Average Case Analysis of QuickSort and Insertion Tree Height using Incompressibility

Tao Jiang, Ming Li, Brendan Lucier

September 26, 2005

Abstract

In this paper we study the Kolmogorov Complexity of a Binary Insertion Tree. We obtain a simple incompressibility argument that yields an asymptotic analysis of average tree height. This argument further implies that the QuickSort algorithm sorts a permutation of n elements in $\Theta(n \log n)$ comparisons on average.

1 Introduction

In this paper we use an incompressibility argument to show that the QuickSort algorithm runs in time $\Theta(n \lg n)$ on average. We do this by showing that the Recursion tree for the QuickSort algorithm, which is isomorphic to the so-called Binary Insertion Tree, has average height $\Theta(\lg n)$. This is interesting because while the QuickSort algorithm has a simple traditional average-case analysis, the traditional analysis of the average height of a Binary Insertion Tree is far more difficult.

In Section 2 we make necessary definitions and give an overview of known and general results. In Section 3 we give our core incompressibility argument as Theorem 3.2. In Section 4 we apply our incompressibility result to give our average case analyses. Conclusions and avenues for future research are outlined in Section 5.

2 Definitions and Known Results

2.1 Permutations

A *permutation* is a sequence of integers such that no integer occurs more than once. Except when otherwise stated, it is assumed that a permutation contains precisely the values $\{1, \ldots, n\}$ for some $n \geq 0$. Given a permutation π, the value $\pi[i]$ refers to the sequence element in position i of π, for $1 \leq i \leq n$.

In permutation π, the element $\pi[1]$ is referred to as the *pivot* for π. The subpermutation π_L is π restricted to values less than $\pi[1]$. The subpermutation π_R is π restricted to values greater than $\pi[1]$. Note that π is uniquely determined by $\pi[1]$, π_L, π_R, and the locations of the elements of π_L or π_R, since we can reconstruct π by shuffling π_L and π_R in the appropriate manner and prepending $\pi[1]$.

2.2 Binary Insertion Trees

Given permutation π, we wish to define the *Binary Insertion Tree* $T(\pi)$. Let $T(\pi)$ be the binary search tree obtained by beginning with the empty tree and inserting the elements of π in order, as leaves. That is, the tree is not rotated as it is created; each new entry is added as a leaf to the existing tree. Let $h(\pi)$ denote the height of $T(\pi)$. Given $y \in \pi$, let $d(y)$ denote the depth of y, $1 \leq d(y) \leq h(\pi)$.

Given $y \in \pi$, we denote by $T(y)$ the subtree of $T(\pi)$ rooted at y consisting of all descendents of y. Let $R(y)$ denote the number of nodes in $T(y)$. Suppose x and z are the (possibly null) children of y in $T(\pi)$.

We say x is the *lesser child* (or simply a *lesser node*) if $R(x) < R(z)$ or if $R(x) = R(z)$ and x is the left child of y. Heuristically, the lesser child of y is the root of the smaller of its two subtrees. We also say that y is *balanced* if $R(x), R(z) > \frac{1}{4}R(y)$. Heuristically, y is balanced if it occurs in the middle half of its range.

A *k-path* of $T(\pi)$ is a sequence $Y = (y_1, \ldots, y_k)$ of nodes such that y_1 is the root of $T(\pi)$ and y_i is the parent of y_{i+1} for each i. Let $B(Y)$ and $U(Y)$ denote the number of balanced and unbalanced nodes in Y, so $B(Y) + U(Y) = k$. Let $L(Y)$ denote the number of lesser nodes in Y.

Now, for each i, if y_i is a lesser child then $R(y_i) \leq \frac{1}{2}R(y_{i-1})$. Since $R(y_1) = n$ and $R(y_j) \geq 1$ for all j, we conclude that

$$L(Y) \leq \lg n. \tag{1}$$

Similarly, if y_i is balanced then $R(y_{i+1}) \leq \frac{3}{4}R(y_i)$. We conclude that

$$B(Y) \leq \log_{\frac{4}{3}} n. \tag{2}$$

2.3 The QuickSort Algorithm

We consider the deterministic QuickSort algorithm, which is described in pseudo-code below.

```
1. QuickSort( Array π )
2.    If |π| = 0 return
3.    Let p = π[1]
4.    Let π_L = (x ∈ π, x < p) in stable order
5.    Let π_R = (x ∈ π, x > p) in stable order
6.    QuickSort(π_L)
7.    QuickSort(π_R)
8.    π = π_L p π_R
```

The value p is referred to as a pivot. The determinism of this algorithm stems from the fact that the pivot always chosen as the first element in a subpermutation. This motivates our definition of a pivot in the previous section.

