# The Knuth-Yao Quadrangle-Inequality Speedup is a Consequence of Total-Monotonicity

Wolfgang W. Bein * Mordecai J. Golin † Lawrence L. Larmore ‡ Yan Zhang §

**Abstract**

There exist several general techniques in the literature for speeding up naive implementations of dynamic programming. Two of the best known are the Knuth-Yao quadrangle inequality speedup and the SMAWK algorithm for finding the row-minima of totally monotone matrices. Although both of these techniques use a quadrangle inequality and seem similar they are actually quite different and have been used differently in the literature.

In this paper we show that the Knuth-Yao technique is actually a direct consequence of total monotonicity. As well as providing new derivations of the Knuth-Yao result, this also permits showing how to solve the Knuth-Yao problem directly using the SMAWK algorithm. Another consequence of this approach is a method for solving *online* versions of problems with the Knuth-Yao property. The online algorithms given here are asymptotically as fast as the best previously known static ones. For example the Knuth-Yao technique speeds up the standard dynamic program for finding the optimal binary search tree of $n$ elements from $\Theta(n^3)$ down to $O(n^2)$, and the results in this paper allow construction of an optimal binary search tree in an online fashion (adding a node to the left or right of the current nodes at each step) in $O(n)$ time per step.

We conclude by discussing how the general technique described here is also applicable to later extensions of the Kuth-Yao result, such as those developed Borchers and Gupta.

*Department of Computer Science, University of Nevada, Las Vegas, NV 89154. Email: bein@cs.unlv.edu.
Research supported by NSF grant CCR-0312093.

†Dept. of Computer Science, Hong Kong UST, Clear Water Bay, Kowloon, Hong Kong. Email golin@cs.ust.hk
Research partially supported by Hong Kong RGC CERG grant HKUST6312/04E.

‡Department of Computer Science, University of Nevada, Las Vegas, NV 89154. Email: larmore@cs.unlv.edu.
Research supported by NSF grant CCR-0312093.

§Dept. of Computer Science, Hong Kong UST, Clear Water Bay, Kowloon, Hong Kong. Email: cszy@cs.ust.hk
Research partially supported by Hong Kong RGC CERG grant HKUST6312/04E.

# 1 Introduction

## 1.1 History

The construction of optimal binary search trees is a classic optimization problem. The input is $2n+1$ weights (probabilities) $p_1, \ldots, p_n, q_0, q_1 \ldots, q_n$; $p_i$ is the weight that a search is for $\text{Key}_i$; such a search is called *successful*. The value $q_i$ is the weight that the search argument is *unsuccessful* and is for an argument between $\text{Key}_i$ and $\text{Key}_{i+1}$ (where we set $\text{Key}_0 = -\infty$ and $\text{Key}_{n+1} = \infty$).

Our problem is to find an *optimal binary search tree* (OBST) with $n$ internal nodes – corresponding to successful searches – and $n+1$ leaves – corresponding to unsuccessful searches – that minimizes the average search time. Let $d(p_i)$ be the depth of internal node corresponding to $p_i$ and $d(q_i)$ the depth of leaf corresponding to $q_i$. Then we want to find a tree that minimizes

$$\sum_{1 \le j \le n} p_j(1 + d(p_j)) + \sum_{0 \le k \le n} q_k\, d(q_k).$$

It is not hard to see that this problem reduces to solving the following recurrence:

$$B_{i,j} = \begin{cases} 0 & \text{if } i = j \\ \sum_{t=i+1}^{j} p_i + \sum_{t=i}^{j} q_j + \min_{i<t\le j} \{B_{i,t-1} + B_{t,j}\} & \text{if } i < j \end{cases} \tag{1}$$

where the cost of the optimal OBST is $B_{0,n}$. The naive way of calculating $B_{i,j}$ requires $\Theta(j-i)$ time, so calculating all of the $B_{i,j}$ would seem to require $\Theta(n^3)$ time. In fact, this is what was done in by Gilbert and Moore in 1956 [8]. More than a decade later, in 1971, it was noticed by Knuth [9] that, using a complicated amortization argument, the $B_{i,j}$ can all be computed using only $\Theta(n^2)$ time. Around another decade later, in the early 1980s, Yao [15, 16] simplified Knuth's proof and, in the process, showed that this *dynamic programming speedup* worked for a large class of problems satisfying a *quadrangle inequality* property.

Many other authors then used the Knuth-Yao technique, either implicitly or explicitly, to speed up different dynamic programming problems. See *e.g.*, [13, 3, 4].

In the 1980s a variety of researchers developed various related techniques for exploiting properties, such as convexity and concavity, to yield dynamic programming speedups; a good early survey is [7]. A high point of this strand of research was the development in the late 1980s of the linear time SMAWK algorithm [1] for finding the row-minima of totally monotone matrices. The work in [6] provides a good survey of the techniques mentioned as well as applications and later extensions. One particular extension we mention (since we will use it later) is the LARSCH algorithm of Larmore and Schieber [10] which, in some cases, permits finding row-minima even when entries of the matrix can implicitly depend upon other entries in the matrix (a case SMAWK cannot handle).

