Disjoint set union

Ajay Kumar

Disjoint set union

Disjoint Set(DS) data structures are representations of sets (which are all disjoint, sharing no elements) with certain functions:
DS is used in various algorithms, such as Kruskal's minimum spanning tree finder.

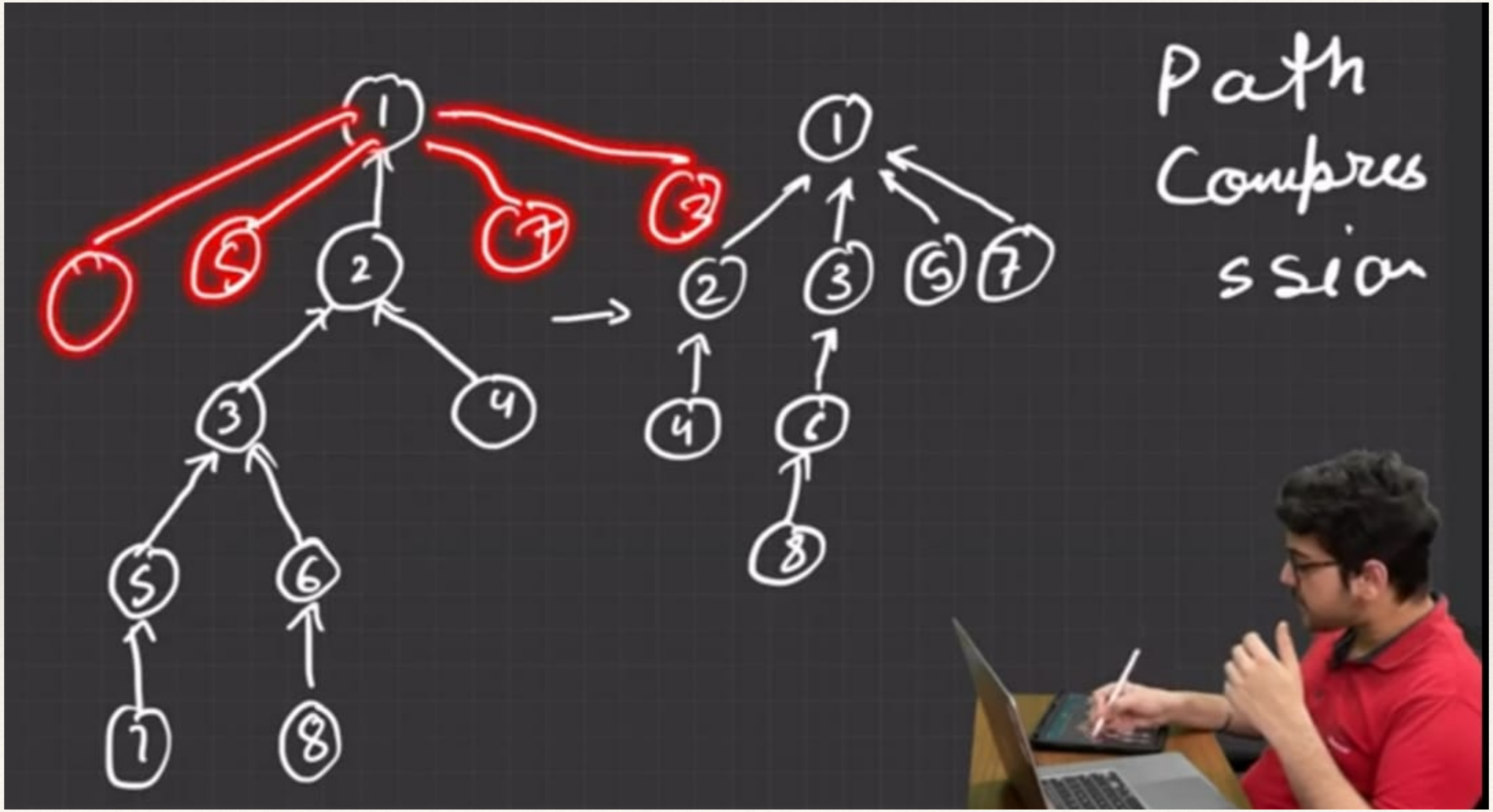
Funtions	Works
FindSet(x)	Find the set of element x
UniteSets(x,y)	Unites the set x and y
MakeSet(x)	Makes a set with element x
Disunion(list)	Removes all elements from other sets and makes a new set with theese elements

