ON THE GENERATION OF RANDOM BINARY SEARCH TREES

Luc Devroye
School of Computer Science
McGill University
and
John Michael Robson
Department of Computer Science
Australian National University

ABSTRACT. We consider the computer generation of random binary search trees with n nodes for the standard random permutation model. The algorithms discussed here output the number of external nodes at each level, but not the shape of the tree. This is important for example when one wishes to simulate the height of the binary search tree. Various paradigms are proposed, including depth-first search with pruning, incremental methods in which the tree grows with random-sized jumps, and a tree growing procedure gleaned from birth-and-death processes. The last method takes $O(\log^4 n)$ expected time.

KEYWORDS AND PHRASES. Binary search tree, height of a tree, probabilistic analysis, expected complexity, simulation, random combinatorial object, point process, recursive procedure.

1991 MATHEMATICS SUBJECT CLASSIFICATIONS: Primary: 68Q25. Secondary: 68U20, 93E30, 00A72, 11K45, 65C05, 65C10, 05C80, 68R++, 68P05, 68P10, 60C05.

Authors' addresses: School of Computer Science, McGill University, 3480 University Street, Montreal, Canada H3A 2A7. Department of Computer Science, Australian National University, G.P.O. Box 4, Canberra, A.C.T. 2601, Australia. Luc Devroye's research was sponsored by NSERC Grant A3456 and FCAR Grant 90-ER-0291.

1. Introduction.

The binary search tree is one of the most frequently used structures in computer science, see e.g. Aho, Hopcroft and Ullman (1983), Knuth (1973) or Cormen, Leiserson and Rivest (1990). A random binary search tree is defined as the random binary tree obtained by consecutive insertion of $X_1, \ldots, X_n$ into an initially empty tree, where $X_1, \ldots, X_n$ is either an i.i.d. sequence of random variables with a fixed density, or an equiprobable random permutation of $\{1, \ldots, n\}$. The height H_n of the tree is the maximal distance between any node and the root (thus, $H_1 = 0$, as the root is at distance 0 from itself, and $H_2 = 1$). In this note, we propose various methods for the generation of random binary search trees. Trees with n nodes necessarily require $\Omega(n)$ time. Since many quantities related to random trees such as their heights grow logarithmically with n, large size trees are required in simulations that attempt to extract asymptotic information. In fact, one often does not care about the actual tree, but rather about the number of nodes at each level. The methods given below take sublinear expected time and output a random vector $(n_0, \ldots, n_m)$ where n_i is the number of external nodes at level i. The height H_n is nothing but $m - 1$, for example. Early studies of H_n include Robson (1979, 1982), Pittel (1984) and Mahmoud and Pittel (1984). See also Mahmoud (1992). While it is known that

$$\frac{H_n}{\log n} \to c \stackrel{\text{def}}{=} 4.33107\ldots \text{a.s.}$$

as $n \to \infty$ (Devroye, 1986, 1987), very little additional information is available regarding H_n, and one is led to simulation in order to study the second order properties of H_n such as its variance, its asymptotic distribution, and so forth. Such simulations require formidable values of n in view of the logarithmic growth of H_n with n. It is thus of great importance to assure that the time and especially the space requirements grow slowly with n.

Constructing the tree by consecutive insertions leads to a $\Theta(n \log n)$ expected time and $\Theta(n)$ space algorithm. One can exploit a certain growth property of the random binary search tree (all external nodes are equally likely to receive a new node), leading to a $\Theta(n)$ time and $\Theta(n)$ space method for growing the tree by direct insertion of new nodes at old leaves. With extra effort, the space requirement can be reduced to $\Theta(\log n)$ expected space.

In section 3, we present a simple method that requires $O(n/\log n)$ expected time and $O(\log n)$ expected space. A shortcut is required as a sublinear time method can-not possibly consider all nodes in the tree. The algorithm presented here uses some sort of depth-first search with tree pruning gleaned from game tree search applications. The idea of simulating random variables via shortcuts that bypass the definition of the random variables has been exploited by many. For example, the maximum of n i.i.d. random variables with a given density f can be generated in expected time $O(1)$ or $O(\log n)$ depending upon whether the distribution function is available at unit cost or not (Schmeiser, 1978; Devroye, 1980). The sum of n i.i.d. random variables can also be generated in time $o(n)$; often, $O(1)$ expected time is achievable, as is demonstrated in Devroye (1986, 1988). A binomial (n, p) random variable representing the outcome of n coin flips is nowadays routinely generated in $O(1)$ expected time (Ahrens and Dieter, 1980; Kachitvichyanukul, 1982; Pokhodzei, 1984, 1988; Devroye, 1987; Kachitvichyanukul and Schmeiser, 1988; Stadlober, 1988, 1989) uniformly bounded over n and p. The result of this paper is just another illustration of the same principle. The method of section 3 is extremely easy to implement, and is competitive with the other methods for values n that are not too large.

In section 4, we recall a simple linear time method based upon growing a random binary search tree by replacing external nodes, i.e., not by insertion from the root down. Probabilistic shortcuts based upon waiting times allow us in section 5 to reduce the expected time to $O(\sqrt{n} \log n)$. This requires efficient generators for a multivariate hypergeometric and a certain waiting time distribution, thereby rendering the programming effort and the overhead a bit heavier. Nevertheless, for medium-sized values of n, the method is very useful.

In section 6, a random binary search tree is grown by imagining that each external node is a living organism that will bear two children and die according to a simple Poisson point process. We then let the time grow by constant amounts, so that the tree grows at an exponential rate. At any given moment, we have a correctly distributed binary search tree, but the size is random. When one stops as soon as the size of the tree is n or larger, the expected time complexity is $O(\log^4 n)$. A modification of the algorithm is introduced to obtain the right size. The constant in $O(\log^4 n)$ is rather large due to the overhead in a multinomial random vector generator used in a bottleneck portion of the algorithm. This last method is useful for extremely large n, such as $n \approx 10^{40}$.

2. Two key properties of random binary search trees.

FACT 1. *If we associate with node i in a random binary search tree an integer S_i denoting the size of the subtree rooted at that node, then for any (i.e., left or right) child j of node i, we have*

$$S_j \stackrel{\mathcal{L}}{=} \lfloor S_i U \rfloor \ ,$$

where $\stackrel{\mathcal{L}}{=}$ means "is distributed as", and U is a uniform $[0,1]$ random variable independent of S_i.

