Sequential and Parallel Subquadratic Work Algorithms for Constructing Approximately Optimal Binary Search Trees

Marek Karpinski[*] Lawrence L. Larmore[†] Wojciech Rytter [‡]

Abstract
A sublinear time subquadratic work parallel algorithm for construction of an optimal binary search tree, in a special case of practical interest, namely where the frequencies of items to be stored are not too small, is given. A sublinear time subquadratic work parallel algorithm for construction of an approximately optimal binary search tree in the general case is also given. Sub-quadratic work and sublinear time are achieved using a fast parallel algorithm for the *column minima* problem for Monge matrices developed by Atallah and Kosaraju. The algorithms given in this paper take $O(n^{0.6})$ time with n processors in the CREW PRAM model. Our algorithms work well if every subtree of the optimal binary search tree of depth $\Omega(\log n)$ has $o(n)$ leaves.

We prove that there is a sequential algorithm with subquadratic average-case complexity, by demonstrating that the "small subtree" condition holds with very high probability for a randomly permuted weight sequence. This solves the conjecture posed in [11] and breaks the quadratic time "barrier" of Knuth's algorithm [10]. This algorithm can also be parallelized to run in average sublinear time with n processors.

1 Introduction

The problem of developing a subquadratic time sequential algorithm computing opt imal binary search trees (the OBST problem) appears to be very hard. Algorithm for finding approximately optimal binary search trees have been found by Allen, Mehlhorn and Unterauer [2, 13, 14]. The results of this paper are largely based on the algorithm for approximately optimal binary search trees given by Larmore [11]. We show that there is a sequential algorithm with subquadratic average-case complexity, where weights are randomly permuted.

The OBST problem is especially interesting in a parallel setting, since there is no known $\mathcal{NC}$ algorithm which solves that problem efficiently, and the problem of finding such a parallel algorithm appears to be very hard [4].

There is an $\mathcal{NC}$ algorithm for the special case of alphabetic trees using n^2 processors [12]. The best known $\mathcal{NC}$ algorithms require $O(n^6)$ work for optimal binary search trees and $O(n^2)$ work for approximately optimal binary search trees [4, 15].

Sublinear time parallel algorithms sometimes have much lower total work than $\mathcal{NC}$ algorithms. In [8] a sublinear time algorithm for the OBST problem whose work is very close to quadratic is given. The fastest known sequential algorithm for the OBST problem is the classical algorithm by Knuth [10], which takes quadratic time. The main theorem of [10] uses, without stating it in those terms, the Monge property of the matrix of subtree costs. A matrix M has the *Monge property* if, for all $i_0 < i_1$ and $j_0 < j_1$ which are within range, $M_{i_0,j_0} + M_{i_1,j_1} \leq M_{i_0,j_1} + M_{i_1,j_0}$. This is essentially the same as the *quadrangle inequality* introduced by Yao [16] which allowed speedup of certain dynamic programming algorithms.

We consider the problem in a parallel setting, using the CREW PRAM model of computation. We present sublinear time subquadratic work parallel algorithms for certain special instances of the OBST problem, We also give sublinear time subquadratic work parallel algorithms which give approximately optimal binary search trees in the general case.

Define a binary search tree to be *ϵ-approximately optimal* if its cost differs by at most ϵ from the cost of the optimal binary search tree. Our main result is:

THEOREM 1.1. *There exists an $O(n^{0.6})$-time parallel algorithm using n processors which computes the optimal binary search tree for any sequence in which all weights are $\Omega(\frac{1}{n})$. Furthermore, there exists an $O(n^{0.6})$-time parallel algorithm using n processors which computes an ϵ-approximately optimal binary search tree for any sequence, where $\epsilon = o(1)$.*

We use terminology from [9], pages 434–435. Let $K_1, \ldots K_n$ be a sequence of n weighted items (keys), which are to be placed in a binary search tree. We are

[*]Dept. of Computer Science, University of Bonn, 53117 Bonn. This research was partially supported by DFG Grant KA 673/4-1, and by ESPRIT BR Grant 7097 and ECUS 030. Email:marek@cs.uni-bonn.de

