JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 41, 525-541 (2025)
DOI: 10.6688/JISE.202505_41(3).0001

Efficient Algorithms for the Increasing Interval Subsequence Problems

GUAN-TING CHEN AND CHANG-BIAU YANG+
Department of Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, 804 Taiwan
E-mail: t36085520@icloud.com; cbyang@cse.nsysu.edu.tw

This paper focuses on the interval sequence problems. 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. Thus, we define four versions of the *longest increasing subsequence* (LIS) problems on the interval sequences. Accordingly, we give a new research direction concerning interval sequences. For each version of the problem, we design an efficient algorithm to solve it 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.

Keywords: interval, interval sequence, longest increasing subsequence, most increasing interval subsequence, longest increasing interval subsequence

1. INTRODUCTION

Data sequences are frequently generated across various applications, and extensive research in this field has a rich history. For example, DNA sequences enable us to decipher the characteristics and lifestyles of species. To uncover the evolutionary relationships between different species, the similarity comparison of DNA sequences may serve as a measurement method. In addition to the character sequences, such as DNA or protein sequences, numeric sequences also play a crucial role in our lives. Examples include time series data, stock prices, and temperature fluctuations. To extract valuable insights from these numerical sequences, researchers have proposed a variety of related problems.

The *longest increasing subsequence* (LIS) problem aims to find the strictly increasing subsequence with the maximum length in a numeric sequence. For example, given a 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 [1] introduced the LIS problem and proposed an $O(n \log n)$-time algorithm for solving it, where n denotes the length of the given sequence.

Received November 15, 2023; revised December 25, 2023; accepted January 16, 2024.
Communicated by Chao-Lin Liu.
+ Corresponding author.
* This research work was partially supported by National Science and Technology Council of Taiwan under contract NSTC 112-2221-E-110-026-MY2.

LIS has been extensively studied and finds diverse applications. The previous researches on the LIS-related problems are listed in Table 1.

Table 1. The related researches on the longest increasing subsequence (LIS) problem and its variants. *n*: input sequence length; *l*: answer length; *w*: size of the maximum antichain.

Year	Author(s)	Time complexity	Note
1961	Schensted [1]	$O(n\log n)$	Young tableau, binary search
1977	Hunt and Szymanski [2]	$O(n\log\log n)$	Match pair, van Emde Boas tree
2000	Bespamyatnikh and Segal [3]	$O(n\log\log n)$	Enumerating all answers
2010	Crochemore and Porat [4]	$O(n\log\log l)$	Split block
2013	Alam and Rahman [5]	$O(n\log n)$	Divide-and-conquer
2017	Kloks *et al.* [6]	$O(wn\log\min(\frac{n}{w},l))$	Partially ordered set
2025	This paper	$O(n\log n)$, $O(n\log^2 n)$	Interval sequence

In 1975, Fredman [7] proved that the comparison operation in the LIS requires at least $n\log n - n\log\log n + O(n)$ units of times. Hunt and Szymanski designed another algorithm to solve the problem in 1977; its time complexity is still $O(n\log n)$. Furthermore, by using the van Emde Boas tree data structure [8], the time complexity can be reduced to $O(n\log\log n)$, when the given sequence is restricted to a permutation of the numbers from 1 through n. In 1999, Aldous and Diaconis [9] proved that the lower bound of the time complexity for solving the LIS problem is $\Omega(n\log n)$ if the sequence is unconstrained. In 2017, Kloks *et al.* [6] proposed an algorithm for the longest ascending sequence which is composed of elements from partial order sets.

Besides the conventional LIS problem, various variants have been proposed in prior researches [10–21]. The *longest increasing circular subsequence* (LICS) problem [10, 11] aims to find the LIS of a circular sequence, in which the given sequence can be rotated so that every position could be the starting point. In the *longest almost increasing subsequence* (LaIS) problem [12–14], a small drop within an increasing subsequence is allowed to exhibit its flexibility.

The *longest common increasing subsequence* (LCIS) problem [15–20] focuses on the similarity by finding the common increasing subsequence between two sequences. Besides, to reflect real-world situations, two new variants were defined in 2024: the *longest wave subsequence with trend* (LWSt) problem and the *longest wave subsequence within r segments* (LWSr) problem [22]. A wave sequence consists of alternating increasing and decreasing subsequences, so the wave problem can be seen as a generalization of the traditional LIS problem.

