Algorithm Design: Expected Time

Ștefan Ciobâcă, Dorel Lucanu

Faculty of Computer Science
Alexandru Ioan Cuza University, Iași, Romania
dlucanu@info.uaic.ro

PA 2019/2020

1. Recap
2. Expected Time for Deterministic Algorithms
3. Case Studies
   - First Occurrence in a List
   - Quicksort
   - Nuts and Bolts
   - Treaps

Plan

Randomized Algorithms

- a particular case of nondeterministic algorithms when the choices are made according to a probability distribution
- `random(n)`
  - returns an $x \in \{0, 1, \ldots, n-1\}$ uniformly chosen (with the probability $\dfrac{1}{n}$)

Random Variables

- A random variable is a function X defined over a set of possible outcomes Ω of a random phenomenon.
- Example (only discrete variables): $D2$ (two dice):
  - random phenomenon: rolling two dice
  - $D2$ returns the sum of the numbers on the two dice
- probability distribution: $X(\Omega) = \{x_0, x_1, x_2, \ldots\}$, $Prob(X = x_i) = p_i$[1]
- Expected Value of X: $E(X) = \sum_i x_i \cdot p_i$
  Properties:
  $E(X + Y) = E(X) + E(Y)$
  $E(X \cdot Y) = E(X) \cdot E(Y)$
  (X and Y independent)

[1] The exact terminology for $\mathrm{Prob}(X = x_i)$ is "probability mass function". Here we use the more general term of probability distribution (the way the total probability of 1 is distributed over all various possible outcomes).

Monte Carlo Algorithms

– may produce incorrect results with some small probability, but whose execution time is deterministic
– if runned multiple times with independent random choices each time, the failure probability can be made arbitrarily small, at the cost of the running time .

Example: primality test

Las Vegas Algorithms

- never produce incorrect results, but whose execution time may vary from one run to another

– random choices made within the algorithm are used to establish an expected running time for the algorithm that is, essentially, independent of the input

Example: k-median

k-median: Las Vegas Algorithms

```
randPartition(out a, p, q) {
  if (p < q) {
    i = p + random(q - p);
    swap(a, i, q);
    return partition(a, p, q);
  }
}

randSelectRec(out a, p, q, k)
{
  j = randPartition(a, p, q);
  if (j == k) return a[j];
  if (j < k) return randSelectRec(a, j+1, q, k);
  return randSelectRec(a, p, j-1, k);
}

randSelect(out a, k)
{
  return randSelectRec(a, 0, a.size()-1, k);
}
```

randSelect: analysis 1/2

exp-time(n, k) - the expected time to find the k-median in an array of length n

exp-time$(n) = \max_k$ *exp-time*(n, k)

Since we are interested in the worst case analysis, we assume that the recursive call chooses always the longest subarray.

Recall that $E(CB) < \frac{3}{4}$

It follows that the expected length of the longest subarray is at most $\frac{3}{4}n$.

randSelect: analysis 2/2

Lemă

The expected length of the array after i call is at most $\left(\frac{3}{4}\right)^i n$.

Theorem

$$exp\text{-}time(n) = O(n)$$

Plan

Motivation

- A solves P, $x \in P$
- worst-case execution time:
$$T_A(n) = \sup\{time(A, p) \mid p \in P \wedge size(p) = n\}$$
- Sometimes the number of instances p with $size(p) = n$ and for which $time(A, p) = T_A(n)$ or $time(A, p)$ is close to $T_A(n)$ is very small.
- For such cases the expected time is more appropriate.

Definition 1/2

- deterministic algorithm : each input x uniquely determines an execution path
- but let us think that the inputs are randomly coming
- $time_A(\_)$ as random variable:
  - random event = the execution of the (deterministic) algorithm for a random input x,
  - the associated value = the execution time
- $time_A^n$ is $time_A$ restricted to instances x with $size(x) = n$

Definition 2/2

- probability distribution: $Prob(time_A^n = t)$
- expected time = expected value of the random variable

$$exp\text{-}time(n) = E(time_A^n)$$

- A particular case: $\{time_A(x) \mid size(x) = n\} = \{t_0, t_1, \ldots\}$, $Pr(time_A^n = t_i) = p_i$

$$exp\text{-}time(n) = \sum_i t_i \cdot p_i$$

Plan

Plan

FIRST OCCURRENCE Problem

Problem FIRST OCCURRENCE

Input: $n, a = (a_0, \ldots, a_{n-1}), z$, all integers.

