1. One additional feature of the list-based implementation of a union-find structure is that it allows for the contents of any set in a partition to be listed in time proportional to the size of the set. Describe how this can be done.

Answer:
Union- find structure can be implemented in two ways either by linked list or with the help of the trees.

For this question, we have implemented set as a linked-list.

Each linked-list has a head node which has following properties:
i. Size of the set.
ii. Name of the set.
iii. Next pointer pointing to the next element of the linked-list. If there are no elements in the linked list, its next pointer will point to null.
iv. Tail pointer pointing to the last element of the linked list. If there are no elements in the linked list, the tail pointer will point to the head of the linked list.

Also, each node present in the linked list has two pointers next and head. head points to the head node of the linked list. This will give us the name of the set in which the element is present. Next pointer points to the next element in the linked list. If there is no element after, the next pointer will point to null.

Suppose we want to get all the elements in the set in which element x is present.

Steps:
a. Declare a temporary variable and store the address of the x.head. Now the temp will point to the head node of the linked list.
b. Set temp equal to temp.next. This will make the temporary variable point to the first element of the linked list.
c. Run a while loop until temp is not equal to null. Print the data pointing to the temporary variable and set temporary variable to temporary.next;

Since we are iterating over all the elements in the linked list the time complexity of the above operation is O(n) where n is the number of elements in the set. The space complexity is O(1).

2. Describe how to implement a union-find structure using extendable arrays, which each contains the elements in a single set, instead of linked lists. Show how this solution can be used to process a sequence of m union-find operations on an initial collection of n singleton sets in O(n log n+m)time.

Answer:
We can implement union find structure with the help of the extendable arrays as well. Let's say we have a set of ten elements.

Let S = {0,1,2,3,4,5,6,7,8,9}
We will use a size array to keep track of the size of the subset.

Initially, each element in the array will have its own subset, so the array and the size of the array will look like this

A = [0,1,2,3,4,5,6,7,8,9]
and
size = [1,1,1,1,1,1,1,1,1,1]

We can do this operation with the help of the makeSet function

```
Algorithm: makeSet(S):
    Input: A set S that contains the elements
    Output: Creating an Array A which stores the name of the subset to which element belongs and the size of the subset
    Create an extendable array A
    for every element in S
        Add element to A
        set size = 1
```

```
Algorithm: find(int A[], int num)
    Input: Array A which contains collections of subsets and num which is used to find the name of the subset to which num belongs
    Output: Name of the subset

    while(A[num] != num)
        num = A[num]
    return num
```

```
Algorithm: UnionFind(int A[], int size[], int x, int y)
    Input: Array A that contains the name of the subset to which element belongs, Array size that contains size of the subset
    Output: Union of the subsets x & y

    root_x ← find(x)
    root_y ← find(y)
    if(size[root_x] > size[root_y]) then
        A[root_y] ← A[root_x]
        size[root_x] += size[root_y]
    else
        A[root_x] ← A[root_y]
        Size[root_y] += size[root_x]
```

From the given algorithms we can see that the UnionFind operation takes logN times to find an element and perform the union operation. So the overall time complexity of the function is O(nlogn + m)

Resources:
https://www.hackerearth.com/practice/notes/disjoint-set-union-union-find/

3.Consider the game of Hex, as in the previous exercise, but now with a twist. Suppose some number, k, of the cells in the game board are colored gold and if the set of stones that connect the two sides of a winning player's board are also connected to k′≤k of the gold cells, then that player gets k′bonus points. Describe an efficient way to detect when a player wins and also, at that same moment, determine how many bonus points they get. What is the running time of your method over the course of a game consisting of n moves?

Answer:
We arrange black pieces along the top (Set BU) and bottom edges (Set BB) and white pieces along the left (Set WL) and right sides of a board (Set WR). This initialization facilitates the search for a winning move. At most O(n) MakeSet operations are used.

For each move, we place a piece in its desired position (i.e. create a set there) and check to see if any surrounding cells are filled by the same coloured pieces. If this is the case, union the newly produced set with the set containing the next cell (there is a maximum constant number of Union operations here).

Then we look to see if the existing component unions are BU and BB or WL and WR. If it does, we have made a winning move and will end the game; otherwise, we will continue. When we locate a winning move, we use the FindSet operation to count the associated gold pieces to this color, as well as all the pieces that are part of Set G (gold pieces), and they are the extra points.

The board initialization takes O(n) time, and each move requires 1 Union and 2 FindSet operations to determine if the border sets are unioned. We'll need k' Find operations after the winning move to compute additional points.

Therefore the union operation takes O(n + k’) time.

4. Suppose we are given a sequence S of n elements, each of which is colored red or blue. Assuming S is represented as an array, give an in-place method for ordering S so that all the blue elements are listed before all the red elements. Can you extend your approach to three colors?

Answer:

The strategy which we are going to use can be used for any amount of colors.

For the explanation to be easier let us assume we have three colors: Yellow, White & Black.

Let us assume we sort the colors by yellow, white and black

Let there be an array representing colors

So, A = [Y, B, B ,W , Y, B, Y,W,Y,Y]

We will initialise a pointer i and j at Array position 0. We will iterate both the pointers until we not find a yellow color. In this case at position 1 there is no yellow color.

Now set j = i + 1 and iterate until we find the yellow color. Here in position 4 we find yellow color. Now we swap the yellow with black color and make i = i + 1.