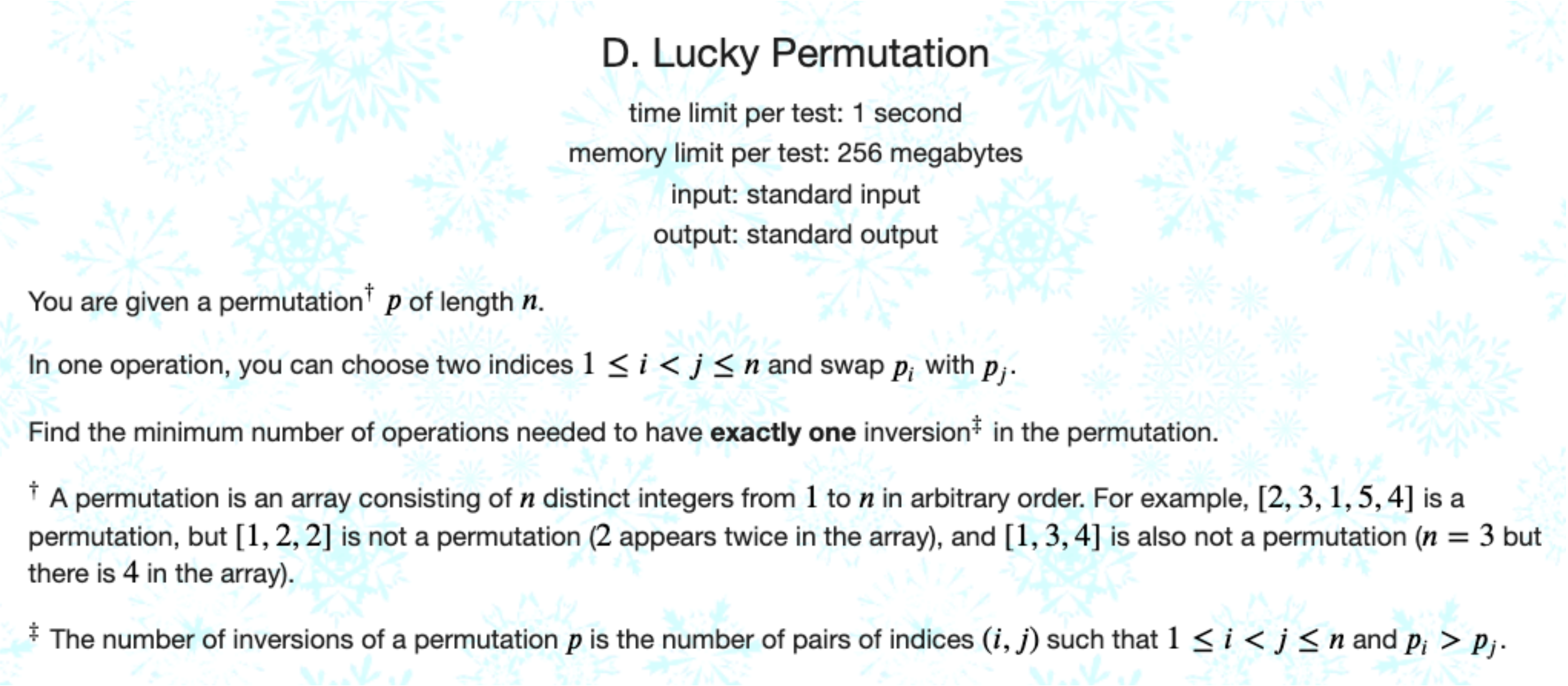
(Note: the disunion operation is not as commonly used as the other operations and is not implemented efficiently, with an O(n) runtime.)

```
int parent[N];
int size[N];
void make(int v){
    parent[v]=v;
}
int find(int v){
    if(v==parent[v]){
        return v;
    }
    //Path compression
    return parent[v]=find(parent[v]);
}
void Union(int a,int b){
    a= find(a);
    b= find(b);
    if(a!=b){
        if(size[a]<size[b]){
            swap(a,b);
        parent[b]=a;
        size[a]+=b;
    }
}
```

