```
int n;
cin >> n;
vector<int> a(n);
for(int i=0; i<n; i++) cin >> a[i];
```

```
class Dsu {
public:
    ll parent[2005]; //map<ll,ll>
    ↪  parent;
    ll siz[2005];    //map<ll,ll>
    ↪  siz;
    void make_set(ll v) {
        parent[v]=v; siz[v]=1;
    }
    ll find_set(ll v) {
        return (v==parent[v])?v:pa⌋
        ↪  rent[v]=find_set(paren⌋
        ↪  t[v]);
    }
    void union_sets(ll a, ll b) {
        a = find_set(a);
        b = find_set(b);
        if (a == b) return;
        if(siz[a]<siz[b])
        ↪  swap(a,b);
        parent[b] = a;
        siz[a]+=siz[b];
        siz[b]=0; //siz.erase(b);
    }
    ll get_size(ll v){
        return siz[find_set(v)];
    }
};
```

```
vector<pair<char, int>> segments;
char prev = s[0];
int cnt = 0;
for(auto c : s) {
    if(c != prev) {
        segments.pb({prev, cnt});
        cnt = 0;
        prev = c;
    }
    cnt++;
}
segments.pb({prev, cnt});
```

```
typedef long long LL;
typedef pair<LL, LL> PLL;

namespace Hashing {
    #define ff first
    #define ss second

    const PLL M = {1e9+7, 1e9+9};
    ↪  ///Should be large primes
    const LL base = 1259;
    ↪  ///Should be larger than
    ↪  alphabet size
    const int N = 1e6+7;
    ↪  ///Highest length of string

    PLL operator+ (const PLL& a, LL
    ↪  x)    {return {a.ff + x,
    ↪  a.ss + x};}
    PLL operator- (const PLL& a, LL
    ↪  x)    {return {a.ff - x,
    ↪  a.ss - x};}
    PLL operator* (const PLL& a, LL
    ↪  x)    {return {a.ff * x,
    ↪  a.ss * x};}
    PLL operator+ (const PLL& a,
    ↪  PLL x)   {return {a.ff +
    ↪  x.ff, a.ss + x.ss};}
    PLL operator- (const PLL& a,
    ↪  PLL x)   {return {a.ff -
    ↪  x.ff, a.ss - x.ss};}
    PLL operator* (const PLL& a,
    ↪  PLL x)   {return {a.ff *
    ↪  x.ff, a.ss * x.ss};}
    PLL operator% (const PLL& a,
    ↪  PLL m)   {return {a.ff %
    ↪  m.ff, a.ss % m.ss};}
    ostream& operator<<(ostream&
    ↪  os, PLL hash) {
        return os<<"("<<hash.ff<<",
        ↪  "<<hash.ss<<")";
    }
    PLL pb[N];      ///powers of
    ↪  base mod M

    ///Call pre before everything
    void hashPre() {
        pb[0] = {1,1};
        for (int i=1; i<N; i++)
        ↪  pb[i] = (pb[i-1] *
        ↪  base)%M;
    }

    ///Calculates hashes of all
    ↪  prefixes of s including
    ↪  empty prefix
    vector<PLL> hashList(string s)
    ↪  {
        int n = s.size();
        vector<PLL> ans(n+1);
        ans[0] = {0,0};
        for (int i=1; i<=n; i++)
        ↪  ans[i] = (ans[i-1] *
        ↪  base + s[i-1])%M;
        return ans;
    }

    ///Calculates hash of substring
    ↪  s[l..r] (1 indexed)
    PLL substringHash(const
    ↪  vector<PLL> &hashlist, int
    ↪  l, int r) {
        return
        ↪  (hashlist[r]+(M-hashli⌋
        ↪  st[l-1])*pb[r-l+1])%M;
    }

    ///Calculates Hash of a string
    PLL Hash (string s) {
        PLL ans = {0,0};
        for (int i=0; i<s.size();
        ↪  i++)  ans=(ans*base +
        ↪  s[i])%M;
        return ans;
    }

    ///Tested on https://toph.co/p⌋
    ↪  /palindromist
    ///appends c to string
    PLL append(PLL cur, char c) {
        return (cur*base + c)%M;
    }

