# Dynamic Interpolation Search

KURT MEHLHORN

*Max-Planck-Institut für Informatik and Universität des Saarlandes, Saarlandes, Germany*

AND

ATHANASIOS TSAKALIDIS

*Computer Technology Institute, Patras, Greece*

Abstract. A new data structure called interpolation search tree (IST) is presented which supports interpolation search and insertions and deletions. Amortized insertion and deletion cost is $O(\log n)$. The expected search time in a random file is $O(\log\log n)$. This is not only true for the uniform distribution but for a wide class of probability distributions.

Categories and Subject Descriptors: E.1 [**Data Structures**]: *trees*; F.2 [**Analysis of Algorithms and Problem Complexity**]

General Terms: Algorithms

Additional Key Words and Phrases: Dynamization, interpolation search, search tree

## 1. *Introduction*

Interpolation search was first suggested by Peterson [11] as a method for searching in sorted sequences stored in contiguous storage locations. The expected running time of interpolation search on random files (generated according to the *uniform* distribution) of size $n$ is $\Theta(\log\log n)$. This was shown by Yao and Yao [14], Pearl et al. [9], and Gonnet et al. [3]. A very intuitive explanation of the behavior of interpolation search can be found in Pearl and Reingold [10] (see also Mehlhorn [7]).

Dynamic interpolation search, that is, data structures that support insertions and deletions as well as interpolation search, was discussed by Frederickson [2], and Itai et al. [5]. Frederickson presents an implicit data structure that supports insertions and deletions in time $O(n^{\epsilon})$, $\epsilon > 0$, and interpolation search with expected time $O(\log\log n)$. The structure of Itai et al. has expected insertion

A preliminary version of this paper was presented at ICALP '85 and published in *Proceedings of ICALP '85*. Lecture Notes in Computer Science, vol. 194. Springer-Verlag, New York, 1985, pp. 22–23.

Authors addresses: K. Mehlhorn, Max-Planck-Institut für Informatik und Fachbereich Informatik, Universität des Saarlandes, D-6600 Saarbrücken, Saarlandes, Germany; A. Tsakalidis, Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece.

Journal of the Association for Computing Machinery, Vol. 40, No. 3, July 1993, pp 621–634.

time $O(\log n)$ and amortized insertion time $O((\log n)^2)$. It is claimed to support interpolation search, although no formal analysis of its expected behavior is given. Both papers assume that files are generated according to the uniform distribution. Other results on dynamic manipulation of sequential files are given by Franklin [1], Melville and Gries [8], Hofri and Konheim [4], and Willard [12].

Willard [13] extended interpolation search into another direction: *nonuniform* distributions. He showed that the $\log\log n$ asymptotic retrieval time of interpolation search does not remain in force for most nonuniform probability distributions. However, he also introduced a variant of interpolation search and showed that its expected running time is $O(\log\log n)$ on *static* $\mu$-random files where $\mu$ is any *regular* probability density. A density $\mu$ is regular if there are constants $b_1$, $b_2$, $b_3$, $b_4$ such that $\mu(x) = 0$ for $x < b_1$ or $x > b_2$ and $\mu(x) \geq b_3 > 0$ and $|\mu'(x)| \leq b_4$ for $b_1 \leq x \leq b_2$. A file is *$\mu$-random* if its elements are drawn independently according to density $\mu$. It is important to observe that Willard's algorithm does *not* have to know the density $\mu$. Rather, its running time is $O(\log\log n)$ on $\mu$-random files provided that $\mu$ is *regular*. Thus, his algorithm is fairly robust.

In this paper, we combine and extend the research mentioned above. We present a new data structure called *interpolation search tree* (IST) that has the following properties:

(1) It requires space $O(n)$ for a file of cardinality $n$. We want to point out that most of the prior work on interpolation search, in particular [11] and [13], assumes an unindexed file and hence uses precisely $n$ units of memory.

(2) The amortized insertion and deletion cost is $O(\log n)$; the expected amortized insertion and deletion cost is $O(\log\log n)$.

(3) The expected search time on files generated by $\mu$-random insertions and random deletions is $O(\log\log n)$ provided that $\mu$ is a *smooth* density. An insertion is $\mu$-random if the key to be inserted is drawn from density $\mu$. A deletion is random if every key present in the current file is equally likely to be deleted. These notions of randomness are called $I_r$ and $D_r$ respectively in Knuth [6].

A density $\mu$ is *smooth* for a parameter $\alpha$, $\frac{1}{2} \leq \alpha < 1$, if there are constants $a$, $b$, and $d$ such that $\mu(x) = 0$ for $x < a$ and $x > b$ and such that for all $c_1, c_2, c_3, a \leq c_1 < c_2 < c_3 \leq b$, and all integers $m$ and $n$ with $m = \lceil n^\alpha \rceil$

$$\int_{c_2-(c_3-c_1)/m}^{c_2} \mu[c_1, c_3](x)\, dx \leq dn^{-1/2},$$

where $\mu[c_1, c_3](x) = 0$ for $x < c_1$ or $x > c_3$ and $\mu[c_1, c_3](x) = \mu(x)/p$ for $c_1 \leq x \leq c_3$ where $p = \int_{c_1}^{c_3} \mu(x)\, dx$.

We want to point out that every regular (in Willard's sense) density is smooth and that there are smooth densities which are not regular.