O(Alpha(n)) =Alpha(n): Reverse Akerman Function
O(Alpha(n)): Amortartized Algorithms
Almost constant
Over a number of operations

- Union by Size: Join the small tree to the big tree
- Union by Rank: Join the trees wrt to the depth
- Path Compression:





D. Lucky Permutation

time limit per test: 1 second

memory limit per test: 256 megabytes

input: standard input

output: standard output

You are given a permutation[†] p of length n.

In one operation, you can choose two indices $1 \le i < j \le n$ and swap p_i with p_j.

Find the minimum number of operations needed to have **exactly one** inversion[‡] in the permutation.

[†] A permutation is an array consisting of n distinct integers from 1 to n in arbitrary order. For example, $[2, 3, 1, 5, 4]$ is a permutation, but $[1, 2, 2]$ is not a permutation (2 appears twice in the array), and $[1, 3, 4]$ is also not a permutation ($n = 3$ but there is 4 in the array).

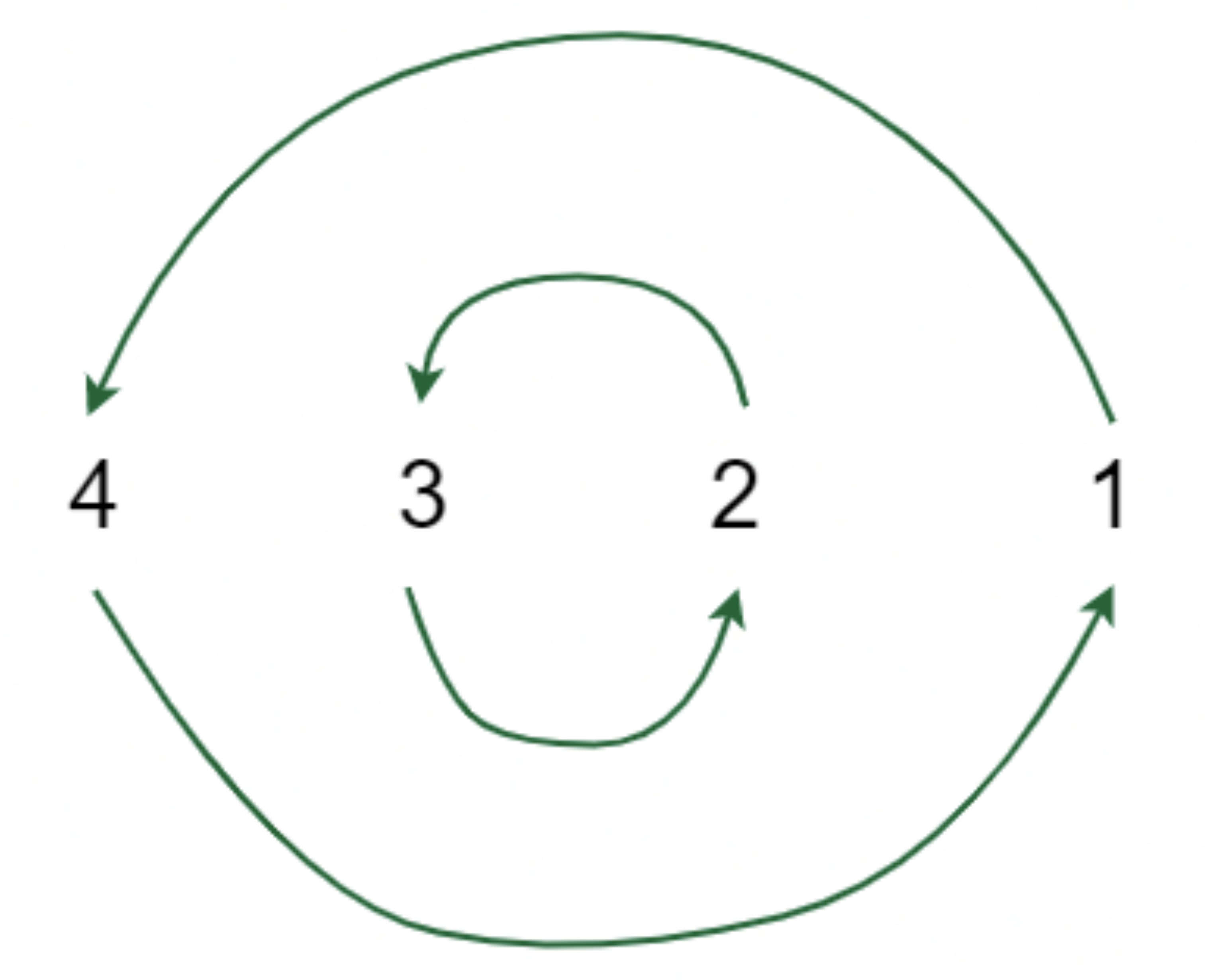
[‡] The number of inversions of a permutation p is the number of pairs of indices (i, j) such that $1 \le i < j \le n$ and $p_i > p_j$.