These previous researches primarily focused on numeric sequences, where each data point is represented by a single value. However, the real-world data often consist of intervals, rather than individual values. Examples include daily temperature ranges and stock price fluctuations (low and high). Fig. 1 shows an example of stock prices, consisting of daily lowest and highest prices. The lowest and highest prices of each day forms an interval, rather than a single value. Therefore, the primary goal of this paper is to investigate

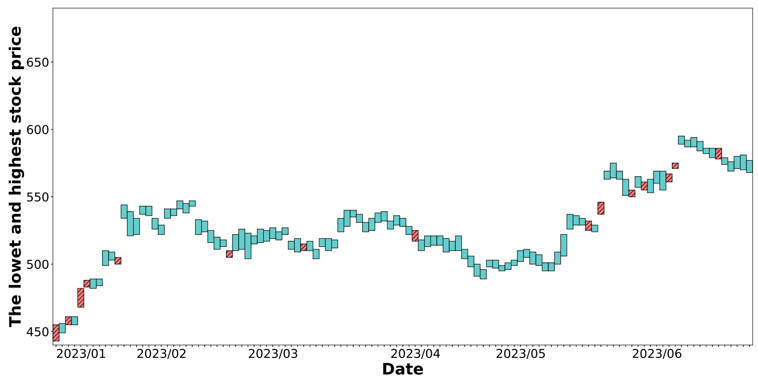

Fig. 1. Daily lowest and highest stock prices of TSMC in the first half of 2023, where the marked result is obtained from MIIS with comparison by full interval.

and address challenges related to the analysis of interval sequences . We aim to import the interval concept to the LIS problem. Accordingly, we will first define the LIS problem on the interval sequences. Through our research, the interval sequences may become more practical for real-world applications.

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

- *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

We design an efficient algorithm to solve each of the above problems 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.

The rest of the paper is structured as follows. In Section 2, we will present the formal definitions of our problems. Section 3 and Section 4 provide the algorithms for solving the MIIS problems and the LIIS problems, respectively. Finally, the conclusion is given in Section 5.

2. PROBLEM DEFINITIONS

We first give the definition of an interval as follows.

Definition 1. (interval) *An* interval $x = [x_s, x_e]$ *is a closed integer interval, ranging from a* starting value x_s *to an* ending value x_e *(inclusive),* $x_s \leq x_e$. *The interval length is* $|x| = x_e - x_s + 1$.

Before handling the increasing subsequence, we need the definition of the comparison of two intervals.

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 $x < y$, or $y > x$.*

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 the basic interval comparison in Definition 2, we derive two comparison schemes as follows.

(1) Full interval: If the full range of one interval is greater than the full range of another interval (the two intervals do not overlap with each other), then we say that the former interval is greater than the latter.

(2) Subinterval: If there exists a subinterval within one interval that is fully greater than a subinterval within another interval, then we say that the former interval is greater than the latter. Here, the length of the subinterval is a predefined constant.

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 $x <_f y$, or $y \, {}_f\!> x$.*

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 $x = [12, 14]$, $y = [15, 17]$, and $z = [14, 17]$, we say that $x <_f y$ with comparison by full interval; we cannot conclude the relation between x and z, 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 $x_s \leq x'_s \leq x'_e \leq x_e$.*

For instance, $[12, 12]$, $[13, 14]$ and $[13, 16]$ are some subintervals of an interval $x = [12, 16]$.

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

For instance, $x = [12, 16]$, $y = [14, 17]$, $z = [9, 18]$ and $c = 3$, we say that $x <_3 y$, because subinterval $x' = [12, 14] \subseteq X$, subinterval $y' = [15, 17] \subseteq y$, and $x' <_f y'$. But, we cannot say the relation between x and z because that neither $x < z$, nor $z < x$. 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* (*maximum quantity*) *of intervals.*

Suppose that $X = \langle [42,50], [20,27], [28,36], [25,33], [40,45], [43,49]\rangle$.Then, the answer of MIIS with comparison by full interval is $\langle [20,27], [28,36], [40,45]\rangle$ with 3 intervals.

Suppose that $Y = \langle [12,16], [9,18], [14,17], [17,19], [6,13]\rangle$ and the constraint constant $c = 3$. The answer of MIIS with comparison by a subinterval is $\langle [12,16], [14,17], [17,19]\rangle$ with 3 intervals, because $[12,16] <_3 [14,17] <_3 [17,19]$. As a note, there exist $[12,14] \subseteq [12,16]$ and $[15,17] \subseteq [14,17]$ that $[12,14] <_f [15,17]$, and there exist $[14,16] \subseteq [14,17]$ that $[14,16] <_f [17,19]$, where the length of each subinterval is 3.

While dealing with the MIIS problem, we consider the maximum number (quantity) of intervals in the answer. Next, our focus shifts to the total length of the intervals involved in the answer, rather than counting the number of the intervals.

Definition 7. (length of an interval sequence) *Given an interval sequence $X = \langle x_1, x_2, x_3, \ldots, x_n\rangle$, where $x_i = [x_{s_i}, x_{e_i}]$, $1 \le i \le n$, 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 8. (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 with the maximum total length.*

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 subinterval length.