(4) The worst-case search time is $O((\log n)^2)$.

(5) The data structure supports sequential access in linear time and operations Predecessor, Successor, and Min in time $O(1)$. In particular, it can be used as a priority queue.

This paper is structured as follows: In Section 2, interpolation search trees are defined and the insertion and deletion algorithms are described and analyzed. In Section 3, we then discuss the retrieval algorithm.

## 2. *The Interpolation Search Tree*

In this section, we introduce interpolation search trees (IST) and discuss the insertion and deletion algorithms.

An IST is a multiway tree where the degree of a node depends on the cardinality of the set stored below it. More precisely, the degree of the root is $\Theta(\sqrt{n})$. The root splits a file into $\Theta(\sqrt{n})$ subfiles. We do not postulate that the subfiles have about equal sizes and in fact our insertion and deletion algorithms will not enforce any such condition. In *ideal* ISTs, the subfiles have about equal sizes. In this case the subfiles have size $\Theta(\sqrt{n})$ and hence the son's of the root of an ideal IST have degree

$$\Theta\left(\sqrt{\sqrt{n}}\right).$$

In particular, ideal ISTs have depth $O(\log\log n)$. The search algorithm for ISTs uses interpolation search in every node of the IST in order to locate the subfile where the search should be continued. Since we want to use interpolation search in the nodes of an IST it is necessary to use arrays for the nodes of an IST. In addition, we associate with each node an approximation to the inverse distribution function that allows us to interpolate perfectly for a sample of values of size $n^{\alpha}$ where $\alpha$, $\frac{1}{2} \leq \alpha < 1$, is a parameter.

*Definition* 1. Let $a$ and $b$ be reals, $a < b$. An interpolation search tree (IST) with boundaries $a$ and $b$ for a set $S = \{x_1 < x_2 < \cdots < x_n\} \subseteq [a, b]$ of $n$ elements consists of

(1) A set REP of representatives $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$, $i_1 < i_2 < \cdots < i_k$ stored in an array REP$[1 \cdots k]$, that is, REP$[j] = x_{i_j}$. Furthermore, $k$ satisfies $\sqrt{n}/2 \leq k \leq 2\sqrt{n}$.
(2) Interpolation search trees $T_1, \ldots, T_{k+1}$ for the subfiles $S_1, \ldots, S_{k+1}$ where $S_j = \{x_{i_{j-1}+1}, \ldots, x_{i_j - 1}\}$ for $2 \leq j \leq k$, $S_1 = \{x_1, \ldots, x_{i_1 - 1}\}$, and $S_{k+1} = \{x_{i_k + 1}, \ldots, x_n\}$. Furthermore, tree $T_j$, $2 \leq j \leq k$, has boundaries $x_{i_{j-1}}$ and $x_{i_j}$, $T_1$ has boundaries $a$ and $x_1$, and $T_{k+1}$ has boundaries $x_k$ and $b$.
(3) An array ID$[1 \cdots m]$, where $m$ is some integer, with ID$[i] = j$ iff REP$[j] < a + i(b - a)/m \leq$ REP$[j + 1]$.

The array REP contains a sample of the file $S$. In ideal ISTs, we require this sample to be equally spaced. Ideal ISTs will be built by the insertion and deletion algorithms at *appropriate* time intervals.

For reasons to become clear below, we require $m = n^{\alpha}$ for some $\alpha$, $\frac{1}{2} \leq \alpha < 1$.

*Definition* 2. An IST with parameter $\alpha$ for set $S$, $|S| = n$, is *ideal* if $i_j = j\lfloor\sqrt{n}\rfloor$ for all $j \geq 1$, if $m = \lceil n^{\alpha} \rceil$, and if the interpolation search trees $T_1, \ldots, T_{k+1}$ are again ideal.

The following lemma is almost immediate.

LEMMA 1. *Let $\frac{1}{2} \leq \alpha < 1$. Then an ideal IST for an ordered set $S$, $|S| = n$, can be built in time $O(n)$ and requires space $O(n)$. It has depth $O(\log\log n)$.*

PROOF. Let $d(n)$ be the depth of an ideal IST for a set of size $n$. Then $d(n) = 1 + d(\sqrt{n})$ and hence $d(n) = O(\log\log n)$. If $P(n)$ is the time to built an ideal IST for a file of size $n$, then clearly $P(n) = O(n^{\alpha}) + \sqrt{n}\,P(\sqrt{n})$. Let $\hat{P}(n) = P(n)/n$. Then $n\hat{P}(n) = O(n^{\alpha}) + n\hat{P}(\sqrt{n})$ and hence $\hat{P}(n) = O(n^{\alpha-1}) + \hat{P}(\sqrt{n})$. Thus, assuming $n = 2^{2^r}$,

$$\hat{P}(n) = O\left(\sum_{i=0}^{r} \left(n^{1/2^i}\right)^{\alpha-1}\right) = O\left(\sum_{i=0}^{r} \left(2^{2^i}\right)^{\alpha-1}\right) = O(1),$$

since $\alpha < 1$. This shows $P(n) = O(n)$. The space requirement is thus also clearly $O(n)$. □

We are now ready for the insertion and deletion algorithms. We follow a very simple principle. Suppose that we have an ideal IST for a file of size $n$. We then leave the root node unchanged for the next $n/4$ insertions and deletions into the file and then build a new ideal IST for the new file. The same principle is applied to the subfiles. Note that this strategy will *not* enforce any balancing criterion between subfiles. In particular, if all insertions go into the same subfile, then the size of this subfile grows from $\sqrt{n}$ to $\sqrt{n} + n/4$. This is in marked contrast to ordinary balanced trees. Deletions are performed by marking (tagging) the element to be deleted. The element is not physically removed. It is removed when a subtree containing it is rebuilt.