    ///Tested on https://toph.co/p⌋
    ↪  /palindromist
    ///prepends c to string with
    ↪  size k
    PLL prepend(PLL cur, int k,
    ↪  char c) {
        return (pb[k]*c + cur)%M;
    }

    ///Tested on https://toph.co/p⌋
    ↪  /chikongunia
    ///replaces the i-th
    ↪  (0-indexed) character from
    ↪  right from a to b;
    PLL replace(PLL cur, int i,
    ↪  char a, char b) {
        return cur + pb[i] *
        ↪  (M+b-a)%M;
    }

    ///Erases c from front of the
    ↪  string with size len
    PLL pop_front(PLL hash, int
    ↪  len, char c) {
```

```
    return (hash +
    ↪  pb[len-1]*(M-c))%M;
}

///Tested on https://toph.co/p⌋
↪  /palindromist
///concatenates two strings
↪  where length of the right
↪  is k
PLL concat(PLL left, PLL right,
↪  int k) {
    return (left*pb[k] +
    ↪  right)%M;
}

PLL power (const PLL& a, LL p)
↪  {
    if (p==0)   return {1,1};
    PLL ans = power(a, p/2);
    ans = (ans * ans)%M;
    if (p%2)    ans =
    ↪  (ans*a)%M;
    return ans;
}

PLL inverse(PLL a)  {
    if (M.ss == 1)  return
    ↪  power(a, M.ff-2);
    return power(a,
    ↪  (M.ff-1)*(M.ss-1)-1);
}

///Erases c from the back of
↪  the string
PLL invb = inverse({base,
↪  base});
PLL pop_back(PLL hash, char c)
↪  {
    return ((hash-c+M)*invb)%M;
}

///Tested on https://toph.co/p⌋
↪  /palindromist
///Calculates hash of string
↪  with size len repeated cnt
↪  times
///This is O(log n). For O(1),
↪  pre-calculate inverses
PLL repeat(PLL hash, int len,
↪  LL cnt) {
    PLL mul =
    ↪  ((pb[len*cnt]-1+M) * i⌋
    ↪  nverse(pb[len]-1+M))%M;
    PLL ans = (hash*mul);
    if (pb[len].ff == 1)
    ↪  ans.ff = hash.ff*cnt;
    if (pb[len].ss == 1)
    ↪  ans.ss = hash.ss*cnt;
    return ans%M;
}
```

```
}

/// Solves https://judge.yosupo.jp⌋
↪  /problem/enumerate_palindromes
using namespace Hashing;
```

```
vi pi(const string& s) {
    vi p(sz(s));
    rep(i,1,sz(s)) {
        int g = p[i-1];
        while (g && s[i] != s[g]) g
        ↪  = p[g-1];
        p[i] = g + (s[i] == s[g]);
    }
    return p;
}

vi match(const string& s, const
↪  string& pat) {
    vi p = pi(pat + '\0' + s), res;
    rep(i,sz(p)-sz(s),sz(p))
        if (p[i] == sz(pat))
        ↪  res.push_back(i - 2 *
        ↪  sz(pat));
    return res;
}
```

```
struct SplayTree {
  struct Node {
    int ch[2] = {0, 0}, p = 0;
    long long self = 0, path = 0;
    ↪  // Path aggregates
    long long sub = 0, vir = 0;
    ↪  // Subtree aggregates
    bool flip = 0;
    ↪  // Lazy tags
  };
  vector<Node> T;

  SplayTree(int n) : T(n + 1) {}

  void push(int x) {
    if (!x || !T[x].flip) return;
    int l = T[x].ch[0], r =
    ↪  T[x].ch[1];

    T[l].flip ^= 1, T[r].flip ^= 1;
    swap(T[x].ch[0], T[x].ch[1]);
    T[x].flip = 0;
  }

  void pull(int x) {
    int l = T[x].ch[0], r =
    ↪  T[x].ch[1]; push(l);
    ↪  push(r);

    T[x].path = T[l].path +
    ↪  T[x].self + T[r].path;
    T[x].sub = T[x].vir + T[l].sub
    ↪  + T[r].sub + T[x].self;
  }

  void set(int x, int d, int y) {
    T[x].ch[d] = y; T[y].p = x;
    ↪  pull(x);
  }

  void splay(int x) {
    auto dir = [&](int x) {
      int p = T[x].p; if (!p)
