Q 1.1)

1. Finding the values of the last two entries of the sub-sequence in using given indices and in constant time.
2. Computing a supposed value of the third last entry using a formula in constant time.
3. Using the computed value, search for the value in using binary search and get the index in in time.

Since , if is the last entry then and would be the second last and the third last in order. If knowing any 2 values of those, we would be able to determine another. In this case, we know and , then .

Overall time complexity is .

Q 1.2)

**Subproblems:** for each and , Let be the problem of determining , the maximum number of elements that could form a beautiful sub-sequence having as the first element of the sub-sequence and as the last element, ignoring the minimum number required to be called beautiful. And , the possible number that could be the next element after in the sub-sequence that is the first element.

**Recurrence:** for and ,

If is matching with any previous , meaning that should be the next element in a beautiful sub-sequence after . Assuming is , then needs to be . And we could get the next possible number () of this sequence by computing . This is done by using binary search for all in time. An array is strictly increasing causing no problems of having duplicated values.

Otherwise, meaning that couldn’t be any of the others next element in the sub-sequence, except from being the second element next after (first element).

If is matching with any previous , as explained above that should be the next element after . Then the maximum number of elements that could form the sub-sequence is the maximum number of elements that form up to then adding by 1 (interpreting having as the new last element).

Otherwise, if could only be the second element in the sub-sequence, then the maximum number of elements in the sub-sequence would only be 2.

Since depends on , we solve problems in increasing order of then . Each interpreting having every number in to be the first element of a beautiful sub-sequence in order to cover all possible cases to form a sub-sequence.

**Base cases:** if then and is undefined.

Interpreting that the first element of a beautiful sub-sequence could have any number as its next element of the sub-sequence and there is currently only 1 element (itself) in the sub-sequence.

The length of the longest beautiful sub-sequence of is the highest since we only care about the number of elements in the sub-sequence, and doesn’t matter which element is at first or at last. However, if it is less than 3 then the length is 0, because the shortest length of a beautiful sub-sequence possible is 3. This could be done by keep tracking and updating of the highest so far (if any 2 sub-sequences have the same number of elements, then both sub-sequences are valid), so it takes constant time.

Overall time complexity is . By iterating through elements in , and for each element, using binary search in time, costing . This is done times for letting each element in being the first element in the sub-sequence, in total.

Note that even though the amount of time iterating is decreased by 1 every time is increased but the time complexity is still . Since

And .

Q 1.3)

By using of the highest that has been tracked. Then and are the first and the last element in the longest beautiful sub-sequence. If highest is less than 3, then there’s no beautiful-subsequences (as explained above).

Creating an array of size then putting and as the first and the last element of the array.

Then using binary search for that and place in the array index before . The idea of why is going to be the number before in the sub-sequence had been explained above as the definition of . Doing this again to find the number before and so on. Until at any point that is from the binary search and , meaning that this is the second element in the sub-sequence, and don’t need to find any more number before this since the first element before this is .

The array we got is a list of all entries in the longest beautiful sub-sequence using logic of Q1.2.

Overall time complexity is . Creating an array of size (at most) in time. And using binary search at most times (in the worst case), for every element in causing it to be . time in total.