*Definition* 3

(a) If $v$ is a node of an IST, then $T_v$ is the subtree with root $v$.
(b) The *weight* $w(v)$ of a node is the number of (marked or unmarked) keys that are stored in $T_v$. The *size* of a node $v$ is the number of *unmarked* keys which are stored in $T_v$.
(c) *Rebuilding* $T_v$ is to replace the subtree $T_v$ by an ideal IST for the set of unmarked keys stored in $T_v$.

In our implementation of interpolation search trees, we associate with each node $v$ of the tree a counter $C(v)$. It counts the number of insertions and deletions into the subfile stored in tree $T_v$. Whenever we rebuild a subtree $T_w$ for some node $w$, we initialize the counters of all nodes of the (new) subtree $T_w$ to zero. Moreover, the counter $C(v)$ *overflows* whenever its value exceeds $W/4$ where $W$ is the size of node $v$ when the tree was built, that is, when the counter $C(v)$ was reset to 0 for the last time.

We execute the insertions and deletions as follows: We first give a short informal description and then give more detailed algorithms below:

*Insertion of a key x*

(1) Find the position where $x$ should be inserted and insert $x$. Increase the counters of all nodes $v$ on the path of search by one.
(2) Find the highest node, say $z$, on the path of search such that $C(z)$ overflows. Rebuild the subtree with root $z$.

*Deletion*

(1) Mark the element to be deleted and increase the counters of all nodes of the search path.
(2) As (2) above.

The *intuition* behind these algorithms is as follows:

(1) Good worst-case behavior for searches can be achieved if the weights along any path from the root to a leaf are geometrically decreasing, because this will guarantee logarithmic depth of the tree. The counters do exactly that; cf. Lemma 2 below. Good worst-case behavior for insertions and deletions (at least in an amortized sense) can be achieved if nodes of size $W$ remain unchanged for a time which is proportional to $W$. This is also achieved by the counters. Note that a tree $T_v$ of (initial) size $W$ is only rebuilt after $W/4$ insertions or deletions into that subtree.

(2) Good average-case behavior for searches can be achieved if interpolation search (or a variant of it) can be used to search the arrays of representatives. In order for that to work, we need to know that the array of representatives stored in a node $v$ is a "fair" sample of the file stored in subtree $T_v$. We achieve this goal by *not* enforcing any criterion that relates the size of the files stored in the direct sons of a node $v$. Note that in ordinary balanced trees an "overflow" at a node $v$ changes not only node $v$ but also $v$'s father. In particular, in ordinary balanced trees an "overflow" at node $v$ changes the set of representatives stored in $v$'s father. We found it impossible to do a satisfactory expected case analysis of any such scheme and therefore have chosen update algorithms that do not enforce any balancing condition between brother nodes. This will imply in particular, that the files that are stored in brother nodes can be treated as independent files.

We give the detailed insertion and deletion algorithm next. In these algorithms, we use *isize*($v$) (for *initial size*) to denote the size of node $v$ when $C(v)$ was reset to zero for the last time. Moreover, we assume that the position where key $x$ has to be inserted or deleted is known. This position can be found using the search algorithm described in the next section.

```
proc Insert ($x$);
  let $T$ be the current tree, let $v_0, v_1, \dots, v_k$ be a path from a new node $v_0$ at the
  appropriate position to the root $v_k$;
  store $x$ into node $v_0$ and set $C(v_0) := \emptyset$; isize($v_0$) := 1;
  for $i$ from 1 to $k$ do $C(v_i) := C(v_i) + 1$ od;
  let $i$ be maximal such that $C(v_i) \geq isize(v_i)/4$;
  if $i$ exists
  then Rebuild ($T_{v_i}$) fi
end
```

and

```
proc Delete ($x$)
  let $T$ be the current tree, let $v_1, \dots, v_k$ be a path from the node $v_1$ containing key $x$
  to the root $v_k$;
  mark key $x$;
  for $i$ from 1 to $k$ do $C(v_i) := C(v_i) + 1$ od;
  let $i$ be maximal such that $C(v_i) \geq isize(v_i)/4$;
  if $i$ exists
  then Rebuild ($T_{v_i}$) fi
end
```

Finally, procedure Rebuild is given by

```
proc Rebuild ($T$)
  build an ideal IST for the set of unmarked keys stored in tree $T$;
  set $C(v) := \emptyset$ for all nodes of the new tree and set isize($v$) to the appropriate value
  of all nodes of the new tree
end
```

From now on we reserve the same **interpolation search tree** for those ISTs that can be generated from the empty initial tree by a sequence of insertions and deletions. We have

LEMMA 2. *The worst-case depth of an IST for a file of n keys is $O(\log n)$. It requires $O(n)$ storage space.*

PROOF. We prove the bound on the depth first. Consider an arbitrary node $v$ of an IST and let $w = \mathit{father}(v)$ be its father node. Then, at most $C(w) \leq \mathit{isize}(w)/4$ insertions and deletions occurred in the subfile represented by node $v$.