Definition 9. (terminal value of an increasing interval subsequence) *Given an increasing interval subsequence $Y = \langle y_1, y_2, y_3, \ldots, y_m\rangle$, where $y_i = [y_{s_i}, y_{e_i}]$, $1 \le i \le m$, the* terminal value *of Y is the last ending value y_{e_m}.*

Suppose that $X = \langle [42,50], [20,27], [3,20], [24,31], [27,36]\rangle$. The answer for LIIS with comparison by full interval is $\langle [3,20], [27,36]\rangle$ with length 28 and terminal value 36. Suppose that $Y = \langle [31,40], [37,45], [25,36], [26,30], [42,51], [49,55], [46,53]\rangle$. The answer for LIIS with comparison by a subinterval is $\langle [25,36], [42,51], [49,55]\rangle$ with length 26 and terminal value 55.

3. MOST INCREASING INTERVAL SUBSEQUENCE

3.1 The Algorithm for Comparison by Full Interval

Our algorithm processes the elements of the input sequence X sequentially, one by one. We have to check whether the newly coming interval overlaps with any existing interval. We utilize the ending value of each interval as its representative in the solution structure T. For example, if $[20,27]$ is a part of the solution, its representative value would be 27. The ending value of the interval can not 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 increasing interval subsequence at each quantity, and to ensure that we have the maximum number (quantity) of intervals found in the answer. We illustrate our algorithm with an example, as shown in Table 2.

Table 2. 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]	<u>**50**</u>		
x_2	[20, 27]	<u>**27**</u>		
x_3	[28, 36]	27	<u>**36**</u>	
x_4	[25, 33]	27	36	
x_5	[40, 45]	27	36	<u>**45**</u>
x_6	[43, 49]	27	36	45

Let u denote the last terminal value in the solution structure T. For each newly coming interval $x_i = [x_{s_i}, x_{e_i}]$, the following cases are considered:

(1) If $u < x_{s_i}$, then x_{e_i} is appended to the end of T to increase the quantity. Examples include x_1, x_3 and x_5.

(2) If $x_{s_i} \leq u \leq x_{e_i}$, it implies that x_i overlaps with the interval represented by u. Then discard x_i. x_6 is an example for this case.

(3) If $x_{e_i} < u$, there exists a successor of x_{e_i} in T, denoted by v. Next, find the predecessor of v, denoted by $pred(v)$. If x_i does not overlap with the interval represented by $pred(v)$, *i.e.* $pred(v) < x_{s_i}$, then replace v by x_{e_i}; otherwise, discard x_{e_i}. x_2 serves as an example for the former subcase, and x_4 exemplifies the latter.

Algorithm 1 presents the pseudo code of the algorithm for solving the MIIS problem with comparison by full interval. T represents the solution structure for maintaining the current terminal value at each quantity. $T.append(a)$ inserts the value a into the end of T. $T.successor(a)$ returns the smallest element greater than a, and $T.predecessor(a)$ returns the largest element less than a.

Theorem 1. *In Algorithm 1, the terminal value recorded in each quantity is the smallest value required to obtain that quantity.*

Proof. We will prove this theorem by induction. Consider an input interval sequence

Algorithm 1 : The algorithm for the MIIS with comparison by full interval.

Input: An interval sequence $X = \langle x_1, x_2, x_3, \ldots, x_n \rangle$, where $x_i = [x_{s_i}, x_{e_i}]$, $1 \le i \le n$.
Output: The number of intervals for the MIIS with comparison by full interval.

```
1: $T.append(-\infty)$, $T.append(x_{e_1})$                ▷ Regard $-\infty$ as the quantity 0
2: for $i \leftarrow 2$ to $n$ do
3:     if $T.end(\,) < x_{s_i}$ then          ▷ $u = T.end(\,)$ is the last terminal value, case 1
4:         $T.append(x_{e_i})$                                  ▷ Append $x_{e_i}$ to the end
5:     else if $x_{e_i} < T.end(\,)$ then                                        ▷ Case 3
6:         $v \leftarrow T.successor(x_{e_i})$
7:         if $T.predecessor(v) < x_{s_i}$ then ▷ $x_i$ does not overlap with the predecessor of $v$
8:             $T.replace(v, x_{e_i})$          ▷ Replace $v$ by $x_{e_i}$, former subcase of case 3
9: return $T.size(\,) - 1$                  ▷ Number of elements in $T$, excluding $-\infty$
```

$X = \langle x_1, x_2, x_3, \ldots, x_n \rangle$. 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 smallest terminal value, and let the current quantity be denoted as q.