In lines 4 and 5, the pivot p is compared with the other $n-1$ elements of π. The values are then partitioned into those less than (and those greater than) p, retaining their order in π. These subpermutations are sorted recursively and recombined to give π in sorted order.

We can view the operation of the QuickSort algorithm on π as a tree, which we will call the QuickSort Tree $T_Q(\pi)$. In this tree, we take p from line 2 as the root, with left and right subtrees $T_Q(\pi_L)$ and $T_Q(\pi_R)$.

Lemma 2.1. *For any π, $T_Q(\pi) = T(\pi)$.*

Proof. We proceed by induction on $|\pi|$. If $|\pi| = 0$ then both $T(\pi)$ and $T_Q(\pi)$ are the empty tree. If $|\pi| > 0$ then the root of $T(\pi)$ is the first element of π, while the root of $T_Q(\pi)$ is the pivot obtained on the first pass of the QuickSort algorithm which is the first element of π. The roots of $T(\pi)$ and $T_Q(\pi)$ are therefore the same.

The left subtree of $T_Q(\pi)$ is simply $T_Q(\pi_L)$, the QuickSort tree of π_L. The left subtree of $T(\pi)$ is created by inserting all values less than the root in the order in which they are encountered in π, which is simply π_L. Thus the left subtree of $T(\pi)$ is $T(\pi_L)$. But $T_Q(\pi_L) = T(\pi_L)$ by induction. Similarly, the right subtrees of $T_Q(\pi)$ and $T(\pi)$ are the same (and equal $T(\pi_R)$). We conclude $T_Q(\pi) = T(\pi)$. □

We can now drop the notation $T_Q(\pi)$ and use $T(\pi)$ to refer to both the QuickSort Tree and the Binary Insertion Tree for π. We shall also carry over the notation from $T(\pi)$ to QuickSort: for each $y \in \pi$, there is an associated QuickSort Subtree $T(y)$. Also, note that $R(y)$ is the size of the subpermutation of which y is a pivot.

Lemma 2.2. *The running time for QuickSort on π is $\sum_{y \in T(\pi)} R(y) - n$.*

Proof. For each y in $T(\pi)$, $R(y)$ is the number of elements being considered when y is chosen as a pivot. Since y is compared with all of these elements besides itself, the number of elements with which y is compared as a pivot is $R(y) - 1$. Thus the total number of comparisons is

$$\sum_{y \in T(\pi)} (R(y) - 1) = \sum_{y \in T(\pi)} R(y) - n$$

as required. □

2.4 Kolmogorov Complexity Results

In this paper, a *string* is a finite binary sequence of 0's and 1's. The length of a string x is the number of characters it contains, denoted $|x|$. String y is a *prefix* (respectively, *suffix*) of string x if there exists string z such that $yz = x$ ($zy = x$). Given a Turing Machine T and string x, $T(x)$ denotes the output of T when x is the initial string on the input tape. Given n strings $x_1, \ldots, x_n$, the value $T(x_1, \ldots, x_n)$ denotes the output of T when given x_i as input strings in a distinguishable way, such as being given on separate input tapes. Finally, we shall not distinguish between integers and binary representations thereof when denoting input to a Turing Machine. That is, for integer i, $T(i)$ is equivalent to $T([i]_2)$ where $[i]_2$ denotes the binary representation of i.

The *Kolmogorov Complexity* of a string x can be informally defined as the length of the shortest binary program that generates x when given as input to a fixed universal Turing Machine. It is a well-known result that this definition is independent of the choice of Universal Turing Machine up to an additive factor [3]. We can therefore denote the Kolmogorov Complexity of a string x unambiguously as $C(x)$. The *Conditional Kolmogorov Complexity* $C(x|y)$ is the length of the shortest binary program that generates x when the additional input y is known.

More formally, take an enumeration $T_1, T_2, \ldots$ of all Turing Machines. Let U be a Universal Turing Machine such that

$$U(i, y) = T_i(y)$$

for all $y \in \{0,1\}^*$ and $i \geq 1$. Then

$$C(x|y) = \min_{q \in \{0,1\}^*} \{|q| \colon U(q, y) = x\}$$

and $C(x) = C(x|\epsilon)$.