Hence, $\mathit{size}(v) \leq \sqrt{\mathit{isize}(w)} + \mathit{isize}(w)/4 \leq \mathit{isize}(w)/2$ since the subfile represented by node $v$ started out with $\sqrt{\mathit{isize}(w)}$ keys when $T_w$ was rebuilt for the last time and has gained at most $\mathit{isize}(w)/4$ additional keys since then. We conclude that the depth of an IST is $O(\log \mathit{isize}(\mathit{root}))$. Since $\mathit{size}(\mathit{root}) \geq \mathit{isize}(\mathit{root}) - C(\mathit{root}) \geq \mathit{isize}(\mathit{root})/2$ and $n = \mathit{size}(\mathit{root})$ we conclude that the depth is $O(\log n)$.

Let us turn to the space bound next. Let $S(n)$ be the maximal storage requirement of an IST of weight $n$. Then

$$S(n) = O(n^{\alpha}) + \max\left\{\sum_{i=1}^{k+1} S(n_i);\ \sum_i n_i = n,\ n_i \leq \frac{n}{2}\right\}$$

Let $\hat{S}(n) = S(n)/n$. Then

$$\begin{aligned} n \cdot \hat{S}(n) &= O(n^{\alpha}) + \max\left\{\sum_{i=0}^{k+1} n_i \hat{S}(n_i);\ \sum_i n_i = n,\ n_i \leq \frac{n}{2}\right\} \\ &\leq O(n^{\alpha}) + \max\left\{\sum_{i=1}^{k+1} n_i \hat{S}\left(\frac{n}{2}\right);\ \sum_i n_i = n\right\} \\ &\leq O(n^{\alpha}) + n\hat{S}\left(\frac{n}{2}\right), \end{aligned}$$

and therefore

$$\begin{aligned} \hat{S}(n) &\leq O(n^{\alpha-1}) + \hat{S}\left(\frac{n}{2}\right) \\ &= O(1), \end{aligned}$$

since $\alpha < 1$. This proves $S(n) = O(n)$.

It remains to relate the size and the weight of the root. Let $m$ be the number of marked keys. Then, $m \leq \mathit{isize}(\mathit{root})/4$, $\mathit{size}(\mathit{root}) \geq \mathit{isize}(\mathit{root}) - m$, and $w(\mathit{root}) = \mathit{size}(\mathit{root}) + m$. Thus $\mathit{size}(\mathit{root}) \geq 3 \cdot \mathit{isize}(\mathit{root})/4$, $m \leq \mathit{size}(\mathit{root})/3$ and hence $w(\mathit{root}) \leq 4 \cdot \mathit{size}(\mathit{root})/3$. This shows that the space requirement is linear. □

Lemma 2 explains why we restrict the parameter $\alpha$ to be strictly less than 1 in AISTs. If $\alpha < 1$, then storage requirement is linear; if $\alpha \geq 1$, then storage requirement is nonlinear.

LEMMA 3. *The amortized cost of an insertion or deletion (not counting the time for the preceding search) is $O(\log n)$, that is, the total cost of the first n insertions and deletions is $O(n \log n)$.*

PROOF. We use the following accounting scheme. Every insertion and deletion puts one token on each of the nodes mentioned in step (1) of the insertion or deletion algorithm. Then every node $v$ of an IST holds $C(v)$ tokens and an insertion/deletion puts down $O(\log n)$ tokens by lemma 2. A token represents the ability to pay for $O(1)$ computing time. Suppose now that we have to rebuild $T_v$ for a cost of $O(w(v))$. Since $w(v) \leq 5/4 \cdot isize(v)$ and $C(v) \geq isize(v)/4$ when $T_v$ is rebuilt we conclude that the tokens on node $v$ suffice to pay for rebuilding tree $T_v$. □

## 3. *Searching in Interpolation Search Trees*

In this section, we discuss how to search in interpolation search trees in expected time $O(\log\log n)$.

The expected case running time is defined with respect to random files and trees that we define next. We also derive some properties of random trees that will be useful for the analysis. Let $\mu$ be a probability density function on the real line. We later put various restrictions on $\mu$. A random file of size $n$ is generated by drawing independently $n$ reals according to density $\mu$. A *random* IST of size $n$ is generated as follows:

(1) Take a random file $F$ of size $n'$ for some $n'$ and build an ideal IST for it.

(2) Perform a sequence $Op_1, \ldots, Op_m$ of $\mu$-random insertions and random deletions on this tree. Clearly, if there are $i$ insertions and $d$ deletions in the sequence then $m = i + d$ and $n = n' + i - d$. An insertion is $\mu$-random if it inserts a random real drawn according to density $\mu$ into the tree. A deletion is random if it deletes a random element from the file, that is, all elements in the file are equally likely to be chosen for the deletion. These notions of randomness are called $I_r$ and $D_r$, respectively, in Knuth [6].

The following two properties of random ISTs are very helpful: Subtrees of random trees are again random and the size of subtree stays *rootic* with high probability.