As we shall soon see, both the Knuth-Yao (KY) and SMAWK techniques rely on an underlying quadrangle inequality in their structure and have a similar "feel". In spite of this, they have until usually been thought of as being different approaches. See, *e.g.*, [12] which uses *both* KY and SMAWK to speed up different problems. In [2] Aggarwal and Park demonstrated a relationship between the KY problem and totally-monotone matrices by building a *3-D monotone matrix* based on the KY problem and then using an algorithm due to Wilber [14] to find *tube* minima in that 3-D matrix. They left as an open question the possibility of using SMAWK directly to solve the KY problem.

The main theoretical contribution of this paper is to show that the KY technique is really just a special case of the use of totally monotone matrices. We first show a direct solution to the KY

problem by decomposing it into $O(n)$ totally-monotone $O(n) \times O(n)$ matrices, permitting direct application of the SMAWK algorithm to yield another $O(n^2)$ solution. After that we describe how the Knuth-Yao technique itself is actually a direct consequence of total-monotonicity of certain related matrices. Finally, we show that problems which can be solved by the KY technique statically in $O(n^2)$ time can actually be solved in an online manner using only $O(n)$ worst case time per step. This is done by using a new formulation of the problem in terms of monotone-matrices, along with the LARSCH algorithm.[1]

### 1.2 Definitions

**Definition 1** *A two dimensional upper triangular array $a(i,j)$, $1 \leq i \leq j \leq n$ satisfies a* quadrangle inequality (QI) *if*

$$a(i,j) + a(i',j') \leq a(i',j) + a(i,j') \quad \textit{for} \quad i \leq i' \leq j \leq j'.$$

Note: In some applications we will write $a_{i,j}$ instead of $a(i,j)$.

**Definition 2** *A $2 \times 2$ matrix is* monotone *if the minimum of the upper row is not to the right of the minimum of the lower row. More formally, $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is* monotone *if $b < a$ implies that $d < c$ and $b = a$ implies that $d \leq c$.*

*A 2-dimensional matrix $M$ is* totally monotone *if every $2 \times 2$ submatrix of $M$ is monotone.*

**Definition 3** *A $2 \times 2$ matrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is* Monge *if $a + d \leq b + c$.*

*A 2-dimensional matrix $M$ is Monge if every $2 \times 2$ submatrix of $M$ is Monge.*

The important observations (all of which can be found in [6]) are

**Observation 1** *An $m \times n$ matrix $M$ is Monge if, for all $1 \leq i < m$ and $1 \leq j < n$,*

$$M(i,j) + M(i+1,j+1) \leq M(i,j+1) + M(j+1,i) \tag{2}$$

**Observation 2** *Every Monge matrix is totally monotone.*

Combining the above leads to the test that we will often use:

**Observation 3** *Let $M$ be an $m \times n$ matrix. $M$ is totally monotone, if for all $1 \leq i < m$ and $1 \leq j < n$,*

$$M(i,j) + M(i+1,j+1) \leq M(i,j+1) + M(j+1,i). \tag{3}$$

### 1.3 Mathematical Framework

Even though both the SMAWK algorithm [1] and the Knuth-Yao (KY) speedup [9, 15, 16] use an implicit quadrangle inequality in their associated matrices, on second glance, they seem quite different from each other.

In the SMAWK technique, the quadrangle inequality is on the entries of a given $m \times n$ *input matrix,* which can be any totally monotone matrix.[2] It is not necessary for the input matrix to

[1] We should point out that, as discussed in more detail at the end of Section 3, an alternative online algorithm to the one presented here could be derived by careful deconstruction of the static Aggarwal-Park [2] method; somehow, this never seems to have been remarked before in the literature.

[2] Note that Monge Matrices satisfy a quadrangle inequality, but in general, a totally monotone matrix may not. However, in practice, most applications of the SMAWK algorithm make use of Monge matrices. If the input matrix is triangular, the missing entries are assigned the default value $\infty$, preserving total monotonicity.

actually be given.All that the SMAWK algorithm requires is that, when needed, individual entries can be calculated in $O(1)$ (amortized) time. The *output* of the SMAWK algorithm is a vector containing the row-minima of the input matrix. If $m \leq n$, the SMAWK algorithm outputs this vector in $O(n)$ time, an order of magnitude speedup of the naive algorithm that scans all $mn$ matrix entries.

The KY technique, by contrast, uses a quadrangle inequality in the upper-triangular $n \times n$ matrix $B_{i,j}$. That is, it uses the QI property of its *result matrix* to speed up the evaluation, via dynamic programming, of the entries in the same result matrix.

More specifically, Yao's result [15] was formulated as follows: For $1 \leq i \leq j \leq n$ let $w(i,j)$ be a given value and

$$B_{i,j} = \begin{cases} 0 & \text{if } i = j \\ w(i,j) + \min_{i<t\leq j} \{B_{i,t-1} + B_{t,j}\} & \text{if } i < j \end{cases} \tag{4}$$

**Definition 4** *$w(i,j)$ is* monotone in the lattice of intervals *if $[i,j] \subseteq [i',j]$ implies $w(i,j) \leq w(i',j')$.*

As an example, it is not difficult to see that the $w(i,j) = \sum_{t=i+1}^{j} p_i + \sum_{t=i}^{j} q_j$ of the BST recurrence (1) satisfies the quadrangle inequality and is monotone in the lattice of intervals.

