Q1)

① Case : greedy strategy 1

Counter Example Input = 2, 5, 9, 7, 6, 1

	greedy algorithm	Optimum sol.^n
① Sol.^	~~9, 7, 2, 1~~ 2, 9, 7, 1	5, 9, 6, 1
② Sum	19	21

② Case : greedy strategy 2

Counter Example Input = 1, 2, 3, 4

	greedy algorithm	optimum sol.^n
① Sol.^n	2, 3	~~4, 3, 1~~ 1, 3, 4
② Sum	5	8

③ Case : greedy Strategy 3

Counter Example Input = 1, 2, 5, 9, 7, 6, 4, 3

	greedy algorithm	optimum sol.ⁿ
① Sol.ⁿ	~~5, 9, 4, 6, 1~~ 1, 5, 9, 6, 4	~~9, 7, 4, 3, 2, 6~~ 1, 2, 9, 7, 4, 3
② Sum	25	26

(Q2) Heart of Algorithm

① **Description**: $S[i]$ = max. possible sum of subsequence till i^{th} array value such that there are no 3 consecutive elements from the original sequence (till i^{th} array value)

② **Recurrence**:

(Assuming original array "A" has values stored in it starting from index = 1 upto index = no. of elements)

$$S[i] = \begin{cases} 0, & i = 0 \\ A[1], & i = 1 \\ A[1] + A[2], & i = 2 \\ \max\left(\max\left(A[i-1] + S[i-3], S[i-2]\right) + A[i], S[i-1]\right), & i >= 3 \end{cases}$$

Base Cases

③ **Solution**: $S[n]$

Q2) Complexity Analysis

The algorithm involves single traversal over the elements of array to compute the desired outcome.

Hence, complexity = $O(n)$

(Q3) Correctness Argument

Let's consider 2 examples.

eg. ① 5, 1, 2, 3, 4

What would be the answer for this = 1 (i.e. '5' needs to be shifted at the rightmost end).

eg. ② 2, 3, 4, 5, 1

What would be the answer for this = 1 (i.e., '1' shifts to the leftmost end).

In both examples, we shift only once & get the desired result (instead of having to do multiple shifting of other array elements). Our intuition here is to see that in eg. ①, we already have a sub-array in desired order (increasing) & ∴, doesn't require any shifting. However, the elements which are out of order, they need to be shifted (1 shift per each out-of-order element) to the correct position.

Similarly for eg., ②, our same intuition leads us to the correct answer.

Now, let's take 1 more example.

eg. ③ 3, 1, 5, 2, 4

Here, Number of shifts reqd. to get the array in ascending order (cards) = 2 (= out-of-order elements)

⇒ 3, 1, 2, 4, **5**
⇒ 1, 2, **3**, 4, 5

We move '5' to the rightmost end & then place '3' in the correct position.

Notice that our intuition still holds valid here, where we have considered the already ordered sub-array as [1, 2, 4] & out_of_order elements = 3, 5.

But also note that [1, 2, 4] is not the only sub-array (or, perhaps a more correct term "subsequence") that is in ascending order. Subsequence [3, 5] is also a valid ordered-set (which also follows our intuition) & so, out-of-order elements = [1, 2, 4], & then for this scenario, the number of shifts reqd. to get desired output = 3 (= no. of out-of-order elements)

greater than 2 (previous shift count)

⇒ **1**, 3, 5, 2, 4
⇒ 1, **2**, 3, 5, 4
⇒ 1, 2, 3, **4**, 5

From this example, we can add to our intuition that choosing a sequence that is in ascending order & is the longest among all such possible subsequences, is the right thing to do.

Reason: Choosing the Longest sub-sequence that is in ascending order (Longest Increasing Sub-sequence - LIS) would ensure that ~~out-of-order~~ number of out-of-order elements for the chosen LIS, would be least as compared to that of other subsequences (of length ≠ longest length), since the elements in chosen LIS are in desired order, so the count of ~~total array values~~ (total array values − elements in LIS) would be the least. This least count of out-of-order elements would ~~req~~ be our answer (minimum shifts required) as each out-of-order element would require 1 shift to ~~be~~ finally be in correct position.

Q3) Complexity Analysis

① Take input : $O(n)$

② LIS ~~dypar~~ algorithm : $O(n^2)$

③ Find max. element in dp array generated by LIS algorithm : $O(n)$

④ Output
= Total elements - maxCount (from point ③) : ~~$O(n)$~~ $O(1)$

Overall Complexity : $O(n^2)$