[†]Department of Computer Science, University of Nevada, Las Vegas, NV 89154–4019, USA. Partially supported by National Science Foundation grants CCR-9112067 and CCR-9503441. Email:larmore@cs.unlv.edu

[‡]Institute of Informatics, Warsaw University, 02–097 Warszawa. Partially supported by DFG Grant Bo 56/142-1. Email:rytter@mimuw.edu.pl

given a sequence α of $2n+1$ weights (probabilities): $q_0, p_1, q_1, p_2, q_2, p_3, \ldots, q_{n-1}, p_n, q_n$ where

- p_i is the probability that K_i is the search argument;
- q_i is the probability that the search argument lies between K_i and K_{i+1}.

Note that $\sum p_i + \sum q_i = 1$. It will be convenient to refer to the *external item* E_i, for $0 \le i \le n$, corresponding to the probability q_i.

Let $Tree(\alpha)$ be the set of all full binary weighted trees with n internal nodes, where the i^{th} internal node (in inorder) has weight p_i, and the i^{th} external node (the leaf, in the left-to-right order) has weight q_i. The tree is "full" in the sense that each internal node has exactly two sons. The keys $\{K_i\}$ are to be stored in internal nodes of this binary search tree. The external nodes correspond to intervals between keys. If T is a such a weighted binary search tree then define the *cost* of T to be $cost(T) = \sum \ell(T, v) \cdot weight(v)$ where the summation is over all nodes of T, and $\ell(T, v)$ is the *level* of the node v in T, defined to be the distance (number of nodes on the path) from the root.

Let $\mathrm{OPT}(\alpha)$ be the set of trees $Tree(\alpha)$ whose cost is minimal. The OBST problem consists of finding any tree $T \in \mathrm{OPT}(\alpha)$. Denote by $obst(i, j)$ the set $\mathrm{OPT}(q_i, p_{i+1}, q_{i+1}, \ldots, q_{j-1}, p_j, q_j)$. The trees which are elements of this set are said to have *width* $|j - i|$. Let $cost(i, j)$ be the cost of a tree in $obst(i, j)$, and let $weight(i, j) = q_i + p_{i+1} + \ldots + p_j + q_j$, for $i < j$. Let $cost(i, i) = weight(i, i) = q_i$.

The values of $cost(i, j)$ are tabulated in an array. The time to compute all values of *cost* is $O(n^2)$, using Knuth's Theorem [10], essentially making use of the Monge property of *cost*, considered as a matrix. Knuth's algorithm can be easily parallelized by computing all entries on a given diagonal of the array in parallel. The following lemma was essentially shown in [8]. It says that costs of all optimal subtrees of width at most ℓ can be efficiently computed in parallel.

Lemma 1.1. *All values $cost(i, j)$ for $|j - i| \le \ell$ can be computed in $O(\ell \cdot \log(\ell))$ time with $O(n/\log(\ell))$ processors.*

A matrix M has the *Monge property* if, for all $i_0 < i_1$ and $j_0 < j_1$ which are within range, $M_{i_0,j_0} + M_{i_1,j_1} \le M_{i_0,j_1} + M_{i_1,j_0}$. Monge matrices arise in a large number of applications. We state several known results concerning Monge matrices.

Lemma 1.2. *If $j_0 \le j_1$, then there exist $i_0 \le i_1$ such that the minimum of column j_0 of M is at i_0, and the minimum of column j_1 of M is at i_1.*

The following result is by Aggarwal, Klawe, Morey, Shor, and Wilber [1]. The algorithm developed in that paper is whimsically known as the "SMAWK" algorithm, using a permutation of the authors' initials.

Lemma 1.3. *If M is an $n \times m$ Monge matrix, all column minima of M can be found in $O(n + m)$ sequential time.*

In the parallel case, we have the following result by Atallah and Kosaraju [3].