**Definition 5** *Let*

$$K_B(i,j) = \max\{t \,:\, w(i,j) + B_{i,t-1} + B_{t,j} = B_{i,j}\},$$

*i.e., the largest index which achieves the minimum in (4).*

Yao then proves two Lemmas (see Figure 1 for an example):

**Lemma 1** (Lemma 2.1 in [15])
*If $w(i\,j)$ satisfies the quadrangle inequality as defined in Definition 1, and is also monotone on the lattice of intervals, then the $B_{i,j}$ defined in (4) also satisfy the quadrangle inequality.*

**Lemma 2** (Lemma 2.2 in [15])
*If the function defined in (4) satisfies the quadrangle inequality then*

$$K_B(i,j) \leq K_B(i,j+1) \leq K_B(i+1,j+1) \quad \textit{for} \quad i < j$$

Lemma 1 proves that a QI in the $w(i,j)$ implies a QI in the $B_{i,j}$. Suppose then that we evaluate the values of the $B_{i,j}$ in the order $d = 1, 2, \ldots, n$, where, for each fixed $d$, we evaluate all of $B_{i,i+d}$, $i = 0, 1, n-d$. Then Lemma 2 says that $B_{i,i+d}$ can be evaluated in time $O(K_B(i+1,i+d) - K_B(i,i+d-1))$. Note that

$$\sum_{i=0}^{n-d} (K_B(i+1,i+d) - K_B(i,i+d-1)) \leq K_B(n-d+1,n) \leq n$$

and thus all entries for fixed $d$ can be calculated in $O(n)$ time. Summing over all $d$, we see that all $B_{i,j}$ can be obtained in $O(n^2)$ time.

As mentioned, Lemma 2 and the resultant $O(n^2)$ running time have usually been viewed as unrelated to the SMAWK algorithm. While they seem somewhat similar (a QI leading to an order of magnitude speedup) they appeared not to be directly connected.

The main theoretical result of this paper is the observation that if the $w(i,j)$ satisfy the QI and are monotone in the lattice of intervals, then the $B_{i,j}$ defined by (4) can be derived as the row-minima of a sequence of $O(n)$ different totally monotone matrices, each of size $O(n) \times O(n)$, where the entries in a matrix depend upon the row-minima of previous matrices in the sequence. In fact, we will show three totally different decomposition of the $B_{i,j}$ into $O(n)$ totally monotone matrices. In particular, our first decomposition, will permit the direct use of SMAWK.

### 1.4 Online Algorithms

Generally, an online problem is defined to be a problem where a stream of outputs must be generated in response to a stream of inputs, and where those responses must be given under a protocol which requires some outputs be given before all inputs are known.

The online versions of the problems in which we are interested are given below. Our goal is to achieve the optimal result, while maintaining the same asymptotic time complexity as the offline versions.

Let $L \leq R$ be given along with values $w(i,j)$ for all $L \leq i \leq j \leq R$ that satisfy the QI and the "monotone on lattice of intervals" property. Let

$$B_{i,j} = \begin{cases} 0 & \text{if } i = j \\ w(i,j) + \min_{i<t\leq j} \{B_{i,t-1} + B_{t,j}\} & \text{if } i < j \end{cases}$$

and assume all $B_{i_j}$ for $L \leq i \leq j \leq R$ have already been calculated and stored.

The **Right-online** problem is:
Given new values $w(i, R+1)$ for $L \leq i \leq R+1$, such that $w(i,j)$ still satisfy the QI and lattice property, calculate all of the values $B_{i,R+1}$ for $L \leq i \leq R+1$.

The **Left-online** problem is:
Given new values $w(L-1, j)$ for $L-1 \leq j \leq R$, such that $w(i,j)$ still satisfy the QI and lattice property, calculate all of the values $B_{L-1,j}$ for $L-1 \leq j \leq R$.

These online problems restricted to the optimal binary search tree would be to construct the OBST for items $\text{Key}_L, \ldots, \text{Key}_R$, and, at each step, add either $\text{Key}_{R+1}$, a new key to the right, or $\text{Key}_{L-1}$, a new key to the left. Every time a new element is added, we want to update the $B_{i,j}$ (dynamic programming) table and thereby construct the optimal binary search tree of the new full set of elements. (See Figure 1.) To achieve this, it is certainly possible to *recompute* the entire table; however this comes at the price of $O(n^2)$ time, where $n = R - L + 1$ is the number of keys currently in the table. What we are interested in here is the question of how one can handle a new key where the extra computational work is neutral to the overall complexity of the problem, *i.e.*, a new key can be added in linear time. Our goal is an algorithm in which a sequence of $n$ online key insertions will result in a worst case $O(n)$ per step to maintain an optimal tree, yielding an overall run time of $O(n^2)$.

Unfortunately, the KY speedup *cannot* be used to do this. The reason that the speedup fails is that the KY speedup is actually an amortization over the evaluation of all entries when done in a particular order. In the online case, adding a new item $n$ to previously existing items $1, 2, \ldots, n-1$, requires using (4) to compute the $n$ new entries $B_{i,n}$, in the fixed order $i = n, n-1, \ldots, 1, 0$ and it is not difficult to construct an example in which calculating these new entries in this order using (4) requires $\Theta(n^2)$ work.

