Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Discrete Mathematics 308 (2008) 2213–2221
DISCRETE MATHEMATICS
www.elsevier.com/locate/disc

Searching monotone multi-dimensional arrays☆

Yongxi Cheng[a,*], Xiaoming Sun[b], Yiqun Lisa Yin[c]

[a] *Department of Computer Science, Tsinghua University, Beijing 100084, China*
[b] *Center for Advanced Study, Tsinghua University, Beijing 100084, China*
[c] *Independent security consultant, Greenwich CT, USA*

Received 15 September 2004; received in revised form 7 March 2006; accepted 24 April 2007
Available online 8 May 2007

Abstract

A d-dimensional array of real numbers is called *monotone increasing* if its entries are increasing along each dimension. Given $A_{n,d}$, a monotone increasing d-dimensional array with n entries along each dimension, and a real number x, we want to decide whether $x \in A_{n,d}$, by performing a sequence of comparisons between x and some entries of $A_{n,d}$. We want to minimize the number of comparisons used. In this paper we investigate this search problem, we generalize Linial and Saks' search algorithm [N. Linial, M. Saks, Searching ordered structures, J. Algorithms 6 (1985) 86–103] for monotone three-dimensional arrays to d-dimensions for $d \geqslant 4$. For $d = 4$, our new algorithm is optimal up to the lower order terms.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Search algorithm; Complexity; Partially ordered set; Monotone multi-dimensional array

1. Introduction

In this paper, we investigate the problem of searching monotone multi-dimensional arrays. Suppose we are given a d-dimensional array of real numbers with n entries along each dimension

$$A_{n,d} = \{a_{i_1,i_2,\ldots,i_d} | i_1, i_2, \ldots, i_d = 1, 2, \ldots, n\}.$$

We say that the array $A_{n,d}$ is *monotone increasing* if its entries are increasing along each dimension. More precisely, if $i_1 \leqslant j_1$, $i_2 \leqslant j_2$,..., $i_d \leqslant j_d$ then $a_{i_1,i_2,\ldots,i_d} \leqslant a_{j_1,j_2,\ldots,j_d}$. In other words, if $P = [n]^d$ is the product of d totally ordered sets $\{1, 2, \ldots, n\}$, then $A_{n,d}$ is consistent with the partially ordered set P.

The *search problem* is to decide whether a given real number x belongs to the array $A_{n,d}$, by comparing x with a subset of the entries in the array. The *complexity* of this problem, denoted by $\tau(n, d)$, is defined to be the minimum over all search algorithms for $A_{n,d}$ of the number of comparisons needed in the worst case. Note that for $d = 1$, this problem reduces to searching a totally ordered set. In this case, the binary search algorithm is optimal and requires at most $\lceil \log_2(n + 1) \rceil$ comparisons in the worst case.

☆ Supported in part by National Natural Science Foundation of China Grant 60553001, 60603005, and National Basic Research Program of China Grant 2007CB807900, 2007CB807901.
* Corresponding author.
E-mail addresses: cyx@mails.tsinghua.edu.cn (Y. Cheng), xiaomings@tsinghua.edu.cn (X. Sun), yiqun@alum.mit.edu (Y.L. Yin).

0012-365X/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.disc.2007.04.067

We first briefly review some previous work. In [3], Linial and Saks presented some general results on the complexity of the above class of search problems. In particular, they proved that for any finite partially ordered set P, the information theoretic bound for the complexity is tight up to a multiplicative constant. In [2] they studied the problems for general finite partially ordered set P and also gave more precise results for the case where $P=[n]^d$, for dimensions $d\geqslant 2$. They observed that for $d=2$, it had been known that $\tau(n,2)=2n-1$ [1]. For $d\geqslant 2$, they showed that the order of $\tau(n,d)$ is $\mathrm{O}(n^{d-1})$. More specifically, they proved that for $d\geqslant 2$,

$$c_2(d)n^{d-1}+\mathrm{o}(n^{d-1})\leqslant\tau(n,d)\leqslant c_1(d)n^{d-1},$$

where $c_1(d)$ is a nonincreasing function of d and upper bounded by 2, and $c_2(d)=\sqrt{(24/\pi)}d^{-1/2}+\mathrm{o}(d^{-1/2})$. The upper bound $c_1(d)n^{d-1}$ was obtained by using a straightforward search algorithm which partitions $A_{n,d}$ into n isomorphic copies of $A_{n,d-1}$, and then searches each copy separately. They also described a more efficient algorithm for $d=3$ and proved the following bounds on $\tau(n,3)$:

$$\left\lfloor\frac{3n^2}{2}\right\rfloor\leqslant\tau(n,3)\leqslant\frac{3n^2}{2}+cn\ln n.$$

In the above inequality, c is a positive constant, and so the bounds are asymptotically tight. An open problem left is whether their search algorithm for $d=3$ can be generalized to higher dimensions.