Lemma 1.4. *If M is an $n \times m$ Monge matrix, all column minima of M can be found in $O(\log n \log m)$ time by $n/\log n$ processors, using the CREW PRAM model of computation.*

2 A general structure of the exact algorithm

The main phase of our algorithm uses a form of dynamic programming quite different from the usual ones for optimal binary search trees (bottom-up computation of the costs of optimal subtrees). The new concept of a "partial tree" is introduced. The costs of all partial trees are computed by processing the potential nodes of the trees in in-order. These potential nodes are contained in a tree which we call an "abstract tree." It is possible that not all of the nodes of the abstract tree correspond to nodes in the optimal binary search tree.

Let $d > 0$ be a given integer. The *abstract d-tree* $\mathcal{T}_d$ is a full regular binary tree which consists of all possible nodes at level at most d, and no nodes at higher levels. Note that $\mathcal{T}_d$ has $m = 2^d - 1$ nodes, which we label $v_1, \ldots v_m$ in in-order.

Partial subtrees. Assume T is a binary search tree and $v \in \mathcal{T}_d$ is identified with an internal node of T containing the key K_i. Then $T' = Partial_T(v, K_i)$ is a subtree of T which consists of all vertices(internal and external) of T preceding the node v in in-order, together with v. We say that T' is a *partial tree terminating in* (v, K_i).

If T" is a partial tree, the *partial cost* of T', written *partial_cost*(T'), is the sum of the path weights for all nodes in T'. Define *partial_cost*(v, i) to be the minimum value of *partial_cost*(T') such that T' is a partial tree terminating in (v, K_i).

Our main algorithm depends on two parameters, ℓ and d, and consists of three phases.

3 Analysis of the algorithm MAIN

The essential part of the algorithm is *Basic-Phase*. We derive recurrence equations, as in dynamic programming, to compute the table *partial_cost*. First we introduce the relation "$\Rightarrow$" between the nodes of the abstract tree $\mathcal{T}_d$. The relation $u \Rightarrow v$ means that there is some binary tree T for which a node identified with v is the immediate in-order successor, in T, of a node identified with u. More formally: let u_1, w_1 be, respectively,

```
ALGORITHM MAIN:
 Preprocessing-Phase: parallel implementation of Knuth's algorithm.
     Compute costs of optimal subtrees of width at most $\ell$.
     Comment: can be done in parallel time $O(\ell)$ due to Lemma 1.1

 Basic-Phase: computation of optimal costs of partial subtrees.
     Assume the nodes of the tree $\mathcal{T}_d$ are listed in in-order $v_1, \ldots, v_m$.
     for $k = 1$ to $m$ do
       for each $i = 1 \ldots n$ do in parallel
         compute $partial\_cost(v_k, i)$ using the parallel algorithm
         of [3] for the corresponding column minima problem.

 Construction-Phase: construction of an optimal binary search tree.
     $global\_cost := \min\{partial\_cost(v, n) + (level(v) + 1) \cdot q_n \ : \ v \in \mathcal{T}_d\}$;
     Comment: $global\_cost$ is the cost of an optimal tree;
     $w :=$ a node $v \in \mathcal{T}_d$ for which minimum is achieved;
     construct an optimal tree knowing $w$ and the table $partial\_cost$.
```

the left and right sons of u in $\mathcal{T}_d$. Let $u_2, u_3, \ldots$ be the rightmost branch starting at u_1 and let $w_2, w_3, \ldots$ be the leftmost branch starting at u_1. Then $u_i \Rightarrow v$ and $v \Rightarrow w_i$ for each i. If $u \Rightarrow v$, we say u is a *predecessor* of v, and v is a *successor* of u. The cost of an optimal partial tree terminating in a given node v depends on the cost of a partial tree terminating in a predecessor of v. Let $\ell(u,v) = \max\{level(u), level(v)\} + 1$. We introduce two auxiliary tables $partial\_cost\_1$ and M_v defined by recurrence equations as follows:

$$partial\_cost\_1(v,i) = \min\{partial\_cost(u, i-1)$$
$$+level(v) \cdot p_i + \ell(u,v) \cdot q_{i-1} \ : \ u \Rightarrow v\}$$