Output: $poz = \begin{cases} \min\{i \mid a_i = z\} & \text{if } \{i \mid a_i = z\} \neq \emptyset, \\ -1 & \text{otherwise.} \end{cases}$

Algorithm for FIRST OCCURRENCE

```
i =  0;
while (a[i] != z) && (i < n-1)
  i = i+1;
if (a[i] == z)  poz = i;
else poz = -1;
```

The number of comparisons for the worst-case is n.

Expected Time for FOAlg 1/2

- instance size: $n = \texttt{a.size}()$
- measured operations: comparisons evolving elements in a
- $\{time_{\mathrm{FOAlg}}(x) \mid size(x) = n\} = \{i \mid 2 \leq i \leq n\}$
- it is difficult to compute the probability of an instance

Assumptions:

- probabilitaty that $z \in \{a_0, \ldots, a_{n-1}\}$ is q și
- probabilitaty that the first occurrence of z is on position i is $\frac{q}{n}$

Expected Time for FOAlg 2/2

$Prob(z \notin \{a_0, \ldots, a_{n-1}\}) = 1 - q$

$Prob(time^n_{\mathrm{FOAlg}}(p) = i) = \dfrac{q}{n} = p_i,\ 2 \leq i < n$

$Prob(time^n_{\mathrm{FOAlg}}(p) = n) = \dfrac{q}{n} + (1 - q) = p_n$

Expected time is:

$$\begin{aligned} \textit{exp-time}(n) &= \sum_{i=2}^{n} p_i x_i \\ &= \sum_{i=2}^{n-1} \frac{q}{n} \cdot i + (\frac{q}{n} + (1-q)) \cdot n \\ &= n - \frac{q}{n} - \frac{n-1}{2} \cdot q \end{aligned}$$

For $q = \dfrac{1}{2}$ we have $\textit{exp-time}(n) = \dfrac{3n+1}{4} - \dfrac{1}{2n}$.

For $q = 1$ we have $\textit{exp-time}(n) = \dfrac{n+1}{2} - \dfrac{1}{n}$.

Plan

1. Recap
2. Expected Time for Deterministic Algorithms
3. Case Studies
   - First Occurrence in a List
   - Quicksort
   - Nuts and Bolts
   - Treaps

Quicksort: description

Design paradigm: divide-et-impera.

Algorithm Quicksort

Input: $S = (a_0, \ldots, a_{n-1})$

Output: a sequence including all elements a_i in increasing order

1. choose $x = a_k \in S$
2. compute
   $S_< = (a_i \mid a_i < x)$
   $S_= = (a_i \mid a_i = x)$
   $S_> = (a_i \mid a_i > x)$
3. sort recursively $S_<$ și $S_>$ producing $Seq_<$ și $Seq_>$, respectively
4. return the sequence $Seq_<$, $S_=$, $Seq_>$

Quicksort: partitioning

Use Lomuto algorithm from *k*-median:

```
partition(out a, p, q)
{
  pivot = a[q];
  i = p - 1;
  for (j = p; j < q; ++j)
    if (a[j] <= pivot) {
        i = i + 1;
        swap(a, i , j);
    }
  swap(a, i+1, q);
  return i + 1;
}
```

Quicksort: algorithm

```
@input:   a = (a[p], ..., a[q])
@output:   a sorted in increasing order
qsortRec(out a, p, q)
{
  if (p < q) {
    k = partition(a, p, q);
    qsortRec(a, p, k-1);
    qsortRec(a, k+1, q);
  }
}

qsort(out a)
{
  qsortRec(a, 0, n-1);
}
```

Quicksort: worst-case time analysis

- instance size: $\mathtt{n} = \mathtt{a.size}()$
- measured operations: compararisons between elements of a
- worst case: the array is sorted
- the number of comparisons for the worst case: $(n-1)+(n-2)+\cdots+1 = O(n^2)$

Quicksort: expected time

Expected time for `qsort` is equal to the expected time of the "randomized quicksort".

