Q1:
Firstly look at the input size, we should know that it cost too much time if we simply delete numbers using list and look at if there is element in N not in A every time. We should think about is there any way to use the data used before. And of course, after Homework 1 I have learned to think from the beginning to improve time efficiency,
The logic of Q1 shall be very clear. After briefly thinking, we can observe that
1, numbers are pairs, the remaining numbers are pairs that are pairs in the original lists.
2, if we define the listA number as A, listN number as N, then when (A,N) is deleted, if you look at the current count of A in the listA, if it it 1, after deleting it is 0, and in the listN the number A MUST be deleted in the next round.
WHY? Because there is no chance of deleting (A, N=A), so when there is one A in listA but deleted, there must be left an N==A in the listN.
3, deleting the numbers for rounds share the same consequence with the delete thoroughly for each numbers.
4, use stack store the elements deleted, and use qsize to know how many have been deleted

Now coding: we create a dictionary to store the original pair {N:A}, then create a counts to count the original occurs of elements in listA, each time there is a deletion, delete the count of the specific element, stack append the deleted number, then search for the next to find is there more deletion, and search for the next round... use recursion to improve time efficiency, with only O(n). Then after traversal. The deletion is accomplished. Finally just count the size of the stack
Q2:
About time complexity, it is simple to observe that only when (digit,n) circumstances, the time complexity is n, in other circumstances, like(n,n)(digit,digit),,all 1
The difficult part that bothered me is how to identify every loop of the function, functions can be 2 ways: paralleling and nesting.
When paralleling, the times are compared, when nesting, the times are added. I figured out a terrific way to solve it;

If we store all the lines as 1(F has n complexity),0((n,n)(digit,digit)) and E in a single list, the most perfect form is to just sum up all the 1s and 0s(if all nested). Now, if you take a closer look at the 0 function, it is pretty simple to notice that they do not make difference to time complexity, so firstly, we make all the E that corresponds to 0 turn to 0, actually you can delete then as well , but i choose this way.

Now there are only functions of 1. Expanding on this, consider this : if there are consistent digits of 1 in the list, this is nesting, if there is forms like[1,E], this is paralleling, and only when the compared function which has more time complexity can replace the old one:
So here is the way, we replace the rest E that corresponds to 1 to -1, why? I decide to sum the element in the list one by one, and there is sum(sum of elements so far) and

maxnum(result), maxnum means that the maximum number of sum when calculating. Once I meet a -1, it means the end of one function, we offset the time complexity by let sum -1, Eventually, key point is paralleling functions are compared. Every time before comparison, because sum-1 every time a function ends, the nesting ones all disappear,no confusion exists

e.g.1 1 1 1 -1 -1 -1 -1 1 -1,this is (On^4)vs(On) right?, when i am calculating, till the eight th one, sum is 0(nests are canceled!), and i compare it with (On), however the rest ones can not be more than max sum, so O(n4)wins

Now coding:
How do i identify pairs? Use stack, because it is LIFO
Every time i read a line, if the line is a function, then not only list append 1 or 0, stack append 1 or 0
Every time i read a E, i pop one element form the stack(which is its pair function), see if this is 1 or 0, if 1, list append-1,if0 list append0 (once again, it is also acceptable to do not consider0 at all, but i want to be consistent)
Then the list i want (has 1,-1,0)appears, and then calculate maximum sum. Output.
This is the most high efficiency and simplest way i can think of.