Assume $u \Rightarrow v$ and u or v is at the bottom level, *i.e.*, $\ell(u,v) = d+1$. Then for each $1 \le i < j$ define:

$$M_v(i,j) = partial\_cost(u,i) + level(v) \cdot p_j$$
$$+cost(i-1,j) + weight(i-1,j) \cdot d$$

If $i \ge j$ then define $M_v(i,j) = \infty$. Let $ColMin(M_v, i)$ be the smallest value in the i^{th} column of M_v. The *basic dynamic programming recurrence* for computing *partial_cost* is as follows:

$$partial\_cost(v,i) =$$
$$\min\{partial\_cost\_1(v,i), \ ColMin(M_v, i)\}$$

LEMMA 3.1. (1) *The matrix M_v satisfies the Monge condition.* (2) *For a given v, the values $ColMin(M_v, i)$, for all $1 \le i \le n$, can be computed in $O(\log^2 n)$ time with $n/\log n$ processors.*

Proof. (1) By Lemma 2.1 of [11], the matrix $\{cost(i-1,j)\}$ has the Monge property. It is simple to verify that $\{weight(i-1,j)\}$ is also Monge. The other two terms are trivially Monge since they depend on only one component. Finally, the sum of Monge matrices is Monge.

(2) All column minima of an $n \times n$ Monge matrix can be computed in $O(\log^2 n)$ time with $n/\log n$ processors by a simple divide-and-conquer algorithm.

LEMMA 3.2. *Assume that the each segment of ℓ consecutive items contains at least one node at depth at most d in the optimal binary search tree. Then the optimal binary search tree can be computed in $O(\log n(2^d + \ell))$ parallel time with n processors, or in $O(n(2^d + \ell))$ sequential time.*

Proof. The partial costs can be computed by traversing the tree $\mathcal{T}_d$, in in-order and applying the basic dynamic programming recurrence. The main point is that the values *ColMin* can be computed for a given node in $O(\log n)$ time with n processors, by Lemma 3.1. This proves the following claim:

Claim. Assume the costs of all optimal subtrees rooted below level d are computed. Then an optimal binary search tree can be constructed in $O(2^d \log n)$ time with n processors.

First the optimal costs of subtrees of width ℓ are computed. Then (in *Basic-Phase*) the partial costs are computed in $O(2^d \log n)$ time with n processors (see Claim). Efficiency is gained by applying the parallel algorithm for the column minima problem.

The first phase runs in $O(\ell \cdot \log n)$ time with $O(n\ell)$ work, and the second phase runs in $O(2^d \log n)$ time with $O(2^d n)$ work.

Finally, the binary search tree is reconstructed using pointers that are saved during computation of *cost* and

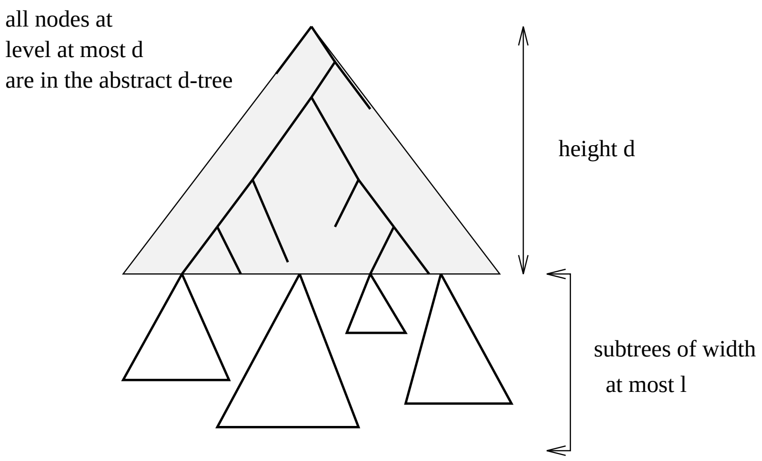

Figure 1: The structure of a binary search tree: $d = \lceil \log_\phi(\frac{1}{\Delta}) \rceil + 2$, where Δ is the smallest total weight of a segment of ℓ consecutive items. The abstract d-tree $\mathcal{T}_d$ is shaded.

