Efficient Algorithms for the Interval Sequence Problems

Guan-Ting Chen
Dept. Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan

Chang-Biau Yang
Dept. Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan
cbyang@cse.nsysu.edu.tw
(Corresponding author)

***Abstract*—This paper focuses on the interval sequence problems. We first define four versions of the *longest increasing subsequence* (LIS) problems on the interval sequences, and then design efficient algorithms for solving them. We propose two ways for comparing a pair of intervals: full interval and subinterval. By combining the interval comparison with the LIS problem, we define two variants of the problem. The *most increasing interval subsequence* (MIIS) problem aims to find the maximum number of intervals involved in the answer. On the other hand, the *longest increasing interval subsequence* (LIIS) problem focuses on the total length of the intervals involved in the answer. We give a new research direction concerning interval sequences. For each variant problem, we design an efficient algorithm to solve each of them in $O(n \log n)$ time, except that the MIIS problem with comparison by a subinterval is solved in $O(n \log^2 n)$ time, where n denotes the number of elements (intervals) in the input interval sequence.**

***Index Terms*—interval, interval sequence, longest increasing subsequence, most increasing interval subsequence, longest increasing interval subsequence**

I. Introduction

The *longest increasing subsequence* (LIS) problem aims to find the strictly increasing subsequence with the maximal length in a numeric sequence. For example, in the sequence $A = \langle 6, 5, 7, 3, 9, 4 \rangle$, the LIS length is 3, with two possible subsequences: $\langle 6, 7, 9 \rangle$ and $\langle 5, 7, 9 \rangle$. In 1961, Schensted [10] introduced the concept and an $O(n \log n)$-time algorithm for solving it. LIS has been extensively studied and has diverse applications.

In addition to the conventional LIS problem, various variants of it have been proposed in prior researches [1, 2, 4–9, 11, 12, 14, 15]. These previous researches focused on numeric sequences, but the real-world data often consist of intervals, rather than individual values. Examples include daily temperature ranges and stock price fluctuations (low and high). Figure 1 shows an example of stock prices and the result obtained from the MIIS problem with comparison by full interval.

There are two kinds of comparison between two intervals: full interval and subinterval. And, the optimization objective may be the count or the length of the answer. Accordingly, we define the following four problems:

- *most increasing interval subsequence* (MIIS) with comparison by full interval
- *most increasing interval subsequence* (MIIS) with comparison by a subinterval
- *longest increasing interval subsequence* (LIIS) with comparison by full interval
- *longest increasing interval subsequence* (LIIS) with comparison by a subinterval

The rest of the paper is structured as follows. Section II and Section III provide the definitions and algorithms of the MIIS problem and the LIIS problem, respectively. Finally, the conclusion is presented in Section V.

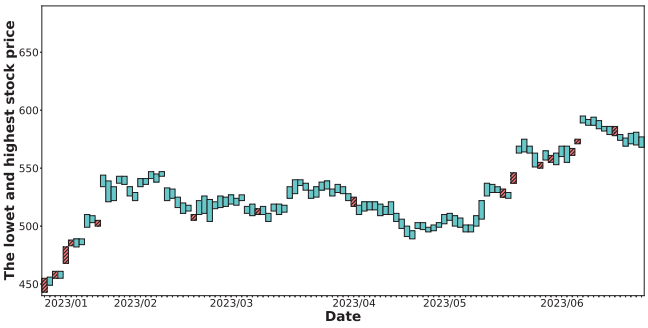

Fig. 1: Daily highest and lowest stock prices of TSMC in the first half of 2023 and the result of the MIIS problem with comparison by full interval.

