 ISBN/978-957-437-895-1(PDF)

# The Longest Wave Subsequence Problem: Generalizations of the Longest Increasing Subsequence Problem *
# (Extended Abstract)

Guan-Zhi Chen[a] and Chang-Biau Yang[a†]
[a]Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan

## Abstract

*In this paper, we define two generalized versions of the longest increasing subsequence (LIS) problem, called the* longest wave subsequence (*LWS*) *problem, including the LWSt and the LWSr problems. Given a numeric sequence $A$ of distinct values and a target trend sequence $T$, the LWSt problem is to find the longest subsequence of $A$, such that it is trend-preserving to the prefix of $T$. The LWSr problem is to find the longest subsequence of $A$ within $r$ segments, where the sequence in each segment is either increasing or decreasing. We propose an $O(n \log n)$-time algorithm for solving the LWSt problem, where $n$ is the length of $A$. The time complexity of our algorithm for solving the LWSr problem is $O(rn \log n)$.*

## 1 Introduction

The *order-preserving pattern matching* (OPPM) problem [1–6] has gained attention. Two numeric strings (or substrings) with the same length are said to be *order-preserving* if the relative rank of each element in one string is equal to that of the corresponding element in the other string. For example, $A = \langle 24,\ 31,\ 42,\ 40,\ 26\rangle$ and $B = \langle 10,\ 18,\ 25,\ 21,\ 16\rangle$ are order-preserving. But, $A = \langle 28, 31, 42, 40, 26\rangle$ and $B = \langle 10, 18, 25, 21, 16\rangle$ are not order-preserving, even though they have the same trend (defined as trend-preserving in this paper).

The *longest increasing subsequence* (LIS) problem also got the attention of researchers in the past [7–17]. Given a sequence $A$ of length $n$, the LIS is the increasing subsequence of $A$ with the maximal length. In 1961, it was first defined by Schensted [16], who also presented an $O(n \log n)$-time algorithm based on the *Young tableau*.

In this paper, we defines two new problems, which are the generalized version of the LIS problem, called the *longest wave subsequence* (LWS) problem. The first one is called the LWSt problem, which finds the LWS with a given trend. Given a numeric sequence $A$ composed of $n$ distinct values and a target trend sequence $T$ of length $n$, the LWSt problem is to find the longest subsequence in $A$ such that it is *trend-preserving* to the prefix of $T$. When the trend sequence $T$ consists of only one type of sequence, such as only increasing, then the LWSt problem is degenerated into the traditional LIS problem. For the LWSt problem, we propose an algorithm with $O(n \log n)$ time, inspired by the algorithms for solving the LIS problem [15, 17].

The second one is the LWS problem within $r$ segments, called the LWSr problem. We define the LWSr problem to find the longest subsequence within $r$ segments of $A$, where the subsequence constituting a segment is either increasing or decreasing. Obviously, when $r = 1$, the LWSr problem is degenerated into the traditional LIS problem. We present an $O(rn \log n)$-time algorithm for solving the LWSr problem.

## 2 Preliminaries

### 2.1 The Longest Wave Subsequence

In this subsection, we introduce the *longest wave subsequence* (LWS) problem. Given a sequence $W = \langle w_1,\ w_2,\ \ldots,\ w_n\rangle$, where $w_i \neq w_j$ for $1 \leq i \neq j \leq n$, we define the *trend sequence*

*This research work was partially supported by the Ministry of Science and Technology of Taiwan under contract MOST 108-2221-E-110-031.
†Corresponding author. E-mail: cbyang@cse.nsysu.edu.tw (Chang-Biau Yang).

$T = \langle t_1, t_2, \ldots, t_n \rangle$ of $W$ in Equation 1.

$$t_i = \begin{cases} 0, & \text{if } w_{i-1} > w_i \text{ and } 2 \le i \le n; \\ 1, & \text{if } w_{i-1} < w_i \text{ and } 2 \le i \le n; \\ \text{complement of } t_2, & \text{if } i = 1. \end{cases} \quad (1)$$

For example, suppose that a sequence $W = \langle 2, 6, 9, 8, 5 \rangle$. Then, its trend sequence is $T = \langle 0, 1, 1, 0, 0 \rangle$. Note that $t_1$ is set as the complement of $t_2$ and $t_1$ is used only for initialization. Here, the trend of $W_{1..3}$ is increasing and $W_{3..5}$ is decreasing.

