```
def find_duplicates(lst):
        L=[]------------------------------O(1)
        R=[]------------------------------O(1)
        for i in range(len(lst)):---------O(n)
                L.append(0)---------------O(1)
        for i in range(len(lst)):---------O(n)
                if L[lst[i]]==0:----------O(1)
                        L[lst[i]]=1----------O(1)
                else:-----------------------O(1)
                        L[lst[i]]+=1---------O(1)
        for i in range(len(lst)):--------O(1)
                if L[i] > 1:---------------O(1)
                        R.append(i)--------O(1)
       return R-------------------------O(1)
```

Runtime: O(n)

```
def romve_all(lst,value):
        end=False----------------------O(1)
        while(end==False):-----------O(n)
                try:-------------------------O(1)
                        lst.remove(value)-O(n)
                except ValueError:----O(1)
                        end = True---------O(1)
```

Runtime: O(n^2)

```
def remove_all(lst, value):
    L=[]------------------------------O(1)
    for i in range(0,len(lst)):------O(n)
        if lst[i] != value:----------O(1)
            L.append(lst[i])-----O(1)
    lst[:]=L--------------------------O(n)
```

Runtime:O(n)

**Show that the following series of *2n* operations takes *O(n)* time: *n* append operations on an initially empty array, followed by *n* pop operations.**

Append one element needs O(1) time, so append n elements need O(n) time. Considering the resizing principle, it's a geometric sequence with the common ratio of 2. For example, append 16 elements will require the list to resize when add the 1, 2, 4, 8, 16 elements. Adding those altogether that will be 2n-1, so it's O(n). The situation is similar to the pop methods, when poping 16 elements, the runtime of each cost is …8…4…2…1, so O(n)+O(n)=O(n)

**Consider a variant to our shrinking strategy, in which an array of capacity *N*, is resized to capacity precisely that of the number of elements, any time the number of elements in the array goes strictly below *N/2*. Show that there exists a sequence of *n* append and/or pop operations on an initially empty MyList object, that requires $\Omega(n^2)$ time to execute**

When capacity = 8, and n =4, append an element will need O(n) time. Pop one, it also needs O(n) time. Pop one, it would then need O(1) time, append one, it will need O(1) time, then form a circle. In this sequence, it will need (n+n+1+1)/4*n=O(n^2).