In this paper, we present new search algorithms for monotone d-dimensional arrays for $d\geqslant 4$, from which we can obtain

$$\tau(n,d)\leqslant\frac{d}{d-1}n^{d-1}+\mathrm{O}(n^{d-2}).$$

The above bound is tight for $d=4$, up to the lower order terms.

The rest of the paper is devoted to the description and analysis of the new algorithms. We start with the case where $d=4$. This special case best illustrates the main idea, and it is also easier to visualize the subspaces that are encountered in the search algorithm. Then we describe the generalized algorithm for $d\geqslant 4$.

Before presenting the technical details, we describe some basic notation and convention that we will follow throughout the paper. In general, we use capital letters to represent sets and small letters to represent numbers. The sets that we need to consider are often subsets of $A_{n,d}$ for which some of the subscripts are fixed, and we use some simple notation to represent them. For example, we use $Q=\{a_{1,i_2,i_3,i_4}\}$ to denote a "surface" of the four-dimensional array $A_{n,4}$ for which the first subscript of a is fixed to be 1. It is understood that all other subscripts range between $[1,n]$, and we often omit the specification "$i_2,i_3,i_4=1,2,\ldots,n$" if it is clear from the context.

2. Searching four-dimensional arrays

In this section, we present a $\frac{4}{3}n^3+\mathrm{O}(n^2)$ algorithm for searching monotone four-dimensional arrays. The algorithm is optimal up to the lower order terms.

We start with a lower bound on $\tau(n,4)$ which will be shown to be asymptotically tight later, followed by the description of an algorithm for partitioning monotone two-dimensional arrays, which will be a useful subroutine for our searching algorithm. Then, we will present the main idea and the details of our search algorithm for four-dimensional arrays.

2.1. A lower bound on $\tau(n,4)$

Using the method in [2], we can calculate a lower bound on $\tau(n,4)$. Let $[n]$ denote the totally ordered set $\{1,2,\ldots,n\}$, and let

$$D_1(n,4)=\{(i_1,i_2,i_3,i_4)\in[n]^4|i_1+i_2+i_3+i_4=2n+1\},$$

$$D_2(n,4)=\{(i_1,i_2,i_3,i_4)\in[n]^4|i_1+i_2+i_3+i_4=2n+2\}.$$

Define $D(n,4)=D_1(n,4)\cup D_2(n,4)$. As shown in [2], $D(n,4)$ is a *section* of $[n]^4$, and a simple adversary argument implies that $\tau(n,4)$ is lower bounded by $|D(n,4)|$. Let

$$X=\{(i_1,i_2,i_3,i_4)\in[2n+1]^4|\ i_1+i_2+i_3+i_4=2n+1\},$$

$$Y_k = \{(i_1, i_2, i_3, i_4) \in X \mid i_k > n\} \quad \text{for } k = 1, 2, 3, 4,$$

$$Z = \{(i_1, i_2, i_3, i_4) \in [n+1]^4 \mid i_1 + i_2 + i_3 + i_4 = n + 1\}.$$

It is easy to see that $|Y_k| = |Z| = \binom{n}{3}$ for $k = 1, 2, 3, 4$. Thus, $|D_1(n, 4)| = |X| - \sum_{k=1}^{4} |Y_k| = \binom{2n}{3} - 4\binom{n}{3} = \frac{1}{3}(2n^3 - 2n)$. Similarly, $|D_2(n, 4)| = \binom{2n+1}{3} - 4\binom{n+1}{3} = \frac{1}{3}(2n^3 + n)$. Therefore,

$$\tau(n, 4) \geqslant |D(n, 4)| = |D_1(n, 4)| + |D_2(n, 4)| = \frac{4}{3}n^3 - \frac{n}{3}.$$

2.2. *Partitioning two-dimensional arrays*

In [2], Linial and Saks gave a simple algorithm for searching an $m \times n$ matrix ($m, n \geqslant 1$) with entries increasing along each row and column. The algorithm needs at most $m + n - 1$ comparisons. We will refer to this algorithm as the *Matrix Search Algorithm*. Since $A_{n,2}$ is isomorphic to an $n \times n$ matrix, given an input x, we can adapt the *Matrix Search Algorithm* to partition $A_{n,2}$ into two subsets S and L using at most $2n - 1$ comparisons, such that S contains entries *smaller* than x and L contains entries *larger* than x. For the sake of completeness, we give detailed description of the new partition algorithm.

Algorithm. *Partition two-Dimensional Array*
Input

- A real number x.
- A monotone two-dimensional array $A_{n,2} = \{a_{i_1,i_2}\}$.