**Definition 1.** (increasing segment and decreasing segment) *Given a sequence $W = \langle w_1, w_2, \ldots, w_n \rangle$ with its trend sequence $T = \langle t_1, t_2, \ldots, t_n \rangle$, where $w_{i'} \neq w_{j'}$ for $1 \le i' \neq j' \le n$, an increasing segment $W^+_{i..j}$ is a substring of $W$ such that $t_{i+1} = t_{i+2} = \ldots = t_j = 1$ and $t_i = t_{j+1} = 0$, for $1 \le i < j \le n$. Similarly, a decreasing segment $W^-_{i..j}$ is a substring of $W$ such that $t_{i+1} = t_{i+2} = \ldots = t_j = 0$ and $t_i = t_{j+1} = 1$, for $1 \le i < j \le n$. Here, $t_{n+1}$ is set as the complement of $t_n$.*

Definition 1 specifies the consecutive elements of the identical trend with the maximal length in a sequence. It clearly implies that a sequence can be decomposed into alternately increasing and decreasing segments. For example, consider a sequence $W = \langle 1, 2, 4, 9, 7, 3, 5, 6, 8 \rangle$, whose trend sequence $T = \langle 0, 1, 1, 1, 0, 0, 1, 1, 1 \rangle$. Thus, $W$ can be decomposed into $W^+_{1..4}$, $W^-_{4..6}$ and $W^+_{6..9}$. $w_4 = 9$ is an overlapping element of $W^+_{1..4}$ and $W^-_{4..6}$. We call 4 as a *turning point*, which is the ending element of the former segment and the starting element of the latter segment.

**Definition 2.** (turning point) *Given a sequence $W$ with its trend sequence $T$, where $t_1 \neq t_2$, $p$ is a turning point if $t_p \neq t_{p+1}$. In other words, at the turning point $p$, $w_p$ is the overlapping element of two neighboring segments in $W$.*

Note that we set $p = 1$ as the turning point of the first segment virtually. In this paper, we discuss two versions of the LWS problem, the LWSt and the LWSr problems in the following.

#### 2.1.1 The Longest Wave Subsequence Problem with Trend

Given two numeric strings (or substrings) $A = \langle a_1, a_2, \ldots, a_n \rangle$ and $B = \langle b_1, b_2, \ldots, b_n \rangle$, let $Rank(a_i, A)$ denote the rank of $a_i$ in $A$ and $Rank(b_i, B)$ denote the rank of $b_i$ in $B$. $A$ and $B$ are said to be *order-preserving* [1–6] if $Rank(a_i, A) = Rank(b_i, B)$, for $1 \le i \le n$. For example, suppose that $A = \langle 24, 31, 42, 40, 26 \rangle$ and $B = \langle 10, 18, 25, 21, 16 \rangle$. Then, we have $Rank(a_i, A)_{i=1,2,\ldots,n} = \langle 1, 3, 5, 4, 2 \rangle$ and $Rank(b_i, B)_{i=1,2,\ldots,n} = \langle 1, 3, 5, 4, 2 \rangle$. Thus, $A$ and $B$ are order-preserving.

Definition 3 describes the *trend-preserving* (TP) relationship between two sequences.

**Definition 3.** (trend-preserving) *Given two numeric sequences $A$ and $B$ with the same length, where the values in each sequence are distinct, $A$ and $B$ are said to be trend-preserving if $A$ and $B$ have the same trend.*

For example, suppose that $A = \langle 28, 31, 42, 40, 26 \rangle$ and $B = \langle 10, 18, 25, 21, 16 \rangle$. $A$ and $B$ have the same trend $T = \langle 0, 1, 1, 0, 0 \rangle$, so they are trend-preserving. However, $A$ and $B$ are not order-preserving since the elements' ranks are not all identical in corresponding positions in $A$ and $B$. For instance, $a_5 = 26$ is the smallest element in $A$, but $b_5 = 16$ is the second smallest element in $B$. It is clear that any two order-preserving sequences are always trend-preserving. On the contrary, two trend-preserving sequences may or may not be order-preserving.