II. Most Increasing Interval Subsequence

A. The Problem Definition

Definition 1. (interval) *An interval $x = [x_s, x_e]$ is a closed integer interval, with a starting point x_s and an ending point x_e, $x_s \leq x_e$. The interval includes x_s and x_e, and the interval length is $|x| = x_e - x_s + 1$.*

Definition 2. (interval comparison) *Given two intervals $x = [x_s, x_e]$ and $y = [y_s, y_e]$, if $x_s < y_s$ and $x_e < y_e$, then we say that $x < y$.*

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

For instance, given $x = [4, 8]$ and $y = [6, 10]$, we have $x_s = 4$ and $x_e = 8$ are smaller than $y_s = 6$ and $y_e = 10$, respectively. Then, we can say that $x < y$. However, we cannot determine the relation between $z = [4, 10]$ and $y = [6, 10]$. In such a situation, their relation remains undefined. Based on Definition 2, the following interval comparison is derived.

Definition 3. (comparison by full interval) *Given two intervals $x = [x_s, x_e]$ and $y = [y_s, y_e]$, if $x_e < y_s$, then we say that $x < y$.*

In Definition 3, we can get the comparison result only if the two compared intervals do not overlap. If the two intervals overlap, the result is defined as undetermined. For example, given $x = [3, 5]$, $y = [6, 9]$, and $z = [4, 7]$, we say that $x < y$ with comparison by full interval; we cannot conclude that $x < z$ or $z < x$, since the intervals overlap. The interval increasing subsequence under Definition 3 can be regarded as a strictly increasing interval sequence.

However, the rigidity of the comparison based on full intervals is a limitation. Therefore, we give an alternative definition that is more flexible than Definition 3.

Definition 4. (subinterval) *A subinterval $x' = [x'_s, x'_e]$ of an interval $x = [x_s, x_e]$, denoted by $x' \subseteq x$, is a closed interval that $x_s \le x'_s \le x'_e \le x_e$.*

For instance, $[4, 4]$, $[4, 7]$, $[5, 8]$ and $[4, 10]$ are some subintervals of an interval $x = [4, 10]$.

Definition 5. (comparison by a subinterval) *Given two intervals x and y and a predefined constant c, we say that $x < y$ if there exist subintervals $x' = [x'_s, x'_e] \subseteq x$, $y' = [y'_s, y'_e] \subseteq y$ such that $x'_e < y'_s$ with Definition 3 and $|x'| = |y'| = c$.*

For instance, $x = [3, 9]$, $y = [5, 11]$, and $c = 3$, we say that $x < y$, because subinterval $x' = [3, 5] \subseteq X$, subinterval $y' = [6, 8] \subseteq y$, and $x' < y'$ by Definition 3. By this way, we can control the result of the increasing subsequence by setting the value of c.

Definition 6. (MIIS problem) *Given an interval sequence $X = \langle x_1, x_2, x_3, \ldots, x_n \rangle$, where $x_i = [x_{s_i}, x_{e_i}]$, $1 \le i \le n$, the* most increasing interval subsequence (*MIIS*) *problem is to find the increasing interval subsequence with the maximum number of intervals.*

B. The Algorithm for Comparison by Full Interval

According to Definition 3, we have to check whether the newly coming interval overlaps any existing interval. We choose the ending value of each interval as its representative in the solution structure T. The ending value of the interval cannot only be used to check if there is any overlap between intervals, but also be used to determine which interval is more extendable. We aim to minimize the terminal value of the interval subsequence, and to ensure that we have the maximal number of intervals found in the answer. We illustrate our algorithm with the example shown in Table I.

In round 1, with $x_1 = [42, 50]$, we start by inserting $x_{1_e} = 50$ into the solution structure. In round 2, when considering $x_2 = [20, 27]$, we first observe that $x_{2_e} = 27 < x_{1_e} = 50$. Therefore, we cannot append x_2 to x_1 to increase the number of elements involved in the answer. Then, we use the ending value 27 to find its successor, which is 50. Next, we replace 50 by $x_{2_e} = 27$ to enhance the extendability of the potential answer. In round 3, $x_{3_s} = 28$ is greater than 27, so we directly extend the quantity to 2.

TABLE I: An example of the algorithm for MIIS with comparison by full interval, where $X = \langle [42, 50], [20, 27], [28, 36], [25, 33], [40, 45], [43, 49] \rangle$. The answer is $\langle [20, 27], [28, 36], [40, 45] \rangle$ with 3 intervals.