Output

- If $x \in A_{n,2}$, output (i_1, i_2) such that $a_{i_1,i_2} = x$.
- If $x \notin A_{n,2}$, output a partition $\{u, v, S, L\}$ of $A_{n,2}$ with the following properties:
    - u and v are two arrays each contains n integers such that $i_1 \leqslant u[i_2]$ iff $a_{i_1,i_2} < x$ and $i_2 \leqslant v[i_1]$ iff $a_{i_1,i_2} < x$.
    - S and L form a partition of $\{(i_1, i_2) | i_1, i_2 \in [n]\}$ such that if $(i_1, i_2) \in S$ then $a_{i_1,i_2} < x$, and if $(i_1, i_2) \in L$ then $a_{i_1,i_2} > x$.

Procedure

- Initially set $S = L = \phi$.
- View $A_{n,2}$ as an $n \times n$ matrix and repeat comparing x with the element e at the top right corner of the current matrix.
    - If $x > e$, then eliminate the first row of the current matrix and put their entries into S.
    - If $x < e$, then eliminate the last column of the current matrix and put their entries into L.
    - If $x = e$, then return this entry and exit.
- Stop when the partition is finished, thus also obtain u and v (see Fig. 1).

We will use the notation u, v, S, L throughout the paper. Sometimes we will introduce subscripts to them to represent the dimension indices to be considered. Ignoring the indices, these four variables have the following useful relations:

$$S = \{(i_1, i_2) | 1 \leqslant i_1 \leqslant u[i_2]\} = \{(i_1, i_2) | 1 \leqslant i_2 \leqslant v[i_1]\}, \quad (1)$$

$$L = \{(i_1, i_2) | u[i_2] < i_1 \leqslant n\} = \{(i_1, i_2) | v[i_1] < i_2 \leqslant n\}. \quad (2)$$

Obviously, $S \cap L = \phi$ and $S \cup L = [n]^2$, hence $|S| + |L| = n^2$. In addition, $|S| = u[1] + \cdots + u[n] = v[1] + \cdots + v[n]$ and $|L| = (n - u[1]) + \cdots + (n - u[n]) = (n - v[1]) + \cdots + (n - v[n])$.

Notice that when $m = 0$ or $n = 0$, we can "search" an $m \times n$ matrix using 0 comparisons. Therefore, based on the *Matrix Search Algorithm* in [2], we have the following lemma that will be useful later.

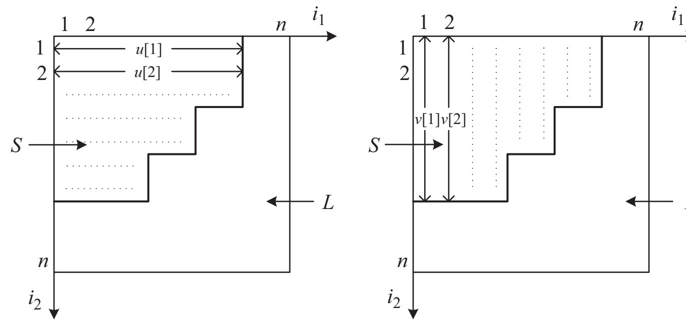

Fig. 1. u, v, S, L: partition of the monotone two-dimensional array $A_{n,2}$.

Lemma 2.1. *For $m, n \geqslant 0$, any $m \times n$ matrix with entries increasing along each dimension can be searched using at most $m + n$ comparisons.*

Proof. If $m = 0$ or $n = 0$, the matrix is empty, thus needs no comparison. If $m, n > 0$, using the *Matrix Search Algorithm*, we can search the matrix using at most $m + n - 1$ comparisons. Therefore, the lemma holds. □

2.3. Main idea of the search algorithm

The main idea of our algorithm for $d = 4$ is to first search the "surfaces" (three-dimensional arrays) of $A_{n,4}$ and then the problem reduces to searching a "smaller" array $A_{n-2,4}$. At a high level, searching the surfaces consists of two major steps:

- Step 1: Select eight special two-dimensional arrays. By using the algorithm *Partition two-dimensional array*, partition each selected two-dimensional array into two subsets L and S, where elements in L are larger than or equal to x, and elements in S are smaller than x.
- Step 2: Search the eight "surfaces" of $A_{n,4}$. The subsets S, L obtained in Step 1 help to "cut" each surface into a sequence of two-dimensional matrices that allows searching with less comparisons.

2.4. Description and analysis of the search algorithm

Now we are ready to present our search algorithm for $d = 4$. As explained in Section 2.3, the algorithm is recursive, and reduces n by two for each recursion. Without loss of generality, we consider the case where $x \notin A_{n,4}$. We first describe the algorithm and then analyze the number of comparisons needed.