LEMMA 4. *Let $\mu$ be a density function with finite support $[a, b]$, let $T$ be a $\mu$-random IST with boundaries a and b and let $T'$ be a subtree of $T$. Then there are reals $c, d$ such that $T'$ is a $\mu[c, d]$-random IST.*

PROOF. The tree $T$ was constructed by starting with the ideal IST $T_0$ for a random file $F_0$ and then applying a random sequence $Op_1, \ldots, Op_m$ of insertions and deletions to it. Let $F_i$ be the file obtained after executing $Op_1, \ldots, Op_i$. Then, $F_i$ is clearly a random file, since insertions insert elements drawn according to $\mu$ and deletions delete all elements in the current file with equal probability.

Let $m' \leq m$ be such that tree $T'$ was built after executing $Op_{m'}$ and was not rebuilt since then. We have $m' = 0$ if $T'$ exists in $T_o$ and was never rebuilt. Let $v$ be the root of $T'$ and let $w$ be the father of $v$. Then, some ancestor of $v$ overflowed when operation $Op_{m'}$ was executed. Let $c$ and $d$ be the boundaries of $T'$. After executing $Op_{m'}$ tree $T'$ is an ideal IST for file $f_{m'}[c, d] := \{x \in F_{m'}; c < x < d\}$. This file is a random file with respect to distribution $\mu[c, d]$. Also $T'$ as it exists after processing $Op_m$ can be obtained from $T'$ as it was created after processing $Op_{m'}$ by applying the operations in $Op_{m'+1}, \ldots, Op_m$ that refer to items between $c$ and $d$. This is a sequence of $\mu[c, d]$-random insertions and random deletions.

LEMMA 5. *Let $\mu$ be a density function with finite support $[a, b]$, let $T$ be a $\mu$-random IST of size $n$, and let $T'$ be a direct subtree of $T$. Then, the size of $T'$ is $O(n^{1/2})$ with probability at least $1 - O(n^{-1/2})$.*

PROOF. Let $n_0$ be the initial size of $T$ at the time of its last rebuilding. Then

$$\frac{3n_0}{4} \le n \le \frac{5n_0}{4}$$

and $m \le n_0/4$ insertions were made into $T$ since $T$ was rebuilt for the last time. Assume that $T'$ is the $i$th subtree of $T$, that is, $T'$ contains the items between $c := REP[i\sqrt{n_0}]$ and $d := REP[(i + 1)\sqrt{n_0}]$. When $T$ was rebuilt for the last time, exactly $\sqrt{n_0}$ elements were stored in $T'$. Since then some number $X$ of additional items were stored in $T'$. The random variable $X$ satisfies

$$E(X) = \sqrt{n_0} \cdot \frac{m}{n_0 + 1} \le \frac{\sqrt{n_0}}{4}$$

$$\mathrm{VAR}(X) \le E(X)\left(1 + \frac{m}{n_0 + 1}\right) \le \frac{\sqrt{n_0}}{2}$$

by Theorem A (cf. appendix).

Using Chebyshev's inequality $(\mathrm{pr}(|X - E(X)| \ge t) \le \sigma^2/t^2$ for a random variable $X$ with expectation $E(X)$ and variance $\sigma^2$ and all $t)$ with $t = n_0^{1/2}$ we conclude

$$\mathrm{pr}\left(X \ge E(X) + n_0^{1/2}\right) \le \frac{n_0^{1/2}}{2 \cdot n_0} = \frac{1}{2} \cdot n_0^{-1/2}$$

This shows that the probability that more than $\frac{5}{4} \cdot n^{1/2}$ additional items were inserted into tree $T'$ is $\le \frac{1}{2} \cdot n_0^{-1/2}$ and hence tree $T'$ has size $O(n^{1/2})$ with probability $1 - O(n^{-1/2})$. □

Lemma 5 illustrates the self-organizing feature of ISTs. The array of representatives in the root node is a sample reflecting the underlying file and hence is fairly dense in regions of high density of the distribution. This leads to the effect that the size of the subtrees stays reasonably balanced no matter what the distribution is.

We can use Lemma 5 to derive a bound on the expected amortized insertion and deletion cost.

LEMMA 6. *Let $\mu$ be a density with finite support $[a, b]$. Then the expected total cost of processing a sequence of $n$ $\mu$-random insertions and random deletions into an initially empty IST is $O(n \log\log n)$, that is, expected amortized insertion and deletion cost is $O(\log\log n)$.*

PROOF. We follow the proof of Lemma 3. Let $f(m)$ be the expected number of tokens put down by the $m$th insertion or deletion. Then

$$f(m) \le 1 + f(O(m^{1/2})) + O(m^{-1/2})O(\log m).$$

This can be seen as follows: If the insertion or deletion goes into a subtree of size $O(m^{1/2})$, then we put down $1 + f(O(m^{1/2}))$ tokens. If it does not, then we put down at most $O(\log m)$ tokens by Lemma 2. The probability of the latter event is $O(m^{-1/2})$ by Lemma 5. Thus, $f(m) = O(\log\log m)$. □

Searching ISTs is quite simple. Suppose that we have to search for $y \in \mathbb{R}$. We can use the approximation ID to the inverse distribution function to obtain a very good estimate for the position of $y$ in the array of representatives. From this initial guess, we can then determine the actual position of $y$ by a simple linear search. The details are as follows:

Let $T$ be an IST with boundaries $a$ and $b$. Let REP$[1 \cdots k]$ and ID$[1 \cdots m]$ be the array of representatives and approximation to the inverse distribution, respectively, associated with the root.

```
$j :=$ ID$[\lfloor((y - a)/(b - a))m\rfloor]$
**while** $y \geq$ REP$[j + 1]$ **and** $j < k$
    **do** $j := j + 1$ **od**
search the $j$th subtree using the same algorithm recursively
```

