The Longest Almost Increasing Subsequence Problem with Sliding Windows

Cheng-Han Ho
Department of Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan

Chang-Biau Yang
Department of Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan
cbyang@cse.nsysu.edu.tw

***Abstract*—In this paper, we first define the *longest almost increasing subsequence with sliding windows* (LaISW), which is a generalized combination of the *longest increasing subsequence with sliding windows* (LISW) problem and the *longest almost increasing subsequence* (LaIS) problem. Given a numeric sequence A, along with a tolerance constant c and a window size w, the LaISW problem aims to find the longest almost increasing subsequence (LaIS) within all windows of size w, where an almost increasing subsequence allows for slight decreases less than c. Then, we propose an efficient algorithm to solve the LaISW problem. In our algorithm, we calculate the change of drop out (occurrence) of each element in the row tower, rather than building the entire row tower. The time and space complexities of our algorithm are O(nL) and O(L), respectively, where n denotes the length of the input sequence, and L denotes the length of the LaISW answer.**

***Index Terms*—longest increasing subsequence, longest almost increasing subsequence, sliding window, row tower, principal row**

I. Introduction

Given a numeric sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$, the *longest increasing subsequence* (LIS) problem [1]–[6] aims to find the increasing subsequence of A with the maximal length. In other words, we want to find the maximum length of the longest subsequence of the given sequence such that every element is greater than the previous element. For example, consider $A = \langle 2,\ 7,\ 5,\ 6,\ 1,\ 9,\ 8,\ 11,\ 10 \rangle$. The LIS answers are $\langle 2,\ 5,\ 6,\ 9,\ 11 \rangle$, $\langle 2,\ 5,\ 6,\ 9,\ 10 \rangle$, $\langle 2,\ 5,\ 6,\ 8,\ 11 \rangle$ or $\langle 2,\ 5,\ 6,\ 8,\ 10 \rangle$, whose lengths are all 5. Note that in LIS, the elements are strictly increasing, which means that there are no duplicate elements.

In 1961, the LIS problem was first defined by Schensted [1], and he proposed an O$(n \log n)$-time algorithm for solving this problem, where n denotes the length of the input sequence. In 1977, Hunt and Szmanski [2] gave an O$(n \log \log n)$-time algorithm by using the van Emde Boas tree (vEB tree) [7] when the input sequence A is a permutation of $\{1, 2, \ldots, n\}$. In 2000, Bespamyatnikh and Segal [3] proposed an O$(n \log \log n)$-time algorithm by using the vEB tree. Their algorithm has the capability to report all LIS answers. In 2010, Crochemore and Porat [5] reduced the time complexity to O$(n \log \log L)$ time in the RAM model, where L denotes the LIS length. In 2013, Alam and Rahman [6] gave a divide-and-conquer method with O$(n \log n)$ time.

In real-world applications, the strictly increasing subsequence in LIS could be relaxed to achieve a longer subsequence. For example, Fig. 1 shows the stock prices of TSMC in the first half year of 2023, where the red curve represents the almost increasing trend. Thus, Elmasry in 2010 [8] first defined the *longest almost increasing subsequence* (LaIS), a generalized variant of the LIS problem, which allows for slight decreases (less than a predefined tolerance constant c) from the maximum element encountered so far. For example, given a sequence $A = \langle 2,\ 7,\ 5,\ 6,\ 1,\ 9,\ 8,\ 11,\ 10 \rangle$ and a tolerance constant $c = 2$, then the LaIS answer is $\langle 2,\ 5,\ 6,\ 9,\ 8,\ 11,\ 10 \rangle$, with length 7. Elmasry also gave an O$(n \log L)$-time algorithm for finding LaIS [8].

The *longest increasing subsequence with sliding windows* (LISW), a variant of LIS, was first defined by Albert *et al.* [9] in 2004. For solving the LISW problem, they presented a *row tower* method with O$(n \log \log n + nL)$ time and O(n) space, where L denotes the answer length. In 2007, Chen *et al.* [10] gave an O(nL)-time algorithm by using the canonical antichain partition. In 2012, Deorowicz *et al.* presented a cover-merge algorithm in O$(n \log \log n + \min(nL, n\lceil L^3/w \rceil) \log \lceil w/L^2 + 1 \rceil)$ time, where w denotes the window size. In 2018, Li *et al.* [11] presented a data structure called *quadruple neighbor list*, which can solve the problem within O(nL) time.

The *longest increasing circular subsequence* (LICS) problem is another LIS variant. For given a numeric sequence A, the LICS problem aims to find the LIS in all rotations of A, where a rotation means removing some prefix elements and appending them at the end, resulting in a circular sequence. If the window size is set to n and the input sequence A is repeated twice, then the LISW algorithm can obviously solve the LICS problem. Therefore, LISW can be seen as a more generalized problem than LICS. The LICS was first defined by Albert *et al.* in 2007, and it was solved in O$(n^{3/2} \log n)$ time [12]. Then, in 2009, Deorowicz [13] presented a cover-merge algorithm with O$(min(nL, n \log n + L^3 \log n))$ time.