X \ q(quantity)		1	2	3
x_1	[42, 50]	**50**		
x_2	[20, 27]	**27**		
x_3	[28, 36]	27	**36**	
x_4	[25, 33]	27	36	
x_5	[40, 45]	27	36	**45**
x_6	[43, 49]	27	36	45

The most complicated case of this problem is shown in round 4 with a newly coming interval $x_4 = [25, 33]$. x_4 overlaps with two intervals. Choosing x_4 would decrease the number of intervals in the answer to 1, so we abandon x_4. In round 5, similar to x_3, we can increase the quantity to 3 by appending 45 to the answer. In round 6, for x_6, we cannot increase the quantity by appending 49 to the answer, neither replace any element already in the solution structure. Finally, we get the answer $\langle [20, 27], [28, 36], [40, 45] \rangle$ with 3 intervals.

Theorem 1. *In the algorithm of the MIIS problem with comparison by full interval, the ending value recorded in each quantity is the minimum for getting that quantity.*

Proof. Consider an interval sequence $X = \langle x_1, x_2, x_3, \ldots, x_n \rangle$. We will prove this theorem by induction. When $n = 1$, the quantity 1 has only one choice, which must be the smallest. By induction hypothesis, for $n = k$, assume that each quantity in structure T has the minimum ending value, and the current quantity is q.

Then, we want to prove that $n = k + 1$ is also true. Let u denote the currently last value in T. Three cases are considered as follows.

1) If $x_{s_{k+1}} > u$, we can extend the quantity to $q+1$ with the ending value $x_{e_{k+1}}$. Therefore, $T[q + 1]$ has one choice for the smallest terminal value, which is $x_{e_{k+1}}$.
2) If $x_{s_{k+1}} \le u \le x_{e_{k+1}}$, it implies that $x_{e_{k+1}}$ overlaps with u. We cannot increase the quantity, and $x_{e_{k+1}}$ cannot replace any element to make the value become smaller. Thus, discarding $x_{e_{k+1}}$ ensures that each quantity in structure T has the smallest terminal value.
3) If $x_{s_{k+1}} \le u$ and $x_{e_{k+1}} \le u$, there should exist a successor of $x_{e_{k+1}}$ in T, denoted by v. Let v be at quantity w. And, let the predecessor of v be denoted by $pred(v)$.

(3.1) If $pred(v) < x_{s_{k+1}}$ (x_{k+1} does not overlap with $pred(v)$), then $x_{e_{k+1}}$ can be extended from $pred(v)$, resulting in a quantity of w. Thus, v can be replaced by $x_{e_{k+1}}$, since v is dominated by $x_{e_{k+1}}$. This replacement with $x_{e_{k+1}}$ only makes the value become smaller.
(3.2) If $pred(v) \geq x_{s_{k+1}}$ (x_{k+1} overlaps with $pred(v)$), then we cannot extend from $pred(v)$ to $x_{e_{k+1}}$. In other words, the possible maximal quantity of $x_{e_{k+1}}$ is $w-1$. However, $pred(v)$ is also at quantity $w-1$ and $pred(v)$ dominates $x_{e_{k+1}}$. Thus, discarding $x_{e_{k+1}}$ ensures that each quantity in structure T has the smallest terminal value.

Through the above three cases, we conclude that the theorem holds. □

Theorem 2. *The MIIS problem with comparison by full interval can be solved in $O(n \log q)$ time and $O(q)$ space, where q denotes the length of the MIIS answer.*

C. The Algorithm for Comparison by a Subinterval

For the MIIS problem with comparison by a subinterval, four conditions need to be satisfied as follows.

(1) The starting and ending values of the intervals involved in the answer are both increasing.

(2) The intervals in the structure should be the most extendable.

(3) If we cannot determine which one is the most extendable, we have to keep all possible intervals in the structure T.