As an immediate corollary of this we have

FACT 2. *Let node i in a random binary search tree have children j and k. Then*

$$\max(S_j, S_k) \stackrel{\mathcal{L}}{=} \lfloor S_i \max(U, 1-U) \rfloor \ , \stackrel{\mathcal{L}}{=} \left\lfloor S_i \frac{1+U}{2} \right\rfloor \ ,$$

where U is a uniform $[0,1]$ random variable independent of S_i.

3. A simple algorithm for the height of a random binary search tree.

3.1. Description. The algorithm keeps a stack of nodes characterized by a pair (l, s), where l is the level of a node (i.e., its distance from the root), and s is the size of the subtree rooted at that node. The actual position of each node in the tree and the actual tree with all its links are never constructed. As we proceed, nodes on the stack are split into two nodes representing the children. A node (l, s) is thus split into $(l+1, s_1)$ and $(l+1, s_2)$ where $s_1 + s_2 = s - 1$. Moreover, s_1 is distributed as $\lfloor sU \rfloor$, where U is uniformly distributed on $[0,1]$. See fact 1 above. The new nodes are put back on the stack with the largest subtree on top. A node (l, s) can at best produce a node at distance $l + s - 1$ from the root, so there is no point in processing or stacking such a node if $l + s - 1$ does not exceed the current value of the height of the tree, i.e., the largest level among all nodes seen thus far. This observation is at the basis of the sublinear expected time: not all nodes in the random binary search tree are expanded; in fact, only a negligible fraction ($O(1/\log n)$) is ever expanded.

```
A sublinear depth-first search algorithm.

makenull (S) (S is a stack).
push ((0,n),S) (put the root onto the stack).
h ← 0 (h is the current value of the height).
while not empty (S) do
  pop ((l,s),S)
  if l+s−1 > h then
    l ← l+1
    if l > h then h ← l
    generate U uniformly on [1/2,1].
    s2 ← ⌊sU⌋, s1 ← s−1−s2.
    if s1 > 1 then if l+s1−1 > h then push ((l,s1),S) .
    if s2 > 1 then if l+s2−1 > h then push ((l,s2),S) .
return h.
```

3.2. Expected stack size. It is easy to verify by induction that every level except the furthest can occur at most once on the stack; the furthest occurs at most twice (just note that the level numbers of the nodes on the stack are strictly increasing except possibly for the top node). This shows without further ado that the maximal stack size is less than one plus the height of the tree, and this in turn is less than $(c+\epsilon)\log n$ almost surely for any $\epsilon > 0$. Furthermore, the expected stack size is not greater than $(c+o(1))\log n$.

3.3. Expected time analysis. Initially, we push and pop a sequence of nodes that correspond to a path down the tree ending with a node with a subtree of size one. After that, the algorithm backtracks by processing a node closer to the root. Let us call L_n the level of this furthest node encountered in the first phase of the algorithm. Let

$$\begin{aligned} S_0 &= n; \\ S_i &= \left\lfloor S_{i-1} \frac{1+U_i}{2} \right\rfloor \quad (i \geq 1) \end{aligned}$$

where $U_1, U_2, \ldots$ are i.i.d. uniform $[0,1]$ random variables. The sequence $S_0, S_1, \ldots$ corresponds to the sizes of the subtrees rooted at the nodes processed on the first pass (see Fact 2). It is clear that $L_n = k$ if and only if $S_{k-1} > 1$ and $S_k = 1$. The asymptotic behavior of L_n is covered by the following result.

FACT 3. *As $n \to \infty$, we have*

$$\frac{L_n}{\log n} \to \frac{1}{\log(e/2)} = 3.258891\ldots \text{ a.s.}$$

as $n \to \infty$. In particular, if $a < 1/\log(e/2)$, then

$$\mathbf{P}\{L_n < a \log n\} = O(n^{-\beta})$$

for some $\beta > 0$ depending upon a.

PROOF. Observe that

$$n \prod_{i=1}^{k} \left(\frac{1+U_i}{2} \right) - k \leq S_k \leq n \prod_{i=1}^{k} \left(\frac{1+U_i}{2} \right) .$$

Hence,

$$\begin{aligned} \mathbf{P}\{L_n \leq k\} &\leq \mathbf{P}\left\{S_k \leq 1\right\} \\ &\leq \mathbf{P}\left\{ n \prod_{i=1}^{k} \left(\frac{1+U_i}{2} \right) \leq k+1 \right\} \\ &\leq \mathbf{P}\left\{ \log n + \sum_{i=1}^{k} \log \left(\frac{1+U_i}{2} \right) \leq \log(k+1) \right\} \\ &= \mathbf{P}\left\{ \frac{1}{k} \sum_{i=1}^{k} \log \left(\frac{1+U_i}{2} \right) \leq -\frac{\log n}{k} + \frac{\log(k+1)}{k} \right\} . \end{aligned}$$

Now take $k = \lceil a \log n \rceil$ with $0 < a < 1/\log(e/2)$. We now apply a large deviation theorem (see e.g., Petrov, 1975, or section 1.9 of Durrett, 1990) which states that if

$Y_1, Y_2, \ldots$ are i.i.d. random variables with mean μ, and if $\mathbf{E} e^{-tY_1} < \infty$ for some $t > 0$, then, for every $\epsilon > 0$, there exists a $\delta > 0$ such that

$$\mathbf{P}\left\{\frac{1}{k}\sum_{i=1}^{k} Y_i < \mu - \epsilon\right\} \leq e^{-k\delta} \ .$$

Now, applied to our situation, noting that

$$\mathbf{E}\log\left(\frac{1+U_i}{2}\right) = \log(2/e)$$

and that

$$-\frac{\log n}{k} + \frac{\log(k+1)}{k} = -\frac{1}{a} + o(1) \ ,$$

we obtain

$$\mathbf{P}\left\{\frac{1}{k}\sum_{i=1}^{k}\log\left(\frac{1+U_i}{2}\right) \leq -\frac{\log n}{k} + \frac{\log(k+1)}{k}\right\} = O(e^{-\delta\log n}) = O(n^{-\delta})$$

for some $\delta > 0$, provided that $-1/a < \log(2/e)$. This proves the second half of Fact 3. In the same manner, one can show that

$$\mathbf{P}\{L_n > a\log n\} = o(1)$$

for all $a > 1/\log(e/2)$. $\square$

Theorem. *The expected time taken by the algorithm given above is $O(n/\log n)$.*