The first version of the LWS problem, the *longest wave subsequence problem with trend*, denoted as LWSt, is defined as follows.

**Definition 4.** (LWSt problem) *Given a numeric sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$ of distinct values and a target trend sequence $T = \langle t_1, t_2, \ldots, t_n \rangle$, with the same length $n$, where $t_i \in \{0, 1\}$ for $1 \le i \le n$, and $t_1 \neq t_2$, the* longest wave subsequence with trend (LWSt) *is a longest subsequence $W = \langle w_1, w_2, \ldots, w_{n'} \rangle$ obtained from $A$ such that $w_{i-1} < w_i$ if $t_i = 1$ and $w_{i-1} > w_i$ if $t_i = 0$, for $2 \le i \le n' \le n$.*

As another view of Definition 4, the LWSt is to find $W$ with the maximal length such that the trend of $W$ is the prefix of $T$. For example, suppose that $A = \langle 4, 6, 1, 7, 3, 5, 8, 2 \rangle$, and $T = \langle 0, 1, 1, 0, 1, 1, 1, 1 \rangle$. The LWSt is $\langle 4, 6, 7, 3, 5, 8 \rangle$, whose trend $\langle 0, 1, 1, 0, 1, 1 \rangle$ is identical to a certain prefix of $T$. In other words, $W$ and the prefix of $T$ are trend-preserving. In addition, when there is only one turning point in the trend sequence, such as $\langle 0, 1, 1, 1, \cdots, \rangle$, the LWSt problem is degenerated into the traditional LIS problem.

#### 2.1.2 The Longest Wave Subsequence Problem within $r$ Segments

**Definition 5.** (LWSr problem) *Given a numeric sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$ of distinct val-*

*ues and the constraint number $r$ of segments, the* longest wave subsequence within $r$ segments (LWSr) *is to find a longest subsequence $W = \langle w_1, w_2, \ldots, w_{n'}\rangle$ obtained from $A$ such that the number of turning points in $W$ is at most $r$.*

In Definition 5, $i$ is a turning point, $i \geq 2$, if and only if $(w_{i-1} - w_i) \times (w_i - w_{i+1}) < 0$, which means that either ($w_{i-1} > w_i$ and $w_i < w_{i+1}$) or ($w_{i-1} < w_i$ and $w_i > w_{i+1}$).

Without loss of generality, it is assumed that the first segment is increasing, that is $w_1 < w_2$. Table 1 shows the LWSr examples with $A = \langle 4, 6, 1, 7, 3, 5, 8, 2\rangle$. When $r = 1$, it is identical to the traditional LIS problem, and the LWSr is $\langle 4, 6, 7, 8\rangle$ or $\langle 1, 3, 5, 8\rangle$ with length 4.

For a given sequence $A$, suppose that the LWSr answers $W$ and $W'$ are required to have exactly $r$ and $r'$ segments, respectively. Then it cannot be guaranteed that $|W| \leq |W'|$ if $r < r'$. For instance, suppose $A = \langle 1, 2, 6, 7, 5, 8, 4, 3\rangle$. When the LWSr is required to have exactly $r = 2$ segments, then the LWSr answer is $\langle 1, 2, 6, 7, 5, 4, 3\rangle$ or $\langle 1, 2, 6, 7, 8, 4, 3\rangle$ with length 7. However, when the LWSr is required to have exactly $r = 3$ segments, then the LWSr answer is $\langle 1, 2, 6, 7, 5, 8\rangle$ with length 6.

## 3 The Algorithm for the Longest Wave Subsequence Problem with Trend

**Definition 6.** (best ending element) [15, 17] *Given a sequence $A$, the best ending element is the last element in an increasing (decreasing) subsequence of $A$ with a certain length, and it is also the minimal (maximal) element.*

We first demonstrate our idea for solving the LWSt problem with the example in Table 2, where the input sequence $A = \langle 4, 6, 1, 7, 3, 5, 8, 2\rangle$ and $T = \langle 0, 1, 1, 0, 1, 1, 1, 1\rangle$. The turning point list is $P = \langle 1, 3, 4\rangle$. The LWSt answer of this example is $\langle 4, 6, 7, 3, 5, 8\rangle$ with length 6.