In this paper, we will first define the *longest almost increasing subsequence with sliding window* (LaISW) problem, a generalized combination of the LISW and LaIS problems. Given a numeric sequence $A = \langle a_1,\ a_2,\ \ldots,\ a_n \rangle$, along

This research work was partially supported by National Science and Technology Council of Taiwan under contract NSTC 112-2221-E-110-026-MY2.

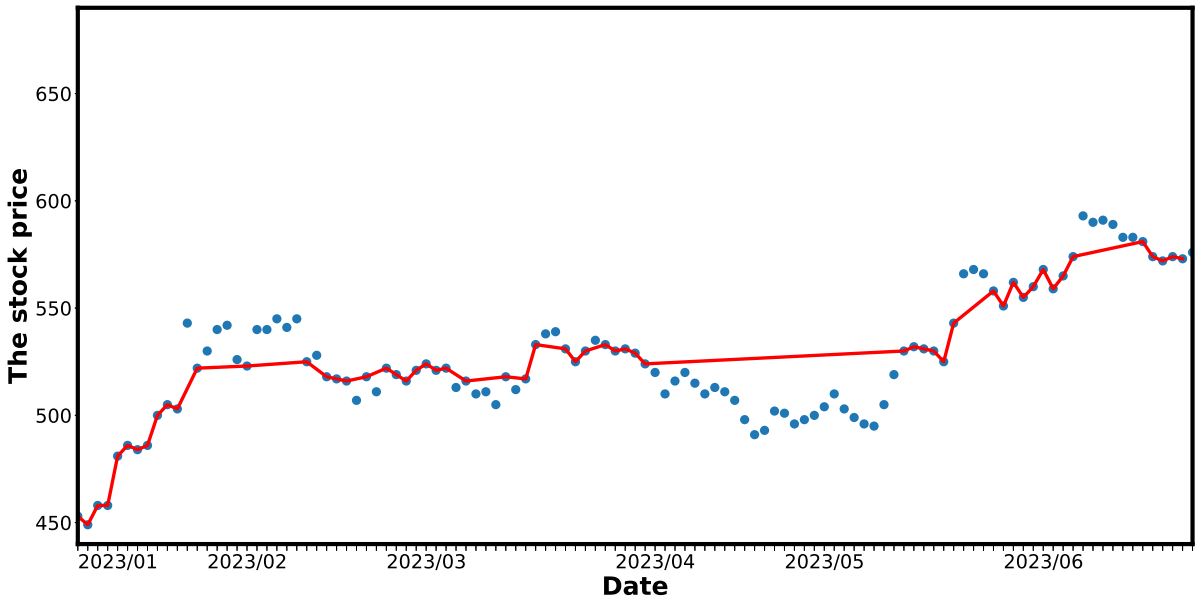

Fig. 1. The stock prices of TSMC (Taiwan Semiconductor Manufacturing Co., Ltd.) from 2023/1/1 to 2023/6/30. The red curve represents the almost increasing trend.

with a tolerance constant c and a sliding window size w, the goal of the LaISW problem is to find the almost increasing subsequence with the maximal length among all substrings of A with length w. Then, we propose an efficient algorithm for solving the LaISW problem. In the algorithm, instead of building the whole row tower, we compute the change of drop out (occurrence) of each element in the row tower. The time and space complexities of our algorithm are $\mathrm{O}(nL)$ and $\mathrm{O}(L)$, respectively.

The rest of this paper is given as follows. In Section II, we briefly review the LaIS problem and the LISW problem. Section III formally defines the LaISW problem. We propose an algorithm for solving it with $\mathrm{O}(nL)$ time and $\mathrm{O}(L)$ space in Section IV and present a complete example in Section V. Finally, Section VI gives the conclusions.

II. Preliminaries

The *longest almost increasing subsequence* (LaIS) problem represents a generalized variant of the LIS problem, first proposed by Elmasry in 2010 [8]. Unlike LIS, LaIS does not require the sequence to be strictly increasing but allows for slight decreases in the sequence as long as they do not exceed the predefined tolerance constant. In essence, the goal of the LaIS problem is to find the *almost increasing subsequence* with the maximal length.

Definition 1. (almost increasing sequence) [8] *Given a numeric sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$ and a tolerance constant c, A is an* almost increasing sequence *(aIS), if $\forall i$, $2 \leq i \leq n$, $a_i > \max\{a_k | 1 \leq k \leq i-1\} - c$.*

For example, suppose we are given a sequence $A = \langle 2, 7, 5, 6, 1, 9, 8, 11, 10 \rangle$ and a tolerance constant $c = 4$, then the LaIS answer is $\langle 2, 7, 5, 6, 9, 8, 11, 10 \rangle$, whose length is 8. If the tolerance constant $c = 2$, then the LaIS answer is $\langle 2, 5, 6, 9, 8, 11, 10 \rangle$, whose length is 7. Note that 5 cannot be appended behind 7, because it is not true for $5 > 7 - c = 5$. Furthermore, the answer obtained from LaIS may be longer than that from LIS because LaIS allows for some slight decreases.