We will see later that the decomposition given in section 3 permits a fully online algorithm with no penalty in performance, *i.e.*, after adding the $n^{\text{th}}$ new key, the new $B_{i,j}$ can be calculated in $O(n)$ worst case time. Furthermore, this will be true for both the left-online and right-online case.

| | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|
| 3 | 0 | 91 | 282 | 499 | 821 |
| 4 | | 0 | 169 | 386 | 686 |
| 5 | | | 0 | 124 | 348 |
| 6 | | | | 0 | 155 |
| 7 | | | | | 0 |

| | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|
| 3 | 3 | 4 | 5 | 5 | 6 |
| 4 | | 4 | 5 | 5 | 6 |
| 5 | | | 5 | 6 | 7 |
| 6 | | | | 6 | 7 |
| 7 | | | | | 7 |

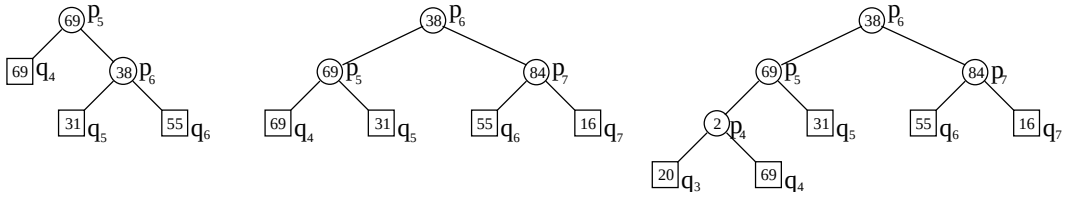


Figure 1: An example of the online case for optimal binary search trees where $(p_4, p_5, p_6, p_7) = (2, 69, 38, 84)$ and $(q_3, q_4, q_5, q_6, q_7) = (20, 69, 31, 55, 16)$. The leftmost table contains the $B_{i,j}$ values; the rightmost one, the $K_B(i, j)$ values. The unshaded entries in the table are for the problem restricted to only keys $5, 6$. The dark gray cells are the entries added to the table when key 7 is added to the right. The light gray cells are the entries added when key 4 is added to the left. The corresponding optimal binary search trees are also given, where circles correspond to successful searches and squares to unsuccessful ones. The values in the nodes are the weights of the nodes (not their keys).

## 2 The First Decomposition

**Definition 6** *For $1 \le d < n$ define the $(n - d + 1) \times (n + 1)$ matrix $D^d$ by*

$$D^d_{i,j} = \begin{cases} w(i, i+d) + B_{i,j-1} + B_{j,i+d} & \text{if } 0 \le i < j \le i + d \le n, \\ \infty & \text{otherwise.} \end{cases} \tag{5}$$

Figure 2 illustrates the first decomposition. Note that (4) immediately implies

$$B_{i,i+d} = \min_{0 \le j \le n} D^d_{i,j} \tag{6}$$

so finding the row-minima of $D^d$ yields $B_{i,i+d}$, $i = 0, \ldots, n - d$. Put another way, the $B_{i,j}$ entries on diagonal $j - i = d$ are exactly the row-minima of matrix $D^d$.

**Lemma 3** *If the function $B_{i,j}$ defined in (4) satisfies the QI then, for each $d \le n$, $D^d$ is a totally monotone matrix*

*Proof*: From Observation 3 it suffices to prove that

$$D^d_{i,j} + D^d_{i+1,j+1} \le D^d_{i+1,j} + D^d_{i,j+1} \tag{7}$$

Note that if $i < j \le j + d$, then from Lemma 1,

$$B_{i,j-1} + B_{i+1,j} \le B_{i+1,j-1} + B_{i,j} \tag{8}$$

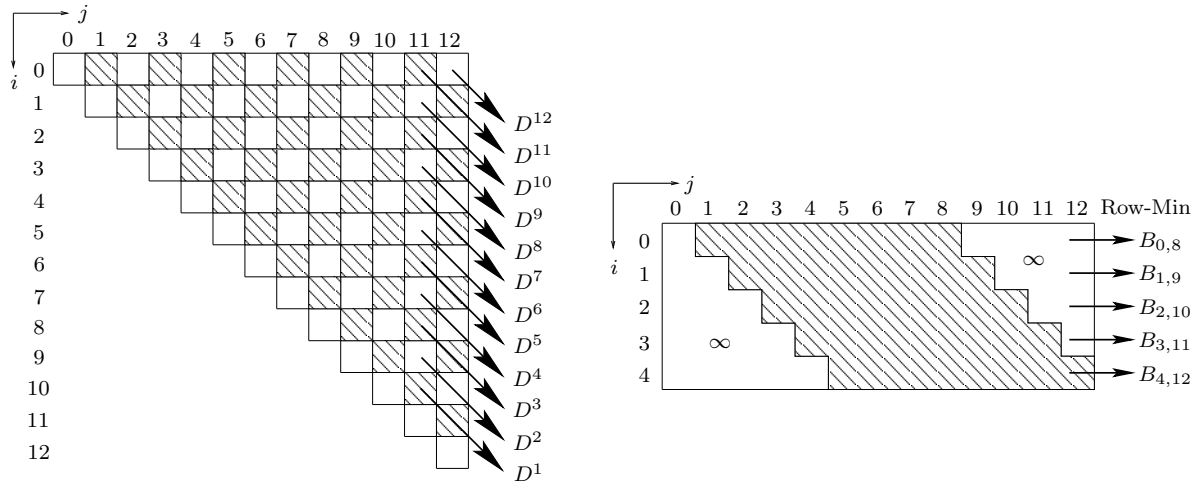


Figure 2: The lefthand figure shows the $B_{i,j}$ matrix for $n = 12$. Each diagonal, $d = i - j$, in the matrix will correspond to a totally monotone matrix $D^d$. The minimal item of row $i$ in $D^d$ will be the value $B_{i,i+d}$. The righthand figure shows $D^8$.

