# Depth of a Random Binary Search Tree with Concurrent Insertions

James Aspnes[1] and Eric Ruppert[2]

[1] Yale University, USA
[2] York University, Canada

**Abstract.** Shuffle a deck of $n$ cards numbered 1 through $n$. Deal out the first $c$ cards into a hand. A player then repeatedly chooses one of the cards from the hand, inserts it into a binary search tree, and then adds the next card from deck to the hand (if the deck is empty). When the player finally runs out of cards, how deep can the search tree be?
This problem is motivated by concurrent insertions by $c$ processes of random keys into a binary search tree, where the order of insertions is controlled by an adversary that can delay individual processes. We show that an adversary that uses any strategy based on comparing keys cannot obtain an expected average depth greater than $O(c + \log n)$. However, the adversary can obtain an expected tree height of $\Omega(c \log(n/c))$, using a simple strategy of always playing the largest available card.

## 1 Introduction

In the worst case, the height of a binary search tree (BST) can be linear in the number of keys that it stores. However, if the tree is constructed by inserting the keys one by one in a random order, then the average node depth, and even the height of the tree, will be logarithmic [9]. Here, we consider how much worse these measures become if insertions are performed concurrently by multiple processes. Consider the tree shown in Fig. 1. Suppose three processes wish to insert keys 12, 13 and 16 simultaneously. The processes will compete to insert their keys into the slot at the right child of the node containing key 11. The shape of the resulting tree depends upon which process succeeds first, and this can be determined by a number of system-dependent factors. For example, the scheduling of steps by different processes is affected by cache misses, timesharing and other events that are difficult to predict or analyze. This scheduling will, in turn, determine which process acquires a lock first (in a lock-based BST implementation) or performs a CAS first (in a CAS-based non-blocking BST implementation), and hence which insertion takes effect first. Similarly, in a transactional memory system, the order of insertions may depend on many details of the implementation of transactional memory that are outside the direct control of the BST insertion algorithm. Since the insertion algorithm has no control over these factors, if we wish to provide a worst-case analysis of the BST being constructed, we can imagine an adversary choosing which of the concurrent insertions occurs first. This will make our analysis sufficiently general to cover any kind of synchronization used to coordinate the insertions by different processes.

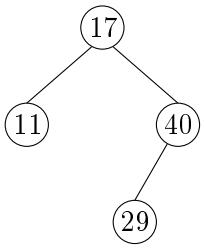


**Fig. 1.** An example binary search tree.

Thus, we consider the following experiment, in which $c$ processes simultaneously insert $n$ random keys into an initially empty BST. Fix an ordered universe $U$ and a probability distribution on that universe. The $c$ processes first choose $c$ keys from $U$ independently at random using this distribution. The $c$ processes attempt to insert these $c$ keys concurrently. The adversary chooses any one of the $c$ values to become the root of the BST (by scheduling the corresponding process $p$ so that it succeeds in installing that value at the root). Process $p$ completes its insertion and draws the next key, again independently at random from the fixed probability distribution, and attempts to insert it. Once again, the adversary can choose any one of the $c$ pending insertions to take effect next, and this procedure is repeated until all $n$ keys have been inserted. We assume that the BST does not permit duplicate keys.

The adversarial scheduler is intended to model difficult-to-predict factors like cache misses or processes being interrupted by higher priority processes. These factors may depend on the memory addresses accessed by the processes, which may in turn depend on the relative order of the random keys chosen by the processes. Beyond this ordering information, the precise values of the keys chosen are unlikely to have any effect on the scheduling of processes. Thus, we assume the adversary is *comparison-based*: it decides which key to insert next based only on the relative order among the $c$ keys of pending operations and the keys that have already been inserted.

As discussed in Mahmoud's monograph [9], if all keys are chosen independently at random from the same distribution, the resulting ranks of the keys form a random permutation of $\{1, \ldots, n\}$, where each permutation is equally likely. Since we consider only comparison-based adversaries, we can envision the concurrent construction of our random BST as follows. We start with an empty BST. We shuffle a deck of $n$ cards labelled with keys 1 to $n$. The first $c$ cards make up the adversary's initial hand. At each step, the adversary chooses one card to play from its hand, inserts that card's key into the BST and, if the deck is not yet empty, replaces the card by drawing the top card from the deck and adding it to the hand. We are interested in two measures of the resulting random BST. The depth of a node is the number of edges along the path from the root to that node. The **average depth** is the sum of all the node depths divided

by $n$. The **height** is the maximum of all node depths. Our goal is to establish bounds on the expected values of these measures, where the expected value is taken over all random permutations.

