UNDERSTANDING THE COMPLEXITY OF INTERPOLATION SEARCH *

Yehoshua PERL
Department of Mathematics and Computer Science, Bar-Ilan University, Ramat Gan, Israel

Edward M. REINGOLD **
Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel

Received 5 January 1977, revised version received 12 September 1977

Interpolation search, analysis of algorithms

1. Introduction

Interpolation search of an ordered table, due originally to Peterson [4], has recently been analyzed. The average number of probes (key comparisons) required when the n keys are drawn from a uniform distribution has been shown to be $\lg \lg n$ [1] in two (independent) papers. Yao and Yao [6] use a very complex combinatorial argument, and show further that interpolation search is, in a sense, optimal. In Perl, Itai, and Avni [3] the analysis is based on proving that the expected error of the j-th probe is at most the square root of the expected error of the $(j-1)$st probe. From this it follows that the expected error in the $(\lg \lg n)$th probe is $O(1)$, and applying martingale techniques from advanced probability theory yields $\lg \lg n$ probes as the average. Neither analysis gives any simple intuition as to why interpolation search has $\lg \lg n$ average behavior.

In this paper we show that a quadratic application of binary search yields a variant of conventional interpolation search that is easily shown to use about 2.4 $\lg \lg n$ probes on the average. The analysis of this variant requires only a simple argument, and the most basic results of probability theory. It thus clarifies the behavior of interpolation search and as such has great pedagogical value.

2. Interpolation search

Interpolation search, as described by Peterson in [4] (see also [2]), works as follows. Suppose we have an ordered table of numeric keys $x_1 < x_2 < \ldots < x_n$, where the keys are drawn independently from a uniform distribution over the range (x_0, x_{n+1}), and suppose that we are to find the index i such that $x_i \leqslant y < x_{i+1}$ for a given value y, $x_0 < y < x_{n+1}$. Since the keys are uniformly distributed, we may expect that about $p = (y - x_0)/(x_{n+1} - x_0)$ of the keys are less than y. Thus we make a probe $y : x_{\lceil pn \rceil}$, and if they are not equal we then apply the same method recursively to the appropriate subtable, $x_1, \ldots, x_{\lceil pn \rceil - 1}$ if $y < x_{\lceil pn \rceil}$, and $x_{\lceil pn \rceil + 1}, \ldots, x_n$ if $y > x_{\lceil pn \rceil}$. In the worst case this method can, of course, require n probes, but on the average only $\lg \lg n$ probes are required ([3] and [6]).

3. Quadratic binary search

The $\lg \lg n$ behavior suggests that it may be possible to view interpolation search as a quadratic application of the binary search technique whose average and

* This research was supported in part by the United States National Science Foundation, Grant GJ–41538.

** On leave from Department of Computer Science, University of Illinois at Urbana–Champaign, Urbana, Il. 61801 USA.

[1] We use $\lg x$ to represent the binary logarithm of x.