Min Swaps to sort an array

Given an array of **N** distinct elements, find the minimum number of swaps required to sort the array.

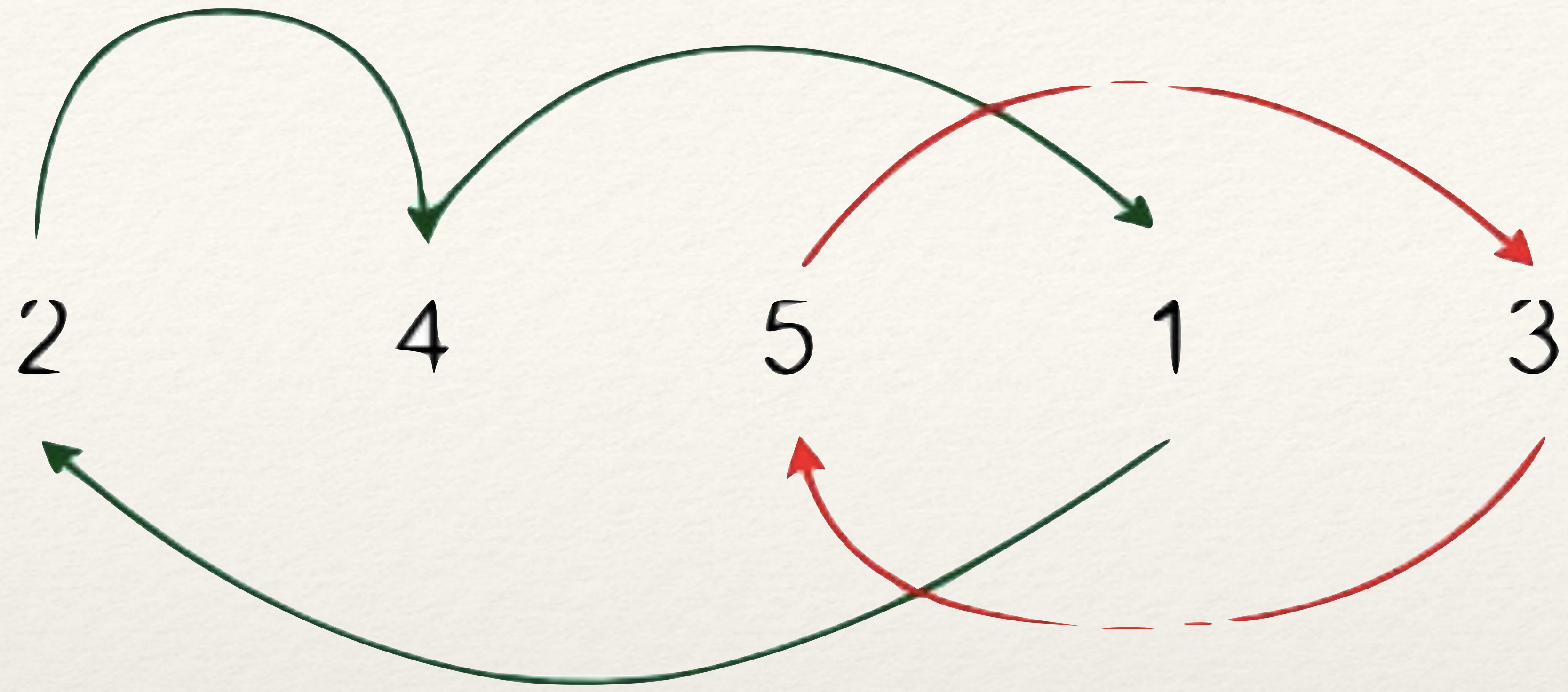
Input: {4, 3, 2, 1}
Output: 2
Explanation: Swap index 0 with 3 and 1 with 2 to form the sorted array {1, 2, 3, 4}
Input: {1, 5, 4, 3, 2}
Output: 2

