COMP3121 Assignment 3 – Q1

1.1

A sequence is beautiful if the sequence has size equal to or greater than 3 and that $3x_i + 5x_{i+1} = x_{i+2}$.

Given a beautiful subsequence $[\dots, a_j, a_k]$, which is a subsequence of array A, consider an algorithm which will compute the index in A 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 a_i in the following equation which can be rearranged as such:

$$\begin{aligned} \mathbf{3}\boldsymbol{a_i} + 5a_j &= a_k, \\ \mathbf{3}\boldsymbol{a_i} &= a_k - 5a_j, \\ \boldsymbol{a_i} &= (a_k - 5a_j) \,/\, 3. \end{aligned}$$

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

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

Due to the criteria of a beautiful subsequence, duplicates in array A 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 A are strictly increasing and positive.

This algorithm will have an overall time complexity of $O(\log n)$ as the time complexity of computing the equation is $O(1)$ and the time complexity of running a binary search over A is $O(\log n)$ as the maximum length of the array is n.

1.2

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

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, $opt(i, j)$, which will return the longest beautiful subsequence which ends at index i and j is the second-last element of the subsequence in A.

To begin, the algorithm will initialise a 2D array DP with the dimensions $n * n$, where $DP[i][j]$ will store the result of $opt(i, j)$.

The algorithm will iterate loop through the values from 0 to j where $j \leq n$ and within each iteration, loop through 0 to i for $i < j$ to iterate over all possible pairs of i and j.

With these values, the algorithm will compute the result $opt(i, j)$ for each possible pair of i and j to be stored in $DP[i][j]$. 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, k, for the third last entry for a beautiful subsequence exists in A for such i and j index values.

If k exists, the algorithm will update $DP[i][j] = DP[k][i] + 1$. This is the basis of the recurrence relation where the length of the subsequence from 0 to j will be a result of the previous existing subsequence.

Or else, if k doesn't exist, $DP[i][j] = 2$. This will also act as a base case as the first few iterations of i and j 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 i and j to completely fill array DP.

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

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

If $maxLength \geq 3$, the algorithm will return $maxLength$.

If $maxLength < 3$, 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 i and j to fill DP with all the values of $opt(i, j)$: $O(n^2)$
    - For each iteration, the algorithm will run algorithm 1.1: $O(\log n)$
- Finally, the final iteration to find $maxLength$: $O(\log n)$

As such, the overall time complexity is the sum of $O(n^2 \log n) + O(n^2)$ which can be simplified to $O(n^2 \log n)$.

1.3

Consider an algorithm which runs in $O(n \log n)$ additional time and lists the entries of the longest beautiful subsequence in A.

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

Then, the algorithm will find the indexes i and j which gave the maximum $DP[i][j]$ (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 B and begin by appending $A[j]$ to the list.

Then, the algorithm will start a while loop with the condition that while $DP[i][j] > 2$:

- Append $A[i]$ to B
- Find the third last element (index k) of the subsequence using the algorithm from question 1.1
- Update the indexes $j = i$ and $i = k$

When this process is finished, finally, append $A[i]$ with the last most updated i to list B.

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

The time complexity of this algorithm begins with the 1.2 which runs in $O(n^2 \log n)$

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 $O(\log n)$ (as per question 1.1) and it will be run n times. Resulting in a time complexity of $O(n \log n)$ additional time.