The correctness of this algorithm can be seen as follows:

Let $i := \lfloor(y - a)m/(b - a)\rfloor = (y - a)m/(b - a) - \epsilon$ for some $0 \leq \epsilon < 1$. Then

$$\text{REP}[j] < a + \frac{i(b - a)}{m} \leq \text{REP}[j + 1].$$

This proves $y \geq \text{REP}[j]$ and hence establishes correctness. The worst-case search time $WT(n)$ in an IST of size $n$ is also easily derived. Clearly, at most $O(\sqrt{n})$-time units are spent in the root array. Also the subfile to be searched has size at most $n/2$ and hence

$$WT(n) \leq O(\sqrt{n}) + WT\left(\frac{n}{2}\right),$$

which solves for $WT(n) = O(\sqrt{n})$. The worst-case search time is easily improved to $O((\log n)^2)$ by using exponential and binary search instead of linear search. More precisely, compare $y$ with REP$[j]$, REP$[j + 2^0]$, REP$[j + 2^1]$, REP$[j + 2^2], \ldots,$ until an $l$ is found with REP$[j + 2^{l-1}] < y \leq$ REP$[j + 2^l]$. Then, use binary search to finish the search. With this modification, worst-case search time in the array is $O(\log n)$ and hence total worst-case search time is $O((\log n)^2)$.

Finally, observe that the number of search steps used by exponential + binary search is $O$ (number of steps taken by linear search) and hence the expected case analysis done below also applies to exponential + binary search. Let us turn to expected search time next.

LEMMA 7. *Let $\mu$ be a smooth density for parameter $\alpha$, $\frac{1}{2} \leq \alpha < 1$, and let $T$ be a $\mu$-random IST with parameter $\alpha$. Then, the expected search time in the root array is $O(1)$.*

PROOF. Suppose that $l + 1$ iterations of the **while**-loop are required. As above, let

$$i := \left\lfloor \frac{(y-a)m}{(b-a)} \right\rfloor = \frac{(y-a)m}{(b-a)} - \epsilon$$

and let $j = \mathrm{ID}[i]$. Then

$$\begin{aligned}\mathrm{REP}[j] &\le y - \frac{\epsilon(b-a)}{m} \le \mathrm{REP}[j+1] \le \mathrm{REP}[j+l+1] < y \\ &\le \mathrm{REP}[j+l+2].\end{aligned}$$

Let $n_0$ be the initial size of tree $T$ and let $F_0$ be the random file which existed when $T$ was rebuilt most recently. Then, at least $l \cdot \sqrt{n_0}$ elements of $F_0$ (all elements in the subtrees $T_{j+1}, \ldots, T_{j+l}$) lie in the interval $[y - (b-a)/m, y]$. The probability of this event is at most $(3 \cdot d/l)^{l\sqrt{n_0}}$ by the claim below and hence the expected number of iterations of the **while**-loop is

$$\sum_{l \ge 1} \min\left(1, \left(3 \cdot \frac{d}{l}\right)^{l \cdot \sqrt{n_0}}\right) = O(1).$$

CLAIM. *Let $\mu$ be a smooth density for parameter $\alpha$, $\frac{1}{2} \le \alpha < 1$, with support $[a, b]$. Let $y \in [a, b]$, let $F$ be a $\mu$-random file of size $n_0$, and let $m = \lceil n_0^\alpha \rceil$. Then*

$$pr\left(\left|F \cap \left[y - \frac{b-a}{m}, y\right]\right| \ge l \cdot \sqrt{n_0}\right) \le \left(\frac{3 \cdot d}{l}\right)^{l\sqrt{n_0}}$$

*for all integer $l$.*

PROOF. Let

$$p = \int_{y-(b-a)/m}^{y} \mu(x)\,dx.$$

Then, $p \le dn_0^{-1/2}$ for some constant $d$ since $\mu$ is smooth. Let $F = \{X_1, \ldots, X_{n_0}\}$.

If $|F \cap [y - (b-a)/m, y]| \ge l \cdot \sqrt{n_0}$, then there must be at least $l \cdot \sqrt{n_0} X_i$'s with $X_i \in [y - (b-a)/m, y]$. For each particular $X_i$ we have $\mathrm{pr}(X_i \in [y - (b-a)/m, y]) = p$. Thus,

$$\begin{aligned}\mathrm{pr}\left(\left|F \cap \left[y - \frac{b-a}{m}, y\right]\right|\right) \ge l \cdot \sqrt{n_0} &\le \binom{n_0}{l \cdot \sqrt{n_0}} p^{l\sqrt{n_0}} \\ &\le \frac{n_0^{l\sqrt{n_0}}}{\left(l \cdot \sqrt{n_0}\right)!} \left(\frac{d}{\sqrt{n_0}}\right)^{l\sqrt{n_0}} \\ &\le \left(\frac{n_0 \cdot e \cdot d}{l \cdot \sqrt{n_0} \cdot \sqrt{n_0}}\right)^{l\sqrt{n_0}},\end{aligned}$$

since $s! \geq (s/e)^s$ for all $s$ by Stirling's approximation,

$$\leq \left(\frac{e \cdot d}{l}\right)^{l \cdot \sqrt{n_0}}. \qquad \square$$

THEOREM 1. *Let $\mu$ be a smooth density for parameter $\alpha$, $\frac{1}{2} \leq \alpha < 1$. Then, the expected search time in a $\mu$-random IST of size n is $O(\log \log n)$.*