M.

LEMMA 3.3. *Let ϕ be the* golden ratio ($\phi = \frac{\sqrt{5}+1}{2} \approx 1.62$). *Assume that the total weight of each segment of ℓ consecutive items is at least Δ. Then an optimal binary search tree can be computed in $O(\log n \max\{\ell \cdot, 2^{\log_\phi(\frac{1}{\Delta})}\})$ parallel time with n processors.*

Proof. Let $d = \lceil \log_\phi(\frac{1}{\Delta}) \rceil + 1$,
Claim. Let $T \in OPT(q_0, p_1, q_1, \ldots, p_n, q_n)$.
If v is an internal node of T and the weight of all items contained in the subtree rooted at v is Δ, then $level_T(v) < \log_\phi(\frac{1}{\Delta}) + 2$.

Let $F_1 = 1$, $F_2 = 1$, $F_3 = 2$, *etc.*, be the Fibonacci numbers. By Theorem 2 of [7], similar to results of [6], a subtree whose root is at level h can have weight at most $2/F_{h+2}$. $F_n > 2\phi^{n-4}$ (see [9] exercise 4, pg. 18), which proves the claim and the lemma.

4 The proof of our main results related to parallel constructions

In this section we prove our main results related to sublinear time parallel computations, as two separate theorems. The proofs consist of manipulating the parameters d, ℓ, and Δ. Let $\beta = 1/(1+\log_2 \phi)$. We have $\beta \approx 0.59023$. Hence $n^\beta \log n = O(n^{0.6})$. We restate Theorem 1.1 precisely:

THEOREM 4.1. *Assume $q_{i-1} + p_i + q_i \geq \frac{\delta}{n}$ for each i, where $\delta > 0$ is a constant. Then an optimal binary search tree can be computed in $O(n^{0.6})$ parallel time with $O(n)$ processors.*

Proof. We apply Lemma 3.3 with $\Delta = \frac{\delta \ell}{n}$. Let $d = \log_\phi(\frac{n}{\ell})$. The work in the first phase is $O(n\ell)$ and in the second phase it is $O(2^d n)$. The algorithm has the smallest minimal work if the phases have nearly equal work. This occurs when $\log_\phi(\frac{n}{\ell}) = \log_2(\ell)$. It can be calculated that in this case $d = \log_2(n) \cdot \beta$. Thus $2^d \log n = O(n^{0.6})$.

THEOREM 4.2. (1) *Let $a > 1$. There is a parallel $O(n^{1+a\cdot\beta} \log n)$-time n-processor algorithm which constructs an $(n^{1-a} \log n)$-approximately optimal binary search tree.* (2) *There is a parallel $O(n^{0.6})$-time n-processor algorithm which constructs an $o(1)$-approximately optimal binary search tree.*

Proof. Let $\delta = n^{-a}$.
Claim. If each item has weight at least δ we can compute the optimal binary search tree in parallel $O(n^{1+a\cdot\beta} \log n)$ time with n processors.

Proof. (of the claim) Take $\ell = n^{a\cdot\beta}$, $\Delta = \ell \cdot \delta$ and apply Lemma 3.3.

We remove a pair consisting of K_i and E_i provided $q_{i-1} + p_i + q_i < \delta$. Iterate this process until $q'_{i-1} + p'_i + q'_i \geq \delta$ for all i in the remaining sequence. Construct an optimal binary search tree T' for the remaining items using the algorithm from Claim 1. The tree T' has height $O(\log n)$ since the weight of each item is sufficiently large. The cost of T' does not exceed the cost of an optimal tree for the whole sequence.

We now attach the deleted items to T', as follows. Suppose $K_i, E_i \ldots K_j, E_j$ is a maximal list of consecutive deleted items. Replace E_{i-1} in T' by an almost regular binary search tree whose items are $E_{i-1}, K_i, \ldots K_j, E_j$.

We increase the cost by $O(n \cdot n^{-a} \cdot \log n)$, which is $O(n^{1-a} \log n)$. This proves (1).

We can take a very close (from below) to $0.6/\beta$ and achieve time $O(n^{0.6})$ with n processors and $o(1)$-

approximation.