(4) For every two intervals involved in the answer, there should exist a non-overlapping subinterval with at least a predefined length c.

Definition 7. (domination of the interval) *For given two intervals x and y, if $x_s \leq y_s$ and $x_e \leq y_e$, we say that x dominates y.*

Each dominated interval in the answer at the same length is deleted, and all non-dominated intervals are kept.

For a newly coming interval a_i, we employ the concept of binary search to decide the appropriate length to be inserted for a_i. Let l and r represent the minimum and the maximum possible quantity, respectively. Then, $m = \frac{l+r}{2}$ is set as the length for checking in the current step. According to the check result of m, we adjust the value of l or r to reduce the check range. While $l > r$, the round will be ended, and l is the quantity for inserting the new interval.

Table II shows an example with constraint $c = 3$. In round 1, the first interval $x_1 = [12, 18]$ is inserted into quantity 1. In round 2, we start to apply the concept of binary search with $l = 1$ and $r = 1$. We first get $m = \frac{l+r}{2} = 1$ as the length for inserting x_2. When $m = 1$, we first check the element in quantity 1. We find that $x_2 = [15, 17]$ and $x_1 = [12, 18]$ cannot dominate each other, so we just insert $[15, 17]$ into the quantity 1 as a candidate.

In round 3, $l = 1$, $r = 1$, and $m = \frac{l+r}{2} = 1$, we first find out the predecessor of $x_3 = [16, 19]$ from the intervals with

TABLE II: An example for the algorithm for MIIS with comparison by a subinterval, where $X = \langle [12, 18], [15, 17], [16, 19], [17, 21], [6, 13] \rangle$ and constraint constant $c = 3$. The answer is $\langle [12, 18], [16, 19], [17, 21] \rangle$ with 3 intervals.

X		q(quantity) 1	2	3
x_1	$[12, 18]$	$[12, 18]$		
x_2	$[15, 17]$	$[15, 17]$ $[12, 18]$		
x_3	$[16, 19]$	$[15, 17]$ $[12, 18]$	$[16, 19]$	
x_4	$[17, 21]$	$[15, 17]$ $[12, 18]$	$[16, 19]$	$[17, 21]$
x_5	$[6, 13]$	$[6, 13]$ ~~$[15, 17]$~~ ~~$[12, 18]$~~	$[16, 19]$	$[17, 21]$

quantity 1. Here, the predecessor is defined as the relation between the ending values. The reasons is proved in Theorem 3. We get the predecessor $x_1 = [12, 18]$. It implies that x_3 should be inserted into a minimum quantity of 2. Next, $l = m + 1 = 2$ and $l > r$, so x_3 is inserted into quantity $l = 2$. Similarly, in round 4 x_4 is inserted into quantity 3.

In round 5, $x_5 = [6, 13]$, we first check $m = \frac{1+3}{2} = 2$. At quantity 2, we cannot find the predecessor of x_5, so we set $r = m - 1 = 1$. Consequently, we determine that x_5 should be inserted into quantity 1. After inserting x_5, we remove each interval in quantity 1 dominated by x_5 to make sure no domination in quantity 1. Finally, the answer is $\langle [12, 18], [16, 19], [17, 21] \rangle$ with 3 intervals.

Theorem 3. *In the MIIS problem with comparison by a subinterval, the predecessor searched by the ending values of the intervals can make sure that the extended interval has the smallest starting value.*

Proof. Suppose that intervals x and y are in the same quantity. By definition, they cannot dominate each other. Without loss of generality, assume that $x_s > y_s$, then we have $x_e < y_e$. Therefore, if the starting values of the intervals are sorted decreasingly, then the ending values should be sorted increasingly. The predecessor interval of a newly coming interval z searched by the ending value should have the smallest starting value to extend. □

In the algorithm, we have ensured that the intervals in the same quantity do not mutually dominate each other. In Theorem 4, we further prove that the intervals of different quantity will not dominate each other.