We show that the expected average depth is $O(c + \log n)$. This bound is tight within a constant factor: if the adversary always chooses the largest key among the pending insertions, the expected average depth will be $\Omega(c + \log n)$. Unlike the case of sequential insertions, we show that the expected height can be significantly larger than the average depth: we prove that the adversary that always chooses the largest key causes the expected height to be $\Omega(c \log(n/c))$.

## 2 Related Work

The height and average depth of randomly constructed BST has been extensively studied and is a classic problem in average-case complexity. Even the earliest papers [1, 15] on BSTs included a discussion of the expected average depth of nodes in a tree built from random keys, showing that it is $O(\log n)$. Robson [13] showed that the expected height of a BST built by inserting a random permutation of $\{1, \ldots, n\}$ is at most $(4.311...) \log n + o(\log n)$. The constant factor was shown to be exact by Devroye [5]. Reed [12] proved an even tighter result, showing that the expected height is $(4.311...) \log n - (1.953...) \log \log n + O(1)$. The variance of the height is also known to be $O(1)$ [6, 12]. More detailed information is known about the exact distribution of node depths in random trees. Mahmoud's monograph [9] provides an overview of many results of this area. The analysis of random trees constructed using deletions as well as insertions has had very limited success. There have been some empirical studies of this scenario, however (see [4]).

In our paper, we start with a random permutation and allow an adversary to reorder the permutation in a constrained way. A complementary scenario was considered by Manthey and Reischuk [10]: they instead begin with a permutation chosen by the adversary and then randomly perturb it (in a limited way) and analyze the expected height of the resulting BST.

Since BSTs play a central role in computer science, concurrent implementations of them are of great practical importance. If only insertions are supported, it is fairly trivial to implement a BST. For example, a non-blocking implementation can be designed using the compare-and-swap primitive as follows. To insert a key $k$, first search for the key $k$. We assume that duplicate keys are not permitted in the BST, so if the key $k$ is found during this search, the insertion terminates without altering the tree. Otherwise, the search reaches a nil child pointer. The process then attempts to install a leaf containing $k$ in place of that nil pointer. If CAS is available, a single CAS can effect this change. Alternatively, for a lock-based implementation, the process can acquire a lock for the child pointer, check that it is still nil, replace the nil pointer, and release the lock. If either of these ways of updating the tree fail (because another process has already replaced the nil pointer with a different non-nil value), then the insertion can simply continue searching down the tree from its current location

and try again to insert the new key $k$. The standard algorithm to search for keys in a BST will work even if insertions are being done concurrently.

In the case of the CAS-based implementation described above, it is easy to see that the number of steps performed by an insertion is proportional to the depth of the node it eventually adds to the tree, since it does a constant amount of work at each step along the path to that node. Thus, the total number of steps to construct a random tree of $n$ nodes is proportional to the sum of the node depths, which has expected value $O(n(c + \log n))$, according to our result. (The total number of CAS steps is $O(nc)$: each of the $n$ successful CAS steps can cause at most one failed CAS step at each other process, for a total of $O(nc)$.)

Coordinating updates to the tree becomes considerably more difficult if deletions can also occur. Ensuring the BST is *balanced*, so that the height of the tree is logarithmic, adds significant additional complications to concurrent implementations of BSTs. Lock-based balanced BSTs have long been studied. As early as 1978, Guibas and Sedgewick [8] sketched a lock-based implementation when they introduced their balanced red-black trees. For a more up-to-date example, see Bronson et al.'s lock-based implementation of an AVL tree [2]. Ellen et al. [7] gave a non-blocking implementation of an unbalanced BST from CAS instructions. There have been several subsequent non-blocking BST implementations, including one that extends the scheme of [7] to yield a balanced BST [3], ensuring that the height is $O(c+\log n)$ whenever there are $n$ keys in the tree and $c$ pending operations on it. However, concurrent balanced trees are considerably more complex than unbalanced ones, and the tree rotation operations that are required to maintain balance incur a significant overhead. In applications where keys will be inserted in random order, our work suggests that this overhead and additional complexity can be avoided by using unbalanced trees without sacrificing good search times in the resulting BST.

## 3 Problem Statement

We are given a **deck** of $n$ cards with unique keys from some ordered universe $U$, which are shuffled according to a uniform random permutation $\pi$. We start with an empty BST. At each step, a player (representing the adversary) chooses to insert into a BST one card from a **hand** consisting of the first $c$ cards in the permutation that have not yet been inserted, or, if fewer than $c$ cards remain, all remaining cards.

In making this decision, the player can only observe the cards in the hand and the tree; it cannot predict the order of the remaining cards in the deck. Cards already inserted in the tree or present in the hand at some step are called **dealt**; the remaining cards are called **undealt**.