PROOF. The running time of the algorithm is appropriately measured by the number of nodes that are ever stacked. On the first pass just described, we push at most $2(L_n+1)$ onto the stack. After this first pass, only nodes with the property that the (l,s) pair satisfies $l+s-1>L_n$ can ever be pushed onto the stack. The expected number of nodes pushed on the stack after the first pass is not more than

$$
\begin{aligned}
&\mathbf{E}\left\{\min\left(n,\sum_{k=1}^{\infty}2^k I_{[N_k\geq 1,k+N_k-1>L_n]}\right)\right\}\\
&\leq \sum_{k\geq a\log n}2^k\mathbf{P}\{N_k\geq 1\}+\sum_{k<a\log n}2^k\mathbf{P}\{a\log n+N_k-1>3.2\log n\}+n\mathbf{P}\{L_n<3.2\log n\}\\
&\stackrel{\text{def}}{=} I+II+III\ ,
\end{aligned}
$$

where N_k is the size of the subtree of a typical node at distance k from the root. Here we took into account that there are potentially 2^k nodes at distance k from the root. Observe next that

$$N_k\stackrel{\mathcal{L}}{=}\lfloor N_{k-1}U_k\rfloor\ ,\ N_0=n\ ,$$

where $U_1,U_2,\ldots$ are i.i.d. uniform $[0,1]$ random variables. Thus,

$$N_k\leq n\prod_{i=1}^{k}U_i\stackrel{\mathcal{L}}{=}e^{\log n-G_k}$$

where G_k is a gamma (k) random variable. We have

$$\mathbf{P}\{N_k\geq 1\}\leq\mathbf{P}\{\log n-G_k\geq 0\}\leq\frac{(\log n)^k e^{-\log n}}{k!(1-\log n/(k+1))}$$

by an inequality for the left tail of the gamma distribution found e.g., in Devroye (1987). As a result of this estimate, we see that for $a>1$,

$$
\begin{aligned}
I&=\sum_{k\geq a\log n}2^k\mathbf{P}\{N_k\geq 1\}\\
&\leq\sum_{k\geq a\log n}\frac{(2\log n)^k e^{-\log n}}{k!(1-1/a)}\\
&=\frac{a}{a-1}e^{\log n}\sum_{k\geq a\log n}\frac{(2\log n)^k e^{-2\log n}}{k!}\\
&=\frac{an}{a-1}\mathbf{P}\{Z\geq a\log n\}\ ,
\end{aligned}
$$

where Z is a Poisson $(2\log n)$ random variable. We will see that we need to take $a\in(2,3.2)$. By Chebyshev's inequality, since $a>2$, we see that

$$\mathbf{P}\{Z\geq a\log n\}\leq\frac{\mathrm{Var}\{Z\}}{(a-2)^2\log^2 n}=\frac{2}{(a-2)^2\log n}\ .$$

We conclude that for $a > 2$,

$$I \le \frac{2an}{(a-1)(a-2)^2 \log n} \ .$$

Next,

$$III = n\mathbf{P}\{L_n < 3.2 \log n\} \le n^{1-\delta}$$

for some $\delta > 0$ by Fact 3. Finally, if $y = \log n - \log\log n - \log(3.2 - a)$,

$$\begin{aligned}
II &= \sum_{k < a \log n} 2^k \mathbf{P}\{a \log n + N_k - 1 > 3.2 \log n\} \\
&\le \sum_{k < 1.1 \log n} 2^{1+1.1 \log n} + \sum_{1.1 \log n \le k < a \log n} 2^k \mathbf{P}\{N_k - 1 > (3.2 - a) \log n\} \\
&\le 2n^{0.77} + \sum_{1.1 \log n \le k < a \log n} 2^k \mathbf{P}\{\log n - G_k > \log(3.2 - a) + \log\log n\} \\
&\le 2n^{0.77} + \sum_{1.1 \log n \le k < a \log n} 2^k \frac{y^k e^{-y}}{k!(1 - y/(k+1))} \\
&\le 2n^{0.77} + 11\, e^y \sum_{1.1 \log n \le k < a \log n} \frac{(2y)^k e^{-2y}}{k!} \\
&\le 2n^{0.77} + 11\, e^y \\
&= 2n^{0.77} + \left(\frac{11}{3.2 - a}\right) \frac{n}{\log n} \ .
\end{aligned}$$

Thus, $I + II + III = O(n/\log n)$. $\square$

4. A simple linear time algorithm.

In Devroye (1986, p.650), a linear time method is given for generating a random binary search tree. The basic algorithm is shown below.

```
m ← 0 (m is the number of levels)
n_0 ← 1
for N := 1 to n do (N is the number of external nodes)
  generate L randomly in {0,...,m}
    according to the frequencies n_0,...,n_m
  n_L ← n_L − 1
  if L = m then m ← m + 1
  n_{L+1} ← n_{L+1} + 2
return (n_0, n_1, ..., n_m)
(the height of the tree is m − 1)
```

In this algorithm, we keep the number of external nodes at each level in an array $(n_0, n_1, \ldots, n_m)$. The expected storage needed for this is $O(\log n)$ since the expected height is $O(\log n)$. The algorithm is based upon the fact that when adding a new node, each of the external nodes is equally likely to receive the node. To generate the random integer L according to the vector of frequencies $(n_0, \ldots, n_m)$, one can trivially proceed in time $O(1+m)$, but this would result in overall expected time $O(n \log n)$. To generate L in $O(1)$ expected time, one has to either use more space or more programming resources. For example, keeping an array with $n_0 + n_1 + \cdots n_m$ entries, of which n_i entries have label i, would enable us to generate L in $O(1)$ time. The overall time and space both are $O(n)$. By a dynamic version of the guide table method (see Chen and Asau, 1974, or Fishman and Moore, 1984, for the raw guide table method), $O(\log n)$ expected space is achievable provided that we can update the guide table in $O(1)$ amortized time. This is easy to achieve if we take care to rebuild the table every $\log n$-th (or m-th) entry. Between rebuilding, the new entries in the table are collected in a simple overflow list; this does not affect the overall linear expected time.

