Sparse Table

OJ : CSES-1647

Given an array of n integers, your task is to process q queries of the form: what is the minimum value in range [a,b]?

Constraints : $1 \le n, q \le 200000$

Given an array of n integers, your task is to process q queries of the form: what is the minimum value in range [a,b]?

Constraints : $1 \leq n, q \leq 200000$

Can you answers queries in O(1) ?

Given an array of n integers, your task is to process q queries of the form: what is the minimum value in range [a,b]?

Constraints : $1 \le n, q \le 200000$

Can you answers queries in O(1) ?

Target Time Complexity

Preprocessing : O(nlogn)
Per query : O(1)
so q query : O(q)

Total Time : O(nlogn + q)

Sparse Table

Given Array x[]={10, 1, 3, 20, 25, -5, 6, -10, 11, 8}

	0	1	2	3	4	5	6	7	8	9
2^0=1 length	10	1	3	20	25	-5	6	-10	11	8
2^1=2 length	1	1	3	20	-5	-5	-10	-10	8	
2^2=4 length	1	1	-5	-5	-10	-10	-10			
2^3=8 length	-10	-10	-10							

Sparse[power_of_two][starting_index] = min of (a[start], a[start+1], ... , a[start+2^(power)-1])

Sparse[1][7]=-10; power=1 ; start = 7 ; len=2^1 = 2 ;
min of(a[7],a[8]) = min of (a[7] , a[7+len-1]) = min of (a[7] , a[7+2-1]) - 5

Sparse[2][3]=-5; power=2 ; start = 3 ; len=2^2 = 4 ;
min of(a[3],a[4],a[5],a[6]) = min of (a[3] , a[3+1] , ... ,a[3+ len -1]) = - 5

x[]={10, 1, 3, 20, 25, -5, 6, -10, 11, 8}

	0	1	2	3	4	5	6	7	8	9
2^0=1 length	10	1	3	20	25	-5	6	-10	11	8
2^1=2 length	1	1	3	20	-5	-5	-10	-10	8	
2^2=4 length	1	1	-5	-5	-10	-10	-10			
2^3=8 length	-10	-10	-10							

Sparse[power_of_two][starting_index] = min of (a[start], a[start+1], ... , a[start+2^(power)-1])

Sparse[0][0]=10;
power=0 ; start=0 ; len=2^0=1 ; min(a[0],...a[0+len-1]) = min of (a[0]) = 10

Sparse[0][4]=25;
power=0 ; start=4 ; len=2^0=1 ; min(a[4],...a[4+len-1]) = min of (a[4]) = 25

Sparse[0][7]=-10;
power=0 ; start=7 ; len=2^0=1 ; min(a[7],...a[7+len-1]) = min of (a[7]) = -10

x[]={10, 1, 3, 20, 25, -5, 6, -10, 11, 8}

	0	1	2	3	4	5	6	7	8	9
$2^0=1$ length	10	1	3	20	25	-5	6	-10	11	8
$2^1=2$ length	1	1	3	20	-5	-5	-10	-10	8	
$2^2=4$ length	1	1	-5	-5	-10	-10	-10			
$2^3=8$ length	-10	-10	-10							

Sparse[power_of_two][starting_index] = min of (a[start], a[start+1], ... , a[start+2^(power)-1])

Filling up The first row;
It is our base case;
Here , power_of_two=0 , len = 2^0 = 1

```
for(int i = 0; i < n; i++) sparse_table[0][i]=x[i];
```

	0	1	2	3	4	5	6	7	8	9
2^0=1 length	10	1	3	20	25	-5	6	-10	11	8
2^1=2 length	1	1	3	20	-5	-5	-10	-10	8	
2^2=4 length	1	1	-5	-5	-10	-10	-10			
2^3=8 length	-10	-10	-10							

Sparse[power_of_two][starting_index] = min of (a[start], a[start+1], ... , a[start+2^(power)-1])

sparse_table[k][i] = min(sparse_table[k-1][i], sparse_table[k-1][i+len]);

Here , len = 2^(i-1)

Sparse[1][3] = min(a[3],a[4]) ;   Sparse[1][5] = min(a[5],a[6]) ;

Sparse[2][3] = min(a[3],a[4],a[5],a[6]);
Sparse[2][3] = min(min(a[3],a[4]) , min(a[5],a[6])) ;

Sparse[2][3] = min(Sparse[1][3],Sparse[1][3+2]) ; len = 2^(2-1) = 2 ;