PROOF. Let $T(n)$ be the expected search time in a $\mu$-random IST if size $n$. Then

$$T(n) = O(1) + T(O(n^{1/2})) + O(n^{-1/2}) \cdot O(n^{1/2})$$

and hence $T(n) = O(\log \log n)$. This can be seen as follows: The expected search time in the root array is $O(1)$ by Lemma 7. The search in the root array identifies a subtree of size $n_1 < n$ to be searched next. This subtree is $\mu$-random by Lemma 4. Also $n_1 = O(n^{1/2})$ with probability exceeding $1 - O(n^{1/2})$. This accounts for the $T(O(n^{1/2}))$ term. Finally, if $n_1$ is not $O(n^{1/2})$, then the search time is bounded by $O(n^{1/2})$. The probability of this event is $O(n^{1/2})$. This accounts for the last term and completes the justification of the recurrence. The recurrence has clearly solution $O(\log \log n)$. □

Theorem 1 encompasses a wide class of distributions. We give two examples. If there are constants $c_1, c_2, c_3, c_4$ such that $\mu(x) = 0$ for $x < c_1$ or $x > c_2$ and $0 < c_3 \leq \mu(x) \leq c_4$ for $c_1 \leq x \leq c_2$, then $\mu$ satisfies the hypothesis of Theorem 1 with $\alpha = 1/2$ and $d = c_4/c_3$. However, even some unbounded distributions satisfy Theorem 1. An example is provided by $\hat{\mu} = \mu[0, b]$ where $\mu(x) = x^{\beta}/(1 + \beta)$, $-1 < \beta < 0$ and $b > 0$. Density $\hat{\mu}$ is smooth for parameter $\alpha \geq 1/2$.

## 4. *Conclusion*

We see two achievements in this paper:

(a) We made interpolation search dynamic, that is, we found a data structure with $O(\log \log n)$ expected search time, $O((\log n)^2)$ worst search time, $O(\log \log n)$ expected amortized insertion and deletion cost, and $O(\log n)$ worst case amortized insertion and deletion cost.
(b) We extended interpolation search to a wider class of density functions.

## *Appendix A*

THEOREM A. *Let $X_1, \ldots, X_n, Y_1, \ldots, Y_m$ be independent random variables drawn according to density $\mu$. Let $(X_{(1)}, \ldots, X_{(n)})$ be obtained by reordering $(X_1, \ldots, X_n)$ according to increasing magnitude and let q, k and l be integers with $0 \leq q \leq q + k \leq n$. Let $L(n, m, q, k)$ denote the number of j such that $X_{(q)} \leq Y_j \leq X_{(q+k)}$. Then*

(a)

$$\Pr(L(n,m,q,k)=l)=\frac{\binom{l+k-1}{l}\binom{m-l+n-k}{n-k}}{\binom{m+n}{m}},$$

(b)

$$E(L(n,m,q,k))=k\cdot\frac{m}{n+1},$$

(c)

$$E\big((L(n,m,q,k))^2\big)=\frac{k\cdot m}{n+1}\cdot\left(1+\frac{(k+1)(m-1)}{n+1}\right),$$

(d)

$$Var(L(n,m,q,k))\leq\frac{k\cdot m}{n+1}\cdot\left(1+\frac{m}{n+1}\right).$$

PROOF

(a) Let $Z_1,\ldots,Z_{n+m}$ be obtained from $X_1,\ldots,X_n,Y_1,\ldots,Y_m$ be reordering according to magnitude. Clearly, for fixed $Z_1,\ldots,Z_{n+m}$, there are $\binom{n+m}{m}$ partitions $\{\{X_1,\ldots,X_n\},\{Y_1,\ldots,Y_m\}\}$ that yield this particular Z-sequence. How many of these partitions satisfy that there are exactly $lY$'s between $X_{(q)}$ and $X_{(q+k)}$? Let us call this number $C$.

The partitions may be identified with the paths through a rectangular lattice with sides $n$ and $m$. An $X$ corresponds to a horizontal edge and a $Y$ to a vertical edge of the path. The partitions contributing to $C$ correspond to paths with the following property. The subpath starting in the first vertex of the path in column $q$ and extending to the first vertex in column $q+k$ contains exactly $l$ vertical edges. The subpath is a path through the rectangular lattice with side $k$ and $l$ ending with an horizontal edge, that is, a path through the rectangular lattice with sides $k-1$ and $l$. If we cut this subpath out of the original path, we obtain a path through a lattice with sides $n-k$ and $m-l$. Conversely, if we take any path through the $n-k$ by $m-l$ lattice, cut the path at column $q$, and insert a path through the $k-1$ by $l$ lattice followed by a horizontal edge then we obtain a path through the $n$ by $m$ lattice which contributes to $C$. Thus, $C=\binom{l+k-1}{k-1}\binom{n-k+m-l}{n-k}$. Since $C$ is independent of the specific values of $Z_1,\ldots,Z_{n+m}$, we conclude

$$\Pr(L(n,m,q,k)=l)=\frac{\binom{l+k-1}{k-1}\binom{n-k+m-l}{n-k}}{\binom{n+m}{n}}$$

(b)

$$
\begin{aligned}
E(L(n,m,q,k)) &= \sum_{l=0}^{m} \frac{\binom{l+k-1}{k-1}\binom{n-k+m-l}{n-k}}{\binom{n+m}{n}} \\
&= \frac{k}{\binom{n+m}{n}} \cdot \sum_{l=0}^{m} \binom{l+k-1}{k}\binom{n-k+m-l}{n-k} \\
&= \frac{k}{\binom{n+m}{n}} \cdot \binom{m+n}{n+1} \\
&= k \cdot \frac{m}{n+1}
\end{aligned}
$$

The next to last equality follows by counting the number of paths through an $n+1$ by $m-1$ grid according to the height at which the path reaches column $k$.