Step 1: Apply the algorithm *Partitioning two-dimensional array* to divide each of the following eight two-dimensional arrays into two subsets (the eight arrays are defined by fixing two of the subscripts to either 1 or n, thus reducing the number of dimensions by two):

$$M_1 = \{a_{i_1,i_2,1,n}\} : S_1, L_1, \quad M_1^* = \{a_{i_1,i_2,n,1}\} : S_1^*, L_1^*,$$

$$M_2 = \{a_{n,i_2,i_3,1}\} : S_2, L_2, \quad M_2^* = \{a_{1,i_2,i_3,n}\} : S_2^*, L_2^*,$$

$$M_3 = \{a_{1,n,i_3,i_4}\} : S_3, L_3, \quad M_3^* = \{a_{n,1,i_3,i_4}\} : S_3^*, L_3^*,$$

$$M_4 = \{a_{i_1,1,n,i_4}\} : S_4, L_4, \quad M_4^* = \{a_{i_1,n,1,i_4}\} : S_4^*, L_4^*.$$

The eight pairs of "mutually complementary" subsets S_k, L_k and S_k^*, L_k^* $(k = 1, 2, 3, 4)$ have the following properties:

$$a_{i_1,i_2,1,n} < x < a_{j_1,j_2,1,n} \quad \text{for } (i_1, i_2) \in S_1 \text{ and } (j_1, j_2) \in L_1,$$

$$a_{n,i_2,i_3,1} < x < a_{n,j_2,j_3,1} \quad \text{for } (i_2, i_3) \in S_2 \text{ and } (j_2, j_3) \in L_2,$$

$$a_{1,n,i_3,i_4} < x < a_{1,n,j_3,j_4} \quad \text{for } (i_3, i_4) \in S_3 \text{ and } (j_3, j_4) \in L_3,$$

$$a_{i_1,1,n,i_4} < x < a_{j_1,1,n,j_4} \quad \text{for } (i_4, i_1) \in S_4 \text{ and } (j_4, j_1) \in L_4,$$

$$a_{i_1,i_2,n,1} < x < a_{j_1,j_2,n,1} \quad \text{for } (i_1, i_2) \in S_1^* \text{ and } (j_1, j_2) \in L_1^*,$$

$$a_{1,i_2,i_3,n} < x < a_{1,j_2,j_3,n} \quad \text{for } (i_2, i_3) \in S_2^* \text{ and } (j_2, j_3) \in L_2^*,$$

$$a_{n,1,i_3,i_4} < x < a_{n,1,j_3,j_4} \quad \text{for } (i_3, i_4) \in S_3^* \text{ and } (j_3, j_4) \in L_3^*,$$

$$a_{i_1,n,1,i_4} < x < a_{j_1,n,1,j_4} \quad \text{for } (i_4, i_1) \in S_4^* \text{ and } (j_4, j_1) \in L_4^*.$$

In addition to the eight pairs of subsets, the algorithm also outputs u_k, v_k and u_k^*, v_k^*, corresponding to S_k, L_k and S_k^*, L_k^*, respectively, with the properties given in Eqs. (1) and (2). For each k, at most $2n-1$ comparisons are needed to partition M_k (M_k^*). Thus, at most $8 \times (2n-1)$ comparisons are needed in this step.

Step 2: Search the following eight three-dimensional surfaces of $A_{n,4}$ (each surface is defined by setting one of the subscripts to either 1 or n, thus reducing the number of dimensions by one):

$$Q_1 = \{a_{1,i_2,i_3,i_4}\}, \quad Q_1^* = \{a_{n,i_2,i_3,i_4}\},$$

$$Q_2 = \{a_{i_1,1,i_3,i_4}\}, \quad Q_2^* = \{a_{i_1,n,i_3,i_4}\},$$

$$Q_3 = \{a_{i_1,i_2,1,i_4}\}, \quad Q_3^* = \{a_{i_1,i_2,n,i_4}\},$$

$$Q_4 = \{a_{i_1,i_2,i_3,1}\}, \quad Q_4^* = \{a_{i_1,i_2,i_3,n}\}.$$

By symmetry, we only need to show how to search Q_1. The algorithm proceeds by fixing $i_3 = i_3'$ for $i_3' = 1, 2, \ldots, n$ and searching each of the two-dimensional arrays $\{a_{1,i_2,i_3',i_4}\}$. A useful observation is that for each i_3', we can restrict the search to a smaller matrix (in contrast to an $n \times n$ matrix) by leveraging on information obtained in Step 1.

Below, we explain the above observation and Step 2 in more detail. Consider an element $a_{1,i_2,i_3',i_4} \in Q_1$. If $(i_2, i_3') \in S_2^*$, then we know that $a_{1,i_2,i_3',i_4} \leqslant a_{1,i_2,i_3',n} < x$. Hence, in order for $a_{1,i_2,i_3',i_4} = x$, it must be the case that $(i_2, i_3') \in L_2^*$, or equivalently, $u_2^*[i_3'] < i_2 \leqslant n$. Similarly, we can conclude that in order for $a_{1,i_2,i_3',i_4} = x$, it must be the case that $(i_3', i_4) \in L_3$, or equivalently, $v_3[i_3'] < i_4 \leqslant n$. Hence, we obtain a constraint on the indices (i_2, i_4). By Lemma 2.1, searching this restricted $(n - u_2^*[i_3']) \times (n - v_3[i_3'])$ matrix needs at most $(n - u_2^*[i_3']) + (n - v_3[i_3'])$ comparisons. Notice that $n - u_2^*[i_3']$ is the number of entries (i_2, i_3)'s in L_2^* with $i_3 = i_3'$, and $n - v_3[i_3']$ is the number of entries (i_3, i_4)'s in L_3 with $i_3 = i_3'$. Thus,

$$(n - u_2^*[i_3']) + (n - v_3[i_3']) = |\{(i_2, i_3) \in L_2^* | i_3 = i_3'\}| + |\{(i_3, i_4) \in L_3 | i_3 = i_3'\}|.$$

