First and Last Occurrence in Sorted Array (Binary Search Variant):

```
function firstnlastoccurrence(arr, target, start, end) {
    if (start > end) {
        return;
    }
    const middle = Math.floor((start + end) / 2);

    if (arr[middle] === target) {
        let s = middle;
        let l = middle;

        while (l > start && arr[l - 1] === target) {
            l--;
        }

        while (s < end && arr[s + 1] === target) {
            s++;
        }

        return { first: l, last: s };
    }

    if (arr[middle] > target) {
        return firstnlastoccurrence(arr, target, start, middle - 1);
    } else {
        return firstnlastoccurrence(arr, target, middle + 1, end);
    }
}

const arr = [1, 3, 5, 5, 5, 5, 67, 123, 125];
console.log(firstnlastoccurrence(arr, 5, 0, arr.length - 1));
```

Removing Duplicates from a Linked List:

```
class Node {
    constructor(data) {
        this.data = data;
        this.next = null;
    }
}

class LinkedList{
    constructor(data) {
        this.head = null
    }
    addFirst(data) {
        const newNode = new Node(data)
        newNode.next = this.head
        this.head = newNode
    }
    size(){
        if(!this.head){

            return count
        }
        let current = this.head
        let count = 0
        while(current){
            current = current.next
            count++
        }
        return count
    }
    print(){
        let current = this.head;
        while(current){
            console.log(current.data)
            current = current.next
        }
    }
    removeOneDuplicate(){
        const m = new Map()
        let current = this.head
        while(current){
           m.set(current.data,(m.get(current.data) || 0) + 1)
           current = current.next
        }
        let s;
        for(const t of m.keys()) {
            if(m.get(t)>1){
                s =  { duplicatevalue:t, numberofDuplicates:   m.get(t)-1}
            }
        }
        let currentForDelete = this.head
        let countDelete = 0
        while(currentForDelete.next && countDelete<s.numberofDuplicates){
            if(currentForDelete.next.data === s.duplicatevalue)
            {
                currentForDelete.next = currentForDelete.next.next
                countDelete++
            }

            else{
                currentForDelete=currentForDelete.next
            }
        }
    }

    removeDuplicate(){
        if(!this.head || !this.head.next){
            return;
        }
        const m = new Map()
        let current = this.head
        let prev = null;
        while(current){
            if(m.get(current.data)){
                if (prev) {
                  prev.next = current.next;
                } else {
                   this.head = current.next;
                }
            }else{
                m.set(current.data, 1)
                prev= current
            }
            current = current.next
         }
    }
}

const linkedlist = new LinkedList();




linkedlist.addFirst(13)
linkedlist.addFirst(10)
linkedlist.addFirst(8)
linkedlist.addFirst(8)
linkedlist.addFirst(5)
linkedlist.addFirst(5)
linkedlist.addFirst(5)
linkedlist.addFirst(3)



//console.log("Size =====>",linkedlist.size())
// linkedlist.print()
//linkedlist.removeOneDuplicate()
linkedlist.removeDuplicate()
linkedlist.print()
```