Once we have iterated over all the positions of the array. It means that we have sorted the yellow color. Now we will again call the sorting function but this time we will pass the sub array and sorting will be done for white color.

Since we are iterating over all the n elements of the array and we are recursively calling the sorting function for m colors. The time complexity of the problem is O(m * n). The space complexity of the above function is O(1).

5. Suppose we are given an n-element sequence S such that each element in S represents a different vote in an election, where each vote is given as an integer representing the ID of the chosen candidate. Without making any assumptions about who is running or even how many candidates there are, design an O(n log n)-time algorithm to see who wins the election S represents, assuming the candidate with the most votes wins.

Answer:

We can use any of the sorting methods for example quick-sort, Merge-sort to sort the elements based on the id.

Once the elements are sorted, we will iterate over the sorted elements to count the maximum number of votes.

Below is the mentioned algorithm to find the winner

```
Algoritihm: Winner(int votes[]):
    Input: Sorted elements of the candidate based on the id
    Ouput: Winner among the candidates

    if(length of the array is equal to 1) then
        return votes[0]
    tie ← false
    max_votes ← 1
    winner ← votes[0]

    i ← 0
    while(i < n) {
        temp ← votes[0]
        Local_sum ← 1
        while((i + 1) < n and votes[i + 1] == temp) {

            i ← i + 1;
            local_sum ← local_sum +  1;
        }
        if(local_sum is equal to max_votes) then
            tie ← true;

        else if(local_sum is greater than max_votes) then {
            Tie ← false;
            Max_votes = local_sum;
            winner ← i;
        }

        i← i + 1
    }

    if(tie)
        return “tie”;
    return winner;
```

The above algorithm runs in O(nlogn) + O(n) which is equivalent to O(nlogn). The space complexity is O(n).

6. **Consider the voting problem from the previous exercise, but now suppose that we know the number k<n of candidates running. Describe an O (n log k)-time algorithm for determining who wins the election.**

Answer:

Since we know how many people are running for the elections we can use a balanced binary tree or hash tables to store the count of each candidate.

Suppose we use a balanced binary tree. Balanced binary tree has two functions: insert and find. We will iterate over all the elements of n. We will first use find to check whether a particular candidate is present in our AVL tree or not. If it is present we will update the counter of that node by 1.

If we could not find that candidate then we will create a new node and initialise the counter to 1 and insert that node in the tree.

After creating an AVL tree, we will iterate over the AVL tree to find the maximum votes.

Since the find and the insert operation in the AVL tree takes logk times and we are iterating over all the n elements.The time complexity of the above algorithm is O(nlogk) and the space complexity is O(k).

7.

Suppose you are given two sorted lists, A and B, of n elements each, all of which are distinct. Describe a method that runs in O(log n) time for finding the median in the set defined by the union of A and B.
Answer:
Since the elements are in the sorted order the first thing which comes in our mind is to use binary Search.

It is given that each element in the list contains elements in each. We will give a generalised solution where the elements in each list is not equal.

Algorithm: findMedian(list A, list B):
Input: Two sorted list A & B
Ouput: Median of the union of A & B

```
    if(A.length > B.length)
        findMedian(B, A)

    n1 ← A.length
    n2 ← B.length
    low ← 0
    high ← n1
    while(low <= high) {

    cut1 ← low + (high - low) / 2
    cut2 ← (n1 + n2 + 1) / 2 - cut1

    left_1 ← 0
    left_2 ← 0
    right_1 ← 0
    right_2 ← 0
    if(cut1 is equal to  0) then
        left_1 = -∞
    else
        left_1 = A[cut1 -1]

    if(cut2 is equal to 0) then
        left_2 = -∞
    else
        left_2 = B[cut2 -1]

    if(cut1 is equal to n1) then
        right_1 = ∞
    else
        right_1 = A[cut1]


    if(cut2 is equal to n2) then
```

```
        right_2 = ∞
    else
        right_2 = B[cut2]

    if(left_1 <= right_2 && left_2 <= right_1) then
        if( (n1 + n2) % 2 is equal to 0) then
            return (max(left_1, left_2) + min(right_1, right_2) ) / 2;

    if(left_1 is greater than right_2) then
        high = cut1 - 1
    else
        low = cut1 + 1;
```

The time complexity of the above algorithm is O(logn) where n is the number of elements and the space complexity is O(1).

8

Suppose University High School (UHS) is electing its student-body president. Suppose further that everyone at UHS is a candidate and voters write down the student number of the person they are voting for, rather than checking a box. Let A be an array containing n such votes, that is, student numbers for candidates receiving votes, listed in no particular order. Your job is to determine if one of the candidates got a majority of the votes, that is, more than n/2votes. Describe an O(n)-time algorithm for determining if there is a student number that appears more than n/2 times in A

Answer:

We can use hash table data structure to solve this problem.

We will use the student number as the key and the number of votes that person has got as the value.Since we know that in the hash table the find and update value runs in a constant time, we will take advantage of it.

```
Algorithm: checkIfStudent(int A[])
    Input: An array A containing n votes
    Output: Return true if a particular student receives a vote greater than n / 2 or else
    return false

    Create a new hash table H
    For each record in A,x:
        if(x present in H):
            Update the value of H for a particular key by value +  1
            if(value of H for a key x is greater than n / 2):
                return true;
        else
            Create a new key in the hashtable H and set its value to 1
    return false;
```