"Randomized Quicksort", intuitively

- canonic example for Las Vegas algorithms

Algoritmul RQS

Input: $S = \{a_0, \ldots, a_{n-1}\}$

Output: elements a_i sorted in increasing order

1. if $n = 1$ returns a_0, otherwise randomly choose $x = a_k \in S$
2. compute
   $S_< = (a_i \mid a_i < a_k)$
   $S_= = (a_i \mid a_i = a_k)$
   $S_> = (a_i \mid a_i > a_k)$
3. recursively sort $S_<$ și $S_>$ producing $Seq_<$ și $Seq_>$, resp.
4. returns the sequence $Seq_<$, $Seq_=$, $Seq_>$

"Randomized Quicksort", algoritmically

```
randPartition(out a, p, q) {
  if (p < q) {
    i = p + random(q - p);
    swap(a, i, q);
    return partition(a, p, q);
  }
}

randQsortRec(out a, p, q) {
  if (p < q) {
    k = randPartition(a, p, q);
    randQsortRec(a, p, k-1);
    randQsortRec(a, k+1, q);
  }
}

RQS(out a) { randQsortRec(a, 0, n-1); }
```

Time analysis of RQS: version 1 (1/4)

Let *rank* be the function s.t. $a_{rank(0)} \leq \ldots \leq a_{rank(n-1)}$.

Define $X_{ij} = \begin{cases} 1 & a_{rank(i)} \text{ and } a_{rank(j)} \text{ are compared} \\ 0 & otherwise \end{cases}$

X_{ij} the number of comparisons between $a_{rank(i)}$ and $a_{rank(j)}$

X_{ij} is a random variable of type indicator

$C(n)$ - random variable that returns the number of comparisons

$C(n) = \sum_{j>i} X_{ij}$

Expected number of comparisons:

$E(C(n)) = E(\sum_{i=0}^{n-1} \sum_{j>i} X_{ij}) = \sum_{i=0}^{n-1} \sum_{j>i} E(X_{ij})$

Time analysis of RQS: version 1 (2/4)

p_{ij} probability that $a_{rank(i)}$ and $a_{rank(j)}$ to be compared

$E(X_{ij}) = p_{ij} \times 1 + (1 - p_{ij}) \times 0 = p_{ij}$

Assume $rank(i) < rank(j)$; $a_{rank(i)}$ and $a_{rank(j)}$ are compared iff the first pivot from

$$a_{rank(i)}, \ldots, a_{rank(j)}$$

is $a_{rank(i)}$ or $a_{rank(j)}$. Otherwise the pivot split the array.

It follows $p_{ij} = \dfrac{2}{j - i + 1}$.

Time analysis of RQS: version 1 (3/4)

$$
\begin{aligned}
\sum_{i=0}^{n-2}\sum_{j>i} p_{ij} &= \sum_{i=0}^{n-2}\sum_{j>i} \frac{2}{j-i+1} \\
&\leq \sum_{i=0}^{n-2}\sum_{k=1}^{n-i-1} \frac{2}{k} \\
&\leq 2\sum_{i=0}^{n-2}\sum_{k=1}^{n-i-1} \frac{1}{k}
\end{aligned}
$$

Time analysis of RQS: version 1 (4/4)

We obtain:

$$\sum_{k=1}^{n-i-1} \frac{1}{k} = H_{n-i-1} = \Theta(\log(n-i-1)),$$

$$\sum_{i=0}^{n-2} \Theta(\log(n-i-1)) = \Theta(\log(n-1)!) = \Theta(n \log n).$$

Theorem

The expected number of comparisons given by RQS is at most $2nH_n = O(n \log n)$.

Exercises

Compute the expected expected time of insertSort and of bubbleSort, respectively. Consider two cases:

- measured operations are swaps
- measured operations are the element moves

When there is a significant difference?

Time analysis of RQS: version 2 (recursiv) 1/2

- sequence size: $q + 1 - p = n$
- probability that the pivot x to be the k-th element: $\frac{1}{n}$
- subprobleme sizes: $k - p = i - 1$ and $q - k = n - i$
- $C(n)$ - the number of comparisons
- let Y_i be the ransom variable the expected number of comparisons given by the recursive calls if the partitioning algorithms returns k, 0 otherwise

$$Y_i = \begin{cases} E(C(i-1)) + E(C(n-i)) & , i = k \\ 0 & i \neq k \end{cases}$$

- $E(Y_i) = \frac{1}{n}(E(C(i-1)) + E(C_(n-i)))$
- the number of comparisons:

$$C(n) = (n-1) + \sum_{i=1}^{n} Y_i$$

Time analysis of RQS: version 2 (recursively) 2/2

- expected number of comparisons:

$$
\begin{aligned}
\textit{exp-time}(n) &= E(C(n)) = E((n-1) + \sum_{i=1}^{n} Y_i) \\
&= \begin{cases} (n-1) + \frac{1}{n}\sum_{i=1}^{n}(E(C(i-1)) + E(C(n-i)) & \text{, if } n \geq 1 \\ 1 & \text{, if } n = 0 \end{cases}
\end{aligned}
$$

Theorem

The expected time for `RQS` is $O(n \log_2 n)$.

