Q4: The first thing that comes to my mind for this is using Stack (there might be better solutions than this)

```
def reverse(self, top):
    stack_l=list()
    temp=top
    while temp:
       stack_l.append(temp)
       temp=temp.next
    temp=top= stack_l.pop()
    while stack_l:
       t= stack_l.pop()
       temp.next= t
       temp= t
    t.next=None
    return top
```

I am using top because I have used a stack, you can use head if you like

Q6. This is exactly like merge-sort. The idea comes from the fact that in merge-sort you have to merge two sorted arrays at all stages. Just modifying the function merge solves the problem:

```
Union(l1,l2):
1. m,n= len(l1), len(l2)
2. i,j,l3 = 0, 0, []
3. while (i+j)<(m+n):
4.   if i==m:
5.      l3.append(l2[j])
6.      j++
7.   else if j==n:
8.      l3.append(l2[i])
9.      i++
10.  else if l1[i]<l2[j]:
11.     l3.append(l1[i])
12.     i++
13.  else if l1[i]>l2[j]:
14.     l3.append(l2[j])
15.     j++
16.  else if l1[i]==l2[j]:
17.     i++
18.     j++
19. return l3
```

```
Intersection(l1,l2):
1. m,n= len(l1), len(l2)
2. i,j,l3 = 0, 0, []
3. while (i+j)<(m+n):
4.   if i==m:
5.      j=n
6.   else if j==n:
7.      i=m
8.   else if l1[i]<l2[j]:
9.      i++
10.  else if l1[i]>l2[j]:
11.     j++
12.  else if l1[i]==l2[j]:
13.     l3.append(l2[j])
14.     i++
15.     j++
16. return l3
```

Q7: This is very simple if you can see that all you need to do is just modify the binary search algorithm. In the usual BS-algorithm you would typically stop searching once you have found the match, the trick here is to keep on searching even if you have found the match (and of course keep track of the position of the last match)

```
def leftpos(x, a):
    l, u= 0, len(a)-1
    pos=-1
    while l<=u:
        mid=(l+u)//2
        if x==a[mid]:
            pos=mid
            u=mid-1
        elif x<a[mid]:
            u=mid-1
        else:
            l=mid+1
    return pos
```

Q9: This is actually a bit difficult, especially if you do not understand Dynamic Programming.

Here we have Watch[i] as the max no of matches that can be watched among {Mi, Mi+1, ..., Mn}. Now note that we have two options right away and we have to choose the best option.

Option 1: Watch the i-th match. We will get to watch one match immediately and as many matches as we can manage if we start with Next[i] , so the total no. of matches we can watch is 1+Watch[Next[i]] (adding 1 since we are watching i-th match immediately)

Option 2: Skip the i-th match and the no. of matches we can watch is Watch[i+1]

Now note that Watch[n]=1 because you would not want to skip the last match. Thus we have that as the base-case and the recurrence is:

Watch[i]= max(1+Watch[Next[i]], Watch[i+1]) (taking the best of the two)

Calculating Watch[] is very easy since it is a one-dimensional array, with the base case: Watch[n]=1
So we could just walk backwards to calculate Watch[n-1], Watch[n-2], ..., Watch[1]. And hence we get our desired answer (i.e., Watch[1]) using DP.