Code :

```
void build_sparse_table(int n)
{
    for(int i = 1; i <= n; i++) sparse_table[0][i]=x[i];

    for(int k = 1; k < LOG; k++) {
      for(int i = 1; i + (1 << k) - 1 <= n; i++) {
          sparse_table[k][i] = min(sparse_table[k-1][i], sparse_table[k-1][i+(1<<(k-1))]);
      }
    }
}
```

	0	1	2	3	4	5	6	7	8	9
$2^0=1$ length	10	1	3	20	25	-5	6	-10	11	8
$2^1=2$ length	1	1	3	20	-5	-5	-10	-10	8	
$2^2=4$ length	1	1	-5	-5	-10	-10	-10			
$2^3=8$ length	-10	-10	-10							

Sparse[power_of_two][starting_index] = min of (a[start], a[start+1], ... , a[start+2^(power)-1])

Query(2,6) :

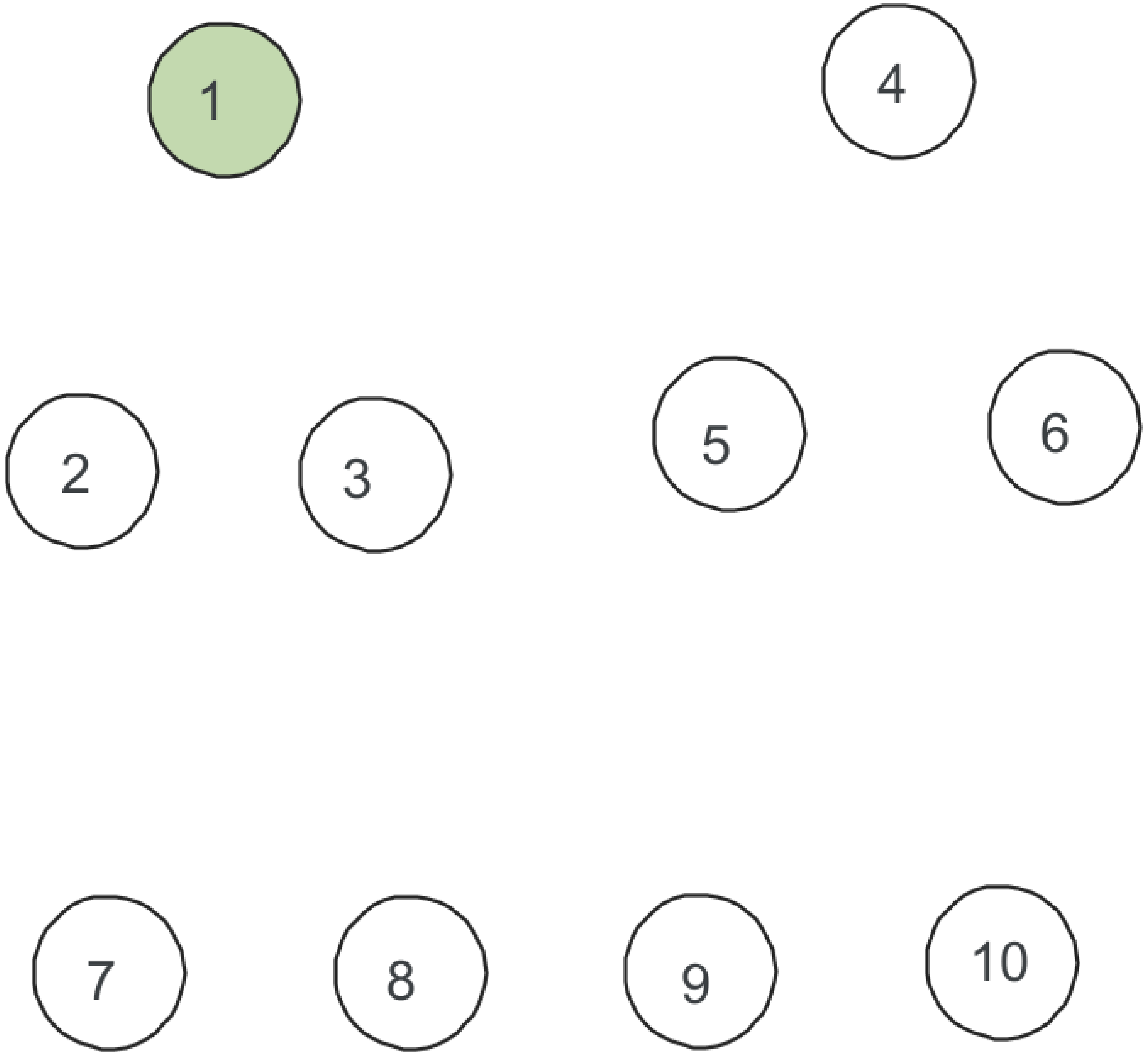
min(a[2],a[3],a[4],a[5],a[6]) = min(min(a[2],a[3],a[4],a[5]) , min(a[3],a[4],a[5],a[6]);
= min(Sparse[2][2], Sparse[2][3]);