Each element $S[x]$ in array $S[\cdot]$ is used to store the best ending element in the LWSt answer with length $x$. According to the turning points, the first range for updating $S[\cdot]$ is $S[1..3]$, which is increasing. First, it sets $S[1] = 4$ for $a_1 = 4$. Next, for $a_2 = 6$, it can extend the LWSt length from $S[1] = 4$, hence it is the best ending element with LWSt length 2 and $S[2] = 6$. Then, $a_3 = 1$ is smaller than $S[1] = 4$. It is the new best ending element with LWSt length 1 and it updates $S[1] = 1$. When $a_4 = 7$ is considered, it can extend from $S[2] = 6$. Thus, it is the best ending element with LWSt length 3. We get $S[3] = 7$, where $p_2 = 3$ is a turning point. Hence, the next range for updating $S[\cdot]$ is shifted to $S[3..4]$, which is decreasing, and $S[3] = 7$ is the starting element of the new range.

While scanning $a_5 = 3$, instead of replacing $S[2] = 6$, it extends the first element $S[3] = 7$ in the decreasing segment, to set $S[4] = 3$. In other words, $a_5 = 3$ is currently the best ending element with LWSt length 4. In addition, we encounter the next turning point $p_3 = 4$. Then, the next range for updating $S[\cdot]$ is shifted to $S[4..8]$, which is increasing, and $S[4] = 3$ is the starting element of the new range. Next, $a_6 = 5$ and $a_7 = 8$ are scanned, and $S[5] = 5$ and $S[6] = 8$, are set respectively. Finally, $a_8 = 2$ replaces $S[4] = 3$ to $S[4] = 2$, as the operation in $S[4..8]$ for the traditional LIS problem.

Note that we need only to update $a_8 = 2$ into the current range $S[4..8]$, rather than any former range, described in Theorem 1. The position of $a_i$ in $S[\cdot]$ can be determined by finding the *successor* of $a_i$ in the current range of $S[\cdot]$ with the binary search scheme. The LWSt answer $\langle 4, 6, 7, 3, 5, 8\rangle$ can be obtained by a simple tracing back technique.

**Theorem 1.** *Suppose that we are given an input sequence $A$ and a trend sequence $T$ with $r$ turning points $P = \langle p_1, p_2, \ldots, p_r\rangle$ and $p_{r+1} = \infty$. After $A_{1..i}$ has been processed, suppose that $S[p_k]$ has been updated, but $S[p_{k+1}]$ has not been updated yet, $1 \leq k \leq r$. Then, when $a_{i+1}$ is processed, the update of $S[p_k..p_{k+1}]$ is sufficient to store the correct best ending elements in $S[p_k..p_{k+1}]$.*

*Proof.* Omitted here. □

By Theorem 1, maintaining only the current range can solve the LWSt problem. Our formal algorithm for solving the LWSt problem is omitted here.

**Theorem 2.** *The LWSt problem can be solved in $O(n \log n)$ time and $O(n)$ space.*

*Proof.* Omitted here. □

## 4 The Algorithm for the Longest Wave Subsequence Problem within $r$ Segments

Without loss of generality, it is assumed that the first segment in the LWSr problem is increas-

Table 1: The LWSr answers of $A = \langle 4, 6, 1, 7, 3, 5, 8, 2\rangle$ for $r = 1$, 2, or 3.

| $r$ | LWSr | length |
|---|---|---|
| 1 | $\langle 4, 6, 7, 8\rangle$, $\langle 1, 3, 5, 8\rangle$ | 4 |
| 2 | $\langle 4, 6, 7, 3, 2\rangle$, $\langle 4, 6, 7, 5, 2\rangle$, $\langle 4, 6, 7, 8, 2\rangle$, $\langle 1, 3, 5, 8, 2\rangle$ | 5 |
| 3 | $\langle 4, 6, 1, 3, 5, 8\rangle$, $\langle 4, 6, 7, 3, 5, 8\rangle$ | 6 |