Approach
This can be easily done by visualizing the problem as a graph. We will have N nodes and an edge directed from node i to node j if the element at the i'th index must be present at the j'th index in the sorted array.



Graph for {4 3 2 1}

The graph will now contain many non-intersecting cycles. Now a cycle with 2 nodes will only require 1 swap to reach the correct ordering, similarly, a cycle with 3 nodes will only require 2 swaps to do so.



Hence, ans = $\Sigma_{i = 1k}$(cycle_size − 1), where **k** is the number of cycles

```cpp
struct dsu {
    vector<ll> e;
    dsu(ll n) : e(n, -1) {}
    bool sameSet(ll a, ll b) { return find(a) == find(b); }
    ll size(ll x) { return -e[find(x)]; }
    ll find(ll x) { return e[x] < 0 ? x : e[x] = find(e[x]); }
    bool join(ll a, ll b) {
        a = find(a), b = find(b);
        if (a == b) return false;
        if (e[a] > e[b]) swap(a, b);
        e[a] += e[b]; e[b] = a;
        return true;
    }
};
```

```cpp
bool solve(){
    ll n;
    cin >> n;
    vector<ll> v(n);
    dsu d(n);
    for(ll i=0 ; i<n ; ++i) cin >> v[i];
    for(ll i=0 ; i<n ; ++i){
        v[i]--;
        d.join(i,v[i]);
    }
    ll cycles = 0;
    set<ll> s;
    for(ll i=0 ; i<n ; ++i)  s.insert(d.find(i));

    cycles = s.size();
    for(ll i=1 ; i<n ; ++i){
        if(d.sameSet(v[i],v[i-1])){
            cout << n-(cycles+1) << endl;
            return true;
        }
    }
    cout << n-(cycles-1) << endl;
    return true;
}
```

- Initiallise the struct

```
int main()
{
    struct car
    {
        char  name[100];
        float price;
    };

    // car1.name -> "xyz"
    //car1.price -> 987432.50
    struct car car1 ={"xyz", 987432.50};

    //printing values using dot(.) or 'member access' operator

    printf("Name of car1 = %s\n",car1.name);
    printf("Price of car1 = %f\n",car1.price);

    return 0;

}
```