and

$$B_{j,i+d} + B_{j+1,i+1+d} \leq B_{j,i+1+d} + B_{j+1,i+d}. \tag{9}$$

Thus,

$$\begin{aligned} D^d_{i,j} + D^d_{i+1,j+1} &= [w(i,i+d) + B_{i,j-1} + B_{j,i+d}] + [w(i+1,i+d+1) + B_{i+1,j} + B_{j+1,i+1+d}] \\ &= w(i,i+d) + w(i+1,i+d+1) + [B_{i,j-1} + B_{i+1,j}] + [B_{j,i+d} + B_{j+1,i+1+d}] \\ &\leq w(i,i+d) + w(i+1,i+d+1) + [B_{i+1,j-1} + B_{i,j}] + [B_{j,i+d+1} + B_{j+1,i+d}] \\ &= [w(i+1,i+d+1) + B_{i+1,j-1} + B_{j,i+1+d}] + [w(i,i+d) + B_{i,j} + B_{j+1,i+d}] \\ &= D^d_{i+1,j} + D^d_{i,j+1} \end{aligned}$$

and (7) is correct (where we note that the right hand side is $\infty$ if $i \not< j$ or $j \not\leq i + d$). □

**Lemma 4** *Assuming that all of the row-minima of $D^1, D^2, \ldots, D^{d-1}$ have already been calculated, all of the row-minima of $D^d$ can be calculated using the SMAWK algorithm in $O(n)$ time.*

*Proof*: From the previous lemma, $D^d$ is a totally monotone matrix. Also, by definition, its entries can be calculated in $O(1)$ time, using the previously calculated row-minima of $D^{d'}$ where $d' < d$. Thus SMAWK can be applied. □

Combined with (6) this immediately gives a new $O(n^2)$ algorithm for solving the KY problem; just run SMAWK on the $D^d$ in the order $d = 1, 2, \ldots, n-1$ and report all of the row-minima.

We point out that this technique cannot help us solve the online problem as defined in subsection 1.4, though. To see why, suppose that items $1, \ldots, n$ have previously been given, new item $n+1$ has just been added, and we need to calculate the values $B_{i,n+1}$ for $i = 0, \ldots, n$. In our formulation this would correspond to *adding a new bottom row to every matrix $D^d$ and creating a new matrix $D^{n+1}$.* In our formulation, we would need to find the row-minima of all of the $n$ new bottom rows. Unfortunately, the SMAWK algorithm only works on the rows of matrices all at once and cannot help to find the row-minima of a single new row.

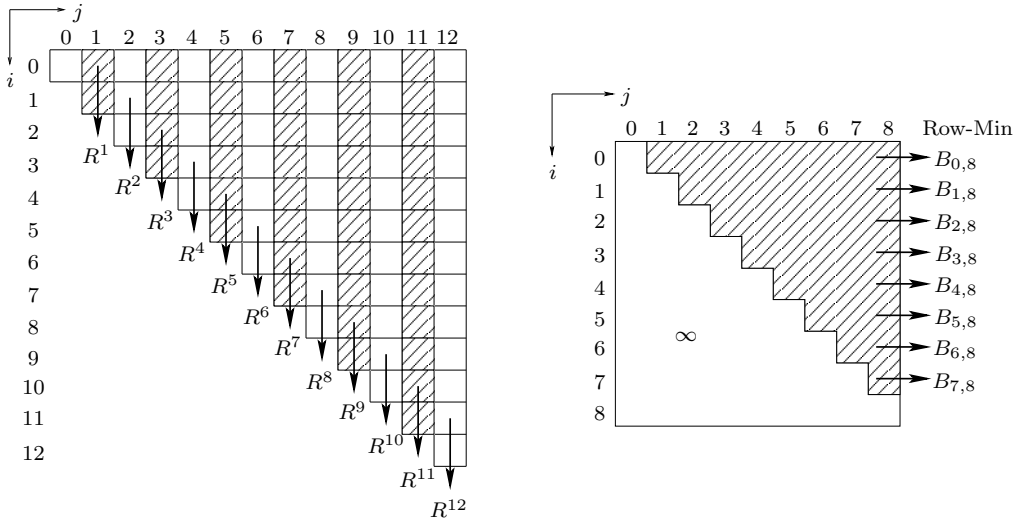


Figure 3: The lefthand figure shows the $B_{i,j}$ matrix for $n = 12$. Each column in the $B_{i,j}$ matrix will correspond to a totally monotone matrix $R^m$. The minimal element of row $i$ in $R^m$ will be the value $B_{i,m}$. The righthand figure shows $R^8$.

# 3 The Second & Third Decompositions

So far we have seen that it is possible to derive the KY *running time* via repeated calls to the SMAWK algorithm. We now see two more decompositions into totally-monotone matrices. These decompositions will trivially imply Lemma 2 (Lemma 2.1 in [15]), which is the basis of the KY speedup. Thus, the KY speedup is just a consequence of total-monotonicity. These new decompositions will also permit us to efficiently solve the online problem given in subsection 1.4.

The second decomposition is indexed by the *rightmost element* seen so far. See Figure 3.