Table 2: An example for the LWSt problem for an input sequence $A = \langle 4, 6, 1, 7, 3, 5, 8, 2\rangle$ and a trend sequence $T = \langle 0, 1, 1, 0, 1, 1, 1, 1\rangle$, whose LWSt answer is $\langle 4, 6, 7, 3, 5, 8\rangle$ with length 6.

| $T$ | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| $A$ \ $S$(length) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 4 | **4** | | | | | | | |
| 6 | 4 | **6** | | | | | | |
| 1 | **1** | 6 | | | | | | |
| 7 | 1 | 6 | **7** | | | | | |
| 3 | 1 | 6 | 7 | **3** | | | | |
| 5 | 1 | 6 | 7 | 3 | **5** | | | |
| 8 | 1 | 6 | 7 | 3 | 5 | **8** | | |
| 2 | 1 | 6 | 7 | **2** | 5 | 8 | | |

ing. The LWSr problem is obviously degenerated into the traditional LIS problem when $r = 1$. When $r = 2$, the answer is divided into two parts by the turning point. The problem can be solved by applying the traditional LIS algorithm forward and backward, respectively, to get two parts of the answer. We find the best turning point $i$, and then combine the LIS length of $A_{1..i}$ and the LDS length of $A_{i+1..n}$.

When $r$ is equal to three or more, the decision of the turing points would become more complicated, and it is obviously not a good idea to generalize the above algorithm with $r = 2$.

In our LWSr algorithm, we denote a 2-tuple $(e, l)$ as the solution status, where $e$ denotes the best ending element and $l$ represents its LWSr length. For example, (7, 4) represents an LWSr of length 4 which ends with element 7.

For solving the LWSr problem, we use $r$ priority queues for storing these 2-tuple elements. Each queue $Q_j$ corresponds to segment $j$, where $1 \leq j \leq r$. For an increasing segment $j$ ($j$ is odd), $Q_j$ is a min-priority queue. For a decreasing segment $j$ ($j$ is even), $Q_j$ is a max-priority queue. In a min-priority (max-priority) queue, the first 2-tuple is the element with the minimum (maximum) $e$ value in the queue. In other words, the 2-tuple elements are stored in the priority quese based on their $e$ values.

In a priority queue, there are two common operations, *Predecessor* and *Successor*. $Predecessor(Q, x)$ gives the previous element of $x$ in $Q$. On the other hand, $Successor(Q, x)$ gives the next element of $x$ in $Q$. They have opposite effects between a min-priority queue and a max-priority queue. For example, given a min-priority queue $Q = \{(1, 3), (3, 4)$ and $(8, 6)\}$. If $Successor(Q, 2)$ is invoked, it would obtain (3, 4), since 3 of 2-tuple (3, 4) is the smallest element which is greater than 2. It also gets (3, 4) by invoking $Predecessor(Q, 2)$ in a max-priority queue $Q$.

**Definition 7.** *For any 2-tuples* $(e_1, l_1)$, $(e_2, l_2) \in Q_j$, $(e_1, l_1) \neq (e_2, l_2)$, *we say that* $(e_1, l_1)$ dominates $(e_2, l_2)$ *in a min-priority* (*max-priority*) *queue* $Q_j$, *if* $e_1 \leq e_2$ *and* $l_1 \geq l_2$ ($e_1 \geq e_2$ *and* $l_1 \geq l_2$). *Any pair of 2-tuples in* $Q_j$ *cannot dominate each other.*

Table 3 illustrates an example of our concept for solving the LWSr problem with $r = 3$ for $A = \langle 4$, 6, 1, 7, 3, 5, 8, 2$\rangle$. The LWSr answer is $\langle 4$, 6, 7, 3, 5, 8$\rangle$ with length 6. The underlined elements indicate the updates. We first insert (4, 1) into $Q_1$ to $Q_3$. It means that there exists an LWSr of length 1 within one segment, two segments and three segments, respectively. The ending element is 4.