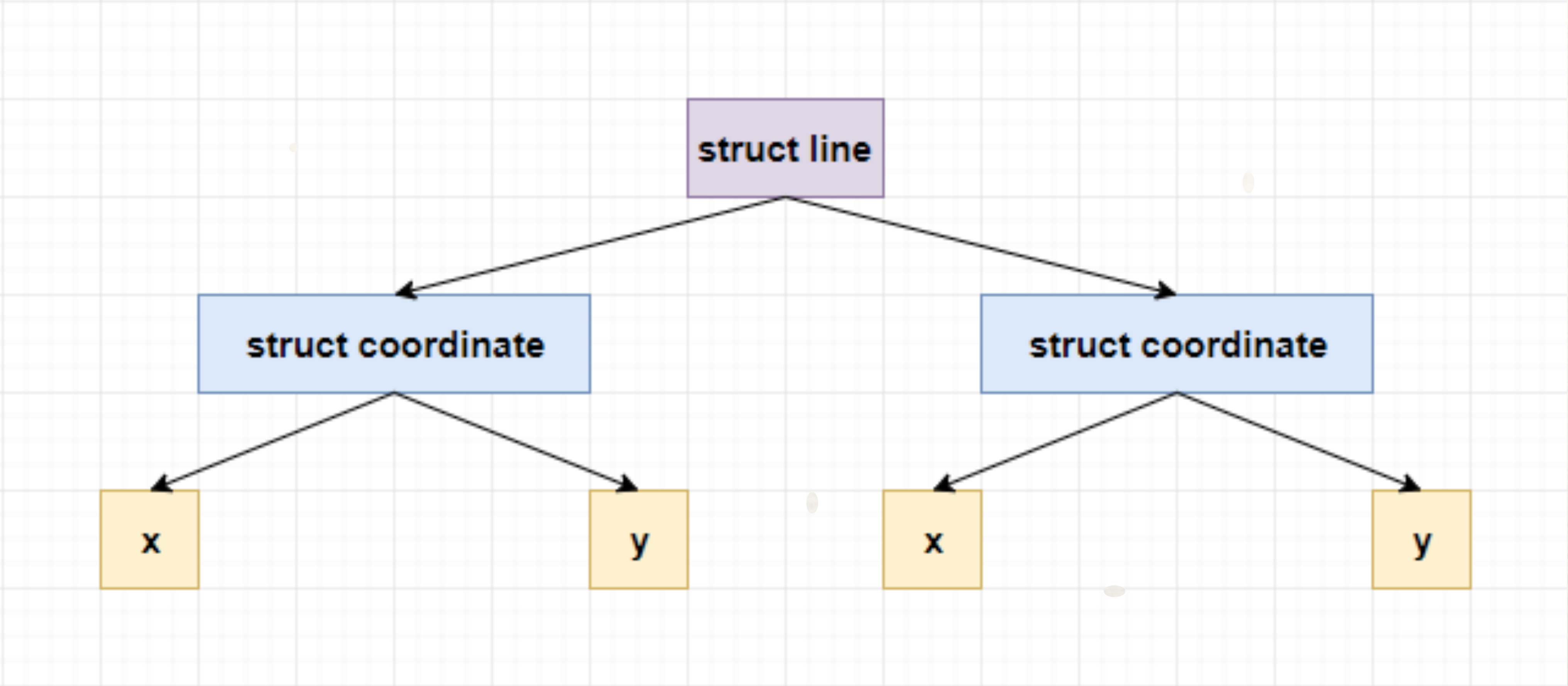
Structure within structure

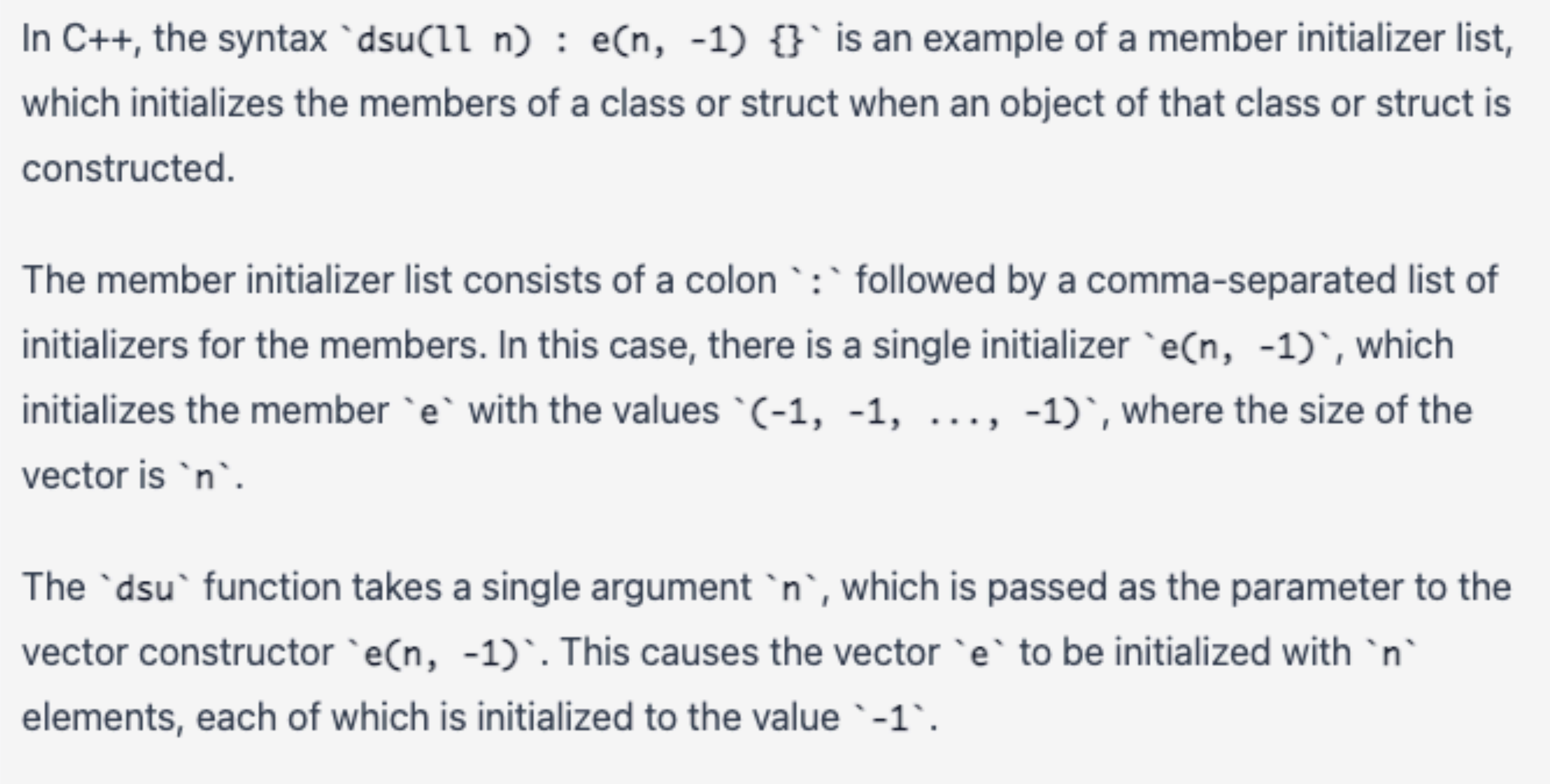
```
struct coordinate{
    int x;
    int y;
};
struct line{
    struct coordinate c1;
    struct coordinate c2;
};

//structure line variable
    struct line l;

    //get coordinate 1
    printf("Enter c1(x and y)\n");
    scanf("%d%d",&l.c1.x,&l.c1.y);

    //get coordinate 2
    printf("Enter c2(x and y)\n");
    scanf(“%d%d",&l.c2.x,&l.c2.y);
    distance = sqrt(pow((l.c2.x - l.c1.x),2) + pow((l.c2.y - l.c1.y),2));
    printf("Distance = %f\n",distance);
```





In C++, the syntax `dsu(ll n) : e(n, -1) {}` is an example of a member initializer list, which initializes the members of a class or struct when an object of that class or struct is constructed.

The member initializer list consists of a colon `:` followed by a comma-separated list of initializers for the members. In this case, there is a single initializer `e(n, -1)`, which initializes the member `e` with the values `(-1, -1, ..., -1)`, where the size of the vector is `n`.

The `dsu` function takes a single argument `n`, which is passed as the parameter to the vector constructor `e(n, -1)`. This causes the vector `e` to be initialized with `n` elements, each of which is initialized to the value `-1`.

```cpp
while (t--) {
        sans = "NO";
        ans = temp = sum = 0;
        cin >> n;
        vll a(n+1,0); vll comp(n+1,0);
        fo(i,1,n){cin>>a[i];}
        vector<bool>vis(n+1,false);
        ll c = 0;
        fo(i,1,n){
            if(!vis[i]){
                ll curr = i;
                c++;
                while(1){
                    if(vis[curr])break;
                    comp[curr] = c;
                    vis[curr] = true;
                    curr = a[curr];
                }
            }
        }
        ll ans = n - c + 1;
        fo(i,2,n) {
            if(comp[i] == comp[i-1]){
                ans-=2;
                break;
            }
        }
        cout<<ans<<"\n";
    }
```