z
int effi_last_pos(const vector<int>&v , int z) {
    int iz = 0;
    int de = v.size() - 1;
    //always v[iz] <= z < v[de]
    while (iz + 1 != de) {
        int mid = (iz + de)/2;
        //iz < mid < de
        if (v[mid] <= z) iz = mid;
        else de = mid;
    }
    return iz;
}

// gets vector v from input chanel
void read_vector(vector<int>& v) {
    int n = v.size();
    for (int i = 0; i < n; ++i) cin >> v[i];
}

int main() {
    int n;
    cin >> n;
    vector<int> v(n);
    read_vector(v);
    int z;
    while (cin >> z) cout << effi_last_pos(v, z) << endl;
}
```