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

1. If $u < x_{s_{k+1}}$, 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_{k+1} overlaps with the interval represented by 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 $x_{e_{k+1}} < u$, 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 $x_{e_{k+1}}$ ($x_{e_{k+1}} < v$). This replacement with $x_{e_{k+1}}$ makes the value become smaller.
   (3.2) If $pred(v) \ge x_{s_{k+1}}$ (x_{k+1} overlaps with $pred(v)$), then we cannot extend from $pred(v)$ to $x_{e_{k+1}}$. In other words, the possible maximum quantity of $x_{e_{k+1}}$ is $w-1$. However, $pred(v)$ is also at quantity $w-1$ and $pred(v)$ dominates $x_{e_{k+1}}$ ($pred(v) < x_{e_{k+1}}$). 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 $O(n \log n)$ time and $O(n)$ space.*

Proof. We use an array to maintain the sorted ending values (terminal value at each quantity). There are two possible situations for each newly coming $x_i = [x_{i_s}, x_{i_e}]$. First, if the

x_{i_s} is larger than the last terminal value in the structure, we increase the quantity by appending x_i to the end. If not, we find the successor of x_{i_e} by binary search in O($\log n$) time. Then, we perform the replacement if the x_{i_s} does not overlap with the predecessor of that successor, which is checked in Line 7 of Algorithm 1. Each check and replacement need only constant time. Therefore, Algorithm 1 can be executed in O($n \log n$) time. □

3.2 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 solution structure T should be the most extendable.

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

(4) For any pair of intervals included in the answer, there should exist non-overlapping subintervals with at least a predefined length c.

Definition 10. (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.*

In the solution structure T, each dominated interval in the answer at the same quantity should be deleted, and all non-dominated intervals are kept. For each non-dominated interval with quantity j, we record its starting and ending values in $T[j]$.

For a newly coming interval $x_i = [x_{s_i}, x_{e_i}]$, we employ the concept of binary search to decide the appropriate quantity m to be inserted for x_i. Here, m is the minimal value such that x_i cannot be extended from any other interval at quantity m. Let l and r represent the minimum and the maximum possible quantities, respectively. Then, $m = \frac{l+r}{2}$ is set as the quantity 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. If x_i can be extended from an interval at quantity m, it means that x_i may be extended from a larger quantity; thus l is set to $m+1$. If x_i cannot be extended from any interval at quantity m, it means that x_i should be extended from a smaller quantity; so r is set to $m-1$. When $l > r$, the round will be ended, and l is the quantity for inserting the new interval.

Table 3 shows an example with constraint $c = 3$. We explain the operations in round 5 for $x_5 = [6, 13]$. Here, $l = 1$ and $r = 3$ initially. We first check quantity $m = \frac{l+r}{2} = 2$. At quantity $m = 2$, since we cannot find the predecessor of x_5, x_5 cannot be extended from quantity $m = 2$. Therefore, 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 ensure no domination in quantity 1. Finally, the answer is $\langle [12, 16]$, $[14, 17]$, $[17, 19] \rangle$ with 3 intervals.

Algorithm 2 shows the pseudo code for solving the MIIS problem with comparison by a subinterval. T is a 2-dimensional solution structure, where the first dimension represents the quantity, and the second dimension represents the ending intervals at that

Table 3. An example for the algorithm for MIIS with comparison by a subinterval, where $X = \langle[12,16],[9,18],[14,17],[17,19],[6,13]\rangle$ and the constraint constant $c = 3$. The answer is $\langle[12,16],[14,17],[17,19]\rangle$ with 3 intervals.

X	q(quantity)	1	2	3
x_1	$[12,16]$	$[12,16]$		
x_2	$[9,18]$	$[12,16]$ $[9,18]$		
x_3	$[14,17]$	$[12,16]$ $[9,18]$	$[14,17]$	
x_4	$[17,19]$	$[12,16]$ $[9,18]$	$[14,17]$	$[17,19]$
x_5	$[6,13]$	$[6,13]$ ~~$[12,16]$~~ ~~$[9,18]$~~	$[14,17]$	$[17,19]$

quantity. $T.size(\ )$ returns the size of the first dimension. $T[m]$ stores the ending intervals at quantity m. $T[m].predecessor(x_i)$ returns the predecessor of interval x_i by checking the ending value x_{i_e}. $T[m].insert(x_i)$ inserts the interval x_i into $T[m]$. $x_i.previous(\ )$ and $x_i.next(\ )$ returns the previous and next intervals, respectively.

Theorem 3. *In Algorithm 2, 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 quantities do not dominate each other.

Theorem 4. *In Algorithm 2, there is no domination between the intervals of different quantities.*

Proof. An interval of a smaller quantity cannot dominate an interval in a larger quantity, since the latter has a better solution with a greater quantity. Now, we aim to prove that an interval of a larger quantity cannot dominate an interval of a smaller quantity. Assume that an interval y is of quantity i and an interval x is of quantity j, where $i < j$ and $x < y$ (x dominates y). In this situation, there should exist another interval z of quantity i that can be extended for x, and $z < x$. This implies that $z < y$ (z dominates y in quantity i). However, all intervals in quantity i do not mutually dominate each other. It leads to that the assumption is not valid. Therefore, there is no domination between the intervals of different quantities. □

Theorem 5. *The MIIS problem with comparison by a subinterval can be solved in $O(n\log^2 n)$ time and $O(n)$ space.*

Algorithm 2 : The algorithm for the MIIS with comparison by a subinterval.

