How Much Work Space Do We Need to Find a Closest Element in an Array in Sublinear Time?

Tetsuo Asano[1] and Hirotaka Ono[2]
[1] Japan Advanced Institute of Science and Technology,
[2] Graduate School of Informatics, Nagoya University.

Abstract

This paper gives an affirmative answer to the following question: Suppose we are given a read-only array of n integers. Given a query integer, we want to find the closest element in the array. Does any array of sublinear work space help it? This is a question we address in this note. We give a data structure of $o(n)$ work space, more exactly, of $o(n)$ words of $O(\log n)$ bits, to answer the closest element query in $o(n)$ time. Furthermore, we can build the auxiliary data structure in polynomial time in n. This result can be extended to any problems for which algorithms are known for solving the problems in $o(n)$ time using a data structure of $O(n)$ space constructed in polynomial time in n. One example is the two-dimensional closest point problem. Given n points in the plane, find the point closest to an arbitrarily given query point. For this problem we know how to use the Voronoi diagram [3] for the given point set to answer the closest point query in $O(\log n)$ time. We also know that the Voronoi diagram can be constructed in $O(n \log n)$ time using $O(n \log n)$ bits. So, we can apply this method to the problem. Another example is the range query problem which counts the number of given points in an arbitrarily given region specified by a simple polygon of a constant number of vertices. On the other hand, our method cannot by applied to the closest point problem in three dimensions since the complexity of the Voronoi diagram is $\Omega(n^2)$ in the three dimensions [3].

1 Introduction

There are a number of studies on space-time tradeoffs on computational problems [1, 2, 4, 6, 7, 8, 9]. One of such results is known concerning the All-Nearest-Larger-Neighbors problem [1], defined as follows: given a read-only array of n integers, find for each element $A[i]$, the location of a nearest element whose key is strictly larger than $A[i]$. An algorithm has been developed which runs in $O(n \log_b n)$ time using $\Theta(b)$ work space for all $b = O(n)$. Another great result was given by Tompa [11] who showed under some computational model there is no polynomial time algorithm for computing the transitive closure of a given boolean matrix using only sublinear work space. For the sorting problem the optimal space-time tradeoff is known [2, 4, 6, 9].

In this paper we consider the following problem: given a read-only array of n integers of $O(\log n)$ bits each, find the element in the array that is closest to an arbitrarily given query integer. We know that $O(\log n)$ time suffices if $O(n)$ work space of $O(\log n)$ bits is available. Copy each element into an auxiliary array and sort them. Then, we can use a binary search on the sorted array to answer the query in $O(\log n)$ time. The other extreme is to find the closest element by scanning all the elements in $O(n)$ time using only $O(1)$ work space.

The question then arises whether it is possible to answer the query in $o(n)$ time by using $o(n)$ auxiliary work space. We do not care how much time is needed to construct the data structure.

We prove it is possible. More precisely, we can build a an auxiliary data structure of $o(n)$ integers of $O(\log n)$ bits each in polynomial time in n to answer such a closest-element query in $o(n)$ time. We can apply our method to any problems for which algorithms are known for solving the problems in $o(n)$ time using an auxiliary data structure of $O(n)$ space constructed in polynomial time in n. One example is the two-dimensional closest point problem. Given n points in the plane, find the point closest to an arbitrarily given query point. We can answer any closest point query in $O(\log n)$ time by computing the Voronoi diagram of $O(n \log n)$ bits. Another example is the range query problem which counts the number of given points in an arbitrarily given region specified by a simple polygon consisting of a constant number of vertices. On the other hand, our method cannot be applied to the closest point problem in three dimensions since the complexity of the Voronoi diagram is $\Omega(n^2)$ in three dimensions [3].

We also consider whether we can speedup sorting by doing some preprocessing using $o(n)$ auxiliary work space. We prove that it is not possible.

2 Closest Element Search

The problem we consider in this paper is described as follows.

[Problem 1] Given n integers of $O(\log n)$ bits each in a read-only array, create an auxiliary data structure of $o(n)$ words of $O(\log n)$ bits so that given any query value x we can find, in sublinear time, the element in the array that is closest to x.