A **play** consists of taking one card from the hand and inserting it into the tree. The game continues for $n$ plays. We formalize the notion of a comparison-based adversary as follows. When deciding which card to play, the player can only observe the relative order of the dealt cards. In particular, if two permutations $\pi$ and $\pi'$ produce the same relative order of their first $t$ cards, then the play after $t$

cards have been dealt will be from the same position in both $\pi$ and $\pi'$. Note that the dealt cards at this time will always be the first $t$ cards in the permutation $\pi$.

We have two measures of performance for an adversary strategy:

1. The **expected height** is the expected maximum depth of any node in the binary search tree after all $n$ plays, where the expectation is taken over all choices of the permutation $\pi$.
2. The **expected average depth** of all nodes in the tree under the same conditions.

## 4 An Upper Bound on Expected Average Depth

In this section, we prove the following upper bound on the expected average depth of a random BST.

**Theorem 1.** *For every comparison-based adversary, the expected average depth of a random BST is* $O(c + \log n)$.

*Proof.* To simplify the argument, we assume that the player is deterministic. This means that when conditioning on various events we do not need to take into account any random choices made by the player. However, the argument applies equally well to a randomized player, as such a player can be described as a random mixture of deterministic players, and hence the expected average depth for a randomized player will be a weighted average of the expected average depths for various deterministic players.

Number the cards from 1 to $n$ in the order they are dealt. Let $A_{ij}$ be the indicator variable for the event that, in the final tree, card $i$ is a proper ancestor of card $j$. Then the depth of $j$ is exactly $\sum_i A_{ij}$, and the total depth of the tree is $\sum_j \sum_i A_{ij}$. Interchanging the summations gives $\sum_i \sum_j A_{ij}$, which shows that the total depth is equal to the sum of the number of proper descendants of each node. In the argument below, we will bound this sum instead, booking the expected number of proper descendants of a card $i$, conditioned on the information available to the player, as soon as it is inserted into the tree.

Let $R$ be the order relation on the shuffled cards, so that $i <_R j$ means that the $i$-th card in the deck is less than the $j$-th card in the deck. Recall that we assume all $n!$ permutations of the deck are equally likely.

Define time $t$, for $c \leq t \leq n$, as the first time at which $t \geq c$ cards have been dealt, of which $c$ are held in the player's hand and $t - c$ have already been inserted in the tree. We will define a supermartingale process that bounds the expected return to the player up to time $n$, and use a separate argument to handle the insertion of the last $c$ cards. (See the appendix for background on supermartingales.)

Let $R_t$ be the subrelation of $R$ that includes only cards 1 through $t$. Since the player is comparison-based, $R_t$ includes all information available to the player at time $t$. No new information is revealed to the player after time $n$, because the last of the $n$ cards is dealt at time $n$. For any undealt card $j > t$, we have that all

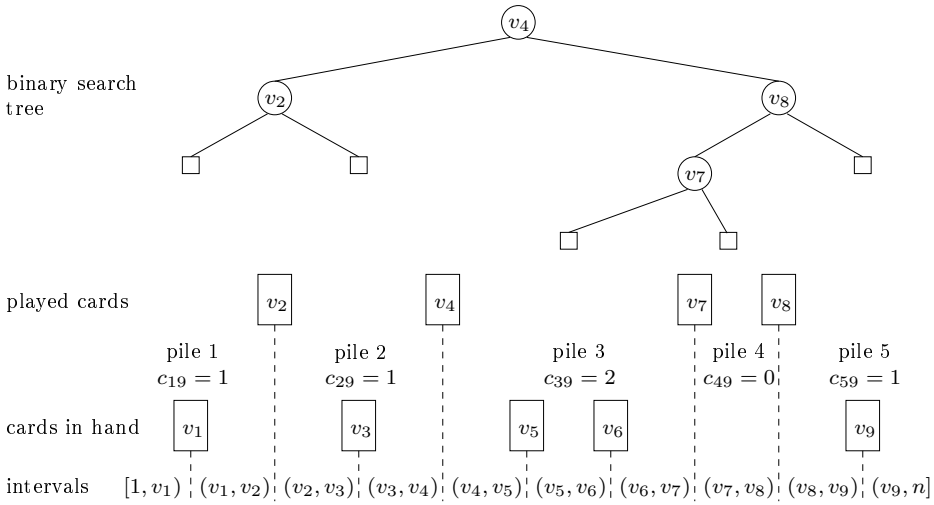


**Fig. 2.** An example when $t = 9$ and $c = 5$. The values on the cards are $v_1 < v_2 < \cdots < v_9$. 4 cards have been inserted into the tree and 5 cards are in the player's hand. The small squares in the tree represent nil pointers.

$(t+1)!$ permutations of $\{1, \ldots, t, j\}$ are equally likely, and conditioning on $R_t$, we get that all $t+1$ possible positions of $j$ relative to cards $1, \ldots, t$ are equally likely. So the probability that $j$ appears in each of these positions is $\frac{1}{t+1}$, and summing over all $n-t$ undealt cards gives an expected number of $\frac{n-t}{t+1}$ undealt cards in each of these positions.