5. Discrete jumps in the simulation: an $\mathbf{O}(\sqrt{\mathbf{n}}\log\mathbf{n})$ method.

5.1. Description. Consider a vector $(n_0, n_1, \ldots, n_m)$ representing the number of external nodes at levels $0, 1, \ldots, m$ respectively, and assume that $n_0 + \ldots + n_m = n$ for now. The previous linear time algorithm is based upon the selection of a uniform random external node, say one at level $k \in \{0, \ldots, m\}$, and the updating of the vector by the rule

$$\begin{cases} n_k \leftarrow n_k - 1 \\ n_{k+1} \leftarrow n_{k+1} + 2 \end{cases} .$$

Imagine that the n original external nodes are white balls in an urn, and that the label of each ball is its level number. A randomly selected (white) ball is removed. If its label is k, two black balls with label $k+1$ are added. This process can be repeated until we pick for the first time a black ball. The number of draws required until this happens is a random variable $T_n \in \{2, \ldots, n+1\}$. We say that T_n has the *waiting-time distribution* with parameter n. We can let the tree growing process jump ahead by T steps at once if we are given T. Indeed, given T, it suffices to draw $T-1$ white balls uniformly and without replacement from the urn. The vector $(D_0, D_1, \ldots, D_m)$ represents the number of balls drawn with labels $0, 1, \ldots, m$ respectively. The distribution of this vector is multivariate hypergeometric; the details on how to generate the vector on a computer will be given later; it suffices for now that this vector can be generated in $O(m)$ expected time uniformly over all n. Now, the vector of external nodes is updated by the rule:

$$\begin{aligned} (n_0, \ldots, n_m, n_{m+1}) \leftarrow\ & (n_0, \ldots, n_m, 0) \\ & - (D_0, D_1, \ldots, D_m, 0) \\ & + 2(0, D_0, \ldots, D_m) . \end{aligned}$$

The single black ball is taken care of by selecting a label L at random from the $T-1$ white ball labels just selected, the k-th label being picked with probability proportional to D_k. This label can be chosen in time $O(m)$ by the trivial algorithm

```
generate X uniformly on {1,...,T-1}.
S ← D_0, L ← 0.
while X > S do
   L ← L + 1
   S ← S + D_L
return L
```

A further update is required:

$$(n_{L+1}, n_{L+2}) \leftarrow (n_{L+1} - 1, n_{L+2} + 2) \ ,$$

where, if $L = m + 1$, we define $n_{m+2} = 0$ before the update and $n_{m+2} = 2$ after the change. The number of external nodes now is $n + T$ instead of n. Iterate this process until we obtain more external nodes than needed. It is a simple matter to get the exact size one wants by simply truncating T in the last iteration. Let us first provide the algorithm in its entirety:

```
N ← 1 (N is the number of external nodes)
m ← 0 (m is the number of levels)
n_0 ← 1
repeat
  generate T ∈ {2,...,N+1} with the waiting time distribution
    with parameter N
  if T + N > n + 1
    then T ← n + 2 − N, S ← −1
    else generate S uniformly in {0,...,T − 1}
  generate a multivariate hypergeometric (D_0,...,D_m)
      with parameters T − 1 and (n_0,...,n_m)
  if D_m > 0
    then m ← m + 1
         (n_0,...,n_m) ← (n_0,...,n_{m−1},0) − (D_0,...,D_{m−1},0) + 2(0,D_0,...,D_{m−1})
    else (n_0,...,n_m) ← (n_0,...,n_m) − (D_0,...,D_m) + 2(0,D_0,...,D_{m−1})
  if S ≠ −1
    then generate a random integer L ∈ {0,...,m}
         from S by inversion according to the frequencies D_0,...,D_m
         n_{L+1} ← n_{L+1} − 1
         if L + 2 ≤ m then n_{L+2} ← n_{L+2} + 2
                      else n_{L+2} ← 2, m ← m + 1
until T + N ≥ n + 1
return (n_0,n_1,...,n_m)
```

The algorithm given above returns a vector with m components n_i, where n_i is the number of external nodes on level i. Clearly, for a random binary search tree with n nodes, we have $n+1$ external nodes, and thus $\sum_i n_i = n+1$. Thus, besides the height of the binary search tree $(m-1)$, we also have information about the distribution of the nodes over the various levels.

A few details have to be ironed out:

- Determine the waiting time distribution for T and show how a random variate T can be generated in constant expected time, bounded uniformly over n.

- Show how one can generate a multivariate hypergeometric random variate.
- Show that the algorithm takes $O(\sqrt{n}\log n)$ expected time units.

5.2. The distribution of T. By using the urn with white balls and black balls, we see that $\mathbf{P}\{T > k\}$ is equal to the probability that in the first k draws only white balls are chosen. Clearly then, if the urn has n white balls to begin with,

$$\mathbf{P}\{T > k\} = \prod_{i=1}^{k-1} \frac{n-i}{n+i} \ , \ 2 \le k \le n+1$$

so that

$$\mathbf{P}\{T = k+1\} = \frac{2k}{n} \frac{\binom{2n}{n+k}}{\binom{2n}{n}} \ , \ 1 \le k \le n \ .$$

Random variate generation can be dealt with by von Neumann's rejection method, which requires a summable function of k that dominates the probability vector given above. In von Neumann's method, worked out below for our case, it suffices to generate random variates from a distribution with probabilities proportional to the bounding vector and stop when for the first time an acceptance condition is satisfied. We first derive an upper bound from the following inequality:

$$\mathbf{P}\{T = k+1\} \le \frac{2k}{n} e^{-2k^2/3n} \ , \ 1 \le k \le n \ .$$

This can be shown as follows. Using $\log(1+u) \ge 2u/(2+u)$ for $u > 0$, we have

$$\begin{aligned}
\mathbf{P}\{T = k+1\} &\le \frac{2kn^{k-1}}{(n+k)^k} \le \frac{2k}{n} e^{-2k^2/(2n+k)} \\
&\le \frac{2k}{n} e^{-2k^2/3n} \\
&\le \frac{2(x+1)}{n} e^{-2x^2/3n} \ , \ k-1 < x \le k \ .
\end{aligned}$$