Here is an affirmative answer to the problem. Our basic idea is to partition a given read-only array $A[0..n-1]$ of length n into blocks (or subarrays) of length b, as shown in Figure 1, and then find the element closest to a query x in each block. Hereafter, we assume for simplicity that b divides n. Otherwise we will fill in dummy elements in the last block.

For each block $A_i = A[ib..(i+1)b-1]$ of length b we prepare an auxiliary array $B_i[0..b-1]$ having indices between 0 and $b-1$ as values to keep the sorting order. That is, we compute b indices $B_i[0..b-1]$ so that the elements are sorted as $A[ib+B_i[0]] \leq A[ib+B_i[1]] \leq \cdots \leq A[ib+B_i[b-1]]$, where $0 \leq B_i[0], \ldots, B_i[b-1] \leq b-1$ are offset indices from the starting index ib. This sequence can be represented by a sequence of b integers of $\log b$ bits, taking $b \log b$ bits in total (see Figure 2).

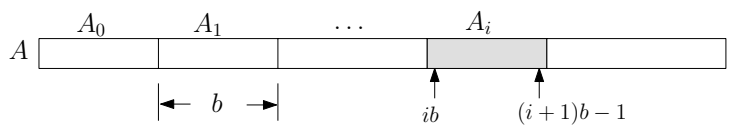

Figure 1: Partition of a given array into blocks of length b.

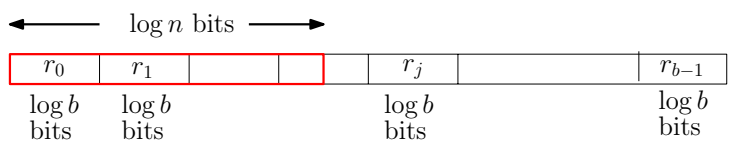

Figure 2: Offset bit array $B_i[0..b-1]$.

In this way we have $b \log b$ bits for each block. Since we have n/b blocks, the total number of bits we need is $(n/b) \times b \log b = n \log b$, which takes $n \log b / \log n = o(n)$ words of $O(\log n)$ bits.

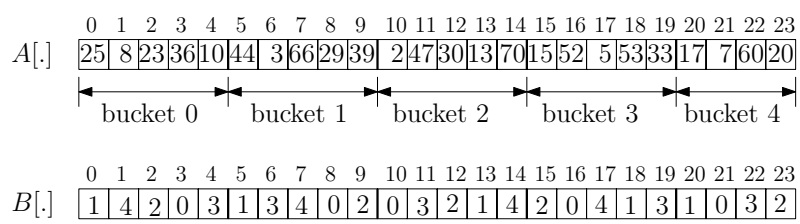

Figure 3: An example of an input array and its corresponding array $B[.]$.

Once we have the offset bit array for a block, we can access any element in the sorted sequence, say, the j-th element by taking its j-th $\log b$ bits. For the time being we assume that this can be done in constant time by preparing for each $j = 0, \ldots, b-1$ a function that takes the j-th $\log b$ bits using sliding operations appropriately in constant time by precomputing another array of length b. Later on we consider the case after removing the assumption.

Then, given a query value x, we search for the element closest to x in each block A_i. We perform binary search to find the element in each block A_i that is closest to x. The binary search is done by

comparing the middle element. As it was explained earlier, we can access to any element in the sorted sequence in constant time. Thus, the binary search is done in $O(\log b)$ time. So, the total query time is $O((n/b)\log b) = O((n\log b)/b) = o(n)$.

The construction time of the data structure is bounded by $O((n/b) \times b\log b) = O(n\log b)$ since we just sort each block using an auxiliary work space of $O(b)$.

Summarizing the results, we have the following theorem.