TABLE I
An example for the LaIS algorithm [8], where $A = \langle 2, 7, 5, 6, 1, 9, 8, 11, 10 \rangle$, and $c = 4$. The LaIS answer is $\langle 2, 7, 5, 6, 9, 8, 11, 10 \rangle$ with length 8.

A \ length	1	2	3	4	5	6	7	8	9
a_1 2	**2**								
a_2 7	2	**7**							
a_3 5	2	**5**	7						
a_4 6	2	5	**6**	7					
a_5 1	**1**	2	6	7					
a_6 9	1	2	6	7	**9**				
a_7 8	1	2	6	7	**8**	9			
a_8 11	1	2	6	7	8	9	**11**		
a_9 10	1	2	6	7	8	9	**10**	11	

Table I shows an example for the LaIS algorithm proposed by Elmasry [8]. In each iteration, a new element is added and it retains the largest element among answers of each length and aims to minimize the largest element. Their LaIS algorithm can be summarized as follows.

Theorem 1. [8] *In the LaIS algorithm, when a new element a_i is added, it is placed before the first value that is greater than a_i. Then, the first element that is greater than or equal to $a_i + c$ is removed.*

The *longest increasing subsequence with sliding windows* (LISW) problem, another variant of the LIS problem, was first defined by Albert *et al.* in 2004 [9]. The LISW answer is

to find the LIS among all substrings of the given sequence with the window size. Albert *et al.* introduced the concept of principal row, row tower and drop out of the LIS to solve the LISW problem. The time complexity of their algorithm for LISW is $O(n \log\log n + nL)$, where L denotes the LISW answer length.

By integrating the LaIS and LISW problems, in this paper, we first define the *longest almost increasing subsequence with sliding windows* (LaISW) problem. Then, by constructing the principal row, row tower and drop out of LaISW, we will solve the LaISW problem with a similar concept.

III. The Problem Definition and Related Properties

Definition 2. (LaISW problem) *Given a numeric sequence, $A = \langle a_1, a_2, \ldots, a_n\rangle$, a tolerance constant c, and a window size w, the* longest almost increasing subsequence with sliding window (LaISW) *problem is to find the longest almost increasing subsequence in all windows $A_{i..i+w-1} = \langle a_i, a_{i+1}, \ldots, a_{i+w-1}\rangle$, where $1 \le i \le n - w + 1$.*

For example, suppose that $A = \langle 9, 1, 2, 5, 3, 5, 10, 8, 2\rangle$, $c = 3$, and $w = 6$. The LaISW answer are $\langle 1, 2, 5, 3, 5, 10\rangle$ with length 6, obtained from $A_{2..7} = \langle 1, 2, 5, 3, 5, 10\rangle$, or $\langle 2, 5, 3, 5, 10, 8\rangle$, obtained from $A_{3..8}$.

Definition 3. (principal row of a substring in LaISW) *Given a numeric sequence $A = \langle a_1, a_2, \ldots, a_n\rangle$, along with a tolerance constant c, the* principal row *of A is the final list built by Theorem 1. Let R_i^j denote the* principal row *of $A_{i..j} = \langle a_i, a_{i+1}, \ldots, a_j\rangle$, $1 \le i \le j \le n$.*

For example, see Table II. Suppose that a sequence $A = \langle 2, 7, 5, 6, 1, 9, 8, 11, 10, 3\rangle$, and the tolerance constant $c = 4$. Then, the principal row $P = R_1^9 = \langle 1, 2, 6, 7, 8, 9, 10, 11\rangle$. In addition, $p_4 = 7$ is the best (smallest) maximum value of the almost increasing subsequence $\langle 2, 7, 5, 6\rangle$ with length 4.

Definition 4. (row tower in LaIS) *Given a numeric sequence $A = \langle a_1, a_2, \ldots, a_n\rangle$, and a tolerance constant c, the* row tower *R of LaIS consists of principal rows $R_1^n, R_2^n, \ldots, R_n^n$.*

Definition 5. (drop out) [9] *Given a sequence A, the* drop out *D records the number of occurrences of each element from the first row in the row tower R.*

The drop out was proposed by Albert *et al.* [9](LISW), used to solve the LISW problem. Here, the concept can still be applied by changing the row tower of LIS to be the row tower of LaIS. For example, the row towers of $A_{1..9}$, $A_{2..9}$, and $A_{2..10}$ with their drop outs are illustrated in Table II.