      ↪  return -1;
      return T[p].ch[0] == x ? 0 :
      ↪  T[p].ch[1] == x ? 1 : -1;
    };
    auto rotate = [&](int x) {
      int y = T[x].p, z = T[y].p,
      ↪  dx = dir(x), dy = dir(y);
      set(y, dx, T[x].ch[!dx]);
      set(x, !dx, y);
      if (~dy) set(z, dy, x);
      T[x].p = z;
    };
    for (push(x); ~dir(x); ) {
      int y = T[x].p, z = T[y].p;
      push(z); push(y); push(x);
      int dx = dir(x), dy = dir(y);
      if (~dy) rotate(dx != dy ? x
      ↪  : y);
      rotate(x);
    }
  }
};

struct LinkCut : SplayTree {
  LinkCut(int n) : SplayTree(n) {}

  int access(int x) {
    int u = x, v = 0;
    for (; u; v = u, u = T[u].p) {
      splay(u);
      int& ov = T[u].ch[1];
      T[u].vir += T[ov].sub;
      T[u].vir -= T[v].sub;
      ov = v; pull(u);
    }
    return splay(x), v;
  }

  void reroot(int x) {
    access(x); T[x].flip ^= 1;
    ↪  push(x);
  }

  void Link(int u, int v) {
    reroot(u); access(v);
    T[v].vir += T[u].sub;
    T[u].p = v; pull(v);
  }

  void Cut(int u, int v) {
    reroot(u); access(v);
    T[v].ch[0] = T[u].p = 0;
    ↪  pull(v);
  }

  // Rooted tree LCA. Returns 0 if
  ↪  u and v arent connected.
  int LCA(int u, int v) {
    if (u == v) return u;
    access(u); int ret = access(v);
    return T[u].p ? ret : 0;
  }

  // Query subtree of u where v is
  ↪  outside the subtree.
  long long Subtree(int u, int v) {
    reroot(v); access(u); return
    ↪  T[u].vir + T[u].self;
  }

  // Query path [u..v]
  long long Path(int u, int v) {
    reroot(u); access(v); return
    ↪  T[v].path;
  }

  // Update vertex u with value v
  void Update(int u, long long v) {
    access(u); T[u].self = v;
    ↪  pull(u);
  }
};
```

```
array<vi, 2> manacher(const string&
↪  s) {
    int n = sz(s);
    array<vi,2> p = {vi(n+1),
    ↪  vi(n)};
    rep(z,0,2) for (int
    ↪  i=0,l=0,r=0; i < n; i++) {
        int t = r-i+!z;
        if (i<r) p[z][i] = min(t,
        ↪  p[z][l+t]);
        int L = i-p[z][i], R =
        ↪  i+p[z][i]-!z;
        while (L>=1 && R+1<n &&
        ↪  s[L-1] == s[R+1])
            p[z][i]++, L--, R++;
        if (R>r) l=L, r=R;
    }
    return p;
}
```

```
int minRotation(string s) {
    int a=0, N=sz(s); s += s;
    rep(b,0,N) rep(k,0,N) {
        if (a+k == b || s[a+k] <
        ↪  s[b+k]) {b += max(0LL,
        ↪  k-1); break;}
        if (s[a+k] > s[b+k]) { a =
        ↪  b; break; }
    }
    return a;
}
```

```
/*
Author: Md. Ashraful Islam
Copyright @ All Rights Reserved
*/

#include <bits/stdc++.h>

#define ll long long
#define ld long double
#define pb push_back
#define pf push_front
#define pll pair<long long, long
↪  long>
#define asort(a) sort(a.begin(),
↪  a.end())
#define arsort(a,n) sort(a, a+n)
#define MAX 2000005
#define MOD 998244353
#define faster {ios_base::sync_wit⌋
↪  h_stdio(false);cin.tie(NULL);c⌋
↪  out.tie(NULL);}
#define endl "\n"
#define pii pair<int, int>
#define rep(i, a, b) for(int i = a;
↪  i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()

using namespace std;
typedef vector<int> vi;

//using namespace __gnu_pbds;
#include
↪  <ext/pb_ds/assoc_container.hpp>
#include