Plan

Problem domain

- n bolts and n nuts
- no way to compare two bolts or two nuts, resp.
- each nut matches exactly one bolt
- when a bolt is compared with a nut, one can decide if they matches, or is larger, or is smaller
- the goal is to find the matching nit for each bolt (or vice versa)

Obs. Recently, in1995, was designed an algoritm that solves the problem in $O(n \log n) time$:

Janos Komlos, Yuan Ma, and Endre Szemeredi. Sorting nuts and bolts in $O(nlogn)$ time, SIAM J. Discrete Math 11(3):347-372, 1998. Technical Report MPI-I-95-1-025, Max-Planck-Institut für Informatik, September 1995.

Simplified version

Find a nut for a given bolt.

Let cmp(N, B) be a function that returns:
-1 if the nut N is less than the bolt B,
0 if the nut N is equal to the bolt B,
1 if the nut N is greater than the bolt B,.

```
NutsAndBolt (NL, B)
{
  n = NL.size();
  for (i=0; i < n - 1; ++i)
    if (cmp(NL[i], B) == 0) return i;
  return n - 1;
}
```

Expected number of comparisons: version 1

assume that the order of nuts in NL (that is the same with the comparing order) is a uniform random variable.

It follows that the probability that the needed nut to be on position i is $\dfrac{1}{n}$ (why?).

Let $C(n)$ be the random variable that returns the number of comparisons. Assume that NL is nonempty and that always there is a nut that matches B.

Possible values for $C(n)$: $1, 2, \ldots, n-1$.

Probability that $C(n) = i$, $1 \leq i < n-1$: $\dfrac{1}{n}$

Probability that $C(n) = n-1$: $\dfrac{2}{n}$

Expected number of comparisons

$$
\begin{aligned}
E(C(n)) &= \sum_{i=1}^{n-2} i \cdot \frac{1}{n} + (n-1) \cdot \frac{2}{n} \\
&= \sum_{i=1}^{n-1} \frac{i}{n} + \frac{n-1}{n} \\
&= \frac{n(n-1)}{2n} + \frac{n-1}{n} \\
&= \frac{n+1}{2} - \frac{1}{n}
\end{aligned}
$$

Expected number of comparisons: version 2 (recursiv) 1/2

If $n > 1$, the first element is always compared(i.e. its probability is 1). Recalll that the probability that the algorithm stops after the first comparison is $\frac{1}{n}$. It follows that the rest of the list is processed with the probability $1 - \frac{1}{n} = \frac{n-1}{n}$.

Y - the random variable that returns $E(C(n-1))$ for the recursive call, and 0 otherwise.

We have:

$C(1) = 0,\ C(n) = 1 + Y$ pentru $n > 1$

which implies

$E(C(n)) = 1 + E(Y) = 1 + \frac{n-1}{n} E(C(n-1))$ pentru $n > 1$.

Expected number of comparisons: version 2 (recursiv) 2/2

Let $c(n) = nE(C(n))$.

It follows: $c(1) = 0$, $c(n) = n + c(n-1)$ for $n > 1$.

We obtain: $c(n) = \sum_{i=2}^{n} i = \dfrac{n(n+1)}{2} - 1$.

which implies $E(C(n)) = \dfrac{c(n)}{n} = \dfrac{n+1}{2} - \dfrac{1}{n}$.

General case, solution 1 (naive)

```
NutsAndBolts (NL, BL) {
  M = emptyList;  n = NL.size();
  forall B in BL {
    i = 0;
    while (i < n)
      if (cmp(NL[i], B) == 0) {
        M.pushBack([B, NL[i]]);
        i = n;
      }
  }
  return M;
}
```

The number of comparisons for the worst case $=$
the expected number of comparisons $=$
$O(n^2)$

The above solution does not use the values -1 and 1 returned by cmp.

General case, solution 2: idea

It is of recursive nature.