**Definition 7** *For $1 \le m \le n$ define the $(m+1) \times (m+1)$ matrix $R^m$ by*

$$R^m_{i,j} = \begin{cases} w(i,m) + B_{i,j-1} + B_{j,m} & \text{if } 0 \le i < j \le m, \\ \infty & \text{otherwise.} \end{cases} \tag{10}$$

Note that (4) immediately implies

$$B_{i,m} = \min_{0 \le j \le m} R^m_{i,j} \tag{11}$$

so finding the row-minima of $R^m$ yields $B_{i,m}$ for $i = 0, \ldots, m$. Put another way, the $B_{i,j}$ entries in column $m$ are exactly the row minima of $R^m$.

The third decomposition is similar to the second except that it is indexed by the *leftmost element* seen so far. See Figure 4.

**Definition 8** *For $0 \le m < n$ define the $(n-m) \times (n-m)$ matrix $L^m$ by*

$$L^m_{j,i} = \begin{cases} w(m,j) + B_{m,i-1} + B_{i,j} & \text{if } m < i \le j \le n, \\ \infty & \text{otherwise.} \end{cases} \tag{12}$$

(For convenience, we set the row and column indices to run from $(m+1) \ldots n$ and not $1 \ldots (n-m)$.) Note that (4) immediately implies

$$B_{m,j} = \min_{m < i \le j \le n} L^m_{j,i} \tag{13}$$

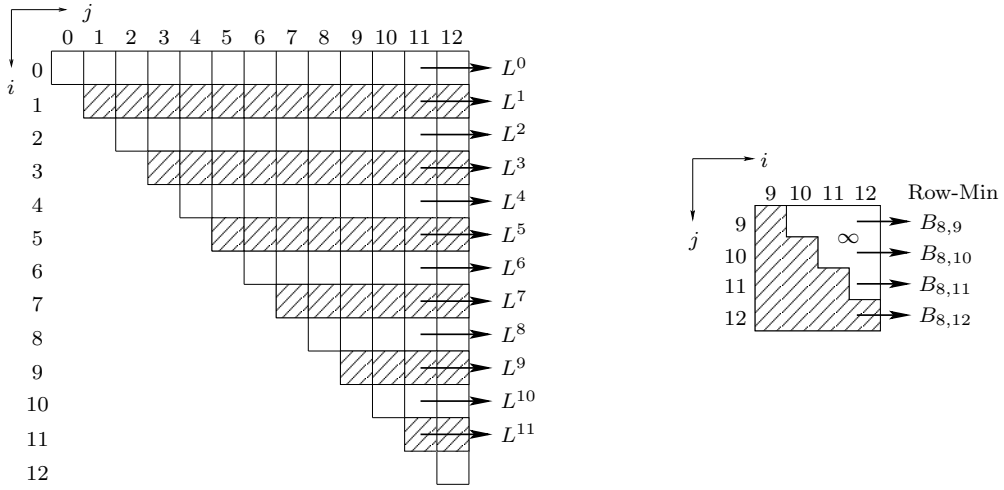


Figure 4: The lefthand figure shows the $B_{i,j}$ matrix for $n = 12$. Each row in the $B_{i,j}$ matrix will correspond to a totally monotone matrix $L^m$. The minimal element of row $j$ in $L^m$ will be the value $B_{m,j}$. The righthand figure shows $L^8$.

so finding the row-minima of $L^m$ yields $B_{m,j}$ for $j = m+1, \ldots, n$. Put another way, the $B_{i,j}$ entries in row $m$ are exactly the row minima of matrix $L^m$.

**Lemma 5** *If the function defined in (4) satisfies the QI then $R^m$ and $L^m$ are totally monotone matrices.*

*Proof*: The proofs are very similar to that of Lemma 3. Note that if $i < j \leq m$, we can again use (8); writing the entries from (8) in boldface gives

$$\begin{aligned} R^m_{i,j} + R^m_{i+1,j+1} &= [w(i,m) + \mathbf{B_{i,j-1}} + B_{j,m}] + [w(i+1,m) + \mathbf{B_{i+1,j}} + B_{j+1,m}] \\ &\leq [w(i+1,m) + \mathbf{B_{i+1,j-1}} + B_{j,m}] + [w(i,m) + \mathbf{B_{i,j}} + B_{j+1,m}] \\ &= R^m_{i+1,j} + R^m_{i,j+1} \end{aligned}$$

and thus $R^m$ is Monge (where we note that the right hand side is $\infty$ if $i \not< j$) and thus totally monotone. If $m < i < j$ then we again use (8) (with $j$ replaced by $j+1$) to get

$$\begin{aligned} L^m_{j,i} + L^m_{j+1,i+1} &= [w(m,j) + B_{m,i-1} + \mathbf{B_{i,j}}] + [w(m,j+1) + B_{m,i} + \mathbf{B_{i+1,j+1}}] \\ &\leq [w(m,j+1) + B_{m,i-1} + \mathbf{B_{i,j+1}}] + [w(m,j) + B_{m,i} + \mathbf{B_{i+1,j}}] \\ &= L^m_{j+1,i} + L^m_{j,i+1} \end{aligned}$$

and thus $L^m$ is Monge (where we note that the right hand side is $\infty$ if $i \not< j$) and thus totally monotone. □