We now review some general results regarding Kolmogorov Complexity. The following result is the core concept behind incompressibility arguments. We include the statement for completeness and future reference; for a proof, see [5].

Lemma 2.3. *Let δ be a positive integer. For every fixed y, every set S with $|S| = m$ has at least $m(1-2^{-\delta})+1$ elements x with $C(x|y) \geq \lfloor \lg m \rfloor - \delta$.*

The next lemma will also be quite useful. It shows that a binary string with linearly more 0s than 1s is linearly compressible. Let $H(\alpha) = -\alpha \lg \alpha - (1-\alpha)\lg(1-\alpha)$ be the entropy function. Given a binary string x, let $n_1(x)$ denote the number of 1s in x.

Lemma 2.4. *Suppose $x \in \{0,1\}^n$ with $n_1(x) \leq \alpha n$ where $\alpha \leq \frac{1}{2}$. Then if n is known there is a self-delimiting encoding $E(x)$ such that $|E(x)| \leq H(\alpha)n + O(\lg n)$.*

Proof. Our string $E(x)$ will specify $n_1(x)$ then give the positions of the 1s by indexing into the $\binom{n}{n_1(x)}$ possibilities. Note $n_1(x)$ can be represented in $\lceil \lg n \rceil$ bits. Also, since $\alpha n \leq \frac{1}{2}n$, $\binom{n}{n_1(x)} \leq \binom{n}{\alpha n}$. We therefore

get

$$\begin{aligned}
|E(x)| &\leq \left\lceil \lg \binom{n}{\alpha n} \right\rceil + \lceil \lg n \rceil \\
&= \lg \left[\frac{n^n}{(\alpha n)^{\alpha n}(n-\alpha n)^{n-\alpha n}} \right] + O(\lg n) + O(\lg n) \quad \text{(using Sterling's Approximation)} \qquad (3) \\
&= n \lg n - \alpha n \lg(\alpha n) - (1-\alpha) n \lg(n - \alpha n) + O(\lg n) \\
&= H(\alpha) n + O(\lg n).
\end{aligned}$$

□

This final lemma is due to Tao [2]. It allows us to encode a set of choices, the number of options for each depending on the previous choices, without requiring extra bits per choice. Suppose we define sets $A_1, \ldots, A_t$ incrementally, such that A_1 is fixed, and A_{i+1} depends on a choice of element $a_i \in A_i$ for each $i \geq 1$. For any incremental choice of values $a = (a_1, \ldots, a_t)$, we say that the corresponding set of options $A_1 \times \cdots \times A_t$ is the *decision space* for a.

Lemma 2.5. *Suppose a is an incremental choice of values with decision space size d. Then a can be encoded in $\lg d + 1$ bits.*

Proof. Suppose $a = (a_1, \ldots, a_t)$ in decision space $A_1 \times \cdots \times A_t$. Let k_i be the index of a_i in A_i for each i. Define integer $k = k_1 + k_2|A_1| + k_3|A_1||A_2| + \cdots + k_t \prod_{j=1}^{t-1} |A_j|$. Then k is a value between 0 and $\prod_{j=1}^{t-1} |A_j|$. Our encoding is simply the binary representation of k, which has length $\lceil \lg[\prod_{j=1}^{t-1} |A_j|] \rceil \leq \lg d + 1$.

To decode $(a_1, \ldots, a_t)$ from k, proceed as follows. Since $|A_1|$ is fixed and known, we can take $k_1 = k \mod |A_1|$. This gives us a_1, which determines $|A_2|$. We now take $k_2|A_1| + k_1 = k \mod |A_1||A_2|$. This gives us k_2 and hence a_2, which determines $|A_3|$. Proceeding in this way, we can retrieve all of $(a_1, \ldots, a_t)$. □

3 Compressing Binary Insertion Trees

We now describe a technical analysis of the Kolmogorov Complexity of Binary Insertion Trees. The key idea behind this analysis is the development of a special encoding scheme for permutations that reflects the structure of a Binary Insertion Tree. We then show how to compress such an encoding if the height of the corresponding tree is sufficiently large. This will imply, by incompressibility, a bound on the average height of Binary Insertion Trees.

3.1 Encoding Permutations

Let π be a permutation of length n. We shall describe π recursively as a series of decisions. We simultaneously show that the corresponding decision space for π has size $n!$, by induction.