(c)

$$
\begin{aligned}
E\big((L(n,m,q,k))^2\big) &= \sum_{l=0}^{m} l^2 \frac{\binom{l+k-1}{k-1}\binom{n-k+m-l}{n-k}}{\binom{n+m}{n}} \\
&= \frac{1}{\binom{n+m}{n}} \cdot \sum_{l=0}^{m} (l \cdot (l-1) + l)\binom{l+k-1}{k-1} \\
&\quad \binom{n-k+m-l}{n-k} \\
&= \frac{k \cdot (k+1)}{\binom{n+m}{n}} \cdot \sum_{l=0}^{m} \binom{l+k-1}{k+1}\binom{n-k+m-l}{n-k} \\
&\quad + \frac{k \cdot m}{n+1} \\
&= \frac{k \cdot (k+1)}{\binom{n+m}{n}} \cdot \binom{n+m}{n+2} + \frac{k \cdot m}{n+1} \\
&= \frac{k \cdot m}{n+1} \cdot \left(1 + \frac{(k+1)(m-1)}{n+2}\right).
\end{aligned}
$$

(d)

$$
\begin{aligned}
\mathrm{Var}(L(n,m,q,k)) &= E\big((L(n,m,q,k))^2\big) - (E(L(n,m,q,k)))^2 \\
&= \frac{k \cdot m}{n+1} \cdot \left(1 + \frac{(k+1)(m-1)}{n+2}\right) \\
&\quad - k^2 \cdot \frac{m^2}{(n+1)^2}
\end{aligned}
$$

$$= \frac{k \cdot m}{n+1} \cdot \left(1 + \frac{(k+1)(m-1)}{n+2} - \frac{k \cdot m}{n+1}\right)$$

$$\leq \frac{k \cdot m}{n+1} \cdot \left(1 + \frac{(k+1) \cdot m}{n+1} - \frac{k \cdot m}{n+1}\right)$$

$$= \frac{k \cdot m}{n+1} \cdot \left(1 + \frac{m}{n+1}\right).$$

## REFERENCES

1. FRANKLIN, W. R. Padded lists· Set operations in expected $O(\log \log N)$ time. *Inf. Proc. Letters 9* (1979), 161–166.
2. FREDERICKSON, G. N. Implicit data structures for the dictionary problem. *J ACM 30*, 1 (Jan 1983), 80–94.
3. GONNET, G., ROGERS, L., AND GEORGE, J. An algorithmic and complexity analysis of interpolation search," *Acta Inf. 13*, 1 (1980), 39–52.
4. HOFRI, M., AND KONHEIM, A. G. Padded lists revisited. *SIAM J. Comput 16* (1987), 1073–1114.
5. ITAI, A., KONHEIM, A. G., AND RODEH, M. A sparse table implementation of priority queues. In *Proceedings of ICALP '81*. Lecture Notes in Computer Science, vol. 115. Springer-Verlag, New York, 1981, pp. 417–431.
6. KNUTH, D. E. Deletions that preserve randomness. *IEEE Trans. Softw. Eng. SE 3* (1977), 351–359.
7. MEHLHORN, K. *Data Structures and Algorithms*. Vol. 1; *Sorting and Searching*. EATCS Monographs in Theoretical Computer Science. Springer-Verlag, New York, 1984.
8. MELVILLE, R., AND GRIES, D. Controlled density sorting. *Inf. Proc. Lett. 10* (1980), 169–172.
9. PEARL, Y., ITAI, A., AND AVNI, H. Interpolation search—A $\log \log N$ search. *Commun. ACM 21*, 7, (1978), 550–554.
10. PEARL, Y., AND REINGOLD, E. M. Understanding the complexity of interpolation search. *Inf. Proc. Lett. 6*, 6 (1977), 219–222.
11 PETERSON, W. W. Addressing for random storage. *IBM J Res. Develop. 1* (1957), 131–132.
12. WILLARD, D. Maintaining dense sequential files in a dynamic environment. In *Proceedings of the 14th Symposium on Theory of Computing*. (San Francisco, Calif., May 5–7). ACM, New York, 1982, pp. 114–122.
13. WILLARD, D. E. Searching unindexed and nonuniformly generated files in $\log \log N$ time. *SIAM J. Comput. 14* (1985), 1013–1029.
14. YAO, A. C., AND YAO, F. F. The complexity of searching an ordered random table. In *Proceedings of the 17th Annual Symposium on the Foundations of Computer Science*. IEEE, New York, 1976, pp. 173–175.

RECEIVED OCTOBER 1984; REVISED NOVEMBER 1991; ACCEPTED DECEMBER 1991