By considering $a_2 = 6$, it would get nothing for invoking $Successor(Q_1, 6)$, which is in a min-priority queue. Note that the first segment is increasing and the second segment is decreasing. It means that 6 can be extended from (4, 1) in segment 1. Thus, we insert (6, 2) into $Q_1$ and we get $Q_1 = \{(4, 1), (6, 2)\}$. In $Q_2$, we invoke $Successor(Q_2, 6) = (4, 1)$, whose value 4 is smaller than 6, and (4, 1) should be replaced. That is, we want to insert (6, 1) into $Q_2$. However,we have an alternative choice by copying (6, 2) from $Q_1$ to $Q_2$, which is (6, 2) and it dominates (6, 1). That is, in a decreasing segment 2, (6, 2) has higher potential than (6, 1) for developing better answer. So, (6, 1) is discarded, and (6, 2) is inserted into $Q_2$. Now, $Q_2 = \{(4, 1), (6, 2)\}$. At this moment, (6, 2) dominates (4, 1). Then, (4, 1) is removed from $Q_2$ and we get $Q_2 = \{(6, 2)\}$. As for $Q_3$, (6, 2) is also

 ISBN/978-957-437-895-1(PDF)

Table 3: An example of the LWSr algorithm within $r = 3$ segments for $A = \langle 4, 6, 1, 7, 3, 5, 8, 2\rangle$.

| length | | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $a_1 = 4$ | $Q_1$ | **(4, 1)** | | | | | |
| | $Q_2$ | **(4, 1)** | | | | | |
| | $Q_3$ | **(4, 1)** | | | | | |
| $a_2 = 6$ | $Q_1$ | (4, 1) | **(6, 2)** | | | | |
| | $Q_2$ | ~~(4, 1)~~ | **(6, 2)** | | | | |
| | $Q_3$ | (4, 1) | **(6, 2)** | | | | |
| $a_3 = 1$ | $Q_1$ | ~~(4, 1)~~ **(1, 1)** | (6, 2) | | | | |
| | $Q_2$ | | (6, 2) | **(1, 3)** | | | |
| | $Q_3$ | ~~(4, 1)~~ | ~~(6, 2)~~ | **(1, 3)** | | | |
| $a_4 = 7$ | $Q_1$ | (1, 1) | (6, 2) | **(7, 3)** | | | |
| | $Q_2$ | | ~~(6, 2)~~ | ~~(1, 3)~~ **(7, 3)** | | | |
| | $Q_3$ | | | (1, 3) | **(7, 4)** | | |
| $a_5 = 3$ | $Q_1$ | (1, 1) | ~~(6, 2)~~ **(3, 2)** | (7, 3) | | | |
| | $Q_2$ | | | (7, 3) | **(3, 4)** | | |
| | $Q_3$ | | | (1, 3) | ~~(7, 4)~~ **(3, 4)** | | |
| $a_6 = 5$ | $Q_1$ | (1, 1) | (3, 2) | ~~(7, 3)~~ **(5, 3)** | | | |
| | $Q_2$ | | | (7, 3) | ~~(3, 4)~~ **(5, 4)** | | |
| | $Q_3$ | | | (1, 3) | (3, 4) | **(5, 5)** | |
| $a_7 = 8$ | $Q_1$ | (1, 1) | (3, 2) | (5, 3) | **(8, 4)** | | |
| | $Q_2$ | | | ~~(7, 3)~~ | ~~(5, 4)~~ **(8, 4)** | | |
| | $Q_3$ | | | (1, 3) | (3, 4) | (5, 5) | **(8, 6)** |
| $a_8 = 2$ | $Q_1$ | (1, 1) | ~~(3, 2)~~ **(2, 2)** | (5, 3) | (8, 4) | | |
| | $Q_2$ | | | | (8, 4) | **(2, 5)** | |
| | $Q_3$ | | | (1, 3) | ~~(3, 4)~~ | ~~(5, 5)~~ **(2, 5)** | (8, 6) |

copied and inserted into $Q_3$. But, (4, 1) and (6, 2) cannot dominate each other, since they are in an increasing segment. Thus, $Q_3 = \{(4, 1), (6, 2)\}$

For the next input element $a_3 = 1$, we find $Successor(Q_1, 1) = (4, 1)$. Hence, we insert (1, 1) into $Q_1$, and remove (4, 1) from $Q_1$, since (4, 1) is dominated by (1, 1). Next, we get nothing by invoking $Successor(Q_2, 1)$. It means that 1 can be extended from (6, 2). Hence, (1, 3) is inserted into $Q_2$. Besides, (1, 3) is also inserted into $Q_3$, because the copied element (1, 3) from $Q_2$ dominates (1, 1) obtained by $Successor(Q_3, 1)$. In addition, the dominated elements (4, 1) and (6, 2) are removed from $Q_3$.