Input: An interval sequence $X = \langle x_1, x_2, x_3, \dots, x_n\rangle$ and a constraint constant c, where $x_i = [x_{s_i}, x_{e_i}]$, $1 \le i \le n$.

Output: The number of intervals for the MIIS with comparison by a subinterval.

```
1:  $T[1].append(x_1)$                                  ▷ Initialization
2:  for $i \leftarrow 2$ to $n$ do
3:      $l \leftarrow 1$, $r \leftarrow T.size(\ )$                   ▷ Size of the first dimension
4:      while $l \le r$ do                  ▷ Binary search for a proper quantity to insert $x_i$
5:          $m \leftarrow \frac{l+r}{2}$
6:          if $t_{pred} \leftarrow T[m].predecessor(x_i)$ exists then    ▷ Predecessor of ending value
7:              if $t_{pred} <_c x_i$ then                  ▷ Can extend the quantity
8:                  $l \leftarrow m+1$
9:              else                                ▷ Cannot extend the quantity
10:                 $r \leftarrow m-1$
11:         else
12:             $r \leftarrow m-1$
13:     $T[l].insert(x_i)$
14:     if $x_i.previous(\ ) < x_i$ then        ▷ $x_i$ is dominated, remove it (by Definition 2)
15:         $T[l].remove(x_i)$
16:     while $x_i < x_i.next(\ )$ do                ▷ Remove intervals dominated by $x_i$
17:         $T[l].remove(x_i.next(\ ))$
18: return $T.size(\ )$
```

Proof. For a newly coming interval, we first we find out the corresponding quantity by using the concept of binary search. It needs O($\log q$) insertion attempts, where q denotes the length of the MIIS answer. For each insertion attempt at a specific quantity, we utilize one predecessor operation to find the potential insertion position, which requires O($\log n$) time. Once the interval is inserted into the corresponding quantity, we proceed to remove dominated intervals. Each interval can be removed at most once. The domination process requires O(n) time throughout the algorithm. Thus, the algorithm takes O($n\log n\log q$) = O($n\log^2 n$) time. The space complexity is O(n) since at most n intervals are stored in the solution structure. □

4. LONGEST INCREASING INTERVAL SUBSEQUENCE

4.1 The Algorithm for Comparison by Full Interval

Our algorithm sequentially handles the elements of the input sequence X, one at a time. Two steps are performed for inserting a newly coming interval x_i into the solution structure T as follows.

(1) Find the possible interval subsequence for extension while ensuring that it does not overlap with x_i. Afterward, insert x_i into T.

(2) Check for the existence of dominated intervals and remove them.

We require another definition of the domination of the LIIS problem with comparison by full interval to characterize the solution structure as the most extendable. This is defined as follows.

Definition 11. (domination of the non-overlapping increasing interval sequence) *Let* $X = \langle x_1,\ x_2,\ x_3,\ \ldots,\ x_m\rangle$ *and* $Y = \langle y_1,\ y_2,\ y_3,\ \ldots,\ y_n\rangle$ *be two non-overlapping increasing interval sequences, with their terminal values* x_{e_m} *and* y_{e_n}*, respectively. Here,* $x_i \cap x_j = \emptyset$ *and* $y_i \cap y_j = \emptyset$ *for* $i \neq j$*. If* $x_{e_m} \leq y_{e_n}$ *and* $|X| \geq |Y|$*, then we say that* X *dominates* Y*.*

The solution structure T consists of 2-tuples, each formed by (*terminal value*, *cumulative length*). Each 2-tuple t represents an increasing interval subsequence, where $t.ter$ denotes the terminal value of the subsequence, and $t.len$ represents the cumulative length, which is the sum of the lengths of the intervals in the subsequence. The 2-tuples in T is sorted with the terminal values in increasing order. All searches for predecessors and successors are based on these terminal values.

Table 4 shows an example of our algorithm. For example, $(36,28)$ represents a possible answer subsequence with a terminal value 36 and a total length 28. $(36,28)$ may consist of one or more intervals, which are not necessarily consecutive.

In rounds 1 and 2, x_1 and x_2 are inserted into T, since they do not overlap with each other. In round 3, we cannot extend $x_3 = [3,20]$ from any existing answer, so we insert $(20,18)$ into the structure temporarily. Then, both $(27,8)$ and $(50,9)$ are dominated by $(20,18)$, so the two dominated 2-tuples are removed. In round 4, $x_4 = [24,31]$ is inserted to increase the cumulative length to 26. In the final round, $x_5 = [27,36]$, we use $x_{s_5} = 27$ 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.