Observe that

$$\begin{aligned}
\int_0^\infty \frac{2x}{n} e^{-2x^2/3n} &= \frac{3}{2} \ ; \\
\int_0^\infty \frac{2}{n} e^{-2x^2/3n} &= \sqrt{\frac{3\pi}{2n}} \ .
\end{aligned}$$

Thus, the density $\frac{2(x+1)}{n} e^{-2x^2/3n}$ on the positive halfline is the mixture of the Maxwell density and the normal density. The rejection method for T can be summarized as follows:

```
generator for T with parameter n

repeat
  generate U,V uniform [0,1]
  if U < (3/2)/(3/2+sqrt(3π/2n))
    then generate an exponential random variate E; set Y ← sqrt(3nE/2)
    else generate N standard normal; set Y ← sqrt(3nN²/4)
  X ← ⌈Y⌉
until Y ≤ n and V (2X+2)/n exp(−2X²/3n) < (2Y/n) (binom(2n, n+Y)/binom(2n, n))
return T ← Y + 1
```

The expected number of iterations in this algorithm is $3/2 + \sqrt{3\pi/4n}$. The expected time is uniformly bounded over n if we can evaluate factorials in $O(1)$ time or if we can verify the acceptance condition in $O(1)$ time. If we accept a model in which simple basic operations take constant time, regardless of the size of the operands. The factorial, evaluated naively, would thus take time proportional to n. One can consult chapter X of Devroye (1986) on this issue; depending upon the situation, one can use a combination of either Stirling's approximation or Binet's approximation with the alternating series acceptance/rejection method. Another property we will require is that $\mathbf{P}\{T > \sqrt{n/4}\} \geq 1/2 > 0$. To see this, observe that with $s = \left\lceil \sqrt{n/4} \right\rceil \leq n$, for $s - 1 \geq 1$,

$$\begin{aligned}
\mathbf{P}\{T > s\} &\geq \left(\frac{n-s+1}{n+s-1}\right)^{s-1} \\
&= \left(1 - \frac{2s-2}{n+s-1}\right)^{s-1} \\
&\geq \left(1 - \frac{2(s-1)^2}{n+s-1}\right) \\
&\geq \frac{1}{2}
\end{aligned}$$

since $s \leq 1 + \sqrt{n/4}$. When $s = 1$ (thus, $n \leq 4$), the inequality we are trying to establish is obviously satisfied, as $T \geq 2$ with probability one.

5.3. Generation of random vectors with a multivariate hypergeometric distribution. The random vector $(D_0, D_1, \ldots, D_m)$ obtained by drawing without replacement k balls from an urn having n_i balls with label i, $0 \le i \le m$, is called a multivariate hypergeometric random vector with parameters k and $(n_0, n_1, \ldots, n_m)$. In a simple hypergeometric situation, we have $m = 1$. For $m = 1$, various generators have uniformly fast expected time per random variate: see for example the generators of Kachitvichyanukul (1982), Kachitvichyanukul and Schmeiser (1985, 1988), Stadlober (1988, 1989) or Devroye (1987). Using these algorithms, we can generate D_0 by drawing from an urn with n_0 balls labeled 0 and $n_1 + \cdots + n_m$ balls labeled "> 0". By repeating this step, we can generate the multivariate hypereometric random variate in expected time $O(m)$ uniformly in all the other parameters.

5.4. Probabilistic analysis. We will show the following.

FACT 4. *The algorithm given above takes expected time $O(\sqrt{n}\log n)$.*

PROOF. The following notation will be used: the number of external nodes at the beginning of the i-th iteration is N_i: thus,

$$1 = N_0 < N_1 < N_2 < \cdots \ .$$

We consider the algorithm without truncation, iterated ad infinitum. Let T_i be the value of the waiting time random variable for the i-th iteration; its parameter is N_{i-1}, and we have

$$N_i = N_{i-1} + T_i \ (i \ge 1) \ .$$

The algorithm halts after iteration k when for the first time $N_k \ge n+1$. The number of iterations is denoted by J_n. Clearly, $J_n > k$ if and only if $N_k < n+1$. Also, by construction, $N_i \ge 2i$ for all i, so that $J_n \le n$. This implies that

$$\mathbf{E}J_n^2 \le \sum_{1 \le i \le n} \mathbf{P}\{J_n^2 \ge i\} \le \inf_k \left(n^2 \mathbf{P}\{N_k < n+1\} + k^2\right) \ .$$

We will see that $\mathbf{E}J_n^2 = O(n)$. Then we continue as follows: if the vector of external nodes at the outset of an iteration is $(n_0, \ldots, n_m)$, then that iteration takes expected time bounded by a universal constant times $m+1$. Clearly, $m \le H_n$, the height of the random tree generated. Hence, the overall expected time is bounded by

$$c\mathbf{E}\{J_n + J_n H_n\}$$

where c is a universal constant. By the Cauchy-Schwarz inequality, we have

$$\mathbf{E}\{J_n H_n\} \le \sqrt{\mathbf{E}J_n^2 \mathbf{E}H_n^2} \ .$$

From Devroye (1986), we recall that $\mathbf{E}H_n^2 = O(\log^2 n)$. Since $\mathbf{E}J_n^2 = O(n)$, Fact 4 is proved.

To show $\mathbf{E}J_n^2 = O(n)$, it clearly suffices to choose $k = \Theta(\sqrt{n})$ and to show that

$$\mathbf{P}\{N_k < n+1\} = O(1/n) \ .$$

Let I_i be the indicator function of the event $T_i > \sqrt{N_{i-1}/4}$. Thus,

$$N_i \ge N_{i-1} + \max\left\{2, \sqrt{N_{i-1}/4}\, I_i\right\} \ , \ i \ge 1 \ .$$