We now look at the expected return to the player, in terms of the number of proper descendants, of playing a particular card at time $t < n$. Because the tree contains $t - c$ nodes, it has $t - c + 1$ null leaf pointers at which we might insert a new card. Let $c_{kt}$, the multiplicity of leaf $k$, be defined as the number of cards in the hand that would be inserted at $k$. We will refer to the set of these $c_{kt}$ cards as the $k$-th pile at time $t$. See Fig. 2 for an example.

Suppose now that we insert one of these $c_{kt}$ cards $i_t$ at leaf $k$. Each of the remaining $c_{kt} - 1$ cards will eventually become a descendant of $i_t$. When $t \le n$, so will an average of $(c_{kt}+1)\frac{n-t}{t+1}$ undealt cards. So for $t \le n$, we have

$$\begin{aligned} \mathrm{E}\left[\sum_j A_{i_t j} \,\middle|\, R_t\right] &= c_{kt} - 1 + (c_{kt}+1)\frac{n-t}{t+1} \\ &= (c_{kt}+1)\left(1 + \frac{n-t}{t+1}\right) - 2 \\ &= (c_{kt}+1)\frac{n+1}{t+1} - 2. \end{aligned} \tag{1}$$

One way to interpret (1) is that $(c_{kt}+1)\frac{n+1}{t+1}$ is the expected number of leaf pointers in the final subtree rooted at $i_t$. Subtracting one gets the number of

nodes in this subtree, and subtracting one again removes the subtree's root $i_t$ to get the number of proper descendants.

The values of $c_{kt}$ may evolve over time in a complex way as new cards are dealt and as the player chooses which cards to play based on the current state. We will track the effect of these choices using a shape function proportional to $\sum_k c_{kt}^2$ that will reflect the player's future ability to accrue expected descendants by playing cards from large piles.

Let $c'_{kt}$ be the number of cards that can be inserted under the $k$-th leaf after $i_t$ is inserted but before any new card is dealt. Playing a card splits $c_{kt}$ into two new piles of size $c'_{k_1t}$ and $c'_{k_2t}$, where $c'_{k_1t} + c'_{k_2t} = c_{kt} - 1$; the remaining piles are unaffected, so that each $c_{\ell t}$ for a pile that does not contain $i_t$ becomes $c'_{\ell' t}$ for some distinct $\ell'$. After playing a card, replacing that card in the hand by dealing a new card adds one to each $c'_{\ell t}$ with probability $\frac{c'_{\ell t}+1}{t+1}$, since the new card is equally likely to fall into any of the $t+1$ intervals shown at the bottom of Fig. 2.

Splitting $c_{kt}$ into $c'_{k_1t}$ and $c'_{k_2t}$ reduces the sum of the squares by at least $2c_{kt} - 1$, since $(c'_{k_1t})^2 + (c'_{k_2t})^2 - c_{kt}^2 = (c'_{k_1t})^2 + (c'_{k_2t})^2 - (c'_{k_1t} + c'_{k_2t} + 1)^2 = -2c'_{k_1t}c'_{k_2t} - 2(c'_{k_1t} + c'_{k_2t} + 1) + 1 \le -(2c_{kt} - 1)$. Dealing a new card to pile $\ell$ increases the sum by $2c'_{\ell t} + 1$. So for $t < n$,

$$
\begin{aligned}
&\mathrm{E}\left[\sum_\ell c_{\ell,t+1}^2 - \sum_\ell c_{\ell t}^2 \;\middle|\; R_t\right] \\
&\le -(2c_{kt} - 1) + \sum_\ell (2c'_{\ell t} + 1)\frac{c'_{\ell t} + 1}{t+1} \\
&= 1 - 2c_{kt} + \frac{2}{t+1}\left(\sum_\ell (c'_{\ell t})^2 + \sum_\ell c'_{\ell t}\right) + \sum_\ell \frac{c'_{\ell t} + 1}{t+1} \\
&\le 1 - 2c_{kt} + \frac{2}{t+1}\left(\left(\sum_\ell c'_{\ell t}\right)^2 + \sum_\ell c'_{\ell t}\right) + \sum_\ell \frac{c'_{\ell t} + 1}{t+1} \\
&= 1 - 2c_{kt} + \frac{2}{t+1}\left((c-1)^2 + (c-1)\right) + 1 \\
&= 2 - 2c_{kt} + \frac{2c(c-1)}{t+1}. \qquad (2)
\end{aligned}
$$

We will now define a supermartingale process $X_c, X_{c+1} \ldots, X_n$ to bound the expected increase in $\sum_i \sum_j A_{ij}$ up to time $n$. In this context, the supermartingale property means that $X_t \ge \mathrm{E}[X_{t+1} \mid R_t]$ for all $t$, from which it can be shown by induction that $X_c \ge \mathrm{E}[X_n \mid R_c]$. We will structure this process so that $X_c$ is a fixed bound and $X_n$ always exceeds $\sum_i \sum_j A_{ij}$.