We point out these two decompositions immediately imply a new proof of Lemma 2 (Lemma 2.1 in [15]) which states that

$$K_B(i,j) \leq K_B(i,j+1) \leq K_B(i+1,j+1). \tag{14}$$

To see this note that $K_B(i,j+1)$ is the location of the rightmost row-minimum of row $i$ in matrix $R^{j+1}$, while $K_B(i+1,j+1)$ is the location of the rightmost row-minimum of row $i+1$ in matrix $R^{j+1}$. Thus, the definition of total monotonicity (Definition 2) immediately gives

$$K_B(i,j+1) \leq K_B(i+1,j+1). \tag{15}$$

Similarly, $K_B(i,j)$ is the rightmost row-minimum of row $j$ in $L^i$ while $K_B(i,j+1)$ is the location of the rightmost row-minimum of row $j+1$ in $L^i$. Thus

$$K_B(i,j) \leq K_B(i,j+1). \tag{16}$$

Combining (15) and (16) yields (14), which is what we want. Since the actual speedup in the KY technique comes from an amortization argument based on (14), we have just seen that the original KY-speedup itself is also a consequence of total monotonicity.

We have still not seen how to actually calculate the $B_{i,j}$ using the $R^m$ and $L^m$. Before continuing, we point out that even though the $R^m$ are totally monotone, their row minima *cannot* be calculated using the SMAWK algorithm. This is because, for $0 < i < j \leq m$, the value of entry $R^m_{i,j} = w(i,m) + B_{i,j-1} + B_{j,m}$, which is dependent upon $B_{j,m}$ which is itself the row-minimum of row $j$ *in the same matrix* $R^m$. Thus, the values of the entries of $R^m$ depend upon the other entries in $R^m$ which is something that SMAWK does not allow. The same problem occurs with the $L^m$.

We will now see that, despite this dependence, we can still use the LARSCH algorithm to find the row-minima of the $R^m$. This will have the added advantage of solving the online problem as well.

At this point we should note that our decompositions $L^m$ could also be derived by careful cutting of the 3-D monotone matrices of Aggarwal and Park [2] along particular planes. Aggarwal and Park used an algorithm of Wilber [14] (derived for finding the maxima of certain concave-sequences) to find various tube maxima of their matrices, leading to another $O(n^2)$ algorithm for solving the KY-problem. In fact, even though their algorithm was presented as a static algorithm, careful decomposition of what they do permits using it to solve what we call the left-online KY-problem. A symmetry argument could then yield a right-online algorithm. This never seems to have been noted in the literature, though. In the next section, we present a different online algorithm, based on our decompositions and the LARSCH algorithm.

# 4 Online Algorithms Without Losing the KY Speedup

To execute the LARSCH algorithm, as defined in Section 3 of [10] we need only that $X$ satisfy the following conditions:

1. $X$ is a totally monotone $n \times m$ matrix.
2. For each row index $i$ of $X$, there is a column index $C_i$ such that for $j > C_i$, $X_{i,j} = \infty$ Furthermore, $C_i \leq C_{i+1}$.
3. Let $C_0 = 0$. If $C_i < j \leq C_{i+1}$ then, for any $k > i$, $X_{k,j}$ can be evaluated in $O(1)$ time *provided that the row minima of the first $i$ rows are already known.*

If these conditions are satisfied, the LARSCH algorithm then calculates all of the row minima of $X$ in $O(n+m)$ time. We can now use this algorithm to derive

**Lemma 6**

- *Given that all values $B_{i,j}$, $m < i \leq j \leq n$ have already been calculated, all of the row-minima of $L^m$ can be calculated in $O(n-m)$ time.*
- *Given that all values $B_{i,j}$, $0 \leq i \leq j < m$ have already been calculated, all of the row-minima of $R^m$ can be calculated in $O(m)$ time.*

*Proof*: For the first part, it is easy to see that $L^m$ satisfies the first two conditions required by the LARSCH algorithm with $C_j = j$. The third condition is equivalent to saying that if $C_i < j \leq C_{i+1}$,

i.e., if $j = i+1$, then "$\forall k > i$, $L^m_{k,j} = L^m_{k,i+1} = w(m,k)+B_{m,i}+B_{i+1,k}$ can be evaluated in $O(1)$ time provided that the row minima of the first $i$ rows are already known". But, $w(m,k)$ and $B_{i+1,k}$ are already known and can be retrieved in $O(1)$ time and $B_{m,i}$ is the row-minimum of row $i$ of $L^m$ so this condition is trivially satisfied. Thus, all of the row-minima of the $(n-m)\times(n-m)$ matrix $L^m$ can be calculated in $O(n-m)$ time. For the second part, set $X$ to be the $(m+1)\times(m+1)$ matrix defined by $X_{i,j} = R^m_{m-i,m-j}$. Then $X$ satisfies the first two LARSCH conditions with $C_i = i-1$. The third condition is equivalent to saying that if $C_i < j \le C_{i+1}$, i.e., if $j = i$, then "$\forall k > i$, $X_{k,j} = R^m_{m-k,m-i} = w(m-k,m) + B_{m-k,m-i-1} + B_{m-i,m}$ can be evaluated in $O(1)$ time provided that the row minima of the first $i$ rows are already known". But $w(m-k,m)$ and $B_{m-k,m-i-1}$ are already known and can be retrieved in $O(1)$ time and $B_{m-i,m}$ is the row-minimum of row $i$ of $X$ so this condition is also trivially satisfied. Thus, all of the row-minima of $X$ and equivalently of $R^m$ can be calculated in $O(m)$ time. □

Note that Lemma 6 immediately solves the "right-online" and "left-online" problems described in subsection 1.4; Given the new values $w(i,R+1)$ for $L \le i \le R+1$, simply find the row minima of $R^{R+1}$ in time $O(R-L)$. Given the new values $w(L-1,j)$ for $L-1 \le j \le R$, simply find the row minima of $L^{L-1}$.