Consider a deterministic sequence d_i determined by $d_0 = 1$,

$$d_i = d_{i-1} + \max\left\{2, \sqrt{d_{i-1}/4}\right\} \ (i \ge 1) \ .$$

By induction, it is easy to see that $N_i \ge d_{B_i}$ where

$$B_i = \sum_{j=1}^{i} I_j \ .$$

Again by induction, we have $d_i \ge (i/8)^2$ for all $i \ge 1$. Thus, $N_i \ge (B_i/8)^2$. The I_j's are dependent. Nevertheless, we have shown that $\mathbf{E}\{I_j | I_1, \ldots, I_{j-1}\} \ge 1/2$, so that by a simple coupling argument, B_i is stochastically dominated by a binomial $(i, 1/2)$ random variable B_i'. Therefore, by Hoeffding's inequality (Hoeffding, 1963),

$$\begin{aligned}
\mathbf{P}\{N_k < n+1\} &\le \mathbf{P}\{B_k < 8\sqrt{n+1}\} \\
&\le \mathbf{P}\left\{B_k - \mathbf{E}B_k < 8\sqrt{n+1} - k/2\right\} \\
&\le \mathbf{P}\left\{B_k - \mathbf{E}B_k < -k/4\right\} \\
&\qquad (\text{if } k \ge 32\sqrt{n+1}) \\
&\le e^{-2(k/4)^2/k} \\
&= e^{-k/8} \ .
\end{aligned}$$

Take $k = \left\lceil 32\sqrt{n+1} \right\rceil$ and conclude that

$$\mathbf{P}\{N_k < n+1\} = O(1/n)$$

as required. In fact, we have shown that

$$\mathbf{E}J_n^2 \leq 16n + 17$$

for all n large enough. $\square$

6. A birth-and-death process method.

6.1. Derivation. In Robson (1979, 1982), simulations were reported that were based upon an ultra-fast algorithm that produces random binary search trees of random size. This method has never been published nor analyzed. Also, the modifications required to produce a random tree of the correct size are discussed in this paper.

Once again, consider a vector $(n_0, n_1, \ldots, n_m)$ representing the number of external nodes at levels $0, 1, \ldots, m$ respectively, and assume that $n_0 + \ldots + n_m = n$ for now. Every external node should be considered as a living element in a birth-and-death process with unit reproduction rate for each element. When an external node gives birth, it produces two new elements (which live at one level below their parent), and it immediately dies, for a net gain of one element. This is nothing but the Yule process (a special case of a pure birth process; see Taylor and Karlin, 1984, p. 215). The n nodes at time t will thus spawn families of offspring at time $t+\Delta$ of i.i.d. sizes $S_1, \ldots, S_n$. The common distribution of the S_i's is that of S, where

$$\mathbf{P}\{S = k\} = \left(1 - e^{-\Delta}\right)^{k-1} e^{-\Delta} \ (k \geq 1) \ .$$

If the state at time t is described by $(n_0, n_1, \ldots, n_m)$, then our purpose is to efficiently generate an updated state at time $t+\Delta$, where Δ is a constant to be selected. The first step is to generate the sizes of the subtrees of external nodes (at time $t+\Delta$) with roots at the elements alive at time t. This leads to the generation of the triangular array of random integers $N(i,j)$, each representing the number of size j subtrees with original root at level i. Thus,

$$\sum_{j=1}^{\infty} N(i,j) = n_i \ , \ 0 \leq i \leq m \ .$$

In fact, $(N(i,1), N(i,2), N(i,3), \ldots)$ is multinomial with parameter n_i and probabilities given by $p_1(\Delta), p_2(\Delta), \ldots$. By repeatedly appealing to a uniformly fast binomial generator, we can generate this vector in expected time bounded by a constant times the expected value of the maximal size subtree $M(n_i)$ for any of the n_i roots. Now, the maximum of n_i i.i.d. geometric random variables described by the probabilities $p_i(\Delta)$, $i \geq 1$, has expected value not exceeding

$$1 + \frac{1 + \log n_i}{\log(1/(1 - e^{-\Delta}))} \ .$$

Since each n_i does not exceed n, we see that, given m, all $N(i,j)$ can be generated in expected time $O(m \log n)$.

The next step in the algorithm consists of generating the correct numbers of external nodes at all levels. This can be done in one sweep from 0 to m. Assume that we are given $N(i,j)$, $j \geq 1$, for fixed i. This leads to $N(i,1)$ external nodes at level i. For fixed $j \geq 2$, we obtain no external nodes at level i. Rather, it is possible to determine how many subtrees rooted at nodes of level $i+1$ this leads to. Indeed, a node at level i with a subtree having j external nodes yields a left child at level $i+1$ which itself has a subtree of (random) size $S \geq 2$, where S is equally likely to take any value between 1 and $j-1$. The size of the subtree rooted at the right child is $j - S$. Of course, we won't have to do this for each node separately. Rather, it is easy to see that we need only generate a multinomial random vector $w_1, w_2, \ldots, w_{j-1}$ with $\sum w_l = N(i,j)$, and equal probabilities. Here w_l represents the number of left child nodes at level $i+1$ having l external nodes in their subtrees. Adding in the sizes of the right subtrees, we see that at level $i+1$, the $N(i,j)$ level i nodes spawn $2N(i,j)$ nodes with $w_l + w_{j-l}$ nodes of size l. A uniformly fast binomial generator can "split" all $N(i,j)$ at level i in this manner in expected time $O(\mathbf{E}M(n_i)^2)$. For fixed Δ, this is $O(\log^2 n)$.

The sizes of the nodes at level $i+1$ can now be updated, and they in turn are split. An iteration thus takes expected time not exceeding $\mathbf{E}H_N$ times $O(\log^2 n)$ where N is the number of external nodes at the end of the iteration.

```
a generator for a random binary search tree
  with ≥ n+1 external nodes

N ← 1, m ← 0, n_0 ← 1
while N < n+1 do
  r ← 1, t_1 ← 0
  processed ← 0, j ← 0
  while processed < N do
    generate a multinomial random vector (u_1, u_2, ..., u_R) with
      parameter n_j and probabilities p_k = e^{-Δ}(1 - e^{-Δ})^{k-1}, k ≥ 1
    N ← N + u_1 + 2u_2 + ··· + Ru_R - n_j
    if R < r then (u_{R+1}, ..., u_r) ← 0, R ← r
    (u_1, u_2, ..., u_r) ← (u_1, u_2, ..., u_r) + (t_1, t_2, ..., t_r)
    n_j ← u_1, processed ← processed +u_1
    j ← j+1
    m ← 2, (t_1, ..., t_R) ← 0
    while m ≤ R do
      generate a multinomial random vector (w_1, ..., w_{m-1})
        with parameter u_m and equal probabilities 1/(m-1)
      (t_1, ..., t_{m-1}) ← (t_1, ..., t_{m-1}) + (w_1, ..., w_{m-1})
      (t_1, ..., t_{m-1}) ← (t_1, ..., t_{m-1}) + (w_{m-1}, ..., w_1)
      m ← m+1
    if ∑_2^R u_m = 0 then r ← 1 else r ← max{i : t_i > 0}
return (n_0, n_1, ..., n_m)
```