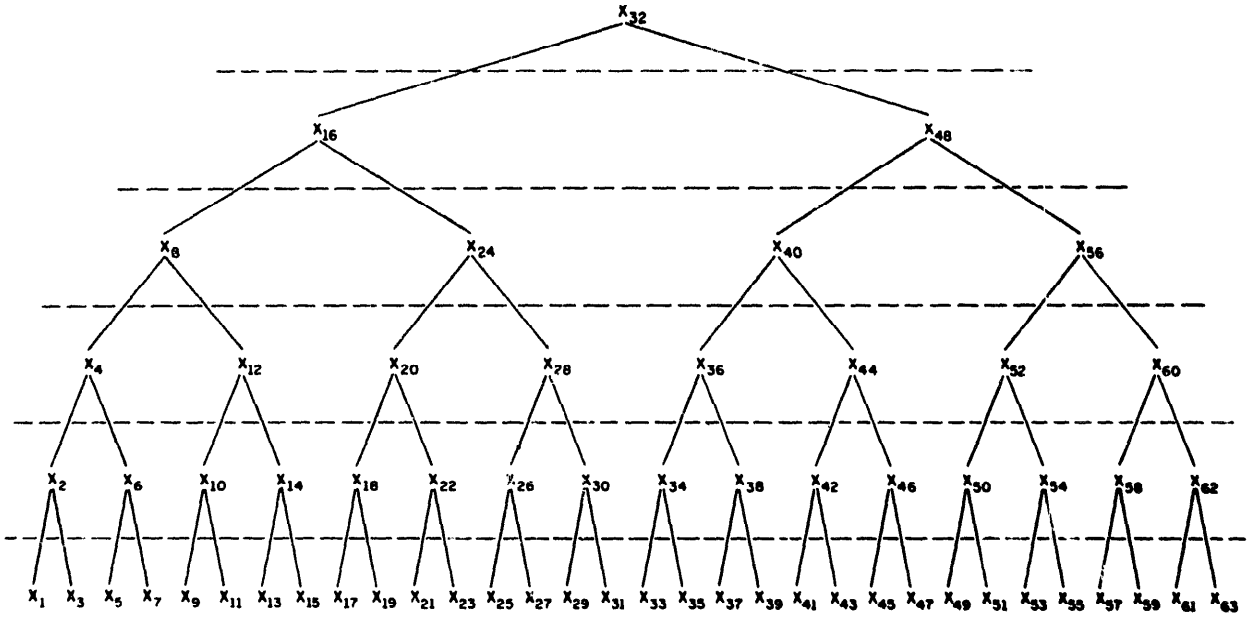

Fig. 1. The binary tree structure implicitly imposed on a table by binary search. The dashed lines separate the levels of the tree.

worst case behavior is well known to be about $\lg n$ (see [2] or [5]). Consider the sequence of keys probed when using binary search on a table of n keys. Since binary search implicitly imposes the structure of a binary tree of $\lg n$ levels on the table (see Fig. 1), the search corresponds to scanning a path down the tree from the root. The length of any such path is bounded by $\lg n$, and this explains the logarithmic behavior of binary search.

We can interpret binary search as performing a sequential search of the vertices on the path down the tree: this suggests that a $\lg \lg n$ search might result from searching the vertices on the path by binary search. Unfortunately, we do not know the path itself before having applied binary search to the entire table of n keys. But, we *can* apply binary search to the *levels* of the tree (shown in Fig. 1 by dashed lines) i.e. "cut" the tree at the middle level. It happens that the resulting algorithm will be more natural if we concentrate on subtrees instead of levels; in other words, we will use all the keys in the top $\frac{1}{2}\lg n$ levels of the tree to partition the tree into $\sqrt{n}$ subtrees of about $\frac{1}{2}\lg n$ levels each. This leads us to Fig. 2. By imagining that each of the subtrees in the lower triangles is drawn in the same way recursively, Fig. 2 can be regarded as considering the table to be a multiway tree of $\lg \lg n$ levels (see [2] or [5]).

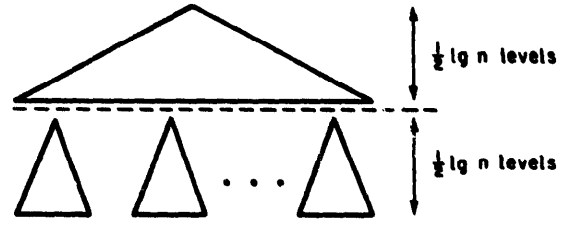

Fig. 2. A schematic version of the tree in Fig. 1. Imagine each of the lower triangles to be drawn in the same way, recursively.

Each triangle shown in Fig. 2 consists of a subtree containing about $\sqrt{n}$ keys from the table. We will see in the next section that using an interpolation probe followed by a sequential search in the upper triangle to determine which of the lower triangles must be searched requires only a constant number of probes, on the average. Applying this method recursively to the appropriate lower triangle (subtable) yields a quadratic binary search that examines at most $\lg \lg n$ levels of the tree and thus uses $O(\log \log n)$ probes on the average.

4. Binary interpolation search

Actually, it is more convenient to shift the table so that the first probe in the subtable on which we are

interpolating (the upper triangle) is at the key $x_{\lceil pn \rceil}$, as in the conventional interpolation search. This then is the binary interpolation search for y, $x_0 < y < x_{n+1}$ in the table $x_1 < x_2 < \ldots < x_n$: The first probe is the comparison $y : x_{\lceil pn \rceil}$, $p = (y - x_0)/(x_{n+1} - x_0)$. If $y > x_{\lceil pn \rceil}$ then y is successively compared with $x_{\lceil pn + i\sqrt{n} \rceil}$, $i = 1, 2, \ldots$ until the smallest i is found, such that $y < x_{\lceil pn + i\sqrt{n} \rceil}$. Similarly, if $y < x_{\lfloor pn \rfloor}$, y is compared successively with $x_{\lceil pn - i\sqrt{n} \rceil}$, $i = 1, 2, \ldots$ In any case, the subtable of size $\sqrt{n}$ thus found is then searched by applying the same technique recursively.