5 The subquadratic average-time sequential algorithm

We consider now the average case complexity. Assume that items $K_1, \dots K_n$ are given, with access probabilities $p_1, \dots, p_n$. We consider the problem of finding an optimal binary search tree for a random permutation of the items. For the sake of simplicity, we assume that every search is successful, *i.e.*, $q_i = 0$. We shall address the general question where the q_i can be positive in the full paper.

In this section, we prove the following, addressing a conjecture of [11].

THEOREM 5.1. *Let probabilities $p_1, p_2, p_3, \dots, p_n$ be given, where the $\{p_i\}$ are randomly permuted. Then there is an $o(n^2)$ time algorithm that computes a binary search tree, where p_i are the weights of the internal nodes and external nodes have weight zero, which is optimal with probability $1 - o(1)$. Furthermore, there is an algorithm which computes an optimal binary search tree in expected time $o(n^2)$.*

We use the following notation: let $x = x_1, \dots x_n$ be any sequence of positive real numbers. Write

$$\begin{aligned} |x| &= n, \text{ the } \textit{length} \text{ of } x \\ \Sigma x &= \sum_{i=1}^{n} x_i, \text{ the } \textit{weight} \text{ of } x \end{aligned}$$

For any $0 < \lambda < 1$, denote by $\text{PREF}\langle x, \lambda \rangle$ the largest prefix y of x such that $\Sigma y < \lambda \Sigma x$. Similarly, denote by $\text{SUFF}\langle x, \lambda \rangle$ the largest suffix z of x such that $\Sigma z < \lambda \Sigma x$.

Definition of λ-trees. For any $0 < \lambda < 1$, define the *λ-tree* on x to be the tree whose root is x and whose nodes are sublists of x, where the left son of y is $\text{PREF}\langle y, \lambda \rangle$ and the right son of y is $\text{SUFF}\langle y, \lambda \rangle$. This generalizes the notion of *min-max tree* introduced by Bayer [5], which is essentially a $\frac{1}{2}$-tree.

LEMMA 5.1. *Let $x = x_1, \dots x_n$ be a sequence of positive real numbers, where $n = |x|$, and let $\tilde{x}$ be a random permutation of x. Let $\pi_i(x)$ be the probability that $\left|\text{PREF}\langle x, \frac{1}{2} \rangle\right| < i$. Then $\pi_i(x) \le \frac{i}{n}$ for $1 \le i \le \frac{n}{2}$.*

By symmetry, note that the $\pi_i(x)$ is also the probability that $\left|\text{SUFF}\langle x, \frac{1}{2} \rangle\right| \le i$.

Proof. If $i = \frac{n}{2}$, then $\pi_i(x) \le \frac{1}{2}$ by symmetry. The rest of the proof is by induction on n. For $n = 1$, the result is trivial. Assume that the statement of the lemma holds for all lists of length $(n-1)$. Let $\tilde{x}_j$ be the smallest item of $\tilde{x}$. Let y be the list of length n obtained from $\tilde{x}$ by subtracting $\tilde{x}_j$ from every item. In the resulting sequence can be many zeros, let us fix one of them. Let x' be the list of length $(n-1)$ obtained by deleting this zero from y. By the inductive hypothesis, the statement of the lemma holds for x'. $\text{PREF}\langle y, \frac{1}{2} \rangle$ will be identical to $\text{PREF}\langle x', \frac{1}{2} \rangle$, except possibly for insertion of the zero.

The zero of y is in one of the first i places with probability $\frac{i}{n-1}$. Recall that $i < \frac{n}{2}$. Then

$$\begin{aligned} \pi_i(y) &= \frac{i}{n}\pi_{i-1}(x') + \frac{n-i}{n}\pi_i(x') \\ &\le \frac{i(i-1)}{n(n-1)} + \frac{(n-i)i}{n(n-1)} = \frac{i}{n} \end{aligned}$$

If $\left|\text{PREF}\langle \tilde{x}, \frac{1}{2} \rangle\right| \le i$, then $\left|\text{PREF}\langle y, \frac{1}{2} \rangle\right| \le i$, since $\tilde{x}$ is obtained from y by adding a constant to each item. Thus $p_i(x) = p_i(\tilde{x}) \le p_i(y)$.