6.2. Probabilistic analysis.

FACT 5. *The algorithm shown above halts in expected time $O(\log^4 n)$.*

PROOF. Consider the overall number of external nodes T_i after i iterations, starting with $T_0 = 1$. T_i is the sum of T_{i-1} i.i.d. geometric random variables with probabilities given by $p_1(\Delta), p_2(\Delta), \ldots$. The expected value of one geometric random variable is e^{Δ}. Thus,

$$\mathbf{E}T_i = e^{\Delta}\mathbf{E}T_{i-1} = e^{i\Delta} \ (i \geq 0) \ .$$

The tree thus grows exponentially quickly. In fact, we know much more. By the derivation given above, the distribution of T_i is geometric itself with parameter $e^{-i\Delta}$. It is perhaps a bit counterintuitive that T_i has a monotonically decreasing discrete density, with the value 1 being most likely. Clearly, if we are aiming for a tree of approximate size n, then we can take $\Delta = \log 2$, and perform $\log_2 n$ iterations. Or we can iterate until for the first time $T_i > n+1$. In the latter case, if the number of iterations is J_n, and the height after stopping is H_N, the overall expected time is bounded by $O(\log^2 n)$ times

$$\mathbf{E}\{J_n H_N\} \leq \sqrt{\mathbf{E}\{J_n^2\}\,\mathbf{E}\{H_N^2\}} = O(\log^2 n) \ ,$$

provided that $\mathbf{E}H_N = O(\log n)$ and that $\mathbf{E}\{J_n^2\} = O(\log^2 n)$. We conclude that the expected complexity is $O(\log^4 n)$. But

$$\mathbf{P}\{J_n > k\} \leq \mathbf{P}\{T_k \leq n+1\} = 1 - \left(1 - e^{-k\Delta}\right)^{n+1}$$

so that

$$\begin{aligned}
\mathbf{E}\{J_n^2\} &= \sum_k 2k\mathbf{P}\{J_n > k\} \\
&\leq k^*(k^*+1) + (n+1)\sum_{k>k^*} 2ke^{-k\Delta} \\
&\leq k^*(k^*+1) + (n+1)\int_{k^*}^{\infty} 2xe^{-x\Delta}\,dx \\
&= k^*(k^*+1) + (n+1)2\Delta^{-2}(1+\Delta k^*)e^{-k^*\Delta} \\
&\leq \left(2 + \frac{\log n}{\Delta}\right)^2 + 2\Delta^{-2}(1+\log(n+1)) \\
&\qquad (\text{Take} \ \ k^* = \lceil 2\log(n+1)/\Delta \rceil)
\end{aligned}$$

$$= O(\log^2 n) \ .$$

Finally, we show that $\mathbf{E}H_N = O(\log n)$. Clearly, for $k > 0$,

$$\begin{aligned}
&\mathbf{P}\{H_N > 4e\log(n+1) + 4ek + 1\} \\
&\quad \le \mathbf{P}\{N > k^2(n+1)^2\} + \mathbf{P}\{H_{k^2(n+1)^2} > 4e\log(n+1) + 4ek + 1\}
\end{aligned}$$

where we used the monotonicity of the tree-building process; here H_j denotes the height of the tree in the birth-and-death process at the moment that it has precisely $j+1$ external nodes. The last probability decreases exponentially quickly with n and k: for $k \ge 2\log n$,

$$\mathbf{P}\{H_n \ge k\} \le \frac{2(2\log n)^k}{n\,k!} \le \frac{2(2e\log n/k)^k}{n}$$

(Devroye, 1987, Theorem 5). Therefore,

$$\begin{aligned}
&\mathbf{P}\{H_{k^2(n+1)^2} > 4e\log(n+1) + 4ek + 1\} \\
&\quad \le 2\left(\frac{4e(\log(n+1) + \log k)}{4e\log(n+1) + 4ek}\right)^{(4e\log(n+1)+4ek+2} \times \frac{1}{k^2(n+1)^2} \\
&\quad \le 2k^{-2}(n+1)^{-2} \ .
\end{aligned}$$

Also,

$$\begin{aligned}
\mathbf{P}\{N > k^2(n+1)^2\} &\le \mathbf{P}\{\cup_{k=0}^{n}[T_k < n+1, T_{k+1} \ge k^2(n+1)^2]\} \\
&\le (n+1)\sup_{1\le k\le n}\mathbf{P}\{T_k < n+1, T_{k+1} \ge k^2(n+1)^2\} \\
&= (n+1)\sup_{1\le k\le n}\sum_{j=1}^{n}\mathbf{P}\{T_k = j\}\mathbf{P}\{T_{k+1} \ge k^2(n+1)^2 | T_k = j\} \\
&\le (n+1)\sup_{1\le k\le n}\mathbf{P}\{T_{k+1} \ge k^2(n+1)^2 | T_k = n\} \\
&= (n+1)\mathbf{P}\{T_1 \ge k^2(n+1)^2 | T_0 = n\} \\
&\le k^{-4}(n+1)^{-3}\mathbf{E}\{T_1^2 | T_0 = n\} \\
&= k^{-4}(n+1)^{-3}\left((ne^{\Delta})^2 + n(e^{\Delta} - 1)\right) \\
&\le \frac{2e^{2\Delta}}{k^4(n+1)} \ .
\end{aligned}$$

Sum the upper bounds over $k > 0$, and conclude that $\mathbf{E}H_N \le 4e\log(n+1) + O(1)$. □

6.3 Stopping at exactly n nodes. To obtain a tree of exactly n nodes from the algorithm above, we need to modify it so that before each iteration copies of N, m and $(n_0, n_1, \ldots, n_m)$ are made and, if the iteration would increase the number of nodes beyond n, the iteration is aborted by returning to the copied values. To preserve the expected time $O(\log^4 n)$, it is also necessary to reduce the value of Δ when it is likely that taking a time step of Δ will give an aborted iteration. A simple approach is to take Δ as $\log((n+N)/2N)$ whenever this is less than $\log 2$ (that is when $N > n/3$), so that the expected number of new nodes created is half of the number required to reach n.