↪  <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

#define ordered_set tree<pair<long
↪  long, long long>,
↪  null_type,less<pair<long long,
↪  long long>>, rb_tree_tag,tree_⌋
↪  order_statistics_node_update >
#define ordered_multiset
↪  tree<unsigned long long,
↪  null_type,less_equal<unsigned
↪  long long>, rb_tree_tag,tree_o⌋
↪  rder_statistics_node_update >


#ifndef ONLINE_JUDGE
#include "dbg.h"
#else
#define dbg(...) {/*temon kichu
↪  na*/}
#endif

#define int long long

void solve() {
        $1
}

signed main() {
    faster;
    bool NO_TEST_CASE = true;
    bool PR1l_CASE = false;
    bool INLINE_CASE = true;

    ll T;
    if(NO_TEST_CASE) {
        T = 1;
    } else {
        cin >> T;
    }
    for(ll t=1; t<=T; t++) {
        if(PR1l_CASE) {
            cout << "Case " << t <<
            ↪  ": ";
            if(!INLINE_CASE) {
                cout << endl;
            }
        }
        solve();
    }
    return 0;
}
```

```
template<class T>
struct RMQ {
    vector<vector<T>> jmp;
    RMQ(const vector<T>& V) :
    ↪  jmp(1, V) {
        for (int pw = 1, k = 1; pw
        ↪  * 2 <= sz(V); pw *= 2,
        ↪  ++k) {
            jmp.emplace_back(sz(V)
            ↪  - pw * 2 + 1);
            rep(j,0,sz(jmp[k]))
            jmp[k][j] = min(jmp[k -
            ↪  1][j], jmp[k - 1][j
            ↪  + pw]);
        }
    }

    T query(int a, int b) {
        assert(a < b); // or return
        ↪  in f i f a == b
        int dep = 31 -
        ↪  __builtin_clz(b - a);
        return min(jmp[dep][a],
        ↪  jmp[dep][b - (1 <<
        ↪  dep)]);
    }
};
```

```
class SegmentTree {
public:
    int *arr, *tree;
    int _size;
    SegmentTree(vector<int> v) {
        int size = v.size();
        _size = size;
        arr = new int[size+10];
        for(int i=1; i<=size; i++)
        ↪  {
            arr[i] = v[i-1];
        }
        tree = new
        ↪  int[(size+10)*4];
        build(1, 1, size);
    }

    void build(int node, int b, int
    ↪  e)
    {
        int left, right, mid;
        if(b == e){
             tree[node] = arr[b];
             return;
        }
        left = node*2;
        right = node*2 + 1;
        mid = (b+e)/2;
        build(left, b, mid);
        build(right, mid+1, e);
        tree[node] = tree[left] +
        ↪  tree[right];
    }

    int query(int node, int b, int
    ↪  e, int i, int j)
    {
        int left, right, mid;
        if(i > j)
             return 0;
        if(i == b && j == e)
             return tree[node];
        mid = (b+e)/2;
        left = 2*node;
        right = 2*node + 1;
        return query(left, b, mid,
        ↪  i, min(j, mid)) +
        ↪  query(right, mid+1, e,
        ↪  max(mid+1, i), j);
    }

    int query(int l, int r) {
        return query(1, 1, _size,
        ↪  l, r);
    }
};
```

```
class SegmentTree {
public:
    int *arr, *tree;
    int _size;
    SegmentTree(vector<int> v) {
        int size = v.size();
        _size = size;
        arr = new int[size+10];
        for(int i=1; i<=size; i++)
        ↪  {
            arr[i] = v[i-1];
        }
        tree = new
        ↪  int[(size+10)*4];
        build(1, 1, size);