here , kbit = 2 , len = 2^2 = 4 , left = 2 , right =6 ;
So , Query(left,right) = min(Sparse[k][left], Sparse[k][right-len+1]);

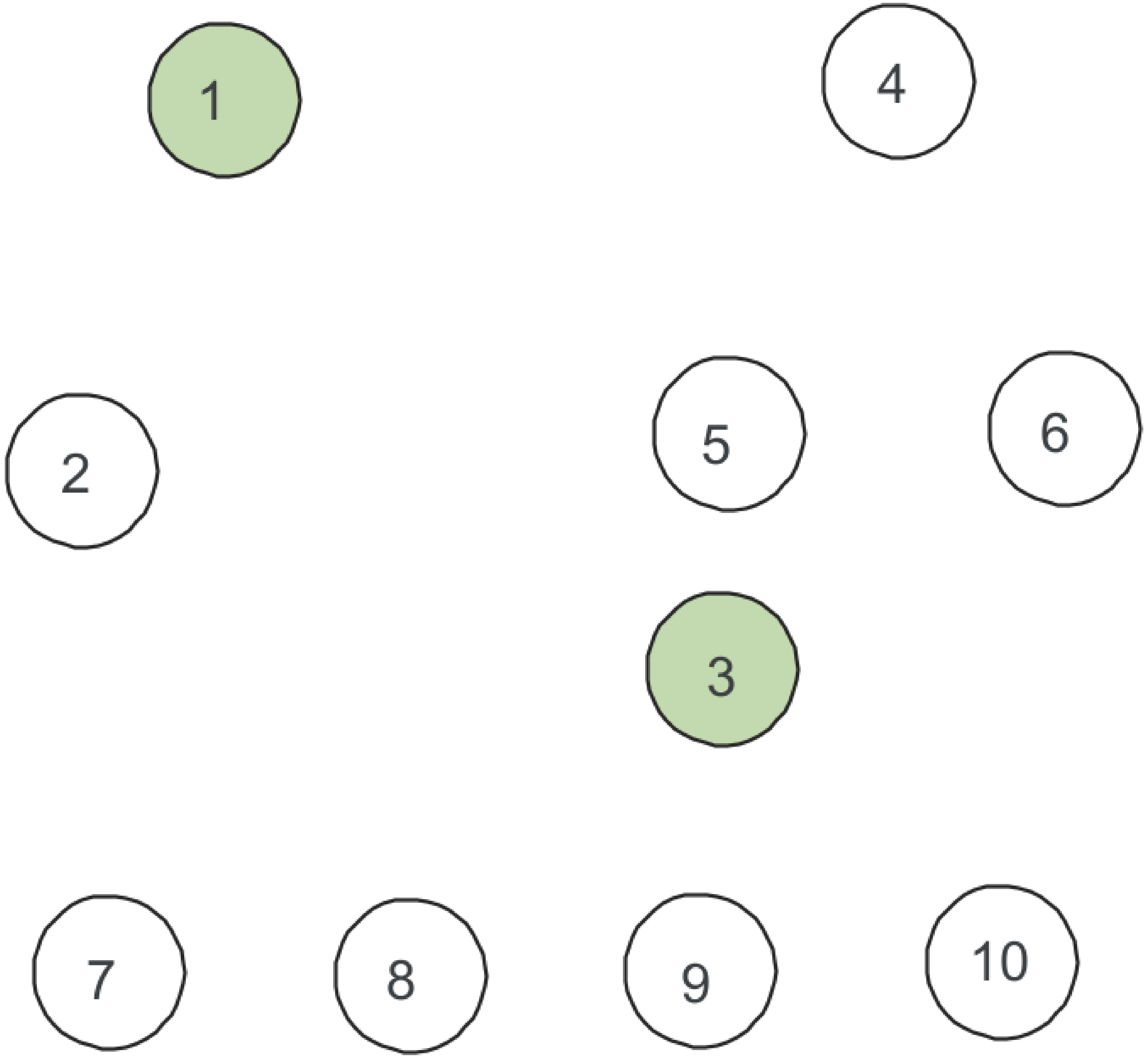
Query Code:

```
int min_query(int L, int R) { // O(1)

        int length = R - L + 1;
    int k = log_2[length];
    return min(sparse_table[k][L], sparse_table[k][R-(1<<k)+1]);

}
```