**Theorem 3.** *The LWSr problem can be solved in $O(rn \log n)$ time and $O(rn)$ space.*

*Proof.* Omitted here. □

## 5 Conclusion and Discussion

The LWSr problem may be used to get the information of a series of stock prices in a stock market. Sometimes we may consider the price trend relationship between two or more stocks. Thus, we may study the *longest common wave subsequence within r segments* (LCWSr) problem in the future. As another research direction, in the LWSr problem, the length of each segment in the answer may be bounded with an upper bound or a lower bound, or both bounds.

## References


[1] T. Chhabra, J. Tarhio, A filtration method for order-preserving matching, Information Processing Letters 116 (2) (2016) 71–74.

[2] S. Cho, J. C. Na, K. Park, J. S. Sim, A fast algorithm for order-preserving pattern matching, Information Processing Letters 115 (2) (2015) 397–402.

[3] P. Gawrychowski, P. Uznański, Order-preserving pattern matching with k mismatches, Theoretical Computer Science 638 (2016) 136–144.

[4] M. M. Hasana, A. S. M. S. Islama, M. S. Rahmana, M. S. Rahman, Order preserving pattern matching revisited, Pattern Recognition Letters 55 (2015) 15–21.

[5] J. Kim, P. Eades, R. Fleischer, S.-H. Hong, C. S. Iliopoulos, K. Park, S. J. Puglisi, T. Tokuyama, Order-preserving matching, Theoretical Computer Science 525 (2014) 68–79.

[6] M. Kubica, T. Kulczyński, J. Radoszewski, W. Rytter, T. Waleń, A linear time algorithm for consecutive permutation pattern matching, Information Processing Letters 113 (12) (2013) 430–433.

[7] M. R. Alam, M. S. Rahman, A divide and conquer approach and a work-optimal parallel algorithm for the lis problem, Information Processing Letters 113 (13) (2013) 470–476.

[8] M. H. Albert, A. Golynski, A. M. Hamel, A. López-Ortiz, S. Rao, M. A. Safari, Longest increasing subsequences in sliding windows, Theoretical Computer Science 321 (2-3) (2004) 405–414.

[9] S. Bespamyatnikh, M. Segal, Enumerating longest increasing subsequences and patience sorting, Information Processing Letters 76 (1-2) (2000) 7–11.

[10] M. Crochemore, E. Porat, Fast computation of a longest increasing subsequence and application, Information and Computation 208 (9) (2010) 1054–1059.

[11] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, S. L. Salzberg, Alignment of whole genomes, Nucleic Acids Research 27 (11) (1999) 2369–2376.

[12] M. L. Fredman, On computing the length of longest increasing subsequences, Discrete Mathematics 11 (1) (1975) 29–35.

[13] J. W. Hunt, T. G. Szymanski, A fast algorithm for computing longest common subsequences, Communications of the ACM 20 (1977) 350–353.

[14] S.-F. Lo, K.-T. Tseng, C.-B. Yang, K.-S. Huang, A diagonal-based algorithm for the longest common increasing subsequence problem, Theoretical Computer Science 815 (2020) 69–78.

[15] U. Manber, Introduction to Algorithms: A Creative Approach, 1st Edition, Addison-Wesley, Boston, USA, 1989.

[16] C. Schensted, Longest increasing and decreasing subsequences, Canadian Journal of Mathematics 13 (1961) 179–191.

[17] I.-H. Yang, C.-P. Huang, K.-M. Chao, A fast algorithm for computing a longest common increasing subsequence, Information Processing Letters 93 (5) (2005) 249–253.