If n is 0 or 1, there is only one possibility for π so no description is necessary, and $1 = 0! = 1!$. If $n > 1$ then we first choose the pivot $p = \pi[1]$, for which there are n choices. We then choose the positions for the $p-1$ elements of π less than p, for which there are $\binom{n-1}{p-1}$ choices. Finally, we recursively describe the permutations π_L of size $p-1$ and π_R of size $n-p$. Induction gives us that π_L and π_R can be reconstructed from their descriptions, and their decision spaces have sizes $(p-1)!$ and $(n-p)!$ respectively. By the discussion of Section 2.1, π can be reconstructed from these four pieces of information, and can therefore be reconstructed from this recursive decision process. Finally, the decision space for π has size $n\binom{n-1}{p-1}(p-1)!(n-p)! = n!$.

Now note that the choices for π_L, π_R, and the value positions depend on the choice of p, but they occur after the choice for p in our description. Thus Lemma 2.5 applies, so we can encode the choices made by this process as $E(\pi)$, with $|E(\pi)| \leq \lg n! + 1$.

3.2 Compression

We now define a method for compressing $E(\pi)$. Suppose $Y = (y_1, \ldots, y_k)$ is a k-path of $T(\pi)$. Let x_Y be a k-bit binary string such that $x_Y[i] = 1$ iff y_i is balanced. Let z_Y be a k-bit binary string such that $z_Y[i] = 1$ iff y_{i+1} occurs in the smaller subrange of y_i for $i < k$. Set $z_Y[k] = 0$.

Define $E(\pi|x_Y, z_Y)$ to be $E(\pi)$ with the following change. Whenever a y_i is chosen as a pivot in the decision process encoded by $E(\pi)$, we modify the set of choices for y_i. We choose y_i only among those values that are balanced (resp: unbalanced) if y_i is balanced (unbalanced). Since half of the values in any given range are unbalanced, this change reduces the size of the decision space by half for each value y_i. Thus the decision space is reduced by a factor of 2^{-k} overall. Lemma 2.5 then implies that $|E(\pi|x_Y, z_Y)| \leq \lg n! - k + 1$.

We now show how to retrieve $E(\pi)$ from $E(\pi|x_Y, z_Y)$, x_Y, and z_Y. Starting at $i = 1$, use bit i of x_Y to determine whether y_i is balanced or not, so the indexing of the choice for y_i can be expanded to a choice among all values. Replace the encoding for y_i with this expanded version. Now bit i of z_Y can be used to determine which child of y_i is y_{i+1}. The choice for y_{i+1} can then be found in $E(\pi|x_Y, z_Y)$ and expanded with the next bit of x_Y, etc.. Thus all modified choices can be expanded, and the result is $E(\pi)$.

We have now shown the following:

Lemma 3.1. *Let Y be a k-path of $T(\pi)$, and define x_Y, z_Y as above. Then $C(\pi|n, x_Y, z_Y) \leq \lg n! - k + O(1)$.*

Proof. Provide $E(\pi|x_Y, z_Y)$ to a a constant-sized program that retrieves $E(\pi)$ as above. Then π can be reconstructed from $E(\pi)$. □

3.3 Tree Height

We now calculate the amount of compression achieved by Lemma 3.1, and use this result to bound the height of $T(\pi)$ for most π.

Theorem 3.2. *There exists a constant c such that any permutation π with $h(\pi) \geq c \lg n$ is $\lg n$-compressible.*

Proof. Let $k = c \lg n$ for sufficiently large c. Suppose $h(\pi) \geq c \lg n$, so $T(\pi)$ has a k-path $Y = (y_1, \ldots, y_k)$. Define strings x_Y, z_Y, and $E(\pi|x_Y, z_Y)$ as in Lemma 3.1.

Now we consider an encoding for x_Y and z_Y. From (2), $n_1(x_Y) = B(Y) \leq \alpha \lg n$, where $\alpha = \frac{1}{\lg(4/3)}$. Similarly, $n_1(z_Y) = L(Y) \leq \lg n$ by (1). Thus $|x_Y z_Y| = 2c \lg n$, but $n_1(x_Y z_Y) \leq (\alpha + 1) \lg n$. By Lemma 2.4 we therefore know $x_Y z_Y$ can be encoded as $E(x_Y z_Y)$ if $c > \alpha + 1$.