Theorem 2.1 *Given a read-only array of n integers of $O(\log n)$ bits each, we can construct a data structure of $O(n\log b/\log n)$ words of $O(\log n)$ bits in $O(n\log b)$ time to find the element in the array closest to an arbitrarily given integer x in $O((n\log b)/b)$ time for any $\omega(1) \le b = O(2^{\log^{\varepsilon} n})$ and any $0 < \varepsilon < 1$ under the assumption that the floor function and all sliding operations mentioned above are done in constant time.*

To achieve our goal of $o(n)$ work space and $o(n)$ query time, we must have $\log b = o(\log n)$ and thus $b = n^{\varepsilon}$ for any $\varepsilon, 1 > \varepsilon > 0$, is too large. One candidate for a valid value of b is $b = O(2^{\log^{\varepsilon} n})$ for any $0 < \varepsilon < 1$. Then, we can achieve sublinear query time $O((n\log 2^{\log^{\varepsilon} n})/2^{\log^{\varepsilon} n)}) = O((n\log^{\varepsilon} n)/2^{\log^{\varepsilon} n)})$ and sublinear work space $O((n\log^{\varepsilon} n)/\log n)$.

What about the case of no such assumption? We show it is possible to implement the floor function and sliding operations both in constant time using $O(b)$ work space.

First we have to compute block number. One way to avoid the floor function is to assume that b divides n. If it does not, then we can add some dummy elements. Or, since we proceed in the array in the increasing order of their block numbers, we do not need to use the floor function to compute the block number. Now, suppose we are dealing with the i-th block A_i starting from ib-th element to $(i+1)b-1$-th element of the array. The block A_i is partitioned into sub-blocks, each containing $s = \lfloor \log n/\log b \rfloor$ numbers of $\log b$ bits between 0 and $b-1$. We pack those s numbers of $\log b$ bits from the head of the sub-block. Then, given an index j between i and $i+b-1$, we have to find the sub-block in which its corresponding number $j-i$ is contained. If we can use the floor function, then the number is given by $\lfloor (j-i)/s \rfloor$. We can find the number by precomputing those results in an array of length b. In this we can avoid the floor function.

How about sliding operations? Suppose we want to know the content of the k-th $\log b$ bits in the j-th sub-block in the i-th block. To extract the information, we prepare s masks so that each of the $\log b$ bits can be extracted. See Figure 4. For this purpose we need an additional work space of $O(s\log n) = O(\log^2 n/\log b)$ bits. We also compute s numbers each corresponding to sliding operations to slide the part to fit to the lowest bit of $\log n$ integer. Once we compute them, we can extract the content by dividing the masked integer by the prepared integer. This is done in constant time. Since we have $n\log b \ge \log^2 n/\log b)$, we can neglect the work space mentioned above. Now, the theorem is restated.

Theorem 2.2 *Given a read-only array of n integers of $O(\log n)$ bits each, we can construct a data structure of $O(n\log b/\log n)$ words of $O(\log n)$ bits in $O(n\log b\log n)$ time to find the element in the array closest to an arbitrarily given integer x in $O((n\log b)/b)$ time for any $\omega(1) \le b = O(2^{\log^{\varepsilon} n})$ and any $0 < \varepsilon < 1$.*

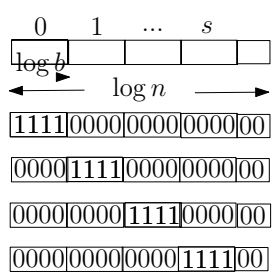

Figure 4: The s different masks to extract s numbers of $\log b$ bits in a sub-block.

Here are our conjectures:

Conjecture 1: For any given read-only array of n integers there is no data structure of $O(n)$ bits to find the element in the array in $o(n)$ time that is closest to an arbitrarily given real value x.

Conjecture 2: For any given read-only array of n integers there is no data structure of $O(n^{1-\varepsilon})$ words of $O(\log n)$ bits to find the element in the array in $O(n^{1-\varepsilon})$ time for any small $\varepsilon > 0$ that is closest to an arbitrarily given integer x.