Curent step:

1. choose a bolt as a pivot and compare it with all the current nuts;
2. take the matched nut and compare it with the remained bolts;
3. after $2(n-1)$ comparisons, split the lists of bolt in two: the list of bolts/nuts less than the matched nut/bolr, and the list of larger ones ;

Recursive calls:

- one call for the list with the bolts/nuts smaller
- one call for the list with the bolts/nuts bigger

Exercise. Write in Alk the above algorithm.

Worst-case analysis

$$
\begin{aligned}
C(0) &= 0 \\
C(n) &= 2n - 1 + \max_{k=1,\ldots,n}(C(k-1) + C(n-k)) \\
&= 2n - 1 + C(n-1)
\end{aligned}
$$

After solving the recurence, we get

$$C(n) = \sum_{i=1}^{n}(2n-1) = O(n^2)$$

Expected number of comparisons 1/3

Assume that the pivot bolt is uniform randomly chosen.
Let B_i be the i-th bolt in the sorted list. Similar N_j.
X_{ij} the random variable (of type indicator), that returns 1 if B_i and N_j are compared, 0 otherwise.
Assume $i < j$. B_i and N_j are compared iff the first pivot from $B_i, \ldots, B_j$ is B_i or B_j (why?).
It follows $E(X_{ij}) = Prob(X_{ij} = 1) = \dfrac{2}{j+1-i}$, $i < j$.
The symetric relation is obtained in the same way: $E(X_{ij}) = \dfrac{2}{i+1-j}$, $i > j$.
If $i = j$, $E(X_{ij}) = 1$ (although these comparisons can be avoided).

Expected number of comparisons 2/3

Since $C(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij}$, it follows

$$
\begin{aligned}
E(C(n)) &= E(\sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij}) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} E(X_{ij}) \\
&= \sum_{i=1}^{n} E(X_{ii}) + 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} E(X_{ij}) \\
&= n + 4 \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{j+1-i} \\
&= n + 4(nH_n - 2n + H_n)
\end{aligned}
$$

Expected number of comparisons 3/3

We used:

$$\begin{aligned}
\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{1}{j+1-i} &= \sum_{i=1}^{n}\sum_{k=2}^{n+1-i}\frac{1}{k}\\
&= \sum_{k=2}^{n}\sum_{i=1}^{n+1-k}\frac{1}{k}\\
&= \sum_{k=2}^{n}\frac{n+1-k}{k}\\
&= (n+1)\sum_{k=2}^{n}\frac{1}{k}-\sum_{k=2}^{n}1\\
&= (n+1)(H_n-1)-(n-1)
\end{aligned}$$

Since $H_n = \sum_{k=1}^{n}\frac{1}{k} = \Theta(\log n)$, it follows

$$E(C(n)) = \Theta(n\log n)$$

Expected number, recursively 1/3

Assume that the pivot is uniform randomly chosen and each bolt B is equiprobable the k-th smallest in the list, i.e. B occurs on position k in the sorted list with the probability $\frac{1}{n}$.

Let Y_k be the random variable that returns the expected number of comparisons for the recursive calls if the pivot is the k-th element, 0 otherwise:

$$Y_k = \begin{cases} E(C(k-1)) + E(C(n-k)) & \text{if the pivot is } B_k \\ 0 & \text{otherwise} \end{cases}$$

We have

$$C(n) = 2n - 1 + \sum_{k=1}^{n} Y_k$$

$$E(Y_k) = \frac{1}{n}(E(C(k-1)) + E(C(n-k))).$$

Expected number, recursively 2/3

$$
\begin{aligned}
E(C(n)) &= E(2n-1+\sum_{k=1}^{n} Y_k)\\
&= 2n-1+\sum_{k=1}^{n} E(Y_k)\\
&= 2n-1+\frac{1}{n}\sum_{k=1}^{n}(E(C(k-1))+E(C(n-k))\\
&= 2n-1+\frac{2}{n}\sum_{k=0}^{n-1} E(C(k))
\end{aligned}
$$

We obtain

$$nC(n) = n(2n-1)+2\sum_{k=0}^{n-1} E(C(k))$$

which implies

$$nC(n) = (n+1)C(n-1)+4n-3$$

Expected number, recursively 3/3

$$
\begin{aligned}
\frac{C(n)}{n+1} &= \frac{C(n-1)}{n} + \frac{4}{n+1} - \frac{3}{n(n+1)} \\
\frac{C(n-1)}{n} &= \frac{C(n-2)}{n-1} + \frac{4}{n} - \frac{3}{(n-1)n} \\
&\dots \\
\frac{C(2)}{3} &= \frac{C(1)}{2} + \frac{4}{2} - \frac{3}{1(2)}
\end{aligned}
$$