Since $E(x_Y z_Y)$ is self-delimiting given n, we get that

$$\begin{aligned} C(\pi|n, p) &\leq |E(x_Y z_Y)| + C(\pi|x_Y, z_Y, n) \\ &\leq 2cH\left(\frac{\alpha+1}{2c}\right)\lg n + O(\lg\lg n) + \lg(n!) - c\lg n + O(1) \quad \text{(by Lemma 2.4, Lemma 3.1)} \\ &= \lg(n!) - c\left(1 - 2H\left(\frac{\alpha+1}{2c}\right)\right)\lg n + O(\lg\lg n). \end{aligned} \tag{4}$$

Thus, if

$$\gamma = c\left(1 - 2H\left(\frac{1+\alpha}{2c}\right)\right) > 1 \tag{5}$$

we get that $C(\pi|n, p) \leq \lg n! - \gamma \lg n + O(\lg\lg n)$, and hence π is $\lg n$-compressible for sufficiently large n. But the value in (5) approaches ∞ as $c \to \infty$, so (5) will be true for sufficiently large c. A computer-aided calculation shows that $c = 16.965\ldots$ is the minimal solution to (5).

□

3.4 Analyzing the Constant Factor

In this section we provide a method for improving the constant c in Theorem 3.2. Let x_Y and z_Y be defined as in the proof of Theorem 3.2. We now present a more complex compression method for x_Y and z_Y which results in a tighter constant c. Encode x_Y as $E(x_Y)$ using the standard compression technique as in Lemma 3.1. Then $|E(x_Y)| = H(\frac{\alpha}{c})c \lg n$, since $n_1(x_Y) \leq \alpha \lg n$.

We now define an encoding $E(z_Y|x_Y)$ for z_Y. Note that if $z_Y[i] = 1$ and $x_Y[i] = 0$ for some $1 \leq i \leq c \lg n$, then y_i is unbalanced and y_{i+1} lies in the smaller subtree of y_i. Thus $R(y_{i+1}) \leq \frac{1}{4}R(y_i)$. We conclude that at most $\log_4 n = \frac{1}{2}\lg n$ values i can satisfy $z_Y[i] = 1$ and $x_Y[i] = 0$. Suppose there are t such values of i. Then, given x_Y, we can encode z_Y by specifying $n_1(z_Y)$ and t, describing the positions of the 1s that occur where x_Y has a 0, then describing the positions of the 1s that occur where x_Y has a 1.

Recall that $n_1(z_Y) \leq \lg n$, so we can first specify $n_1(z_Y)$ and t in $O(\lg \lg n)$ bits. We then specify the choices for locations of 1s in z_Y. There are $\binom{c \lg n - n_1(x_Y)}{t}$ and $\binom{n_1(x_Y)}{n_1(z_Y)-t}$ choices for locations of 1s in z_Y where the corresponding digit of x_Y is 0 or 1, respectively. Subject to the constraints on $n_1(z_Y)$, $n_1(x_Y)$, and t, this number of choices is maximized when $n_1(x_Y) = \alpha \lg n$, $n_1(z_Y) = \lg n$, and $t = \frac{1}{2}\lg n$.

We now have that

$$\begin{aligned} |E(z_Y|x_Y)| &\leq O(\lg \lg n) + \lg \binom{(c-\alpha)\lg n}{\frac{1}{2}\lg n} + \lg \binom{\alpha \lg n}{\frac{1}{2}\lg n} \\ &\leq H\left(\frac{1}{2(c-\alpha)}\right)(c-\alpha)\lg n + H\left(\frac{1}{2\alpha}\right)\alpha \lg n + O(\lg \lg n) \end{aligned} \tag{6}$$

using the same analysis as in Lemma 2.4.

But now π can be encoded as $E(x_Y)E(z_Y|x_Y)E(\pi|x_Y, z_Y)$. We conclude that

$$\begin{aligned} C(\pi|n,p) &\leq |E(x_Y)| + E(x_Y|z_Y)| + C(\pi|x_Y, z_Y, n) \\ &\leq H\left(\frac{\alpha}{c}\right)c\lg n + H\left(\frac{1}{2(c-\alpha)}\right)(c-\alpha)\lg n + H\left(\frac{1}{2\alpha}\right)\alpha \lg n + O(\lg \lg n) + \lg(n!) - c\lg n + O(1) \\ &= \lg(n!) - \left(c - H\left(\frac{\alpha}{c}\right)c - H\left(\frac{1}{2(c-\alpha)}\right)(c-\alpha) - H\left(\frac{1}{2\alpha}\right)\alpha\right)\lg n + O(\lg \lg n). \end{aligned} \tag{7}$$

Thus, if

$$\left(c - H\left(\frac{\alpha}{c}\right)c - H\left(\frac{1}{2(c-\alpha)}\right)(c-\alpha) - H\left(\frac{1}{2\alpha}\right)\alpha\right) > 1 \tag{8}$$

we get that π is $\lg n$-compressible for sufficiently large n, as in Theorem 3.2. We can therefore take c to be the minimal solution to (8). This minimal solution is $c = 15.498\ldots$, which is indeed better than the constant $16.965\ldots$ that results from (5).