Theorem 4. *In the structure of MIIS with comparison by a subinterval, there is no domination between the intervals of different lengths.*

Proof. An interval of a shorter length cannot dominate the interval in a larger length, since the latter has a better solution

with a greater length. Now, we want to prove that an interval of larger length cannot dominate an interval of shorter length. Suppose that an interval y is of length i and an interval x is of length j, where $i < j$ and $x < y$ (x dominates y). In this situation, there should exist another interval z of length i that can be extended for x, and $z < x$. This implies that $z < y$ (z dominates y in length i). However, all intervals in length i do not mutually dominate each other. It leads to that the assumption is not valid. Therefore, there is no domination in the structure after each round. □

Theorem 5. *The MIIS problem with comparison by a subinterval can be solved in $O(n \log^2 n)$ time and $O(qn)$ space, where q denotes the length of the MIIS answer.*

III. Longest Increasing Interval Subsequence

A. The Problem Definition

Definition 8. (length of an interval sequence) *Given an interval sequence $X = \langle x_1, x_2, x_3, \ldots, x_n \rangle$, where $x_i = [x_{s_i}, x_{e_i}]$, $1 \leq i \leq n$, the length of X, denoted as $|X|$, is the sum of the lengths of all intervals, but the length of the overlapping part is counted exactly once.*

Definition 9. (LIIS problem) *Given an interval sequence $X = \langle x_1, x_2, x_3, \ldots, x_n \rangle$, the* longest increasing interval subsequence *(LIIS) problem is to find the increasing interval subsequence whose total length is maximal.*

Now, we focus on the total length of the intervals involved in the increasing subsequence, not the number of the intervals. In other words, the answer should contain as large range as possible. For this purpose, we desire any part of intervals to extend the answer length as long as they meet Definition 2. Therefore, we do not set any constraint to limit the length.

B. The Algorithm for Comparison by Full Interval

Two steps are performed for inserting a newly coming interval x_i into the answer. First, we find possible interval subsequence for extension while ensuring that they do not overlap with each other. In the second step, after inserting x_i, we check for any domination and remove dominated intervals.

We use 2-tuple, formed by (terminal value, cumulative length), to maintain the solution structure and check for the existence of any domination. In Definition 10, we define the domination of the LIIS problem with comparison by full interval to retain the solution structure being the most extendable.

Definition 10. (domination of the cumulative length) *Given two increasing interval sequences $X = \langle x_1, x_2, x_3, \ldots, x_n \rangle$ and $Y = \langle y_1, y_2, y_3, \ldots, y_m \rangle$ with their terminal values x_{n_e} and y_{m_e}, respectively, if $x_{n_e} \leq y_{m_e}$ and $|X| \geq |Y|$, then we say that X dominates Y.*

According to Definition 10, when X ends at a smaller integer but is longer than Y, the newly considered interval should be extended from X, rather than Y, if the addition does not produce any overlap. As a result, Y becomes useless because it is dominated by X.

TABLE III: An example of the algorithm for LIIS with comparison by full interval, where $X = \langle [42, 50], [20, 27], [3, 20], [23, 31], [27, 36] \rangle$. The answer is $\langle [3, 20], [27, 36] \rangle$ with length 28.

X		$\lvert x_i \rvert$	(terminal value, cumulative length)		
x_1	[42, 50]	9	**(50, 9)**		
x_2	[20, 27]	8	**(27, 8)**	(50, 9)	
x_3	[3, 20]	18	**(20, 18)**	~~**(27, 8)**~~	~~**(50, 9)**~~
x_4	[23, 31]	8	(20, 18)	**(31, 26)**	
x_5	[27, 36]	10	(20, 18)	(31, 26)	**(36, 28)**

See Table III as an example. Each 2-tuple element in the structure represents a possible answer subsequence. The terminal value can help us to check if there is any overlap and determine which is the most extendable. We sort the 2-tuples with the terminal values increasingly. All the searching of predecessor and successor are also based on the terminal values.