To design an efficient algorithm for solving LaISW, we first develop the following theorem.

Theorem 2. *Given a numeric sequence $A = \langle a_1, a_2, \ldots, a_n\rangle$ and a tolerance constant c, in the LaIS row tower R, each row R_{i+1}^j can be obtained by removing exactly zero or one element from R_i^j, $1 \le i < i+1 \le j \le n$.*

The formal proof is omitted here due to the page limitation. The proof of this theorem can be found in [14].

Based on Theorem 2, the following corollary can be easily obtained. In other words, for each element in the row tower, if it dies, it will never be reborn.

Corollary 1. *In the LaIS row tower R of a sequence A , R_j^k is a subsequence of R_i^k, $1 \le i < j \le |A|$.*

IV. The Algorithm

To slide the window with a fixed size w, three operations are performed in order: (1) *removal operation*: delete the first element of the original window; (2) *insertion operation*: add the next element into the principal row; (3) *update operation*: update the drop out from the old one to the new one.

Definition 6. (removal operation from the row tower) *When the first element of a given sequence A is deleted, the* removal operation *deletes the first row of the LaIS row tower R, subtracts 1 from each element in the drop out D. Finally, it removes each element in D if it becomes 0 after the subtraction.*

See Table II as an example. Transitioning from $A_{1..9}$ to $A_{2..9}$ by removing the first element $a_1 = 2$ is equivalent to deleting the first row of the row tower for $A_{1..9}$. Therefore, the drop out D can be obtained by subtracting one from each element. That is, $\langle 5, 1, 3, 2, 7, 6, 9, 8\rangle$ for $A_{1..9}$ becomes $\langle 4, 0, 2, 1, 6, 5, 8, 7\rangle$, then we remove any zeros to form $\langle 4, 2, 1, 6, 5, 8, 7\rangle$ for $A_{2..9}$.

When a new element is added, we first perform the *insertion operation* on the old principal row.

Definition 7. (insertion operation of the principal row) [8] *Let b be the newly element added to the end of the new window. b is first inserted into the principal row P in the front of the successor of b. Then, the first element in P greater than or equal to $b + c$ is deleted.*

An example is shown in Table II for illustrating the addition of a new element, from the middle window to the right one. The old principal row (first row on the middle window) is $\langle 1, 6, 7, 8, 9, 10, 11\rangle$. In the right window, a new element $b = 3$ is added. $b = 3$ is inserted before its successor 6, and then 7 is deleted since 7 is the first element greater than or equal to $b + c = 3 + 4 = 7$. After this insertion operation, the new principal row (first row on the right) becomes $\langle 1, 3, 6, 8, 9, 10, 11\rangle$.

Next, we define the *shift path* for updating the drop out.

Definition 8. (shift path of drop out) *When a new element b is inserted to an old principal row $P = \langle p_1, p_2, \cdots, p_m\rangle$, along with drop out $D = \langle d_1, d_2, \cdots, d_m\rangle$ and tolerance constant c, the* shift path of drop out *is represented by the index sequence $\langle i_1, i_2, \cdots, i_{m'}\rangle$, $m' \le m$, that p_{i_1} is the first element greater than or equal to $b + c$, and $d_{i_k} < d_{i_{k+1}}$, for $1 \le k \le m' - 1$. In the shift path, $d_{i_{k+1}}$ is the next element greater than d_{i_k} in D.*

TABLE II
EXAMPLES FOR ROW TOWER, REMOVAL, INSERTION AND UPDATE WITH $A = \langle 2, 7, 5, 6, 1, 9, 8, 11, 10, 3\rangle$, $c = 4$ AND $w = 9$.

sequence	$A_{1..9}$: **2**, 7, 5, 6, 1, 9, 8, 11, 10	$A_{2..9}$: 7, 5, 6, 1, 9, 8, 11, 10	$A_{2..10}$: 7, 5, 6, 1, 9, 8, 11, 10, **3**
row tower R	R_1^9: 1, 2, 6, 7, 8, 9, 10, 11 R_2^9: 1, 6, 7, 8, 9, 10, 11 R_3^9: 1, 6, 8, 9, 10, 11 R_4^9: 1, 8, 9, 10, 11 R_5^9: 1, 8, 9, 10, 11 R_6^9: 8, 9, 10, 11 R_7^9: 8, 10, 11 R_8^9: 10, 11 R_9^9: 10	R_2^9: 1, 6, **7**, 8, 9, 10, 11 R_3^9: 1, 6, **8**, 9, 10, 11 R_4^9: 1, **8**, 9, 10, 11 R_5^9: 1, **8**, 9, 10, 11 R_6^9: **8**, 9, 10, 11 R_7^9: **8**, 10, 11 R_8^9: **10**, 11 R_9^9: **10**	R_2^{10}: 1, 3, 6, 8, 9, 10, 11 R_3^{10}: 1, 3, 6, 9, 10, 11 R_4^{10}: 1, 3, 9, 10, 11 R_5^{10}: 1, 3, 9, 10, 11 R_6^{10}: 3, 9, 10, 11 R_7^{10}: 3, 10, 11 R_8^{10}: 3, 11 R_9^{10}: 3 R_{10}^{10}: 3
drop out D	5, 1, 3, 2, 7, 6, 9, 8	4, 2, **1**, **6**, 5, **8**, 7	4, 9, 2, 1, 5, 6, 7