    }

    void build(int node, int b, int
    ↪  e)
    {
        int left, right, mid;
        if(b == e){
             tree[node] = arr[b];
             return;
        }
        left = node*2;
        right = node*2 + 1;
        mid = (b+e)/2;
        build(left, b, mid);
        build(right, mid+1, e);
        tree[node] =
        ↪  min(tree[left],
        ↪  tree[right]);
    }

    int query(int node, int b, int
    ↪  e, int i, int j)
    {
        int left, right, mid;
        if(i > j)
             return INT_MAX;
        if(i == b && j == e)
             return tree[node];
        mid = (b+e)/2;
        left = 2*node;
        right = 2*node + 1;
        return min(query(left, b,
        ↪  mid, i, min(j, mid)),
        ↪  query(right, mid+1, e,
        ↪  max(mid+1, i), j));
    }

    int query(int l, int r) {
        return query(1, 1, _size,
        ↪  l, r);
    }

    void update(int node, int b,
    ↪  int e, int i, int newvalue)
    {
        if (i > e || i < b)
            return;
        if (b >= i && e <= i) {
            tree[node] = newvalue;
            return;
        }
        int Left = node * 2;
        int Right = node * 2 + 1;
        int mid = (b + e) / 2;
        update(Left, b, mid, i,
        ↪  newvalue);
        update(Right, mid + 1, e,
        ↪  i, newvalue);
        tree[node] =
        ↪  min(tree[Left],
        ↪  tree[Right]);
    }

    void update(int i, int val) {
        update(1, 1, _size, i,
        ↪  val);
    }
};
```

```cpp
class SortedArray {
    ordered_multiset arr;
public:
    long long size() { return arr.size(); }
    void operator += (long long x) { arr.insert(x); }
    long long operator < (long long x) { return arr.order_of_key(x); }
    long long operator <= (long long x) { return arr.order_of_key(x+1); }
    long long operator > (long long x) { return arr.size() - arr.order_of_key(x+1); }
    long long operator >= (long long x) { return arr.size() - arr.order_of_key(x); }
    long long LR(long long l, long long r) { return max((*this <= r) - (*this < l), 0LL); }
    long long lR(long long l, long long r) { return LR(l+1, r); }
    long long Lr(long long l, long long r) { return LR(l, r-1); }
    long long lr(long long l, long long r) { return LR(l+1, r-1); }
    long long operator [] (long long i) { return *arr.find_by_order(i); }
};
```

```cpp
struct SuffixArray {
    vi sa, lcp;
    SuffixArray(string& s, int lim=256) { // or basic string<int>
        int n = sz(s) + 1, k = 0, a, b;
        vi x(all(s)), y(n), ws(max(n, lim));
        x.push_back(0), sa = lcp = y, iota(all(sa), 0);
        for (int j = 0, p = 0; p < n; j = max(1LL, j * 2), lim = p) {
            p = j, iota(all(y), n - j);
            rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
            fill(all(ws), 0);
            rep(i,0,n) ws[x[i]]++;
            rep(i,1,lim) ws[i] += ws[i - 1];
            for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
            swap(x, y), p = 1, x[sa[0]] = 0;
            rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
                (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