We initialize the structure with the terminal value $x_{1_e} = 50$ and cumulative length $|x_1| = 9$, represented by $(50, 9)$. In round 2, $x_2 = [20, 27]$, x_{2_e} is smaller than x_{1_e}, so we cannot extend the length from x_1. There is no domination between x_2 and the existing answer $(50, 9)$, because the former has better (smaller) terminal value, but the latter has better (larger) cumulative length. So we insert x_2 into the structure as a potential answer, denoted as $(27, 8)$.

In round 3, we cannot extend $x_3 = [3, 20]$ from any existing answer, so we insert $(20, 18)$ into the structure. Then, both $(27, 8)$ and $(50, 9)$ are dominated by $(20, 18)$, so the two dominated elements are removed. In round 4, we insert $x_4 = [23, 31]$ to increase the cumulative length to 26. In the final round, $x_5 = [27, 36]$, we use x_{5_s} to find the predecessor by the terminal value. x_5 can be extended from its predecessor $(20, 18)$ to become $(36, 28)$. Without domination in the structure, we get the final answer from $(36, 28)$, which consists of $\langle [3, 20], [27, 36] \rangle$ with length 28.

The pseudo code of the algorithm for LIIS with comparison by full interval is omitted here.

Theorem 6. *The LIIS problem with comparison by full interval can be solved in $O(n \log n)$ time and $O(n)$ space.*

C. The Algorithm for Comparison by a Subinterval

In this algorithm, when dealing with a newly coming interval, we need to examine both the closest non-overlapping and overlapping intervals, because we do not know which interval subsequence will get a better answer. If we choose the overlapping intervals to extend, we should combine them to reduce the time and space complexities. Here, to record a potential solution, we use one 3-tuple (terminal value, subtotal length of the united intervals, cumulative length). The first tuple is the terminal value of the interval subsequence; the second tuple means the subtotal length of the several united

intervals ending at the terminal value; the last tuple represents the cumulative length of the whole increasing interval subsequence.

In this problem, we use two different domination operations. This problem is based on the Definition 9, so we also use Definition 10 to remove the 3-tuples with less extendibility. However, we allow the overlapping intervals in this problem, so Definition 10 cannot handle all situations in the LIIS problem with comparison by a subinterval. We define another type of domination, the containment of intervals, as follows.

Definition 11. (containment and domination) *Given two intervals x and y, if $x_s \leq y_s$ and $x_e \geq y_e$, we say that x* contains *y, and x* dominates *y.*

If two intervals are overlapping and increasing, we can unite them into a single interval. If a short interval is fully contained in a long interval, then the short one is dominated and it should be removed.

Table IV shows an example of our algorithm for LIIS with comparison by a subinterval. First, we insert $x_1 = [31, 40]$ into the answer structure and represent it as a 3-tuple $(40, 10, 10)$. In round 2, for $x_2 = [37, 45]$, we find that the successor of $x_{s_2} = 37$ is 40, which overlaps with x_2. We unite the two overlapping intervals and get a new combined interval ending at 45 with length 15. Thus, $(45, 15, 15)$ is stored in the structure, replacing the original $(40, 10, 10)$.

In round 3, $x_3 = [25, 36]$ cannot be extended from $(45, 15, 15)$, so we directly insert $(36, 12, 12)$, representing x_3 itself, into the structure. There is no domination between $(45, 15, 15)$ and $(36, 12, 12)$. In round 4, we encounter a shorter interval $x_4 = [26, 30]$, which is fully contained in $x_3 = [25, 36]$. According to Definition 11, the 3-tuple $(30, 5, 5)$ for x_4 is dominated by $(36, 12, 12)$, hence $(30, 5, 5)$ is removed.