Disjoint Set Union

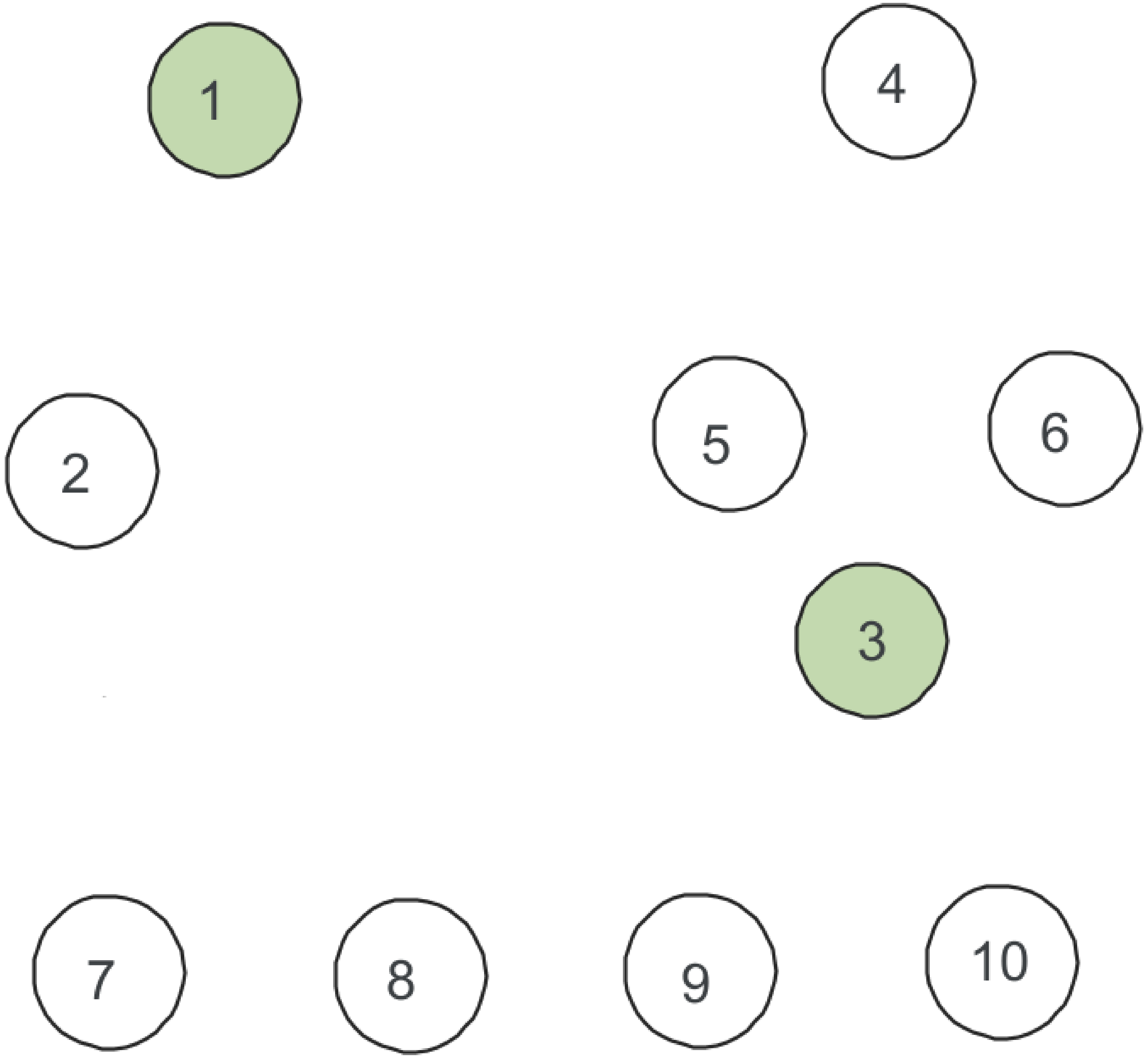


Queries :
1. Find_parent(5) = 5
2. Union(7,8)
3. Find_Parent(8)=7
4. Union(1,2)
5. Union(2,3)
6. same_component(2,7)?=fal
7. Union(4,6)
8. Union(5,6)
9. Union(5,9)
10. Find_Parent(9) =4
11. Union(3,8)
12. same_component(2,7)?=tr
13. same_component(4,9)?=tr
14. Union(1,3)
15. Union(9,10)
16. Find_Parent(7)
17. Union(8,9)
18. Find_Parent(10)