See Table II. The shift path of drop out is $\langle i_1, i_2, i_3\rangle = \langle 3, 4, 6\rangle$. $p_{i_1} = p_3 = 7$ is the first element greater than or equal to $b+c = 7$. In D, the next element greater than $d_3 = 1$ is $d_4 = 6$, and the next element greater than $d_4 = 6$ is $d_6 = 8$.

Definition 9. (update operation of the drop out) *When a new element b is inserted to an old principal row $P = \langle p_1, p_2, \cdots, p_m\rangle$, along with drop out $D = \langle d_1, d_2, \cdots, d_m\rangle$ and tolerance constant c, the* update operation *gets the new drop out $D' = \langle d'_1, d'_2, d'_3, \cdots\rangle$ from the old drop out D with* Equation 1.

$$\begin{cases} d'_i = d_i & \text{if } p_i \le b; \\ d'_i = w & \text{if } i \text{ is the inserted position index of } b \text{ in } P; \\ d'_i = d_{i-1} & \text{if } b+1 \le p_{i-1} \le b+c-1; \\ d'_{i_{k+1}} = d_{i_k} & \text{if } i_k \text{ is in the shift path and } k \ge 1; \\ d'_i = d_i & \text{otherwise.} \end{cases} \tag{1}$$

Theorem 3. *The new drop out can be correctly obtained by Equation 1.*

The formal proof is omitted here. The detailed proof can be seen in [14].

See Table II for an example of the update operation. When by a new element $b = a_{10} = 3$ is added (from the middle window to the right one), the row tower of $A_{2..9}$ can be updated to the row tower of $A_{2..10}$ with a brute-force method. By Definition 7, the brute-force method can insert $b = 3$ into each row of the row tower and removes the first element (if any) greater than or equal to $b+c = 3+4 = 7$. After the new row tower has been built with one row by one row, the new drop out can be easily established by counting the occurrence of each element in the row tower.

Equation 1 is more efficient for getting the new drop out. In the example, $b = 3$ is the newly appended element into $A_{2..9}$ (from the middle to the right). For $A_{2..9}$, the principal row $P = \langle p_1, p_2, \cdots, p_m\rangle = \langle 1, 6, 7, 8, 9, 10, 11\rangle$, where $m = 7$, and the drop out $D = \langle 4, 2, 1, 6, 5, 8, 7\rangle$. According to Equation 1, five cases of the update operation are considered to build the new drop out D' after $b = 3$ is appended as follows.

Case 1: $p_i \le b$. We have that $p_1 = 1 \le b = 3$. So we get $d'_1 = d_1 = 4$. In this case, the number of occurrences of p_1 remains unchanged, since $b = 3$ is inserted after p_1 in P.

Case 2: i is the inserted position index of b in P. In this example, $b = 3$ should be inserted before its successor 6. In other words, $i = 2$. Thus, we get $d'_2 = w = 9$, since $b = 3$ is always alive in the row tower.

Case 3: $b+1 \le p_{i-1} \le b+c-1$. We can get that $b+1 = 4 \le p_2 = 6 \le b+c-1 = 6$. That is, $p_{i-1} = p_{3-1}$ satisfies the condition. We get $d'_3 = d_2 = 2$, meaning that the survival duration of $p_2 = 6$ is not affected by the insertion of $b = 3$, except that p_2 is shifted to the next position $i = 3$ in the new principal row.

Case 4: i_k is in the shift path, $k \ge 1$. As mentioned before, the shift path of this example is $\langle i_1, i_2, i_3\rangle = \langle 3, 4, 6\rangle$. We get $d'_{i_2} = d'_4 = d_{i_1} = d_3 = 1$, and $d'_{i_3} = d'_6 = d_{i_2} = d_4 = 6$.

Now, we explain how the shift path works. See the middle of Table II. $p_3 = 7$ is the first element in $R_2^9 = P$ greater than or equal to $b+c = 7$. From D, we get $d_3 = 1$. Accordingly, $p_3 = 7$ is alive in the first row of the row tower, that is, R_2^9. Then, in the second row through the 6th row ($R_3^9 \sim R_7^9$), the first element greater than or equal to $b+c = 7$ is $p_4 = 8$. In the new row tower, it means that $p_4 = 8$ should be alive in the first row, since $p_3 = 7$ will be deleted, and $p_4 = 8$ will be deleted in the second through the 6th rows. Thus, the old drop out value $d_3 = 1$ of $p_3 = 7$ is copied to become the new drop out value $d'_{i_2} = d'_4$ of $p_4 = 8$.