When i_3 ranges over $1, 2, \ldots, n$, we obtain that the total number of comparisons needed to search Q_1 is at most $N(Q_1) = |L_2^*| + |L_3|$ (see Fig. 2).

Similarly, if $a_{n,i_2,i_3,i_4} \in Q_1^*$ equals to x, it must be the case that $(i_2, i_3) \in S_2$ and $(i_3, i_4) \in S_3^*$, it follows that the total number of comparisons needed to search Q_1^* is at most $N(Q_1^*) = |S_2| + |S_3^*|$.

Using similar arguments, the numbers of comparisons needed for searching the above eight three-dimensional arrays are

$$N(Q_1) = |L_3| + |L_2^*|, \quad N(Q_1^*) = |S_2| + |S_3^*|,$$

$$N(Q_2) = |L_4| + |L_3^*|, \quad N(Q_2^*) = |S_3| + |S_4^*|,$$

$$N(Q_3) = |L_1| + |L_4^*|, \quad N(Q_3^*) = |S_4| + |S_1^*|,$$

$$N(Q_4) = |L_2| + |L_1^*|, \quad N(Q_4^*) = |S_1| + |S_2^*|.$$

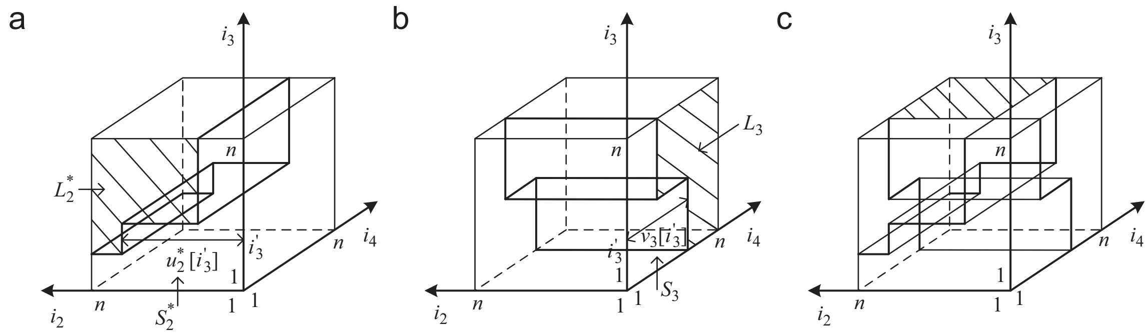

Fig. 2. Searching the three-dimensional surface $Q_1 = \{a_{1,i_2,i_3,i_4}\}$ of $A_{n,4}$. (a) Partition of $M_2^* = \{a_{1,i_2,i_3,n}\}$ into S_2^* and L_2^*; (b) partition of $M_3 = \{a_{1,n,i_3,i_4}\}$ into S_3 and L_3; (c) the 'pyramid' composed of a sequence of two-dimensional matrices to be searched.

Therefore, the total number of comparisons needed for searching these eight arrays is at most

$$\sum_{k=1}^{4}(|S_k| + |L_k| + |S_k^*| + |L_k^*|) = 4 \times 2n^2 = 8n^2.$$

Steps 1 and 2 leave an $(n-2)^4$ array

$$A_{n-2,4} = \{a_{i_1,i_2,i_3,i_4} | i_1, i_2, i_3, i_4 = 2, \ldots, n-1\}.$$

Hence, we have for $n > 2$,

$$\tau(n,4) \leqslant \tau(n-2,4) + 8n^2 + 8(2n-1).$$

From this recursion we can get (see Eq. (4) for the derivation)

$$\tau(n,4) \leqslant \tfrac{4}{3}n^3 + \mathrm{O}(n^2). \tag{3}$$

3. Searching d-dimensional arrays

The algorithm for four-dimensional arrays can be generalized to higher dimensions ($d \geqslant 4$). The main idea is quite similar: the $2d$ "surfaces" ($(d-1)$-dimensional arrays) of $A_{n,d}$ can be searched using $2dn^{d-2} + \mathrm{O}(n^{d-3})$ comparisons. We achieve this in two steps. First, select $2d$ special $(d-2)$-dimensional arrays and partition each of them into two subsets S and L. Second, we search the $2d$ "surfaces". The subsets $\{S, L\}$ will help cut some part of each surface, i.e., reduce the comparison number. In particular, if we fix $(d-3)$ subscripts, the resulting part is a smaller matrix (in contrast to an $n \times n$ matrix). An $a \times b$ matrix can be searched using at most $a + b$ comparisons (Lemma 2.1), adding them up for all the $2d$ "surfaces", we can get the desired upper bound.