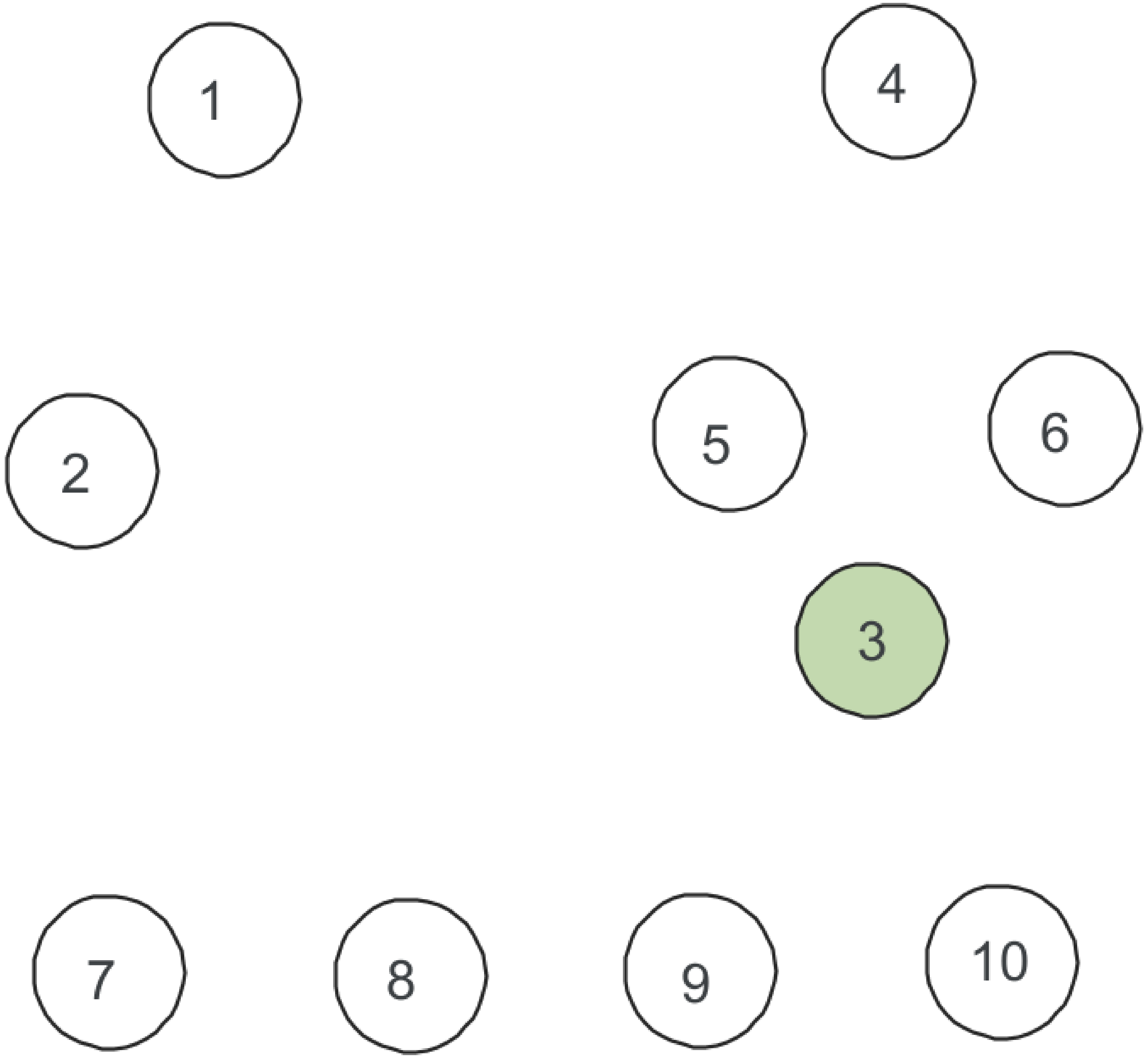

```
Find_Parent(int x)
{
   if(x==parent[x])return x;
   else return Find_Parent(parent[x]);
}
```

Complexity : O(N)