Let $\chi_{it}$ be the indicator variable for the event that the $i$-th card dealt from the deck is in the tree at time $t$. This decision is made based on values observable in $R_{t-1}$.

Define

$$X_t = U_t + V_t + W_t,$$

where

$$U_t = \sum_{i=1}^{n} \chi_{it} \, \mathrm{E}\left[ \sum_{j=1}^{n} A_{ij} \;\middle|\; R_t \right]$$

is the expected total descendants of all nodes already inserted,

$$V_t = \frac{n+1}{2(t+1)} \sum_{\ell=1}^{t-c+1} c_{\ell t}^2$$

is a scaled version of the shape factor discussed above, that will offset changes to $U_t$ that depend on which card is played at time $t$, and

$$W_t = \sum_{s=t}^{n-1} \left( 2\frac{n+1}{s+1} - 2 + (n+1)\frac{c(c-1)}{(s+1)^2} \right).$$

pays for the total expected changes to $U_t$ and $V_t$ that do not depend on which card is played at time $t$.

Let us now demonstrate that $X_c, \ldots, X_n$ is in fact a supermartingale with respect to $R_c, \ldots, R_n$. We will start by considering $U_{t+1}$. Since $\chi_{i,t+1}$ is completely determined by $R_t$, we have

$$\begin{aligned}
\mathrm{E}\left[U_{t+1} \mid R_t\right] &= \sum_{i=1}^{n} \mathrm{E}\left[ \chi_{i,t+1} \, \mathrm{E}\left[ \sum_{j=1}^{n} A_{ij} \;\middle|\; R_{t+1} \right] \;\middle|\; R_t \right] \\
&= \sum_{i=1}^{n} \chi_{i,t+1} \, \mathrm{E}\left[ \mathrm{E}\left[ \sum_{j=1}^{n} A_{ij} \;\middle|\; R_{t+1} \right] \;\middle|\; R_t \right] \\
&= \sum_{i=1}^{n} \chi_{i,t+1} \, \mathrm{E}\left[ \sum_{j=1}^{n} A_{ij} \;\middle|\; R_t \right] \\
&= U_t + \mathrm{E}\left[ \sum_{j=1}^{n} A_{i_t j} \;\middle|\; R_t \right] \\
&= U_t + (c_{kt}+1)\frac{n+1}{t+1} - 2, \qquad (3)
\end{aligned}$$

where $k$ is the number of the pile that contains $i_t$. In the second-to-last step, we use the fact that only $\chi_{i_t}$ changes between $t$ and $t+1$. The last step applies (1).

Now we turn to $V_t$. For $t < n$, use (2) to get

$$\begin{aligned}
\mathrm{E}\left[V_{t+1} \mid R_t\right] &= \frac{n+1}{2(t+2)} \mathrm{E}\left[\sum_{\ell=1}^{t-c+2} c_{\ell,t+1}^2 \,\middle|\, R_t\right] \\
&< \frac{n+1}{2(t+1)} \mathrm{E}\left[\sum_{\ell=1}^{t-c+2} c_{\ell,t+1}^2 \,\middle|\, R_t\right] \\
&\le \frac{n+1}{2(t+1)} \left(\sum_{\ell=1}^{t-c+1} c_{\ell t}^2 + 2 - 2c_{kt} + \frac{2c(c-1)}{t+1}\right) \\
&= V_t + \frac{n+1}{t+1} - c_{kt}\frac{n+1}{t+1} + (n+1)\frac{c(c-1)}{(t+1)^2}. \qquad (4)
\end{aligned}$$

When we add $U_{t+1}$ and $V_{t+1}$ together, the $c_{kt}\frac{n+1}{t+1}$ terms on the right-hand sides of (3) and (4) cancel out, so we are left with

$$\mathrm{E}\left[U_{t+1} + V_{t+1} \mid R_t\right] < U_t + V_t + 2\frac{n+1}{t+1} - 2 + (n+1)\frac{c(c-1)}{(t+1)^2}. \qquad (5)$$

Since the extra terms on the right-hand side of (5) are precisely the value of $W_t - W_{t+1}$, we have $\mathrm{E}\left[X_{t+1} \mid R_t\right] = \mathrm{E}\left[U_{t+1} + V_{t+1} + W_{t+1} \mid R_t\right] < U_t + V_t + W_t = X_t$, and the supermartingale property holds. It follows that $\mathrm{E}\left[X_n\right] = \mathrm{E}\left[U_n + V_n + W_n\right] \le U_c + V_c + W_c = X_c$.