First we describe how to select and partition the $(d-2)$-dimensional arrays. Define $M_1 = \{a_{i_1,i_2,\ldots,i_d} \in A_{n,d} | i_{d-1} = 1, i_d = n\}$. Consider the case where $x \notin M_1$. For fixed $i_2 = i_2', i_3 = i_3', \ldots, i_{d-3} = i_{d-3}'$ (where $i_2', i_3', \ldots, i_{d-3}' \in [n]$ are constants) we can get two integer arrays $u[n]$ and $v[n]$ such that

$$a_{i_1,i_2,\ldots,i_d}|_{i_2=i_2',\ldots,i_{d-3}=i_{d-3}';\ i_{d-1}=1,i_d=n} < x \quad \text{for } i_1 \leqslant u[i_{d-2}],$$

$$a_{i_1,i_2,\ldots,i_d}|_{i_2=i_2',\ldots,i_{d-3}=i_{d-3}';\ i_{d-1}=1,i_d=n} > x \quad \text{for } i_1 > u[i_{d-2}],$$

$$a_{i_1,i_2,\ldots,i_d}|_{i_2=i_2',\ldots,i_{d-3}=i_{d-3}';\ i_{d-1}=1,i_d=n} < x \quad \text{for } i_{d-2} \leqslant v[i_1],$$

$$a_{i_1,i_2,\ldots,i_d}|_{i_2=i_2',\ldots,i_{d-3}=i_{d-3}';\ i_{d-1}=1,i_d=n} > x \quad \text{for } i_{d-2} > v[i_1],$$

by using the algorithm *Partitioning two-dimensional array*, at most $2n-1$ comparisons are needed for each fixed $i_2, \ldots, i_{d-3}$. Thus using at most $n^{d-4}(2n-1)$ comparisons we can get two integer arrays u_1 and v_1 of sizes n^{d-3} such that

- If $i_1 \leqslant u_1[i_2, \ldots, i_{d-2}]$, then $a_{i_1,i_2,\ldots,i_d}|_{i_{d-1}=1, i_d=n} < x$.

  Otherwise, $a_{i_1,i_2,\ldots,i_d}|_{i_{d-1}=1, i_d=n} > x$.
- If $i_{d-2} \leqslant v_1[i_1, \ldots, i_{d-3}]$, then $a_{i_1,i_2,\ldots,i_d}|_{i_{d-1}=1, i_d=n} < x$.

  Otherwise, $a_{i_1,i_2,\ldots,i_d}|_{i_{d-1}=1, i_d=n} > x$.

  Thus, we can partition $[n]^{d-2}$ into two subsets S_1 and L_1 such that

- $a_{i_1,i_2,\ldots,i_d}|_{i_{d-1}=1, i_d=n} < x$ for $(i_1, \ldots, i_{d-2}) \in S_1$.
- $a_{i_1,i_2,\ldots,i_d}|_{i_{d-1}=1, i_d=n} > x$ for $(i_1, \ldots, i_{d-2}) \in L_1$.

  Obviously, we have

- $(i_1, \ldots, i_{d-2}) \in S_1$ if and only if $i_1 \leqslant u_1[i_2, \ldots, i_{d-2}]$ (also $i_{d-2} \leqslant v_1[i_1, \ldots, i_{d-3}]$).
- $(i_1, \ldots, i_{d-2}) \in L_1$ if and only if $i_1 > u_1[i_2, \ldots, i_{d-2}]$ (also $i_{d-2} > v_1[i_1, \ldots, i_{d-3}]$).

Next we describe the algorithm for searching d-dimensional arrays $A_{n,d}$, for $d \geqslant 4$. Without loss of generality, we consider the case where $x \notin A_{n,d}$.

Step 1: Partition each of the following $2d$ $(d-2)$-dimensional arrays into two subsets.

$$M_k = \{a_{i_1,i_2,\ldots,i_d} | i_{k-2} = 1, i_{k-1} = n\}: \quad S_k, L_k,$$

$$M_k^* = \{a_{i_1,i_2,\ldots,i_d} | i_{k-2} = n, i_{k-1} = 1\}: \quad S_k^*, L_k^*,$$

$k = 1, 2, \ldots, d$ (here i_{k-2} means $i_{(k-2) \bmod d}$, and i_{k-1} means $i_{(k-1) \bmod d}$).