3 Generalization of the Result

Now, we are going to extend the result above into a class of more general problems. Consider a class of problems which is described by n input elements such as numbers or points and for which algorithms are known for answering any query on the input set in $Q(n) = o(n)$ time using $O(n)$ work space. One such example is the one dimensional range query problem in which a set of n integers are given and we are requested to count the number of the elements contained in an arbitrarily given interval specified by two integers. Another example is the two-dimensional closest point problem in which n points are given and for any given query point p we need to find the point in the set that is closest to p. For the two problems above, algorithms which run in $Q(n) = O(\log n)$ query time and $O(n)$ work space, i.e., $O(n \log n)$ bits.

Theorem 3.1 *Let $P(n)$ be a problem defined by n elements for which an algorithm is known for answering any query stated above in $Q(n) = o(n)$ time using an auxiliary data structure of $O(n)$ words of $O(\log n)$ bits each after preprocessing in polynomial time $C(n)$. Then, for any $\omega(1) \leq b = O(2^{\log^{\varepsilon} n})$ and any $0 < \varepsilon < 1$ we can construct a data structure of $O((n \log b)/\log n) = o(n)$ words of $O(\log n)$ bits in polynomial time $O((n/c)C(n/c))$ so that we can answer any such query in $O((n/b)Q(n/b)) = o(n)$ time.*

We summarize the related results in the following table.

(1) Closest point problem in 1D, Range query problem in 1D, and Closest point problem in 2D [3]:

	Known algorithm	Our method
Query time	$Q(n) = O(\log n)$	$O((n/b)\log b)$
Work space	$W(n) = O(n)$	$O(n \log b/\log n)$
Construction time	$C(n) = O(n \log n)$	$O(n \log b)$

(2) 2D rectangular query problem [5]:

	Known algorithm	Our method
Query time	$Q(n) = O(\sqrt{n})$	$O(n/\sqrt{b})$
Work space	$W(n) = O(n)$	$O((n/b)\log b/\log n)$
Construction time	$C(n) = O(n \log n)$	$O(n \log b)$

(3) 2D triangular range query problem [12]:

	Known algorithm	Our method
Query time	$Q(n) = O(\sqrt{n}\log^3 n)$	$O(n \log^3 b/\sqrt{b}))$
Work space	$W(n) = O(n)$	$O(n \log^2 b/\log n))$
Construction time	$C(n) =$ polynomial time	polynomial time

4 Extension to Higher Dimensions

Given a set of n points in the d-dimensional space, can we build an auxiliary data structure of $o(n)$ work space that can find the point in the given set that is closest to any given point in $o(n)$ time? A simple extension of our solution for the two-dimensional case does not work since we need $\Theta(n^{\lceil d/2 \rceil})$ space to store the Voronoi diagram in the d-dimensional space [3, 10]. Extension of range query algorithm in higher dimensions is also difficult since the best known algorithm uses $O(n \log^{d-1} n)$ storage to answer a rectangular query in $O(\log^d n)$ time [10].

5 Space-Tine Tradeoff

Consider the following problem:

Problem 2 Suppose we are given n integers in a read-only array. For an input sequence $x_0, x_1, \ldots, x_{m-1}$ of m integers, report for each $x_i, i = 0, \ldots, m-1$ the element in the array closest to x_i.

A naive algorithm that examines all the elements for each x_i runs in $\Theta(mn)$ time using $O(1)$ work space. So, the space-time product is $\Theta(mn)$. If we can use $O(n)$ work space, we copy each element into the read/write array and then sort them. Then, for each x_i we can find the closest element by binary search. This algorithm runs in $O(n \log n + m \log n)$ time and $O(n)$ space. Thus, the space-time product is $(n(m+n)\log n)$, which is worse than the naive algorithm.

Is there any algorithm that achieves $o(mn)$ for space-time product? We give an affirmative answer to this question.

Our algorithm uses $O(b \log b / \log n)$ space, i.e., $O(b \log b / \log n)$ words of $O(\log n)$ bits each, where b is a parameter satisfying $b = o(n)$. We first divide a given sequence of m integers into m/b subsequences of length b. Then, we find for each element of the subsequence the closest element by applying the previous algorithm and auxiliary data structure.