Let us look now at $U_n$, $V_n$, and $W_n$. We have

$$\begin{aligned}
U_n &= \sum_{i=1}^{n} \chi_{in} \mathrm{E}\left[\sum_{j=1}^{n} A_{ij} \,\middle|\, R_n\right] \\
&= \sum_{i=1}^{n} \chi_{in} \sum_{j=1}^{n} A_{ij}.
\end{aligned}$$

This misses all pairs $ij$ where $i$ is among the $c$ cards left in the hand at time $n$. However, at this point the remaining play of the game is purely deterministic, and it is straightforward to see that the player's optimal strategy is to play the cards under each leaf in increasing order, adding $\sum_{k=1}^{n-c+1} \binom{c_{kn}}{2} \le \sum_{k=1}^{n-c+1} \frac{1}{2} c_{kn}^2 = V_n$ to the total. So we have $U_n + V_n \ge \sum_{ij} A_{ij}$. But $W_n = 0$, so this means $X_n \ge \sum_{ij} A_{ij}$.

Now let us return to the start of the process. At time $c$, we have $\chi_{ic} = 0$ for all $i$, so $U_c = 0$. We have only one $c_{\ell c}$, which equals $c$, so $V_c = \frac{(n+1)c^2}{2(c+1)} = O(cn)$.

Using $H(n)$ to denote the $n$th harmonic number,

$$\begin{aligned} W_c &= \sum_{s=c}^{n-1} \left( 2\frac{n+1}{s+1} - 2 + (n+1)\frac{c(c-1)}{(s+1)^2} \right) \\ &< 2(n+1)\left(H(n) - H(c)\right) + (n+1)\cdot c(c-1)\cdot \int_{x=c}^{\infty} \frac{1}{x^2} dx \\ &= O(n\log(n/c)) + (n+1)\cdot c(c-1)\cdot \frac{1}{c} \\ &= O(cn + n\log n). \end{aligned}$$

It follows that $\mathrm{E}\left[\sum_{ij} A_{ij}\right] \leq \mathrm{E}\left[X_n\right] \leq X_c = O(cn + n\log n)$. Dividing by $n$ to convert the total to an average then gives the claimed bound. □

### 4.1 A Matching Lower Bound

A simple strategy for the comparison-based adversary gets expected $\Omega(c+\log n)$ average depth, matching the bound in Theorem 1.

**Theorem 2.** *There is a comparison-based adversary that yields an expected average depth of $\Omega(c + \log n)$.*

*Proof.* Consider the comparison-based adversary that always inserts the largest card in its hand. Let $m$ be the upper median in the initial hand of $c$ cards (i.e., the $\lceil \frac{c+1}{2} \rceil$th smallest card in the hand). With probability at least $1/2$, there are at least $\lfloor n/2 \rfloor$ cards in the deck that are smaller than $m$. When this occurs, $m$ is placed at depth at least $\lceil c/2 \rceil - 1$ and at least $\lfloor n/2 \rfloor$ keys are placed in the left subtree of $m$. Even if that subtree is perfectly balanced, the leaf nodes of that subtree alone have a total path length of at least $(n/4)(\lceil c/2 \rceil + \lfloor \log \lfloor n/2 \rfloor \rfloor - 1) = \Omega(n(c+\log n))$. So, with probability $1/2$, the average depth of all nodes will be at least $\Omega(c+\log n)$, and hence the expected average depth will also be $\Omega(c+\log n)$. □

## 5 A Lower Bound on Expected Height

In this section, we show a lower bound on the expected height of a BST obtained by the particular adversary strategy that always plays the largest available card.

We first introduce some notation from the calculus of finite differences that will be useful in the proof. The **forward difference operator** $\Delta f$ is defined by

$$\Delta f(k) = f(k+1) - f(k).$$

This operator satisfies the summation by parts formula,

$$\begin{aligned}
&\sum_{k=m}^{n} a(k)\Delta b(k) \\
&= \sum_{k=m}^{n} a(k)b(k+1) - \sum_{k=m-1}^{n-1} a(k+1)b(k+1) \\
&= a(n+1)b(n+1) - a(m)b(m) - \sum_{k=m}^{n} b(k+1)\Delta a(k). \qquad (6)
\end{aligned}$$

The **falling factorial** $(x)_c$ is defined by

$$(x)_c = x(x-1)(x-2)\dots(x-c+1).$$

Then, we have

$$\begin{aligned}
\Delta(k-1)_c &= (k)_c - (k-1)_c \\
&= (k)(k-1)_{c-1} - (k-c)(k-1)_{c-1} \\
&= c(k-1)_{c-1}. \qquad (7)
\end{aligned}$$