FACT 6. *The algorithm modified in this way halts in expected time $O(\log^4 n)$.*

PROOF. To prove the bound $O(\log^4 n)$ on the expected time, it is only necessary to show that the bound $\mathbf{E}\{J_n^2\} = O(\log^2 n)$ still holds. First we count only non-aborted iterations. By the reasoning above, the bound holds for the number of iterations to reach $n/3$ nodes. After $n/3$ nodes, each iteration has value of Δ such that the expected number of new nodes is half that required to reach n and it is easy to see that there is a constant $c > 0$ such that each iteration has probability at least c of adding at least half of that expected number (since the rate of creation of new nodes is at least that of a Poisson process creating N nodes per unit time). But the maximum number of iterations after $n/3$ nodes which add at least a quarter of the $n - N$ nodes still required is $O(\log n)$ so that the expectation $\mathbf{E}\{J_n^2\} = O(\log^2 n/c^2) = O(\log^2 n)$ as required.

To handle aborted iterations, we use a similar argument: Each iteration has a probability at least $1/2$ of not being aborted; hence including aborted iterations can increase $\mathbf{E}\{J_n^2\}$ by at most a factor of 4. $\square$

In practice it would be sensible to halt this process when the number of nodes required to be added was small (say $< \sqrt{n}\log^2 n$) and finish the simulation with the method of section 5.

Acknowledgment.

We thank the referees.

References.

A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, Reading, Mass., 1983.

J. H. Ahrens and U. Dieter, "Sampling from binomial and Poisson distributions: a method with bounded computation times," *Computing*, vol. 25, pp. 193–208, 1980.

H. C. Chen and Y. Asau, "On generating random variates from an empirical distribution," *AIIE Transactions*, vol. 6, pp. 163–166, 1974.

T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Boston, 1990.

L. Devroye, "Generating the maximum of independent identically distributed random variables," *Computers and Mathematics with Applications*, vol. 6, pp. 305–315, 1980.

L. Devroye, "A note on the height of binary search trees," *Journal of the ACM*, vol. 33, pp. 489–498, 1986.

L. Devroye, *Non-Uniform Random Variate Generation*, Springer-Verlag, New York, 1986.

L. Devroye, "A simple generator for discrete log-concave distributions," *Computing*, vol. 39, pp. 87–91, 1987.

L. Devroye, "Branching processes in the analysis of the heights of trees," *Acta Informatica*, vol. 24, pp. 277–298, 1987.

L. Devroye, "A simple generator for discrete log-concave distributions," *Computing*, vol. 39, pp. 87–91, 1987.

L. Devroye, "Generating sums in constant average time," in: *Proceedings of the 1988 Winter Simulation Conference*, ed. M. A. Abrams, P. L. Haigh and J. C. Comfort, pp. 425–431, IEEE, San Diego, CA., 1988.

R. Durrett, *Probability: Theory and Examples*, Wadsworth and Brooks, Pacific Grove, CA, 1991.

G. S. Fishman and L. R. Moore, "Sampling from a discrete distribution while preserving monotonicity," *The American Statistician*, vol. 38, pp. 219–223, 1984.

W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, pp. 13–30, 1963.

V. Kachitvichyanukul, "Computer Generation of Poisson, Binomial, and Hypergeometric Random Variates," Ph.D. Dissertation, School of Industrial Engineering, Purdue University, 1982.

V. Kachitvichyanukul and B. W. Schmeiser, "Computer generation of hypergeometric random variates," *Journal of Statistical Computation and Simulation*, vol. 22, pp. 127–145, 1985.

V. Kachitvichyanukul and B. W. Schmeiser, "Binomial random variate generation," *Communications of the ACM*, vol. 31, pp. 216–222, 1988.

V. Kachitvichyanukul and B. W. Schmeiser, "Algorithm 668 H2PEC: sampling from the hypergeometric distribution," *ACM Transactions on Mathematical Software*, vol. 14, pp. 397–398, 1988.

D. E. Knuth, *The Art of Computer Programming, Vol. 3 : Sorting and Searching*, Addison-Wesley, Reading, Mass., 1973.

H. M. Mahmoud, *Evolution of Random Search Trees*, John Wiley, New York, 1992.

H. Mahmoud and B. Pittel, "On the most probable shape of a search tree grown from a random permutation," *SIAM Journal on Algebraic and Discrete Methods*, vol. 5, pp. 69–81, 1984.

V. V. Petrov, *Sums of Independent Random Variables*, Springer-Verlag, Berlin, 1975.

B. Pittel, "On growing random binary trees," *Journal of Mathematical Analysis and Applications*, vol. 103, pp. 461–480, 1984.

B. B. Pokhodzei, "Beta- and gamma-methods of modelling binomial and Poisson distributions," *USSR Computational Mathematics and Mathematical Physics*, vol. 24(1), pp. 114–118, 1984.

B. B. Pokhodzei, "A note on beta-methods for simulating binomial distributions," *USSR Computational Mathematics and Mathematical Physics*, vol. 28, pp. 207–208, 1988.

J. M. Robson, "The height of binary search trees," *The Australian Computer Journal*, vol. 11, pp. 151–153, 1979.

J. M. Robson, "The asymptotic behaviour of the height of binary search trees," *Australian Computer Science Communications*, p. 88, 1982.

B. W. Schmeiser, "Generation of the maximum (minimum) value in digital computer simulation," *Journal of Statistical Computation and Simulation*, vol. 8, pp. 103–115, 1978.

E. Stadlober, "Sampling from Poisson, binomial and hypergeometric distributions: ratio of uniforms as a simple fast alternative," Habilitationsschrift, Institute of Statistics, Technical University of Graz, Austria, 1988.

E. Stadlober, "Binomial random variate generation: a method based on ratio of uniforms," *American Journal of Mathematical and Management Sciences*, vol. 9, pp. 1–20, 1989.

E. Stadlober, "Ratio of uniforms as a convenient method for sampling from classical discrete distributions," in: *Proceedings of the 1989 Winter Simulation Conference*, ed. E. A. MacNair, K. J. Musselman and P. Heidelberger, pp. 484–489, IEEE, Washington, D.C., 1989.

H. M. Taylor and S. Karlin, *An Introduction to Stochastic Modeling*, Academic Press, Orlando, FL, 1984.