Algorithm 3 presents the algorithm for LIIS with comparison by full interval. $T.predecessor(a)$ finds the 2-tuple with the largest terminal value less than a. $T.insert(t)$ inserts the 2-tuple t into T in the order of $t.ter$. $t.previous(\ )$ and $t.next(\ )$ returns the previous and next 2-tuples of t, respectively. $T.end(\ )$ directly returns the last 2-tuple, and its cumulative length is the answer if the final round is finished.

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

Proof. We utilize a red-black tree to maintain the 2-tuples. With this tree structure, we can find the predecessor in O($\log n$) time. The existing predecessor represents a non-overlapping interval subsequence to extend by the newly coming interval. Before insertion, there is no domination in the structure. Therefore, the predecessor should be the most potential for the newly coming interval to extend. From Lines 2 to 5 in Algorithm 3, we find out if there exists a predecessor of x_{i_e} for extension, which needs O($\log n$) time. In domination, we check if the intervals dominate with each other. The domination occurs at

Algorithm 3 : The algorithm for the LIIS with comparison by full interval.

Input: An intervals 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 full interval.

```
1:  for $i \leftarrow 1$ to $n$ do
2:      if $t_{pred} \leftarrow T.predecessor(x_{s_i})$ exists then
3:          $t \leftarrow (x_{e_i}, t_{pred}.len + |x_i|)$          ▷ Append $x_i$ to the solution
4:      else
5:          $t \leftarrow (x_{e_i}, |x_i|)$          ▷ Create an independent solution
6:      $T.insert(t)$
7:      if $t.previous(\ )$ dominates $t$ then
8:          $T.remove(t)$
9:          continue
10:     while $t$ dominates $t.next(\ )$ do
11:         $T.remove(t.next(\ ))$
12: return $T.end(\ ).length$          ▷ Total length
```

most $n-1$ times throughout the algorithm, requiring $O(n)$ removals in total. Therefore, the algorithm can be executed in $O(n \log n)$ time with $O(n)$ space. □

Table 4. An example of the algorithm for LIIS with comparison by full interval, where $X = \langle [42,50], [20,27], [3,20], [24,31], [27,36] \rangle$. The answer is $\langle [3,20], [27,36] \rangle$ with length 28, represented by (36, 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	[24, 31]	8	(20, 18)	**(31, 26)**	
x_5	[27, 36]	10	(20, 18)	(31, 26)	**(36, 28)**

4.2 The Algorithm for Comparison by a Subinterval

In the solution structure T, we use a 2-tuple (*terminal value*, *cumulative length*) to represent a potential solution. This representation is consistent with the algorithm used for comparison by full interval in the previous subsection.

To further characterize the solution structure and its extendability, we need another domination scheme, defined as follows.

Definition 12. (domination of the overlapping increasing interval sequence) *Let $X = \langle x_1, x_2, x_3, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, y_3, \ldots, y_n \rangle$ be two overlapping increasing interval sequences, with their terminal values x_{e_m} and y_{e_n}, respectively. Here, $|x_i \cap x_{i+1}| \geq 0$ and $|y_j \cap y_{j+1}| \geq 0$ for $1 \leq i \leq m-1$ and $1 \leq j \leq n-1$. If $|X| \geq |Y|$ and $|X| - |Y| \geq x_{e_m} - y_{e_n}$, then we say that X dominates Y.*

Table 5. An example of the algorithm for LIIS with comparison by a subinterval, where $X = \langle[31,40],[37,45],[25,36],[26,30],[42,51],[49,55],[46,53]\rangle$. The answer is $\langle[25,36],[42,51],[49,55]\rangle$, represented by (55, 26), with length 26.

	X	(terminal value, cumulative length)		
x_1	[31, 40]	**(40, 10)**		
x_2	[37, 45]	~~(40, 10)~~	**(45, 15)**	
x_3	[25, 36]	**(36, 12)**	(45, 15)	
x_4	[26, 30]	~~**(30, 5)**~~	(36, 12)	(45, 15)
x_5	[42, 51]	(36, 12)	~~(45, 15)~~	**(51, 22)**
x_6	[49, 55]	(36, 12)	~~(51, 22)~~	**(55, 26)**
x_7	[46, 53]	(36, 12)	~~**(53, 20)**~~	(55, 26)

In this algorithm, when dealing with a newly arriving interval x_i, we examine both the closest non-overlapping and overlapping intervals, because we do not know which interval subsequence will yield a better answer. So, we attempt an extension from the predecessor of the starting value of x_i for a non-overlapping extension, and the successor of the starting value of x_i for an overlapping extension. Then, the superior extension is selected for x_i. Subsequently, the domination operations are performed to remove the dominated 2-tuples.