Define $\phi = 1 - \frac{1}{2^5}$ and $\psi = 1 - \frac{1}{2^{10}}$. Note that $(1-\phi)^2 = 1 - \psi$, and $1 - \phi^2 < 2^{-4}$.

LEMMA 5.2. *If y is any randomly permuted list of weights, then $\left|\text{PREF}\langle y, \frac{3}{4} \rangle\right| \le \psi |y|$ with probability at least ϕ^2.*

We shall present the proof, which makes use of Lemma 5.1 twice, in the full paper.

LEMMA 5.3. *If x is any sequence of positive real numbers, and if $\tilde{x}$ is a randomly chosen permutation of x, then the probability that all nodes of the $\frac{3}{4}$-tree of $\tilde{x}$ at depth $2m$ have length at most $n\psi^m$ is at least $1 - 2\phi^{2m}$.*

Proof. By Lemma 5.2, a depth $2m$ node in the $\frac{3}{4}$-tree has length at most $n\psi^m$ with probability at least

$$\begin{aligned} & 1 - \sum_{i=0}^{m-1} \binom{2m-1}{i} (\phi^2)^i (1-\phi^2)^{2m-1-i} \\ \ge \; & 1 - \sum_{i=0}^{m-1} \binom{2m-1}{i} (\phi^2)^m (1-\phi^2)^{2m} \\ \ge \; & 1 - \sum_{i=0}^{m-1} \binom{2m-1}{i} (\phi^2)^m (2^{-4})^{m-1} \\ = \; & 1 - 2^{-2m+2} (\phi^2)^m \end{aligned}$$

There are at most 2^{2m-1} nodes at level $2m$. The result follows.

By Theorem 2 of [7], we have:

LEMMA 5.4. *Assume we have an optimal binary search tree whose list of items is z, and y is the list of items of a subtree rooted at a node of depth 2. Then $\Sigma y \le \frac{2}{5} \cdot \Sigma z$.*

Note that $\frac{2}{5} + \frac{1}{4} < \frac{3}{4}$, a fact which is used in the next lemma.

LEMMA 5.5. *Let y be the list of nodes of a subtree S of the optimal binary search tree over x whose root is at depth $2d-1$. Then y is a sublist of a node at depth d in the $\frac{3}{4}$-tree of x.*

Proof. The proof is by induction. If $d = 1$, the statement is trivial. If $d > 1$, let z be the list of nodes for the subtree T rooted at the *grandfather* of the root of S.

By Lemma 5.4 we have $\Sigma y \leq \frac{2}{5}\Sigma z$. By the inductive hypothesis, z is contained in a sublist w which is a node at depth $d-1$ of the $\frac{3}{4}$-tree of x. Write $w = uyv$. Since $\Sigma y \leq \frac{2}{5}\Sigma w$, either $\Sigma u \geq \frac{1}{4}\Sigma w$ or $\Sigma v \geq \frac{1}{4}\Sigma w$. Thus, y contained in either the right son or the left son of w in the $\frac{3}{4}$-tree.

Directly from Lemmas 5.5 and 5.3, we obtain:

Lemma 5.6. *If x is any sequence of positive real numbers, and if $\tilde{x}$ is a randomly chosen permutation of x, then the probability that all subtrees rooted at depth $4m-1$ of the optimal tree over $\tilde{x}$ have fewer than $n\psi^m + m$ nodes is at least $1 - 2\phi^{2m}$.*

Lemma 5.6 directly implies existence of subquadratic expected work algorithms, since each subtree of an optimal binary search tree at sufficiently large depth contains a sufficiently small number of items. Then we can apply the algorithms presented before.