In the old row tower, $p_4 = 8$ is alive in the first through the 6th rows ($R_3^9 \sim R_7^9$). In these six rows, the next greater element $p_6 = 10$, will not be deleted. However, $p_6 = 10$ will be deleted in the 7th row, since $p_6 = 10$ is the first element greater than or equal to $b+c = 7$. That is, the survival range of $p_6 = 10$ is the original survival range of $p_4 = 8$. Hence, $d_4 = 6$ of $p_4 = 8$ is copied to become the new drop out value $d'_{i_3} = d'_6$ of $p_6 = 10$. As a short summary, in the shift path, an old drop out value can be shifted to become a new drop out value.

Case 5: The remaining cases are those that are greater than or equal to $b+c$ but are not in the shift path. In this example, $p_5 = 9$ and $p_7 = 11$ do not fall in the above cases. Thus, we

get $d'_5 = d_5 = 5$, and $d'_7 = d_7 = 7$. Since they will not be deleted, the drop out will naturally remain unchanged.

The algorithm for calculating the LaISW length is formally presented in Algorithm 1. The *update* operation can be done by Equation 1.

Algorithm 1 Main Algorithm for Computing the LaISW length

Input: A numeric sequence A = $\langle a_1, a_2, \ldots, a_n \rangle$, a tolerance constant c, and a window size w.
Output: The LaISW length

```
{first window}
for i = 1 to w do
    P' ← insertion(P, a_i, c) {continuously add new elements}
    D ← update(P, P', D, a_i, c, i) {update drop out by Equation 1}
    P ← P'
end for
length ← size(D)
{remaining windows}
for i = w + 1 to n do
    P', D ← removal(P, D) {removal operation}
    P'' ← insertion(P', a_i, c) {insertion operation}
    D ← update(P', P'', D, a_i, c, w) {update by Equation 1}
    length ← max(length, size(D)) {keep maximal length}
    P ← P''
end for
return length
```

It is easy to see that both time and space complexities of each *removal*, *insertion*, *update* and *shift* are O(L), where L denotes the length of the LaISW answer.

Theorem 4. Algorithm 1 *can solve the LaISW problem in O(nL) time and O(L) space, where n denotes the length of the input sequence, and L denotes the LaISW length.*

V. A Complete Example

We illustrate our LaISW algorithm with a complete example shown in Table III. In the first window, we consider $A_{1..6}$ = $\langle 9, 1, 2, 5, 3, 5 \rangle$. After constructing the whole row tower, we obtain the principle row P = $\langle 1, 2, 3, 5, 5 \rangle$ and drop out D = $\langle 2, 3, 5, 4, 6 \rangle$, with an aISW length 5.

When sliding to the second window $A_{2..7}$ = $\langle 1, 2, 5, 3, 5, 10 \rangle$, three operations are performed as follows.

(1) Removal operation: Initially, D = D_1^6 = $\langle 2, 3, 5, 4, 6 \rangle$. After decrementing each element by one, D becomes $\langle 1, 2, 4, 3, 5 \rangle$ temporarily.

(2) Insertion operation: The current principal row is P = R_1^6 = $\langle 1, 2, 3, 5, 5 \rangle$, and 10 is the newly inserted element. Since 10 is greater than all elements in P, it is inserted at the end without removing any element. We get $P' = R_2^7$ = $\langle 1, 2, 3, 5, 5, 10 \rangle$.

(3) Update operation: The first five elements of P are all less than 10. Therefore, the first five elements in D remain unchanged, resulting in $D'_{1..5}$ = $D_{1..5}$ = $\langle 1, 2, 4, 3, 5 \rangle$. Then, the drop out of the new element is added at the sixth position. So we set $d'_6 = 6$, which is the window size. Thus, we get D' = $\langle 1, 2, 4, 3, 5, 6 \rangle$.

When sliding to the third window $A_{3..8}$ = $\langle 2, 5, 3, 5, 10, 8 \rangle$, the same three operations are performed as follows.

(1) Removal operation: Initially, the old principal row P = R_2^7 = $\langle 1, 2, 3, 5, 5, 10 \rangle$ and D = $\langle 1, 2, 4, 3, 5, 6 \rangle$. After decrementing each elements by one, D becomes $\langle 0, 1, 3, 2, 4, 5 \rangle$ temporarily. Since d_1 becomes 0, it is removed, resulting in that P becomes $\langle 2, 3, 5, 5, 10 \rangle$, and D becomes $\langle 1, 3, 2, 4, 5 \rangle$.