Let $x_{ib}, x_{ib+1}, \ldots, x_{(i+1)b-1}$ be such a subsequence. We put those elements into an array $B[0..b-1]$ and sort them in the increasing order. Let $B[p[0]] \leq B[p[1]] \leq \cdots \leq B[p[b-1]]$ be the resulting sorted sequence. Then, for each element $a[i]$ in the read-only array we locate it in the sorted sequence. We can compute, for each $B[p[j]]$, the lower highest element $a[i]$ such that $a[i] \leq B[p[j]]$ and there is no element between $a[i]$ and $B[p[j]]$, and the upper lowest element $a[i']$ such that $a[i'] \geq B[p[j]]$ and there is no element between $a[i']$ and $B[p[j]]$. Let $LH[j]$ and $UL[j]$ be candidates for the lower highest and upper lower element for $B[p[j]]$.

We locate $a[i]$ in the sequence $< B[p[0]], B[p[1]], \ldots, B[p[b-1]] >$. Whenever $B[p[j-1]] \leq a[i] \leq B[p[j]]$ and $a[LH[j]] < a[i]$ then we update $LH[j]$ by i. Similarly, whenever $B[p[j]] \geq a[i] \geq B[p[j+1]]$ and $a[UL[j]] > a[i]$ then we update $UL[j]$ by i. After locating $a[ib], a[ib+1], \ldots, a[(i+1)b-1]$, we do the following operations:

for each $j = 0, 1, \ldots, b-1$ **do**
  if $LH[j]$ is undefined **then** $LH[j] = LH[j-1]$;
for each $j = b-1, b-2, \ldots, 0$ **do**
  if $UL[j]$ is undefined **then** $UL[j] = UL[j-1]$;

In the algorithm above we implicitly assumed that no two elements in the array $B[0..b-1]$ coincide and no two elements in the array $a[0..n-1]$ are the same. The algorithm certainly computes the lower highest and upper lowest element for each $B[p[j]]$. If the lower highest element $a[i]$ of $B[p[j]]$ is in the interval $[B[p[j-1]], B[p[j]]]$ then it is certainly found in the algorithm. If $a[i] < B[p[j-1]]$ then this implies that there is no $a[i']$ between $B[p[j-1]]$ and $B[p[j]]$ and thus in this case $B[p[j-1]]$ and $B[p[j]]$ shares the lower highest element. The **for** loop above does this processing.

Finally, we compare the lower highest element and higher lowest element for each $B[p[j]]$ and report the closer one as an output.

The running time of the algorithm is given by

$\frac{m}{b} \times O(b \log b + n \log b) = O(\frac{mn}{b} \log b)$,

and the work space required is

$O(b \log b / \log n)$

as was seen previously. So, the space time product is

$S \times T = O(\frac{mn}{b} \log b) \times O(\frac{b \log b}{\log n}) = O(mn \frac{log^2 b}{\log n})$.

If $b = o(2^{\sqrt{\log n}})$ then we have $\log^2 b = o((\sqrt{\log n})^2) = o(\log n)$ and thus we have

$S \times T = o(mn)$.

Theorem 5.1 *There is an algorithm for solving the following problem with space-time product being $o(mn)$.*
Problem: Suppose we are given n integers in a read-only array. For any input sequence $x_0, x_1, \ldots, x_{m-1}$ of m integers, report for each $x_i, i = 0, \ldots, m-1$ the element in the array closest to x_i.

6 Speeding up Sort Using Partial Information

In this section we consider how much we can speedup sorting a given array of n elements by appropriate preprocessing using work space of S bits in total. More precisely, given a read-only array of n integers of $O(\log n)$ bits each, by preprocessing them and storing the results in an auxiliary data structure of S bits in total, we would like to output those input data in a sorted order faster than usual, i.e., in $o(n \log n)$ time even in the worst case using only comparisons between input elements. Our conclusion is that it is impossible to speedup sorting unless work space of $O(n \log n)$ bits is available, that is, unless we sort a given array in advance and store the sorted sequence in a separate array of $O(n \log n)$ bits.

