Q 1.1)

A. Finding the values of the last two entries of the sub-sequence in A using given indices j and k in constant time.

B. Computing a supposed value of the third last entry using a formula $\frac{A[k]-5A[j]}{3}$ in constant time.

C. Using the computed value, search for the value in A using binary search and get the index in A in $O(\log n)$ time.

Since $3x_i + 5x_{i+1} = x_{i+2}$, if x_{i+2} is the last entry then x_{i+1} and x_i 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 x_{i+2} and x_{i+1}, then $x_i = \frac{x_{i+2}-5x_{i+1}}{3}$.

Overall time complexity is $O(1 + 1 + \log n) = O(\log n)$.

Q 1.2)

Subproblems: for each $1 \leq i \leq n$ and $i \leq j \leq n$, Let $P(i,j)$ be the problem of determining $opt(i,j)$, the maximum number of elements that could form a beautiful sub-sequence having $A[i]$ as the first element of the sub-sequence and $A[j]$ as the last element, ignoring the minimum number required to be called beautiful. And $s(i,j)$, the possible number that could be the next element after $A[j]$ in the sub-sequence that $A[i]$ is the first element.

Recurrence: for $i > 0$ and $i \leq j \leq n$,

$$s(i,j) = \begin{cases} 3A[k] + 5A[j], & if\ k < j\ AND\ A[j] = s(i,k) \\ 3A[i] + 5A[j], & otherwise. \end{cases}$$

If $A[j]$ is matching with any previous $s(i,k)$, meaning that $A[j]$ should be the next element in a beautiful sub-sequence after $A[k]$. Assuming $A[k]$ is x_i, then $A[j]$ needs to be x_{i+1}. And we could get the next possible number (x_{i+2}) of this sequence by computing $3A[k] + 5A[j]$. This is done by using binary search for all $k < j \leq n$ in $O(\log n)$ time. An array A is strictly increasing causing no problems of having duplicated values.

Otherwise, meaning that $A[j]$ couldn't be any of the others next element in the sub-sequence, except from being the second element next after $A[i]$ (first element).

$$opt(i,j) = \begin{cases} opt(i,k) + 1, & if\ k < j\ AND\ A[j] = s(i,k) \\ 2, & otherwise. \end{cases}$$

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

Otherwise, if $A[j]$ 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 $opt(i,j)$ depends on $\{opt(i,k)|k<j\}$, we solve problems in increasing order of j then i. Each i interpreting having every number in A 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 $i=j$ then $opt(i,j)=1$ and $s(i,j)$ 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 A is the highest $opt(i,j)$, 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 i,j of the highest opt 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 $O(n^2 \log n)$. By iterating through n elements in A, and for each element, using binary search in $O(\log n)$ time, costing $O(n * \log n)$. This is done n times for letting each element in A being the first element in the sub-sequence, $O(n * n \log n) = O(n^2 \log n)$ in total.

Note that even though the amount of time iterating j is decreased by 1 every time i is increased but the time complexity is still $O(n^2 \log n)$. Since

$$(n-0)+(n-1)+(n-2)+\cdots+\big(n-(n-1)\big)=\frac{n(n+1)}{2}$$

And $O\left(\frac{n^2+n}{2}\right)=O(n^2)$.

Q 1.3)

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

Creating an array of size $opt(i, j)$ then putting $A[i]$ and $A[j]$ as the first and the last element of the array.

Then using binary search for k that $s(i, k) | k < j \; AND \; s(i, k) = A[j]$ and place $A[k]$ in the array index before $A[j]$. The idea of why $A[k]$ is going to be the number before $A[j]$ in the sub-sequence had been explained above as the definition of s. Doing this again to find the number before $A[k]$ and so on. Until at any point that q is from the binary search and $opt(i, q) = 2$, meaning that this $A[q]$ is the second element in the sub-sequence, and don't need to find any more number before this $A[q]$ since the first element before this is $A[i]$.

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 $O(n \log n)$. Creating an array of size (at most) n in $O(n)$ time. And using binary search at most n times (in the worst case), for every element in A causing it to be $O(n * \log n) = O(n \log n)$. $O(n + n \log n) = O(n \log n)$ time in total.