**Theorem 3.** *There is a comparison-based adversary strategy that yields a BST with expected height $\Omega(c\log(n/c))$.*

*Proof.* We consider the leftmost path in the BST, assuming the adversary always plays the largest available card. Let $L_i$ be the value on the $i$-th card that appears in the leftmost path (starting from the root of the BST), and let $X_i$ be the total number of cards whose values are strictly less than $L_i$ (i.e., $X_i = L_i - 1$). Let $X_0 = n$. When the leftmost path has length $i$, then no card less than $L_i$ has yet been played, and as long as the hand contains any card greater than $L_i$, playing this card does not increase the length of the leftmost path. It follows that the leftmost path increases only when the hand consists entirely of cards less than $L_i$, and that $L_{i+1}$ is the largest of the $\min(c, X_i)$ cards present in the hand at this time.

This can be used to show that $H(X_i)$ does not drop too quickly on average, giving a lower bound on the expected length of the leftmost path. Here, $H(n)$ denotes the $n$th harmonic number, $H(n) = \sum_{i=1}^{n} \frac{1}{i}$.

We now consider the effect of playing a card less than $L_i$. Suppose the value of the random variable $X_i$ is $x$. For any $k \le x$, the probability that $c$ cards chosen uniformly without replacement from the $x$ smallest cards are all at most $k$ is exactly $(k)_c/(x)_c$, and the probability that the largest of these $c$ cards is exactly $k$ is $(k)_c/(x)_c - (k-1)_c/(x)_c = \Delta(k-1)_c/(x)_c$. Now let us compute, for $x \ge c$,

$$\Pr[X_{i+1} = k-1 \mid X_i = x] = \Delta(k-1)_c/(x)_c, \text{ for } c \le k \le x,$$

and

$$
\begin{aligned}
&\mathrm{E}[H(X_{i+1}) \mid X_i = x] \\
&= \sum_{k=c}^{x} H(k-1)\Delta(k-1)_c/(x)_c \\
&= \frac{1}{(x)_c}\left(H(x)(x)_c - H(c-1)(c-1)_c - \sum_{k=c}^{x}(k)_c \Delta H(k-1)\right) \qquad \text{by (6)} \\
&= H(x) - \frac{1}{(x)_c}\cdot\sum_{k=c}^{x}\frac{(k)_c}{k} \\
&= H(x) - \frac{1}{(x)_c}\cdot\sum_{k=c}^{x}(k-1)_{c-1} \\
&= H(x) - \frac{1}{(x)_c}\cdot\sum_{k=c}^{x}\frac{\Delta(k-1)_c}{c} \qquad \text{by (7)} \\
&= H(x) - \frac{1}{c\cdot(x)_c}((x)_c - (c-1)_c) \\
&= H(x) - \frac{1}{c}.
\end{aligned}
$$

Let $Y_i = H(X_i) + i/c$. Then

$$
\begin{aligned}
\mathrm{E}[Y_{i+1} \mid X_0, \ldots, X_i] &= \mathrm{E}[H(X_{i+1}) \mid X_0, \ldots, X_i] + \frac{i+1}{c} \\
&= H(X_i) - \frac{1}{c} + \frac{i+1}{c} \\
&= H(X_i) + \frac{i}{c} \\
&= Y_i
\end{aligned}
$$

The remainder of the proof uses martingales, which are described in the appendix. The sequence $\{Y_i\}$ is a martingale with respect to $\{X_i\}$. Let $\tau$ be the first index at which $X_\tau \leq c-1$. Then $\tau$ is not only a lower bound on the depth of the tree, but is also a stopping time with respect to $\{X_i\}$. It follows from Doob's Optional Stopping Theorem that $\mathrm{E}[Y_\tau] = \mathrm{E}[Y_0] = H(n)$. So, we have $H(n) = \mathrm{E}[Y_\tau] = \mathrm{E}[H(X_\tau) + \tau/c] = \mathrm{E}[H(X_\tau)] + \mathrm{E}[\tau]/c$. Solving for $\mathrm{E}[\tau]$ gives $\mathrm{E}[\tau] = c\,(H(n) - \mathrm{E}[H(X_\tau)]) \geq c\,(H(n) - H(c-1)) = c\sum_{k=c}^{n}\frac{1}{k} = \Omega\,(c\log(n/c))$. □

## 6 Conclusion

We considered a node-oriented (or internal) BST, where keys are stored both in internal nodes and leaves. Some concurrent implementations of BSTs are based on leaf-oriented (or external) BSTs, where the keys are stored only in the leaves,

and internal nodes are used only to direct searches to the appropriate leaf. Our height lower bound extends to leaf-oriented trees. It would be interesting to see whether our average depth upper bound does too.

