# COMP3121 Assignment 3 – Q1

**1.1**  
A sequence is beautiful if the sequence has size equal to or greater than 3 and that .

Given a beautiful subsequence , which is a subsequence of array , consider an algorithm which ­will compute the index in of the third last entry of the sub-sequence.

This algorithm will begin by computing the value third last entry of the subsequence. Following the criteria of a beautiful subsequence, the value of the third last entry can be represented as in the following equation which can be rearranged as such:

Then, the algorithm can perform a binary search over for and return the index of in .

Thus, computing the index in of the third last element in the beautiful subsequence.

Due to the criteria of a beautiful subsequence, duplicates in array do not matter as they will create the same beautiful subsequence if it fits the equation.

In the case that the subsequence does have duplicate values, it will not affect the index as the values in are strictly increasing and positive.

This algorithm will have an overall time complexity of as the time complexity of computing the equation is and the time complexity of running a binary search over is as the maximum length of the array is .

**1.2**Given the array , consider an algorithm which computes the length if the longest beautiful subsequence of .

This algorithm will utilise a dynamic programming approach and use memoisation to store previous computations to use in future computations to avoid unnecessary repetition.

This problem can be divided into subproblems, , which will return the longest beautiful subsequence which ends at index and is the second-last element of the subsequence in .

To begin, the algorithm will initialise a 2D array with the dimensions , where will store the result of .

The algorithm will iterate loop through the values from 0 to where and within each iteration, loop through 0 to for to iterate over all possible pairs of and .

With these values, the algorithm will compute the result for each possible pair of and to be stored in . This is done using the algorithm from question 1.1.

With the use of an if statement, the algorithm will use *algorithm 1.1* to determine whether the index, , for the third last entry for a beautiful subsequence exists in for such and index values.

If exists, the algorithm will update . This is the basis of the recurrence relation where the length of the subsequence from 0 to will be a result of the previous existing subsequence.

Or else, if doesn’t exist, . This will also act as a base case as the first few iterations of and as the earlier indexes do not have enough elements to create a beautiful subsequence as the subsequence must be at least of length 3.

The algorithm will iterate through all possible values of and to completely fill array .

Once these iterations are complete, the algorithm will initialise a variable and iterate through every index in to find the maximum value in the 2D array.

After finding the maximum value, the algorithm will run a final check if is greater than or equal to 3.

If , the algorithm will return .

If , the algorithm will return 0 as the beautiful subsequence must be at least 3 elements long.

To compute the overall time complexity of this algorithm, the following time complexities are considered:

* Initial iterations of and to fill with all the values of :
  + For each iteration, the algorithm will run algorithm 1.1:
* Finally, the final iteration to find

As such, the overall time complexity is the sum of ) which can be simplified to .

**1.3**Consider an algorithm which runs in additional time and lists the entries of the longest beautiful subsequence in .

To begin, this algorithm will use the algorithm from question 1.2 to determine the length of the longest beautiful subsequence in .

Then, the algorithm will find the indexes and which gave the maximum (from 1.2). This will provide the indexes for the elements which make up the second-to-last and last elements in the longest beautiful subsequence.

To store the entries of the subsequence, the algorithm will initialise list and begin by appending to the list.

Then, the algorithm will start a while loop with the condition that while :

* Append to
* Find the third last element (index ) of the subsequence using the algorithm from question 1.1
* Update the indexes and

When this process is finished, finally, append with the last most updated to list .

To complete this process, the algorithm will reverse list and return it, resulting in a list of the entries.

The time complexity of this algorithm begins with the 1.2 which runs in

The additional time is determined mainly by the while loop and the algorithm which is run per iteration. The time complexity of the algorithm is (as per question 1.1) and it will be run times. Resulting in a time complexity of additional time.