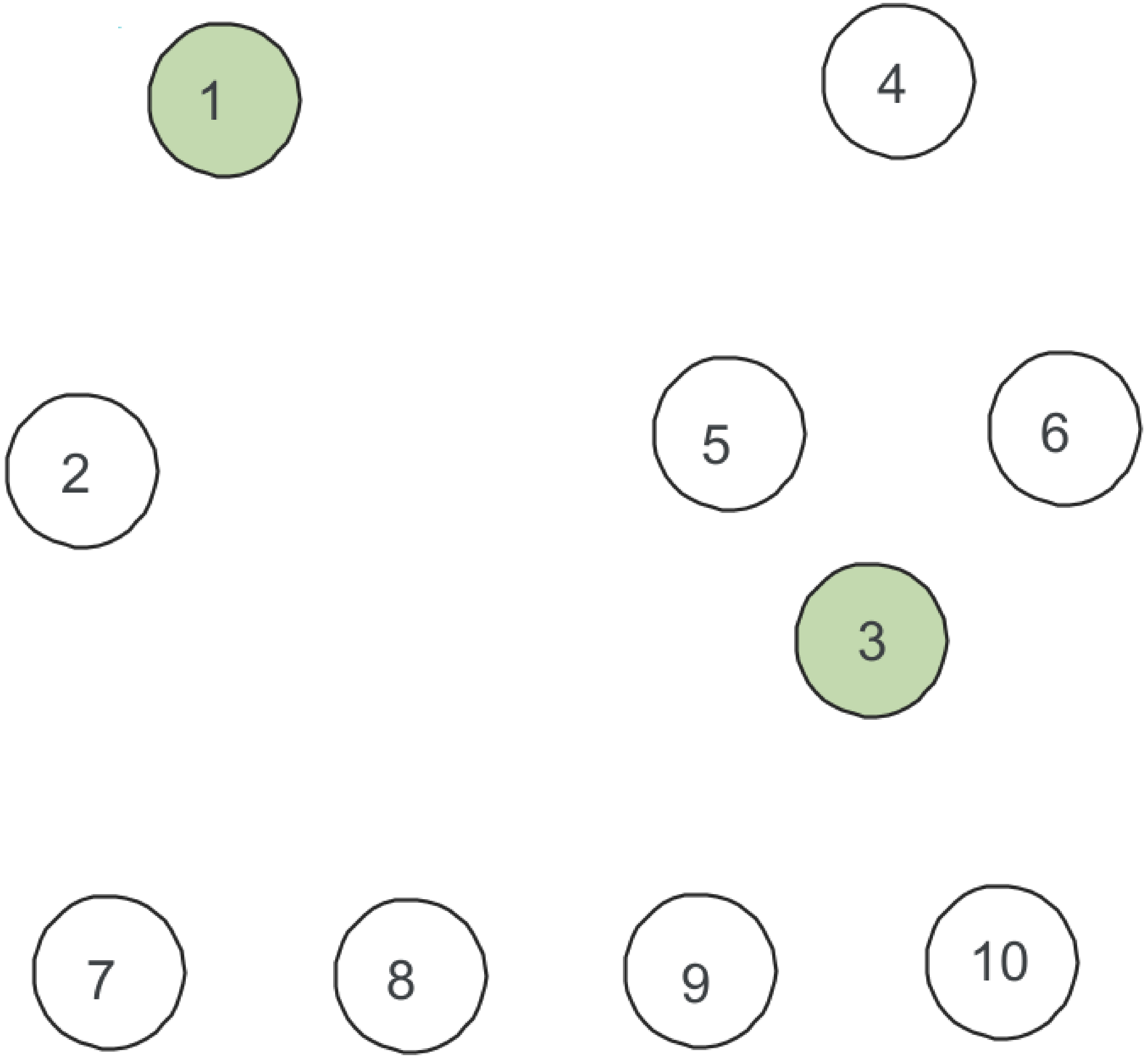

```
Find_Parent(int x)
{
   if(x==parent[x])return x;
   else return parent[x]=Find_Parent(parent[x]);
}
```

Use of Path
Compression to make
Complexity : O(1)




```
Void Union(int x,int y)
{
    int parent_x=Find_Parent(x);
    int parent_y=Find_Parent(y);
    if(parent_x!=parent_y)
    {
        Parent[ parent_y ] = parent_x;
    }
}
```

Complexity : O(1)




```
bool same_component(int x,int y)
{
    int parent_x=Find_Parent(x);
    int parent_y=Find_Parent(y);
    if(parent_x==parent_y) return true;
    else return false

}
```

Complexity : O(1)