Although the comparison-based adversaries discussed here are intended to model schedulers accurately (and pessimistically), it might be interesting to see whether even stronger malicious adversaries could force the height or average depth of random trees to grow higher by using the actual values of the keys. For example, if keys are drawn uniformly at random from the interval $[0, 1]$, and the initial hand consisted of $c = 3$ cards labelled with $0.03, 0.45$ and $0.54$, the adversary would be better off choosing $0.03$ as the root to ensure a more lopsided tree, whereas if the initial hand contained $0.46$, $0.55$ and $0.97$ the adversary should choose $0.97$ as the root.

*Acknowledgements* Funding for the second author was provided by the Natural Sciences and Engineering Research Council of Canada.

## References

1. A.D. Booth and A.J.T. Colin. On the efficiency of a new method of dictionary construction. *Information and Control*, 3(4):327–334, December 1960.
2. Nathan G. Bronson, Jared Casper, Hassan Chafi, and Kunle Olukotun. A practical concurrent binary search tree. In *Proc. 15th ACM Symposium on Principles and Practice of Parallel Programming*, pages 257–268, 2010.
3. Trevor Brown, Faith Ellen, and Eric Ruppert. A general technique for non-blocking trees. In *Proc. 19th ACM Symposium on Principles and Practice of Parallel Programming*, pages 329–342, 2014.
4. J. Culberson and J.I. Munro. Explaining the behaviour of binary search trees under prolonged updates: A model and simulations. *The Computer Journal*, 32(1):68–75, 1989.
5. Luc Devroye. A note on the height of binary search trees. *Journal of the ACM*, 33(3):489–498, 1986.
6. Michael Drmota. An analytic approach to the height of binary search trees II. *Journal of the ACM*, 50(3):333–374, May 2003.
7. Faith Ellen, Panagiota Fatourou, Eric Ruppert, and Franck van Breugel. Non-blocking binary search trees. In *Proc. 29th ACM Symposium on Principles of Distributed Computing*, pages 131–140, 2010.
8. Leo J. Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *Proc. 19th IEEE Symposium on Foundations of Computer Science*, pages 8–21, 1978.
9. Hosam M. Mahmoud. *Evolution of Random Search Trees.* John Wiley & Sons, 1992.
10. Bodo Manthey and Rüdiger Reischuk. Smoothed analysis of binary search trees. *Theoretical Computer Science*, 378(3):292–315, June 2007.
11. Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*, chapter 12. Cambridge University Press, 2005.
12. Bruce Reed. The height of a random binary search tree. *Journal of the ACM*, 50(3):306–332, May 2003.

13. J.M. Robson. The height of binary search trees. *Australian Computer Journal*, 11(4):151–153, November 1979.
14. David Williams. *Probability with Martingales*. Cambridge University Press, 1991.
15. P.F. Windley. Trees, forests and rearranging. *The Computer Journal*, 3(2):84–88, 1960.

## A Background on Martingales

Here, for the sake of completeness, we present background information about martingales that is used in our paper, following the presentation of Mitzenmacher and Upfal's textbook [11].

We say that a sequence $Y_0, Y_1, \ldots$ of random variables is a **martingale** with respect to another sequence $X_0, X_1, \ldots$ of random variables if for all $n \geq 0$

- $Y_n$ is a function of $X_0, X_1, \ldots, X_n$,
- $\mathrm{E}\left[\,|Y_n|\,\right] < \infty$, and
- $\mathrm{E}\left[Y_{n+1} \mid X_0, X_1, \ldots, X_n\right] = Y_n$.

The last property is called the **martingale property**. When $X_n$ contains all the information in the $X_i$ for $i < n$, we can write it more succinctly as $\mathrm{E}\left[Y_{n+1} \mid X_n\right] = Y_n$.

A random variable $\tau$ that takes values from $\mathbb{N}$ is a **stopping time** with respect to $X_0, X_1, \ldots$ if, for all $n \geq 0$, the event $\tau = n$ depends only on $X_0, X_1, \ldots, X_n$.

See [14, Sect. 10.10] for a proof of Doob's Optional Stopping Theorem, of which the following is a special case.

**Theorem 4.** *If $Y_0, Y_1, \ldots$ is a martingale and $\tau$ is a stopping time, both with respect to $X_0, X_1, \ldots$, and $\tau$ is bounded, then $\mathrm{E}\left[Y_\tau\right] = \mathrm{E}\left[Y_0\right]$.*

For some applications, it makes sense to replace the martingale property with an inequality. A **supermartingale** is a process defined as above except that $Y_n \geq \mathrm{E}\left[Y_{n+1} \mid X_0, \ldots, X_n\right]$; where a martingale stays the same on average, a supermartingale is non-increasing on average. A straightforward induction shows that supermartingales satisfy $Y_k \geq \mathrm{E}\left[Y_n \mid X_0, \ldots, X_k\right]$ whenever $k \leq n$.