We obtain

$$\frac{C(n)}{n+1} = \frac{C(1)}{2} + \Theta(\log n) - \Theta(1)$$

which implies

$$C(n) = \Theta(n \log n)$$

Reduction to sorting

- two bolts are compared using the nuts (how ?)
- similarly, two nuts are compare using the bolts (how ?)
- define comparing functions (and compute their execution time)
- call a generic sorting algorithm, having the two comparing functions as arguments
- return $(BL[i], NL[i])$ from the sorted lists

Reducing sorting to NutsAndBolts

- duplicate the list to be sorted
- a copy plays the role of nuts, the other one that of nuts
- call NutsAndBolts
- merge the lists returned by NutsAndBolts a sorted list of the initial elements

Plan

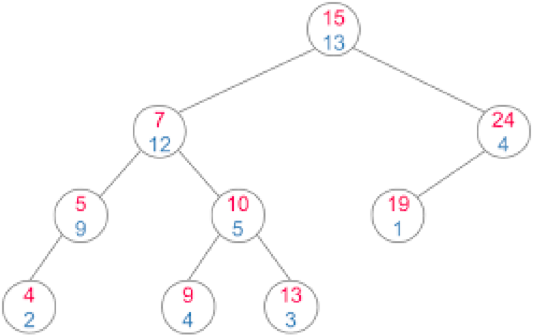
Definition

- combine binary search tree with max-heap (or min-heap)
- binary search structure is given by a `key` associated to each data
- max-heap structure is given by a priority `pri` associated to each data
- searching is performed using the binary search structure

Main idea of use: assign higher priorities to frequently searched elements

Raimund Seidel and Cecilia R. Aragon. Randomized search trees. Algorithmica, 16(4/5):464-497, 1996. Available at
https://faculty.washington.edu/aragon/pubs/rst96.pdf .

Example



Insertion

- insert the new element into a leaf node using the binary search structure
- restore the max-heap property using priorities

Deletion

- its operation way is dual to that of inserting
- the element to be deleted is moved into a leaf using both structures, by rotations
- delete the leaf

Expected number of comparisons 1/3

- assume that the keys are $1, 2, \ldots, n$
- indicator random variable: A_{ij} returns 1 iff i is an ancestor of j
- i will have maximum priority in $[\min(i,j)..\max(i,j)]$
- it follows that $E(A_{ij}) = \dfrac{1}{|i-j|+1}$
- the number of comparison for a node j (searched/inserted/deleted) is proportional to the depth of j
- define $depth(j) = \sum_i A_{ij} - 1$ (the root has the depth 0)
- $depth(j) =$ the number of comparisons to find the key j

Expected number of comparisons 2/3

$$
\begin{aligned}
E(depth(j)) &= E(\sum_i A_{ij} - 1) \\
&= \sum_i E(A_{ij}) - 1 \\
&= \sum_i \left(\frac{1}{|i-j|+1} \right) - 1 \\
&= \sum_{i=1}^{j} \left(\frac{1}{|i-j|+1} \right) + \sum_{i=j+1}^{n} \left(\frac{1}{|i-j|+1} \right) - 1 \\
&= \sum_{k=1}^{j} \frac{1}{k} + \sum_{k=2}^{n-j+1} \frac{1}{k} - 1 \\
&= H_j + H_{n-j+1} - 2
\end{aligned}
$$

Expected number of comparisons 3/3

$\max_j depth(j)$ is reached for $j = \dfrac{n+1}{2}$, which is equal to

$$2H_{\frac{n+1}{2}} - 2 = 2\ln n + O(1)$$

This is an upper bound for the number of comparisons.

A randomized tree is obtained by uniform random generating of priorities (real numbers in $[0,1]$) at insertion

If the nodes are inserted in the decreasing order of priorities, then the result is a treap.

The above analysis holds also for random binary search trees (random insertions, no priorities)..

Quicksort again

Consider the following sorting algorithm:

`T` = empty binary search tree
randomly insert the keys in `T`
returns the inorder list of `T`

Show that `RQS` can be transformed into a randomized algorithm for building binary search trees.

Is the expected time preserved?

The End