In round 5, $x_5 = [42, 51]$ is extended from the predecessor of $x_{s_5} = 42$, which is $(36, 12, 12)$. This extension yields a longer result, $(51, 10, 22)$. In round 6, $x_6 = [49, 55]$ can be extended from the successor of $x_{s_6} = 49$, which is $(51, 10, 22)$, resulting in a longer solution $(55, 14, 26)$. Finally, we obtain the answer length 26, ending at 55, consisting of $\langle[25, 36],\ [42, 51],\ [49, 55]\rangle$. In a short summary, for a newly coming interval, we need to attempt extension from the predecessor of the starting value (for a non-overlapping extension), and the successor of the starting value (for an overlapping extension).

Algorithm 1 illustrates the algorithm for LIIS with comparison by a subinterval. T is a data structure, and t denotes a 3-tuple element in T. Each 3-tuple is composed of three values: the terminal value $t.terminal$, the length $t.sublen$of united intervals ending at terminal value $t.terminal$, and the cumulative length $t.length$. $t.previous()$ and $t.next()$ returns the previous and next 3-tuples of the current t, respectively. $T.insert(t)$ inserts the 3-tuple t into T in order. $T.predecessor(a)$ and $T.successor(a)$ finds the 3-tuple with the closest terminal values less and larger than a, respectively.

Algorithm 1 The algorithm for the LIIS with comparison by a subinterval.

Input: An interval sequence $X = \langle x_1,\ x_2,\ x_3,\ \ldots,\ x_n\rangle$, where $x_i = [x_{s_i}, x_{e_i}]$, $1 \leq i \leq n$.
Output: Length of the LIIS with comparison by a subinterval.

for $i \leftarrow 1$ to n **do**
  $t_{cur} \leftarrow (x_{e_i}, |x_i|, |x_i|)$
  if $t_{pred} \leftarrow T.predecessor(x_{s_i})$ exist **then**
    $t_{cur} \leftarrow (x_{e_i}, |x_i|, t_{pred}.length + |x_i|)$
  if $t_{suc} \leftarrow T.successor(x_{s_i})$ exist **then**
    if $t_{cur}.length < t_{cur}.terminal - t_{suc}.terminal + t_{suc}.length$ **then**
      $temp \leftarrow t_{cur}.terminal - t_{suc}.terminal$
      $t_{cur} \leftarrow (x_{e_i}, temp + t_{suc}.sublen, temp + t_{suc}.length)$
  $T.insert(t_{cur})$
  if $t_{cur}.previous()$ dominates t_{cur} or $t_{cur}.next()$ contain t_{cur} **then**
    $T.remove(t_{cur})$
    continue
  while t_{cur} dominates $t_{cur}.next()$ **do**
    $T.remove(t_{cur}.next())$
  while t_{cur} contain $t_{cur}.previous()$ **do**
    $T.remove(t_{cur}.previous())$
return $T.end().length$ ▷ Total length

Theorem 7. *The LIIS problem with comparison by a subinterval can be solved in $O(n \log n)$ time and $O(n)$ space.*

IV. Data Structures and Time Complexities

In this section, we summarize all the data structures and the time complexities of the algorithms for solving the interval sequence problems.

In the MIIS problem with comparison by full interval, we use an array to maintain the sorted ending values, and each step in the algorithm, the order is still retained. So, we can find a predecessor or successor in O($\log n$) time. Thus, the total time complexity is O($n \log n$).

In the MIIS problem with comparison by a subinterval, we have to keep one or more intervals in each quantity and locate the predecessor based on the ending value. Therefore, we use a red-black tree to manage the intervals within each quantity and an array to handle the trees. Each binary search for a proper quantity to insert a newly coming element requires O($\log n$) insertion attemps, and each insertion attemp needs O($\log n$) times. So, the total time complexity is O($n \log^2 n$).

In the LIIS problem, we also have to find the predecessor and successor, but the tuples can be inserted in any order. If we use an array, where an insertion needs O(n) time, it will require O(n^2) time. Therefore, we use the red-black tree to maintain the sorted order of the tuples. It results in the time complexity O($n \log n$) for both versions of the problems.

TABLE IV: An example of the algorithm for LIIS with comparison by a subinterval, where $X = \langle[31, 40], [37, 45], [25, 36], [26, 30], [42, 51], [49, 55]\rangle$. The answer is $\langle[25, 36], [42, 51], [49, 55]\rangle$, represented by (55, 14, 26), with length 26.