        }
        for (int i = 0, j; i < n - 1; lcp[x[i++]] = k)
            for (k && k--, j = sa[x[i] - 1];
                    s[i + k] == s[j + k]; k++);
    }
};
```

```cpp
struct SuffixTree {
    static const int ALPHA = 28;
    int toi(char c) { return c - 'a'; }
    string a;

    vector<vector<int>> t;
    vector<int> l, r, p, s;
    int v = 0, q = 0, m = 2;

    void ukkadd(int i, int c) {
    suff:
        if (r[v] <= q) {
            if (t[v][c] == -1) {
                t[v][c] = m; l[m] = i;
                p[m++] = v; v = s[v]; q = r[v]; goto suff;
            }
            v = t[v][c]; q = l[v];
        }
        if (q == -1 || c == toi(a[q])) q++;
        else {
            l[m + 1] = i; p[m + 1] = m; l[m] = l[v]; r[m] = q;
            p[m] = p[v]; t[m][c] = m + 1;
            t[m][toi(a[q])] = v;
            l[v] = q; p[v] = m; t[p[m]][toi(a[l[m]])] = m;
            v = s[p[m]]; q = l[m];
            while (q < r[m]) { v = t[v][toi(a[q])]; q += r[v] - l[v]; }
            if (q == r[m]) s[m] = v; else s[m] = m + 2;
            q = r[v] - (q - r[m]); m += 2; goto suff;
        }
    }

    SuffixTree(string a) : a(a) {
        int N = 2 * sz(a) + 10;
        t.assign(N, vector<int>(ALPHA, -1));
        l.assign(N, 0); r.assign(N, sz(a));
        p.assign(N, 0); s.assign(N, 0);

        fill(t[1].begin(), t[1].end(), 0);
        s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;

        rep(i, 0, sz(a)) ukkadd(i, toi(a[i]));
    }

    pii best;
    int lcs(int node, int i1, int i2, int olen) {
        if (l[node] <= i1 && i1 < r[node]) return 1;
        if (l[node] <= i2 && i2 < r[node]) return 2;
        int mask = 0, len = node ? olen + (r[node] - l[node]) : 0;
        rep(c, 0, ALPHA) if (t[node][c] != -1)
            mask |= lcs(t[node][c], i1, i2, len);
        if (mask == 3)
            best = max(best, {len, r[node] - len});
        return mask;
    }

    static pii LCS(string s, string
    ↪  t) {
        SuffixTree st(s +
        ↪  (char)('z' + 1) + t +
        ↪  (char)('z' + 2));
        st.lcs(0, sz(s), sz(s) + 1
        ↪  + sz(t), 0);
        return st.best;
    }
};
```

```
struct RollbackUF {
    vi e; vector<pii> st;
    RollbackUF(int n) : e(n, -1) {}
    int size(int x) { return
    ↪  -e[find(x)]; }
    int find(int x) { return e[x] <
    ↪  0 ? x : find(e[x]); }
    int time() { return sz(st); }
    void rollback(int t) {
        for (int i = time(); i -->
        ↪  t;)
            e[st[i].first] =
            ↪  st[i].second;
        st.resize(t);
    }
    bool join(int a, int b) {
        a = find(a), b = find(b);
        if (a == b) return false;
        if (e[a] > e[b]) swap(a,
        ↪  b);
        st.push_back({a, e[a]});
        st.push_back({b, e[b]});
        e[a] += e[b]; e[b] = a;
        return true;
    }
};
```

```
vi Z(const string& S) {
    vi z(sz(S));
    int l = -1, r = -1;
    rep(i,1,sz(S)) {
        z[i] = i >= r ? 0 : min(r -
        ↪  i, z[i - l]);
        while (i + z[i] < sz(S) &&
        ↪  S[i + z[i]] == S[z[i]])
            z[i]++;
        if (i + z[i] > r)
            l = i, r = i + z[i];
    }
    return z;
}
```