Theorem 6.1 *Given n integers of $O(\log n)$ bits in a read-only array, we would like to prepare some other array of S bits to speedup sorting those values. Sorting the n integers based on comparisons between input integers takes $\Omega(n \log n)$ time in the worst case as long as $S = o(n \log n)$.*

Proof: Initially, there are $n!$ different permutations, which can be partitioned into 2^S equivalent classes using S bits. Then, an algorithm in T steps is followed to get a unique permutation. Since we can explore at most two possibilities in each step, our search space is bounded by 2^T. Thus, we have

$$\frac{n!}{2^S \times 2^T} \leq 1.$$

So, we have

$$2^{S+T} \geq n! \simeq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \geq \left(\frac{n}{e}\right)^n = 2^{n \log(n/e)},$$

which implies $S+T \geq n \log(n/e)$. This means that $T = \Omega(n \log n)$ must hold as long as $S = o(n \log n)$.

7 Conclusions and Future Work

In this paper we have considered how much work space should be used to find the element in a given read-only array of n integers that is closest to an arbitrarily given query value. We devised an algorithm and auxiliary data structure of $o(n)$ words of $O(\log n)$ bits each to answer such a query in $o(n)$ time. We found a considerable gap in space complexity between $O(n)$ and $o(n)$. More precisely, query time is $O(\log n)$ using binary search if $O(n)$ work space is available, but it is almost $O(n)$ even if we use almost $O(n)$ work space. We extended this result to a two dimensional case where n points are given in a read-only array and the task is to find the point closes = t to an arbitrarily given query point. We have also obtained similar results for two-dimensional range counting problems.

No lower bound on the time and space tradeoff is known. One of the future works is to establish some lower bound on this problem.

Acknowledgment

The work was partially supported by KAKENHI No.24106008 of the Ministry of Education, Science, Sports and Culture. The first author would like to thank Masafumi Yamashita, David Kirkpatrick, Yota Otachi, and Ryuhei Uehara for their stimulating discussion.

References

[1] T. Asano and D. G. Kirkpatrick: "Time-Space Tradeoffs for All-Nearest-Larger-Neighbors Problems," Proc. Workshop on Algorithm and Data structure, pp:71-72, 2013.

[2] T. Asano, A. Elmasry, J. Katajainen: "Priority Queues and Sorting for Read-Only Data," Proc. TAMC (Int. Conf. on Theory and Applications of Models of Computation) 2013, pp:32-41, 2013.

[3] F. Aurenhammer: "Voronoi Diagrams – A Survey of a Fundamental Geometric Data Structure," ACM Computing Surveys, 23(3), pp:345-405, 1991.

[4] P. Beame, "A general sequential time-space tradeoff for finding unique elements," *SIAM Journal on Computing*, 20, pp:270-277, 1991.

[5] J. L. Bentley: "Multidimensional binary search trees used for associative searching," Commun. ACM 18, 9, pp:509-517, 1975.

[6] A. Borodin, M.J. Fischer, D.G. Kirkpatrick, N. Lynch, and M. Tompa: "A time-space tradeoff for sorting on non-oblivious machines," J. of Computer and System Science, 22, 3, pp:351-364, 1981.

[7] G. N. Frederickson, "Upper and lower bounds for time-space tradeoffs in sorting and selection," *Journal of Computer and System Sciences*, 34, pp:19-26, 1987.

[8] J. Munro and M. Paterson, "Selection and sorting with limited storage," Theoret. Comput. Sci., 12, pp:315-323, 1980.

[9] J. Pagter and T. Rauhe, "Optimal time-space tradeoffs for sorting," *IEEE Annual Symposium on Foundations of Computer Science*, pp:264-268, 1998.

[10] F.P. Preparata and M.I. Shamos: "Computational Geometry: An Introduction," Springer-Verlag, 1985.

[11] M. Tompa: "Two familiar transitive closure algorithms which admit no polynomial time, sublinear space implementations," SIAM J. on Computing, 11(1), pp.130-137, 1982.

[12] E. Welzl: "Partition Trees for Triangle Counting and Other Range Searching Problems," SCG '88 Proceedings of the fourth annual symposium on Computational Geometry, pp:23-33, 1988.