(2) Insertion operation: The current P = $\langle 2, 3, 5, 5, 10 \rangle$, and the next element to be inserted is $b = 8$. 8 will be inserted before its successor, which is 10. Since no element in P is greater than or equal to $b + c = 8 + 3 = 11$, no element is removed. We get $P' = R_3^8$ = $\langle 2, 3, 5, 5, 8, 10 \rangle$.

(3) Update operation: The first four elements of P are smaller than $b = 8$. Therefore, the first four elements in D remain unchanged, resulting in $D'_{1..4}$ = $D_{1..4}$ = $\langle 1, 3, 2, 4 \rangle$. Then, the new element $b = 8$ is added at the fifth position, so d_5 = window size = 6. Finally, for the sixth position, since $P_6 = 10 \leq b + c - 1 = 8 + 3 - 1 = 10$, we get $d'_6 = d_5$ = 5. Then we get D' = $\langle 1, 3, 2, 4, 6, 5 \rangle$.

Sliding to the last window $A_{4..9}$ = $\langle 5, 3, 5, 10, 8, 2 \rangle$, the same three operations are performed as follows.

(1) Removal operation: P becomes $\langle 3, 5, 5, 8, 10 \rangle$, and D becomes $\langle 2, 1, 3, 5, 4 \rangle$.

(2) Insertion operation: The current P = $\langle 3, 5, 5, 8, 10 \rangle$. The next element $b = 2$ will be inserted before its successor 3. $p_2 = 5$ is the first element in P greater than or equal to $b + c = 2 + 3 = 5$, so $p_2 = 5$ is removed. We get $P' = R_4^9$ = $\langle 2, 3, 5, 8, 10 \rangle$.

(3) Update operation: With P = $\langle 3, 5, 5, 8, 10 \rangle$ and D = $\langle 2, 1, 3, 5, 4 \rangle$, we can identify the shift path $\langle i_1, i_2, i_3 \rangle$ = $\langle 2, 3, 4 \rangle$, corresponding to $\langle p_2, p_3, p_4 \rangle = \langle 5, 5, 8 \rangle$ and $\langle d_2, d_3, d_4 \rangle = \langle 1, 3, 5 \rangle$. They are calculated as follows. First, $i_1 = 2$ because p_2 = 5 is the first element greater than or equal to $b + c = 2 + 3 = 5$. Next, i_2 = 3 because $d_3 = 3$ is the first element greater than $d_2 = 1$. Then, $i_3 = 4$ because $d_4 = 5$ is the first element greater than $d_3 = 3$. The first position is for the new element, so d'_1 = window size = 6. The second element, p_2 = 3, is less than $b+c = 2+3 = 5$, so $d'_2 = d_1$ = 2. The remaining elements are all greater than or equal to $b + c = 2 + 3 = 5$. The third element belongs to the shift path, so its drop out is obtained by shifting from the previous element in the shift path. That is, $d'_3 = d_2$ = 1. The fourth element, 8, is also in the shift path, so $d'_4 = d_3$ = 3. The fifth element, 10, does not belong to the shift path, so $d'_5 = d_5$ = 4. Thus, after sliding to the last window, we have $P' = R_4^9$ = $\langle 2, 3, 5, 8, 10 \rangle$, D' = $\langle 6, 2, 1, 3, 4 \rangle$, and the aISW length is 5.

After executing the algorithm for the sliding window, the LaISW length can be obtained by $\max\{5, 6, 6, 5\}$ = 6.

TABLE III
A COMPLETE EXAMPLE OF THE LAISW ALGORITHM WITH $A = \langle 9, 1, 2, 5, 3, 5, 10, 8, 2\rangle$, $c = 3$, AND $w = 6$. THE LAISW LENGTH IS 6.

window	window 1 ($A_{1..6}$) 9, 1, 2, 5, 3, 5	window 2($A_{2..7}$) 1, 2, 5, 3, 5, 10	window 3 ($A_{3..8}$) 2, 5, 3, 5, 10, 8	window 4 ($A_{4..9}$) 5, 3, 5, 10, 8, 2
row tower R	1, 2, 3, 5, 5 1, 2, 3, 5, 5 2, 3, 5, 5 3, 5, 5 3, 5 5	1, 2, 3, 5, 5, 10 2, 3, 5, 5, 10 3, 5, 5, 10 3, 5, 10 5, 10 10	2, 3, 5, 5, 8, 10 3, 5, 5, 8, 10 3, 5, 8, 10 5, 8, 10 8, 10 8	2, 3, 5, 8, 10 2, 3, 8, 10 2, 8, 10 2, 10 2 2
principal row P	1, 2, 3, 5, 5	1, 2, 3, 5, 5, 10	2, 3, 5, 5, 8, 10	2, 3, 5, 8, 10
drop out D	2, 3, 5, 4, 6	1, 2, 4, 3, 5, 6	1, 3, 2, 4, 6, 5	6, 2, 1, 3, 4
aISW length	5	6	6	5

VI. CONCLUSION

In this paper, we incorporate the concept of 'almost increasing' into the LISW problem, so that the integrated problem not only finds strictly increasing subsequence but also allows for slight decreases. This incorporation makes the algorithm more flexible. If we desire to find the LICS (longest increasing circular subsequence) in the almost increasing (LaICS) problem, we can directly apply the LaISW algorithm we propose. However, this is a preliminary approach, and there could be more efficient methods available for solving the LaICS problem.