Theorem 5.2. *For some constants $\alpha, \beta > 0$ there is an $O(n^{2-\alpha})$-time algorithm which finds an optimal binary search tree over a randomly permuted list of length n in with probability $1 - O(n^{-\beta})$.*

Proof. Pick m such that $2^{4m} = \Theta(n\psi^m)$. Let α, β be such that $2^{4m} = n^{1-\alpha}$ and $n^\beta = \left(\frac{1}{\phi}\right)^{2m}$. It is simple to show that $\alpha = \Theta(1)$ and $\beta = \Theta(1)$. By Lemma 3.2 we are done.

Theorem 5.3. *For some constant $\gamma > 0$ there is an algorithm that constructs an optimal binary search tree over a randomly permuted list x of length n in expected time $O(n^{2-\gamma})$*

Proof. Construct the $\frac{3}{4}$-tree for x level by level, halting at that level d where every node at level d has length at most 2^{2d}. By Lemma 5.5, all subtrees of the optimal binary tree at level $2d$ have length at most 2^{2d}. By Lemma 3.2, there is an $O(n2^{2d})$-time sequential algorithm which constructs an optimal binary search tree.

By Lemma 5.3, $1 - \log_n(2^{2d}) = \Theta(1)$ with probability $n^{-\Theta(1)}$. Thus, the expected time of the algorithm is $O(n^{2-\Theta(1)})$.

Finally, we remark that the algorithms given by Theorems 5.2 and 5.3 can be efficiently parallelized, running in sublinear (or average sublinear) time with n processors.

References

[1] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber, Geometric applications of a matrix-searching algorithm, *Algorithmica* **2** (1987), pp. 195–208.

[2] B. Allen, Optimal and near-optimal binary search trees, *Acta Inform.* **18** (1982), pp. 255–263.

[3] M. J. Atallah, S. R. Kosaraju, Parallel computation of row minima for monotone matrices, *Journal of Algorithms* **13** (1992), pp. 394–413.

[4] M. J. Atallah, S. R. Kosaraju, L. L. Larmore, G. L. Miller, and S-H. Teng, Constructing trees in parallel, *Proc. 1st ACM Symposium on Parallel Algorithms and Architectures* (1989), pp. 499–533.

[5] P. J. Bayer, Improved bounds on the costs of optimal and balanced binary search trees, Project MAC Technical Memorandum 69, MIT (1975).

[6] R. Güttler, K. Mehlhorn, and W. Schneider, Binary search trees: average and worst case behavior, *Elektr. Informationsverarb Kybernetik* **16** (1980), pp. 579–591.

[7] D. S. Hirschberg, L. L. Larmore, and M. Molodowitch, Subtree weight ratios for optimal binary search trees, TR 86-02, ICS Department, University of California, Irvine (1986).

[8] M. Karpinski and W. Rytter, On a sublinear time parallel construction of optimal binary search trees, *Proceedings of the 19th International Symposium on Mathematical Foundations of Computer Science*, LNCS 841 (ed. I. Privara, B. Rovan, P. Ruzicka) (1994), pp. 453–461.

[9] D. E. Knuth, *The Art of Computer Programming*, Addison–Wesley (1973).

[10] D. E. Knuth, Optimum binary search trees, *Acta Informatica* **1** (1971), pp. 14–25.

[11] L. L. Larmore, A subquadratic algorithm for constructing approximately optimal binary search trees, *Journal of Algorithms* **8** (1987), pp. 579–591.

[12] L. L. Larmore, T. M. Przytycka, and W. Rytter, Parallel construction of optimal alphabetic trees, *Proceedings of the 5th ACM Symposium on Parallel Algorithms and Architectures* (1993), pp. 214–223.

[13] K. Mehlhorn, Nearly optimal binary search trees, *Acta Informatica* **5** (1975), pp. 287–295.

[14] K. Unterauer, Dynamic weighted binary search trees, *Acta Informatica* **11** (1979), pp. 341–362.

[15] W. Rytter, Efficient parallel computations for some dynamic programming problems, *Theoretical Comp. Sci.* **59** (1988), pp. 297–307.

[16] F. F. Yao, Efficient dynamic programming using quadrangle inequalities, *Proceedings of the 12th ACM Symposium on Theory of Computing* (1980), pp. 429–435.