We get $2d$ pairs of mutually complementary subsets S_k, L_k and S_k^*, L_k^* with the following properties:

$$a_{i_1,\ldots,i_d}|_{i_{k-2}=1, i_{k-1}=n} < x < a_{j_1,\ldots,j_d}|_{j_{k-2}=1, j_{k-1}=n}$$
$$\text{for } (i_k, \ldots, i_{k+d-3}) \in S_k \quad \text{and} \quad (j_k, \ldots, j_{k+d-3}) \in L_k,$$

$$a_{i_1,\ldots,i_d}|_{i_{k-2}=n, i_{k-1}=1} < x < a_{j_1,\ldots,j_d}|_{j_{k-2}=n, j_{k-1}=1}$$
$$\text{for } (i_k, \ldots, i_{k+d-3}) \in S_k^* \quad \text{and} \quad (j_k, \ldots, j_{k+d-3}) \in L_k^*,$$

$k = 1, 2, \ldots, d$ (here i_{k+d-3} means $i_{(k+d-3) \bmod d}$, etc.).

For the pair S_k and L_k, we have two $(d-3)$-dimensional arrays u_k and v_k such that, if $i_k \leqslant u_k[i_{k+1}, \ldots, i_{k+d-3}]$ then $(i_k, \ldots, i_{k+d-3}) \in S_k$ else $(i_k, \ldots, i_{k+d-3}) \in L_k$; if $i_{k+d-3} \leqslant v_k[i_k, \ldots, i_{k+d-4}]$ then $(i_k, \ldots, i_{k+d-3}) \in S_k$ else $(i_k, \ldots, i_{k+d-3}) \in L_k$, $k = 1, 2, \ldots, d$. Similarly, we have u_k^* and v_k^* for the pair S_k^* and L_k^*, for $k = 1, 2, \ldots, d$.

In this step, we obtain $4d$ $(d-3)$-dimensional arrays u_k, v_k, u_k^*, v_k^* $(k = 1, 2, \ldots, d)$, using at most $2dn^{d-4}(2n-1)$ comparisons.

Step 2: Search the following $2d$ $(d-1)$-dimensional surfaces of $A_{n,d}$, which are defined by fixing one of the subscripts to either 1 or n.

$$Q_k = \{a_{i_1,\ldots,i_d} | i_k = 1\}, \quad Q_k^* = \{a_{i_1,\ldots,i_d} | i_k = n\},$$

$k = 1, 2, \ldots, d$.

By symmetry, we only need to consider searching $Q_1 = \{a_{1,i_2,\ldots,i_d}\}$.

If $a_{1,\ldots,i_d} \in Q_1$ equals to x, we have $(i_3, \ldots, i_d) \in L_3$ and $(i_2, \ldots, i_{d-1}) \in L_2^*$. For fixed $i_3 = i_3', \ldots, i_{d-1} = i_{d-1}'$ (where $i_3', \ldots, i_{d-1}' \in \{1, 2, \ldots, n\}$), there exist two integers $u = u_2^*[i_3', \ldots, i_{d-1}']$ and $v = v_3[i_3', \ldots, i_{d-1}']$ such

that only when $u < i_2 \leqslant n$, $(i_2, i'_3, \ldots, i'_{d-1}) \in L_2^*$; only when $v < i_d \leqslant n$, $(i'_3, \ldots, i'_{d-1}, i_d) \in L_3$. Thus for fixed $i_3 = i'_3, \ldots, i_{d-1} = i'_{d-1}$, only when $u < i_2 \leqslant n$ and $v < i_d \leqslant n$, $a_{1,i_2,i'_3,\ldots,i'_{d-1},i_d}$ possibly equal to x. Searching this $(n-u) \times (n-v)$ matrix needs at most $(n-u)+(n-v)$ comparisons. Notice that $n-u$ is the number of elements $(i_2, \ldots, i_{d-1})$'s in L_2^* with $i_3 = i'_3, \ldots, i_{d-1} = i'_{d-1}$, and $n-v$ is the number of elements $(i_3, \ldots, i_d)$'s in L_3 with $i_3 = i'_3, \ldots, i_{d-1} = i'_{d-1}$. Thus $(n-u)+(n-v) = |\{(i_2, \ldots, i_{d-1}) \in L_2^* | i_3 = i'_3, \ldots, i_{d-1} = i'_{d-1}\}| + |\{(i_3, \ldots, i_d) \in L_3 | i_3 = i'_3, \ldots, i_{d-1} = i'_{d-1}\}|$. When $(i_3, \ldots, i_{d-1})$ ranges over all elements in $[n]^{d-3}$, we obtain that the total number of comparisons needed to search Q_1 is at most $N(Q_1) = |L_3| + |L_2^*|$.

Similarly for $Q_1^* = \{a_{n,i_2,\ldots,i_d}\}$, we have $N(Q_1^*) = |S_2| + |S_3^*|$.