In the future, there is potential to integrate additional concepts into the LISW framework, such as the *longest wave subsequence* (LWS) [15]. This expansion would not only allow LaISW to tolerate slight decreases but also add the possibility of alternating between increasing and decreasing subsequences. In such a way, the algorithms may be more flexible for practical applications. Furthermore, the goal of the LaISW problem is to process a single input sequence. It could be extended to involve two input sequences, where one of the inputs may be significantly longer than the other. It may require us to slide the shorter sequence to identify the *longest common almost increasing subsequence* (LCaIS) [16]–[18] across all windows from the longer sequence.

REFERENCES

[1] C. Schensted, "Longest increasing and decreasing subsequences," *Canadian Journal of Mathematics*, vol. 13, pp. 179–191, 1961.

[2] J. W. Hunt and T. G. Szymanski, "A fast algorithm for computing longest common subsequences," *Communications of the ACM*, vol. 20, pp. 350–353, 1977.

[3] S. Bespamyatnikh and M. Segal, "Enumerating longest increasing subsequences and patience sorting," *Information Processing Letters*, vol. 76, no. 1-2, pp. 7–11, 2000.

[4] C.-T. Tseng, C.-B. Yang, and H.-Y. Ann, "Minimum height and sequence constrained longest increasing subsequence," *Journal of Internet Technology*, vol. 10, no. 2, pp. 173–178, 2009.

[5] M. Crochemore and E. Porat, "Fast computation of a longest increasing subsequence and application," *Information and Computation*, vol. 208, no. 9, pp. 1054–1059, 2010.

[6] M. R. Alam and M. S. Rahman, "A divide and conquer approach and a work-optimal parallel algorithm for the lis problem," *Information Processing Letters*, vol. 113, no. 13, pp. 470–476, 2013.

[7] P. van Emde Boas, R. Kaas, and E. Zijlstra, "Design and implementation of an efficient priority queue," *Mathematical Systems Theory*, vol. 10, no. 1, pp. 99–127, 1976.

[8] A. Elmasry, "The longest almost-increasing subsequence," *Information Processing Letters*, vol. 110, no. 16, pp. 655–658, 2010.

[9] M. H. Albert, A. Golynski, A. M. Hamel, A. López-Ortiz, S. Rao, and M. A. Safari, "Longest increasing subsequences in sliding windows," *Theoretical Computer Science*, vol. 321, no. 2-3, pp. 405–414, 2004.

[10] E. Chen, L. Yang, and H. Yuan, "Longest increasing subsequences in windows based on canonical antichain partition," *Theoretical Computer Science*, vol. 378, no. 3, pp. 223–236, 2007.

[11] Y. Li, L. Zou, H. Zhang, and D. Zhao, "Longest increasing subsequence computation over streaming sequences," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 6, pp. 1036–1049, 2018.

[12] M. H. Albert, M. Atkinson, D. Nussbaum, J.-R. Sack, and N. Santoro, "On the longest increasing subsequence of a circular list," *Information Processing Letters*, vol. 101, no. 2, pp. 55–59, 2007.

[13] S. Deorowicz, "An algorithm for solving the longest increasing circular subsequence problem," *Information Processing Letters*, vol. 109, no. 12, pp. 630–634, 2009.

[14] C.-H. Ho, "The longest almost increasing subsequence problem with sliding windows," in *Master's Thesis, Department of Computer Science and Engineering, National Sun Yat-sen University*, Kaohsiung, Taiwan, 2023.

[15] G.-Z. Chen and C.-B. Yang, "The longest wave subsequence problem: Generalizations of the longest increasing subsequence problem," in *The 37th Workshop on Combinatorial Mathematics and Computation Theory*, Kaohsiung, Taiwan, 2020, pp. 28–33.

[16] M. T. H. Bhuiyan, M. R. Alam, and M. S. Rahman, "Computing the longest common almost-increasing subsequence," *Theoretical Computer Science*, vol. 930, pp. 157–178, 2022.

[17] J. M. Moosa, M. S. Rahman, and F. T. Zohora, "Computing a longest common subsequence that is almost increasing on sequences having no repeated elements," *Journal of Discrete Algorithms*, vol. 20, pp. 12–20, 2013.

[18] T. T. Ta, Y.-K. Shieh, and C. L. Lu, "Computing a longest common almost-increasing subsequence of two sequences," *Theoretical Computer Science*, vol. 854, pp. 44–51, 2021.