Table 5 shows an example of our algorithm. First, we insert $x_1 = [31,40]$ into T, represented by a 2-tuple $(40,10)$. In round 2, $x_2 = [37,45]$ overlaps with x_1 by finding the successor of $x_{s_2} = 37$, which is 40. The extension from x_1 for x_2 increases the length 5, resulting a 2-tuple $(45,15)$ with a terminal value of 45 and a length of 15. Then, by Definition 12, $(40,10)$ is dominated by $(45,15)$. Thus, $(40,10)$ is removed.

In round 3, $x_3 = [25,36]$ is inserted and represented by $(36,12)$. In round 4, $x_4 = [26,30]$, represented by $(30,5)$, is dominated by $(36,12)$. Hence, $(30,5)$ is removed.

In round 5, $x_5 = [42,51]$ has a non-overlapping extension from $(36,12)$, found by the predecessor of $x_{s_5} = 42$. x_5 also has an overlapping extension from $(45,15)$, found by the successor of $x_{s_5} = 42$. The better extension is from $(36,12)$, yielding $(51,22)$. In addition, $(45,15)$ is dominated by $(51,22)$, so the former is deleted. In round 6, $x_6 = [49,55]$ can be extended from $(51,22)$, resulting in $(55,26)$. And $(51,22)$ is removed since it is dominated by $(55,26)$. x_7 can be extended from $(36,12)$, producing $(53,20)$, but it is dominated by $(55,26)$ and removed. Finally, we obtain the answer length of 26, ending at 55, consisting of $\langle[25,36], [42,51], [49,55]\rangle$.

Algorithm 4 illustrates the algorithm for LIIS with comparison by a subinterval. The solution structure T and the operations are the same as those in Algorithm 3.

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

Proof. We use a red-black tree to maintain the 2-tuples in the solution structure T and sort the 2-tuples increasingly by their terminal values. Then, we can find each predecessor or successor in $O(\log n)$ time. Each interval can only be removed at most once. As a result, the domination is performed at most $n-1$ times throughout the algorithm. Therefore, Algorithm 4 can be solved in $O(n \log n)$ time and $O(n)$ space. $\square$