Using similar arguments, the numbers of comparisons needed for searching the above $2d$ surfaces are

$$N(Q_k) = |L_{k+2}| + |L_{k+1}^*|, \quad N(Q_k^*) = |S_{k+1}| + |S_{k+2}^*|,$$

$k = 1, 2, \ldots, d$ (here L_{k+2} means $L_{(k+2) \bmod d}$, etc.).

Thus, searching these $2d$ $(d-1)$-dimensional surfaces needs at most

$$\sum_{k=1}^{d} \{|L_{k+2}| + |L_{k+1}^*| + |S_{k+1}| + |S_{k+2}^*|\} = d \times 2n^{d-2} = 2dn^{d-2}$$

comparisons.

Steps 1 and 2 leave an $(n-2)^d$ d-dimensional array

$$A_{n-2,d} = \{a_{i_1,\ldots,i_d} | i_1, \ldots, i_d = 2, \ldots, n-1\}.$$

Hence the generalized recursion is

$$\tau(1, d) = 1,$$

$$\tau(2, d) \leqslant 2^d,$$

$$\tau(n, d) \leqslant \tau(n-2, d) + 2dn^{d-2} + 2dn^{d-4}(2n-1) \quad \text{for } n > 2.$$

From the recursion, for fixed d there exists a constant C and $\varepsilon \in \{1, 2\}$ (the value of ε depends on the parity of n) such that

$$\begin{aligned}
\tau(n, d) &\leqslant \tau(n-2, d) + 2dn^{d-2} + 4dn^{d-3} \\
&\leqslant \tau(n-4, d) + 2d(n^{d-2} + (n-2)^{d-2}) + 4d(n^{d-3} + (n-2)^{d-3}) \\
&\leqslant \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
&\leqslant C + 2d(n^{d-2} + (n-2)^{d-2} + \cdots + \varepsilon^{d-2}) + 4d(n^{d-3} + (n-2)^{d-3} + \cdots + \varepsilon^{d-3}) \\
&\leqslant C + d((n+1)^{d-2} + n^{d-2} + (n-1)^{d-2} + \cdots + 1^{d-2}) + 4dn^{d-2} \\
&\leqslant C + d \times \int_1^{n+2} t^{d-2}\,\mathrm{d}t + 4dn^{d-2} \\
&= C + \frac{d}{d-1}(n+2)^{d-1} - \frac{d}{d-1} + 4dn^{d-2} \\
&= \frac{d}{d-1} n^{d-1} + \mathrm{O}(n^{d-2}).
\end{aligned}$$

Therefore,

$$\tau(n, d) \leqslant \frac{d}{d-1} n^{d-1} + \mathrm{O}(n^{d-2}), \quad d = 4, 5, \ldots . \tag{4}$$

The following theorem summarizes our main results.

Theorem 3.1. *For* $n \geqslant 1$ *and* $d \geqslant 4$, $\tau(n, d) \leqslant (d/(d-1))n^{d-1} + \mathrm{O}(n^{d-2})$.

In particular, for $d = 4$, $\frac{4}{3}n^3 - n/3 \leqslant \tau(n, 4) \leqslant \frac{4}{3}n^3 + \mathrm{O}(n^2)$.

4. Discussion

In this paper we give an algorithm for searching monotone d-dimensional ($d \geqslant 4$) arrays $A_{n,d}$, which requires at most $(d/(d-1))n^{d-1} + \mathrm{O}(n^{d-2})$ comparisons. For $d = 4$, it is optimal up to the lower order terms.

For $d = 5$, let $D(n, 5) = \{(i_1, i_2, i_3, i_4, i_5) \in [n]^5 | i_1 + i_2 + i_3 + i_4 + i_5 = \lfloor \frac{5}{2}(n+1) \rfloor\} \cup \{(i_1, i_2, i_3, i_4, i_5) \in [n]^5 | i_1 + i_2 + i_3 + i_4 + i_5 = \lfloor \frac{5}{2}(n+1) \rfloor + 1\}$, then the best known lower bound on $\tau(n, 5)$ can be shown to be $|D(n, 5)| = \frac{115}{96}n^4 + \mathrm{O}(n^3)$. However, applying the techniques in this paper, a $\frac{115}{96}n^4 + \mathrm{O}(n^3)$ search algorithm for $A_{n,5}$ has not been found (our algorithm requires $\frac{5}{4}n^4 + \mathrm{O}(n^3)$ comparisons in the worst case). It may be interesting to tighten the bounds for $d > 4$.

Acknowledgments

The authors are grateful to Andy Yao for introducing this interesting problem and helpful discussions. Also, we would like to thank the referees for their valuable suggestions and comments.

References

[1] R.L. Graham, R.M. Karp, Unpublished, CA, 1968.
[2] N. Linial, M. Saks, Searching ordered structures, J. Algorithms 6 (1985) 86–103.
[3] N. Linial, M. Saks, Every poset has a central element, J. Combin. Theory Ser. A 40 (1985) 195–210.