X		$\lvert x_i\rvert$	(terminal value, |united interval|, cumulative length)		
x_1	[31, 40]	10	**(40, 10, 10)**		
x_2	[37, 45]	9	**(45, 15, 15)**		
x_3	[25, 36]	12	**(36, 12, 12)**	(45, 15, 15)	
x_4	[26, 30]	5	~~(30, 5, 5)~~	(36, 12, 12)	(45, 15, 15)
x_5	[42, 51]	10	(36, 12, 12)	(45, 15, 15)	**(51, 10, 22)**
x_6	[49, 55]	7	(36, 12, 12)	(45, 15, 15)	**(55, 14, 26)**

V. Conclusion

In this paper, we focus on the combination of interval sequences and the longest increasing subsequence (LIS) problems. Accordingly, we define the most and the longest increasing interval subsequence (MIIS and LIIS) problems, and then design efficient algorithms to solve them. It is worth noting that our definitions of interval comparison can accommodate various scenarios. In other words, anyone can propose his own definition of the interval comparison. For example, given two intervals $x = [x_s, x_e]$ and $y = [y_s, y_e]$, if we do not care about the increasing property of the starting values of an interval sequence, we can only compare x and y by their subintervals x' and y' with constraint c.

Taking the interval sequence problem as a start, it may be further applied to various fields. In order to assess the similarity between the two interval sequences, the *longest common increasing subsequence* [3, 6, 7, 9, 13–15] may be a good direction for further study. Considering the cyclic nature of yearly temperatures, we may apply the concept of intervals to the *longest increasing circular subsequence* [4]. The interval as a widely used element, it has the potential to be employed in many various fields.

References

[1] M. H. Albert, M. D. 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.

[2] W.-T. Chan, Y. Zhang, S. P. Y. Fung, D. Ye, and H. Zhu, "Efficient algorithms for finding a longest common increasing subsequence," *Proceedings of the 16th International Symposium on Algorithms and Computation (ISAAC 2005)*, Sanya, Hainan, China, 2005. Also in *Lecture Notes in Computer Science*, Vol. 3827, pp. 665–674, 2005.

[3] W.-T. Chan, Y. Zhang, S. P. Y. Fung, D. Ye, and H. Zhu, "Efficient algorithms for finding a longest common increasing subsequence," *Journal of Combinatorial Optimization*, Vol. 13, pp. 277–288, 2007.

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

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

[6] M. Kutz, G. S. Brodal, K. Kaligosi, and I. Katriel, "Faster algorithms for computing longest common increasing subsequences," *Journal of Discrete Algorithms*, Vol. 9, No. 4, pp. 314–325, 2011.

[7] S.-F. Lo, K.-T. Tseng, C.-B. Yang, and K.-S. Huang, "A diagonal-based algorithm for the longest common increasing subsequence problem," *Theoretical Computer Science*, Vol. 815, pp. 69–78, 2020.

[8] 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.

[9] Y. Sakai, "A linear space algorithm for computing a longest common increasing subsequence," *Information Processing Letters*, Vol. 99, No. 5, pp. 203–207, 2006.

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

[11] 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.

[12] 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.

[13] C.-B. Yang and R. C.-T. Lee., "A fast algorithm for computing a longest common increasing subsequence," *Journal of the Chinese Institute of Engineers*, Vol. 10, No. 6, pp. 691–699, 1987.

[14] I.-H. Yang, C.-P. Huang, and K.-M. Chao, "A fast algorithm for computing a longest common increasing subsequence," *Information Processing Letters*, Vol. 93, No. 5, pp. 249–253, 2005.

[15] D. Zhu, L. Wang, T. Wang, and X. Wang, "A simple linear space algorithm for computing a longest common increasing subsequence," *IAENG International Journal of Computer Science*, Vol. 45, No. 3, pp. 472–477, 2018.