4 Applications to Binary Tree Analysis

We now apply Theorem 3.2 of the previous section to obtain asymptotic analyses of Binary Insertion Tree height and QuickSort runtime. It should be noted that these results are not new: an analysis of Binary Insertion Tree height can be found at [1], and the traditional analysis of the QuickSort algorithm is well-known (see, for example, [4]). Although these results are not new, we feel that our analysis of average Binary Insertion Tree height is simpler than that in [1]. We therefore present these alternate proofs as an illustration of how easily one can obtain asymptotic analyses using incompressibility.

Theorem 4.1. *The average height of a binary insertion tree over permutations on n elements is $\Theta(\lg n)$.*

Proof. The lower bound follows trivially from the height of a balanced binary tree. From Theorem 3.2 we know that any $\lg n$-incompressible permutation π has $h(\pi) < c \lg n$. But by Lemma 2.3 there are $(1-2^{-\lg n})n!$

such permutations. Thus only $(2^{-\lg n})n! = (1/n)n!$ permutations can have tree heights larger than $c \lg n$. Since tree heights are trivially bounded by n, we get that the average tree height is no more than

$$\frac{1}{n!}\left[\left(\left(1 - 2^{-\lg n}\right) n!\right) c \lg n + \left(\left(\frac{1}{n}\right) n!\right) n\right] < c \lg n + 1 = O(\lg n) \tag{9}$$

as required. □

Theorem 4.2. *The average runtime of the QuickSort algorithm over all permutations on n elements is $\Theta(n \lg n)$.*

Proof. The lower bound follows trivially from the lower bound on sorting algorithms. By Lemma 2.2 we know that the runtime of QuickSort is less than $\sum_{y \in T(\pi)} R(y)$. But then, since $T(y)$ are disjoint for all y at the same depth, the total cost of QuickSort is

$$C_Q(\pi) < \sum_{y \in T(\pi)} R(y) = \sum_{i=1}^{h(\pi)} \sum_{\substack{y \in T(\pi) \\ d(y)=i}} R(y) < \sum_{i=1}^{h(\pi)} n = nh(\pi). \tag{10}$$

Since $h(\pi) = O(\lg n)$ on average by Theorem 4.1, the average cost for QuickSort is $nO(\lg n) = O(n \lg n)$. □

5 Conclusions and Future Research

We have used Kolmogorov Complexity to show that the average height of a Binary Insertion Tree is $O(\lg n)$. It follows from this result that the average cost of QuickSort is $O(n \lg n)$. While this QuickSort result is certainly interesting and encouraging for the success of the incompressibility method, there are simpler (although, perhaps, less illuminating) analyses that achieve a better constant factor.

The traditional analysis of the average height of a Binary Insertion Tree, however, is not so simple. This problem was first solved in [1] and the analysis is quite difficult. We feel that the Kolmogorov Complexity approach presented in this report is much simpler, and hence a useful contribution to the area of study.

An avenue of future research is to apply the methods of this paper to analyze other algorithms involving binary trees. In particular, the method by which we both specified a path in the tree and gave balancing information about it (i.e. the string xz in the proof of Theorem 3.2) in fewer bits than nodes in the path may prove useful in future analyses. Also, Lemma 2.5 was shown to be quite powerful, and likely has other uses as well.

References

[1] Devroye, Luc. A note on the height of binary search trees. *JACM, 33(3)*, 489-498, July 1986.

[2] Jiang, Tao. Personal Correspondence. July 2005.

[3] Li, Ming and Vitanyi, Paul. *An introduction to Kolmogorov complexity and its applications*, Springer-Verlag, New York, 2nd Edition, 1997.

[4] Skiena, Steven. *Analysis of Algorithms, Lecture 5 - quicksort*, 1997. Retrieved July 2005 from [http://www.cs.sunysb.edu/∼algorith/lectures-good/node5.html].

[5] Vitanyi, Paul. Analysis of Sorting Algorithms by Kolmogorov Complexity (A Survey). December 18, 2003.