We must determine the expected number C of probes required to determine the subtable of $\sqrt{n}$ keys. As in [3] and [6], assume that the n keys $x_1, x_2, \ldots, x_n$ are drawn independently from a uniform distribution over (x_0, x_{n+1}). Then, since the x_i are independent and since $p = (y - x_0)/(x_{n+1} - x_0)$ is the probability that any one of them is less than or equal to y, the probability that exactly j out of the n are less than y is obviously $\binom{n}{j} p^j (1-p)^{n-j}$ (see, for example, Feller [1]). The number of keys expected to be less than or equal to y is thus

$$\mu = \sum_{j=0}^{n} j \binom{n}{j} p^j (1-p)^{n-j} = pn ,$$

with variance

$$\sigma^2 = \sum_{j=0}^{n} (j - pn)^2 \binom{n}{j} p^j (1-p)^{n-j} = p(1-p)\, n .$$

Since $0 < p < 1$, we have $p(1-p) \leqslant \frac{1}{4}$ and hence $\sigma^2 \leqslant \frac{1}{4}n$.
We find C as follows.

$$C = \sum_{i \geqslant 1} i \cdot \Pr(\text{exactly } i \text{ probes are used to determine the subtable})$$

$$= \sum_{i \geqslant 1} \Pr(\text{at least } i \text{ probes are used to determine the subtable})$$

But at least two probes are always used (at $x_{\lceil pn \rceil}$ and $x_{\lceil pn \pm \sqrt{n} \rceil}$), and for $i \geqslant 3$ we have, by definition

Pr (at least i probes are used to determine the subtable)

$$< \Pr(|(\text{location of } y) - pn| > (i-2)\sqrt{n})$$

which is bounded above by $1/4(i-2)^2$ by Chebyshev's inequality (see, for example, [1])

$$\Pr(|x - \mu| \geqslant t) \leqslant \frac{\sigma^2}{t^2} ,$$

for a random variable x with mean μ and variance σ^2. Thus

$$C \leqslant 2 + \sum_{i \geqslant 3} \frac{1}{4} \frac{1}{(i-2)^2} = 2 + \frac{\pi^2}{24} \approx 2.4 .$$

Let $T(n)$ be the average number of probes used by binary interpolation in searching a random table of n keys for y. Since the subtables of size $\sqrt{n}$ are again random, we have

$$T(n) \leqslant C + T(\sqrt{n})$$

and thus $T(n) \leqslant 2.4 \lg \lg n$.

5. Concluding remarks

It is possible to sharpen the foregoing analysis by using more accurate approximations than Chebyshev's inequality. For example, by using the DeMoivre-Laplace limit theorem (see, for example, [1]) we find that for $i \geqslant 3$

Pr (at least i probes are used to determine the subtable)

$$\approx 1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{i-2}{\sqrt{p(1-p)}}} e^{-\frac{1}{2}x^2} \, dx .$$

Using estimates for the "tail" of the normal distribution (again, see for example [1]) we have that this is

$$\leqslant \frac{1}{t\sqrt{2\pi}} e^{-\frac{1}{2}t^2} \Bigg|_{t = \frac{i-2}{\sqrt{p(1-p)}}} ,$$

and thus

$$\leqslant \frac{1}{2\sqrt{2\pi}} \frac{1}{i-2} e^{-2(i-2)^2}$$

since $p(1-p) \leqslant \frac{1}{4}$. This gives

$$C \leqslant 2 + \sum_{i \geqslant 3} \frac{1}{2\sqrt{2\pi}} \frac{1}{i-2} e^{-2(i-2)^2} \approx 2.027029 .$$

There is a negligible probability that conventional interpolation search will require proportional to n probes in the worst case. The number of probes required in the worst case by binary interpolation

search is obviously only $\sqrt{n} + O(n^{1/4})$. This is unimportant, however, for we can get a search technique whose cost is at most twice optimal in both the average *and* worst cases: apply conventional interpolation search and binary search alternatingly or in parallel. This requires at most 2 lg lg n and 2 lg n probes in the average and worst cases, respectively. Such techniques to improve the worst case at the expense of the average case are of interest only if the distribution in the table is suspect. If the distribution is uniform, there is little utility in trying to improve the worst case of conventional interpolation search, since as shown in [3], the probability that even a few more than lg lg n probes are required is very small.

The importance of the binary interpolation search is not its behavior, but rather that it contributes to the understanding of the behavior of interpolation search.

References

[1] W. Feller, An Introduction to Probability Theory and its Applications (3rd edition) (John Wiley, New York, 1968).

[2] D.E. Knuth, The Art of Computer Programming, Volume 3, Searching and Sorting (Addison–Wesley, Reading, MA, 1973).

[3] Y. Perl, A. Itai and H. Avni, Interpolation Search – A log log n Search, Comm. ACM, to appear.

[4] W.W. Peterson, Addressing for random access storage, IBM J. Res. Develop. 1 (1957) 131–132.

[5] E.M. Reingold, J. Nievergelt and N. Deo, Combinatorial Algorithms: Theory and Practice (Prentice-Hall, Englewood Cliffs, N.J., 1977).

[6] A.C. Yao and F.F. Yao, The Complexity of Searching an Ordered Random Table, Proceedings of the Seventeenth Annual Symposium on Foundations of Computer Science (1976) 173–177.