We have therefore just shown that *any* dynamic programming problem for which the KY speedup can statically improve run time from $\Theta(n^3)$ to $O(n^2)$ time can be solved in an online fashion in $O(n)$ time per step. That is, online processing incurs no penalty compared to static processing. In particular, the optimum binary search tree (as illustrated in 1.4), can be maintained in $O(n)$ time per step as nodes are added to both its left and right.

# 5 A further application

In [5], Borchers and Gupta extend the Knuth-Yao quadrangle inequality. One of their major applications is finding an optimal Rectilinear Steiner Minimal Arborescence (RSMA) of a *slide*. A slide is a set of points $(x_i, y_i)$ such that, if $i < j$, then $x_i < x_j$ and $y_i > y_j$. A Rectilinear Steiner Arborescence is a directed tree in which each edge either goes up or to the right. In [11] it was shown that the minimum cost Rectilinear Steiner Arborescence connecting slide-points $(x_i, y_i), (x_{i+1}, y_{i+1}), \ldots (x_j, y_j)$ satisfies

$$L(i,j) = \min_{i \le s < j} \left( L_{i,s} + L_{s+1,j} + x_{s+1} - x_1 + y_s - y_j \right). \tag{17}$$

[11] solved this recurrence in $O(n^3)$ time. Even though (17) is not in the KY form (4) Borchers and Gupta [5] were still able to show that $L_{i,j}$ satisfies a quadrangle inequality. This sufficed to show that Lemma 2 still holds for $L_{i.j}$ and thus permitted deriving an $O(n^2)$ algorithm for calculating all of the $L_{i,j}$. In fact, they showed that if $L(i,j)$ is given by a DP recurrence of the form

$$L(i,j) = \min_{i \le s < j} \left( w(i,s,j) + L_{i,s} + L_{s+1,j} \right). \tag{18}$$

where $w(i,s,j)$ satisfies generalized versions of the quadrangle inequality and monotonicity on integer lattices property, then Yao's result could alsways be generalized to apply. Note that the major difference between (4) and (18) is that (18) allows $w(\,)$ to depend upon the splitting index $s$ and be brought *inside* the min .

We point out that even though we derived our results for problems that satisfy the KY form (4) it is quite straightforward to show all of the results in this paper can be extended to work for all $L_{i,j}$ that satisfy (18) with the Borchers and Gupta speedup conditions. In particular, this yields an $O(n)$ per-step worst-case online algorithm for constructing the new RSMA when adding points to the left and right of a slide.

## References


[1] Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987.

[2] Alok Aggarwal and James Park. Notes on searching in multidimensional monotone arrays. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 497–512, 1988.

[3] M. J. Atallah, S. R. Kosaraju, L. L. Larmore, G. L. Miller, and S.-H. Teng. Constructing trees in parallel. In *Proceedings of the First Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 421–431, 1989.

[4] Amotz Bar-Noy and Richard E. Ladner. Efficient algorithms for optimal stream merging for media-on-demand. *SIAM Journal on Computing*, 33(5):1011–1034, 2004.

[5] Al Borchers and Prosenjit Gupta. Extending the quadrangle inequality to speed-up dynamic programming. *Information Processing Letters*, 49(6):287–290, 1994.

[6] Rainer E. Burkard, Bettina Klinz, and Rudiger Rudolf. Perspectives of Monge properties in optimization. *Discrete Applied Mathematics*, 70(2):95–161, 1996.

[7] Zvi Galil and Kunsoo Park. Dynamic programming with convexity, concavity and sparsity. *Theoretical Computer Science.*, 92(1):49–76, 1992.

[8] E. N. Gilbert and E. F. Moore. Variable length encodings. *Bell System Technical Journal*, 38:933–968, 1959.

[9] Donald E. Knuth. Optimum binary search trees. *Acta Informatica*, 1:14–25, 1971.

[10] Lawrence L. Larmore and Baruch Schieber. On-line dynamic programming with applications to the prediction of RNA secondary structure. *Journal of Algorithms*, 12(3):490–515, 1991.

[11] Sailesh K. Rao, P. Sadayappan, Frank K. Hwang, and Peter W. Shor. The rectilinear steiner arborescence problem. *Algorithmica*, 7(2-3):277–288, 1992.

[12] Amir Said. Efficient alphabet partitioning algorithms for low-complexity entropy coding. In *Proceedings of the 2005 Data Compression Conference*, pages 183–192, 2005.

[13] Russell L. Wessner. Optimal alphabetic search trees with restricted maximal height. *Information Processing Letters*, 4(4):90–94, 1976.

[14] Robert Wilber. The concave least-weight subsequence problem revisited. *Journal of Algorithms*, 9(3):418–425, 1988.

[15] F. Frances Yao. Efficient dynamic programming using quadrangle inequalities. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, pages 429–435, 1980.

[16] F. Frances Yao. Speed-up in dynamic programming. *SIAM Journal on Matrix Analysis and Applications (formerly SIAM Journal on Algebraic and Discrete Methods)*, 3(4):532–540, 1982.