Algorithm 4 : 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 $x_i = [x_{s_i}, x_{e_i}]$, $1 \le i \le n$.
Output: Length of the LIIS with comparison by a subinterval.
1: for $i \leftarrow 1$ to $n$ do
2:     $t \leftarrow (x_{e_i}, |x_i|)$
3:     if $t_{pred} \leftarrow T.predecessor(x_{s_i})$ exists then          ▷ Non-overlapping
4:         $t \leftarrow (x_{e_i}, t_{pred}.len + |x_i|)$
5:     if $t_{suc} \leftarrow T.successor(x_{s_i})$ exists then          ▷ Overlapping
6:         $l' \leftarrow t_{suc}.len + x_{e_i} - t_{suc}.ter$          ▷ Adding non-overlapping part
7:         if $t.len < l'$ then
8:             $t \leftarrow (x_{e_i}, l')$
9:     $T.insert(t)$
10:    if $t.previous(\,)$ dominates $t$ or $t.next(\,)$ dominates $t$ then
11:        $T.remove(t)$          ▷ Remove the dominated 2-tuple
12:        continue
13:    while $t$ dominates $t.next(\,)$ do
14:        $T.remove(t.next(\,))$
15:    while $t$ contains $t.previous(\,)$ do
16:        $T.remove(t.previous(\,))$
17: return $T.end(\,).length$          ▷ Total length
```

Table 6. The data structures, time complexities and space complexities of the MIIS and LIIS algorithms.

Problem		Data Structure	Time	Space
MIIS	full interval	array	$O(n \log n)$	$O(n)$
	subinterval	array red-black tree	$O(n \log^2 n)$	$O(n)$
LIIS	full interval	red-black tree	$O(n \log n)$	$O(n)$
	subinterval	red-black tree	$O(n \log n)$	$O(n)$

5. CONCLUSION

In this paper, we focus on the generalized combination of interval sequences and the longest increasing subsequence (LIS) problems. We define the most and the longest increasing interval subsequence (MIIS and LIIS) problems, and then design efficient algorithms to solve them. We summarize our algorithms in Table 6.

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 one does not care about the increasing property of the starting values of an interval sequence, the comparison between x and y can be based solely on their subintervals x' and y' with a width of 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* [15, 16, 18–20, 23] may be a good direction for further study. Additionally, considering the cyclic nature of yearly temperatures, we may apply the concept of intervals to the *longest increasing circular subsequence* [11]. Given the widespread use of intervals, they hold the potential for application in numerous fields.

REFERENCES

1. C. Schensted, "Longest increasing and decreasing subsequences," *Canadian Journal of Mathematics*, Vol. 13, 1961, pp. 179-191.
2. J. W. Hunt and T. G. Szymanski, "A fast algorithm for computing longest common subsequences," *Communications of the ACM*, Vol. 20, 1977, pp. 350-353.
3. S. Bespamyatnikh and M. Segal, "Enumerating longest increasing subsequences and patience sorting," *Information Processing Letters*, Vol. 76, 2000, pp. 7-11.
4. M. Crochemore and E. Porat, "Fast computation of a longest increasing subsequence and application," *Information and Computation*, Vol. 208, 2010, pp. 1054-1059.
5. 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, 2013, pp. 470-476.
6. T. Kloks, R. B. Tan, and J. van Leeuwen, "Tracking maximum ascending subsequences in sequences of partially ordered data," Technical Report UU-CS-2017-010, Department of Information and Computing Sciences, Utrecht University, Netherlands, 2017.
7. M. L. Fredman, "On computing the length of longest increasing subsequences," *Discrete Mathematics*, Vol. 11, 1975, pp. 29-35.
8. P. van Emde Boas, R. Kaas, and E. Zijlstra, "Design and implementation of an efficient priority queue," *Mathematical Systems Theory*, Vol. 10, 1976, pp. 99-127.
9. D. Aldous and P. Diaconis, "Longest increasing subsequences: from patience sorting to the baik-deift-johansson theorem," *Bulletin of the American Mathematical Society*, Vol. 36, 1999, pp. 413-432.
10. 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, 2007, pp. 55-59.
11. S. Deorowicz, "An algorithm for solving the longest increasing circular subsequence problem," *Information Processing Letters*, Vol. 109, 2009, pp. 630-634.
12. A. Elmasry, "The longest almost-increasing subsequence," *Information Processing Letters*, Vol. 110, 2010, pp. 655-658.
13. 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, 2013, pp. 12-20.
14. 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, 2021, pp. 44-51.

15. 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, 2005, pp. 249-253.
16. Y. Sakai, "A linear space algorithm for computing a longest common increasing subsequence," *Information Processing Letters*, Vol. 99, 2006, pp. 203-207.
17. W.-T. Chan, Y. Zhang, S. P. Y. Fung, D. Ye, and H. Zhu, "Efficient algorithms for finding a longest common increasing subsequence," in *Proceedings of the 16th International Symposium on Algorithms and Computation*, LNCS 3827, 2005, pp. 665-674.
18. M. Kutz, G. S. Brodal, K. Kaligosi, and I. Katriel, "Faster algorithms for computing longest common increasing subsequences," *Journal of Discrete Algorithms*, Vol. 9, 2011, pp. 314-325.
19. 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, 2018, pp. 472-477.
20. 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, 2020, pp. 69-78.
21. C.-T. Tseng, C.-B. Yang, and H.-Y. Ann, "Minimum height and sequence constrained longest increasing subsequence," *Journal of Internet Technology*, Vol. 10, 2009, pp. 173-178.
22. G.-Z. Chen, C.-B. Yang, and Y.-C. Chang, "The longest common wave subsequence problem: Generalizations of the longest increasing subsequence problem," *International Journal of Foundations of Computer Science*, 2024.
23. 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, 2007, pp. 277-288.

Guan-Ting Chen received his BS degree in Computer Science and Information Engineering from National Central University, Taoyuan, Taiwan, in 2021. He later received his MS degree in Computer Science and Engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 2023. His primary research interest is computer algorithms.

Chang-Biau Yang received his BS degree in Electronic Engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1982, and his MS and Ph.D. degrees in Computer Science from National Tsing Hua University, Hsinchu, Taiwan, in 1984 and 1988, respectively. In 1990, he joined National Sun Yat-sen University as a faculty member. He has been the chairman of the Department of Computer Science and Engineering from 2001 to 2004, the deputy dean of Academic Affairs from 2007 to 2008, and the Director of Office of Library and Information Services from 2009 to 2011. From 2011 to 2023, he served as the Chairman of the Collegiate Programming Examination (CPE) Committee under the Association of Taiwan Computer Programming Contest (TCPC), and from 2019 to 2022, as the Chairman of the Association of Taiwan Computer Programming Contest (TCPC). Through his dedicated efforts in CPE, he significantly contributed to enhancing the programming skills of computer science students across Taiwanese universities. He got the Best Service Award from the IEEE Tainan Section in 2014, and the Outstanding Professor Award from the Chinese Institute of Engineers in 2019. He is currently a distinguished professor in the Department of Computer Science and Engineering at National Sun Yat-sen University in Taiwan since 2019. His research interests include computer algorithms, bioinformatics and parallel processing. Especially, he was devoted to the study of the sequence-related problems with wide applications on bioinformatics.