Question 1:

1.1

1. In order to find the smallest value gap between two elements in a list, First we need to sort the list, because for a given element X, the smallest gap between itself and another element can only be 1 of the 2 possibilities: X and the biggest element within those are smaller than X or X and the smallest element within those are bigger than X.
2. Use a sorting method that takes $O(nlogn)$
3. After sorting, we go through the list again and record every gap between two neighbor elements, which takes linear time
4. We get the smallest gap and that is the answer
5. Total time: still $O(nlogn)$

1.2

1. Augmented BST can do this. Use the elements in the list to build an augmented BST as preprocessing.
2. For query: first we get the 'min-gap' after we build the tree and store it. For each insertion, we get the inserted element and compare the 2 gaps of it with its 2 neighbors, get the smaller one, compare it to 'min-gap', if it is smaller, update 'min-gap', else, discard it and keep 'min-gap'. Upon query, we return 'min-gap' in constant time.
3. For update: As stated in the lecture, when building the tree, store subtree sizes rather than ranks in the nodes. So subtree sizes can be updated when inserting and rebalancing. And each insertion in augmented BST takes $O(logn)$

1.3

1. For each node, add 'min-gap of subtree', 'min-element of subtree' and 'max-element of subtree' to it.
2. Upon each deletion, we update these infomation in the current node and go all the way up to the root and update infomation in each node. So the depth is at most $logn$ with time $O(1)$ for each level. So total time for one deletion is $O(logn)$

Question 2:

1. Choose every element from the list one by one, from left to right. For each element, insert it into an augmented BST.
2. For each insertion: find the element's rank, then we know how many elements in the current tree is bigger than this element (total number - rank).
3. So for this element, this is exactly the same number as how many elements in the list that are both on its left and bigger than itself. Add this number to the variable 'sum' (default value: 0)
4. Return 'sum' when every element in the list is inserted, and this is the 'flip' we want.
5. Runtime: O(logn) for each element, O(nlogn) in total

Question 3:

- Define subproblem: D[i,j] as edit distance between prefixes A[0,1,...,i] and B[0,1,...,j].
- So we can compute D[i,j] in terms of smaller subproblems:

$$D[i,j] = \begin{cases} max\,\{i,j\} & \text{if } min\,\{i,j\} = 0 \\ D[i-1,j-1] & \text{if A[i]=B[j]} \\ 1 + min\,\{D[i-1,j-1], D[i-1,j], D[i,j-1]\} & \text{otherwise} \end{cases} \tag{1}$$

- Runtime: subproblem number: mn, each takes constant time, so total runtime: O(mn)