Randomized Algorithms I

Sorting & Searching

Pat Morin

Carleton University

University of Sydney

National ICT Australia

Randomized Algorithms

What?: An algorithm that makes random choices during its execution

Why?: Randomness can make many tasks easier (or even possible)

How?: The tools of probability theory are used to analyze properties of randomized algorithms

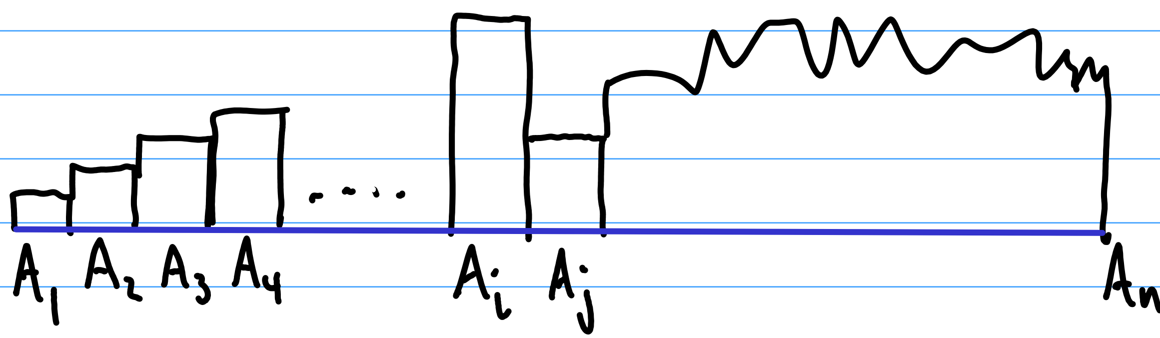
When?: Since the 1960's (see Quicksort, 1961)

Where?: Randomization has applications in almost all branches of algorithms

Who?: Everybody's doin' it

Insertion Sort

```
Sort(A_1, ..., A_n)
for i ← 1 to n-1 do
    j ← i+1
    while j>1 and A_{j-1} > A_j
        swap A_{j-1} ↔ A_j
        j ← j-1
```



Analysis of Insertion Sort

```
Sort(A_1, ..., A_n)
    for i ← 1 to n-1 do
        j ← i+1
        while j > 1 and A_{j-1} > A_j
            swap A_{j-1} ↔ A_j
            j ← j-1
```

How often does each line execute?

Average Case Analysis of InsertionSort

```
Sort(A_1,...,A_n)
1:   for i ← 1 to n-1 do
2:       j ← i+1
3:       while j>1 and A_{j-1} > A_j
4:           swap A_{j-1} ↔ A_j
5:           j ← j-1
```

What if $A_1,\dots,A_n$ is a random permutation of $1,\dots,n$?

Let $I_{x,y} = \begin{cases} 1 & \text{if } x \text{ and } y \text{ are swapped in line 4} \\ 0 & \text{otherwise} \end{cases}$

Then we are interested in $\displaystyle\sum_{x=1}^{n-1} \sum_{y=x+1}^{n} I_{x,y}$ (Inversions of $A_1 \dots A_n$)

$= \#$ times line 4 executes

$$E\left[\sum_{x=1}^{n-1}\sum_{y=x+1}^{n} I_{x,y}\right] = \sum_{x=1}^{n-1}\sum_{y=x+1}^{n} E[I_{x,y}]$$

$$= \sum_{x=1}^{n-1}\sum_{y=x+1}^{n} 1\cdot \Pr\{y \text{ appears before } x\}$$

$$= \sum_{x=1}^{n-1}\sum_{y=x+1}^{n} \frac{1}{2}$$

$$= \frac{1}{2}\binom{n}{2} = \frac{n^2}{4} - \frac{n}{4}$$

∴ InsertionSort does half as many swaps on a random input

Randomized Insertion Sort

```
Sort(A_1, ..., A_n)
    Permute(A_1, ..., A_n)                      } O(n)
    for i ← 1 to n-1 do                         } n-1
        j ← i+1                                 } n-1
        while j > 1 and A_{j-1} > A_j           }
            swap A_{j-1} ↔ A_j                  } 1/2 (n choose 2) = 1/4 n^2 - n/4
            j ← j-1                             }
```

$O(n)$

$n-1$

$n-1$

$\frac{1}{2}\binom{n}{2} = \frac{1}{4}n^2 - \frac{n}{4}$

```
Permute(A_1, ..., A_n)
    for i = 1 to n-1 do
        j ← random(i, ..., n)
        swap A_i ↔ A_j
```

$O(n)$

$\therefore$ The additional work is $O(n)$ but eliminates $\frac{n^2}{2}$ comparisons.

Finding the Minimum

```
FindMin(A_1,..., A_n)
1: m ← ∞
2: for i=1 to n do
3:     if A_i < m then
4:         m ← A_i
5: return m
```

If $A_1, \ldots, A_n$ is a random permutation of $1, \ldots, n$
How many times does line 4 execute?

$$I_i = \begin{cases} 1 & \text{if } A_i = \min\{A_1, \ldots, A_i\} \\ 0 & \text{otherwise} \end{cases} \quad (A_i \text{ is a "record"})$$

$$E[I_i] = 1/i$$

Finding the Minimum

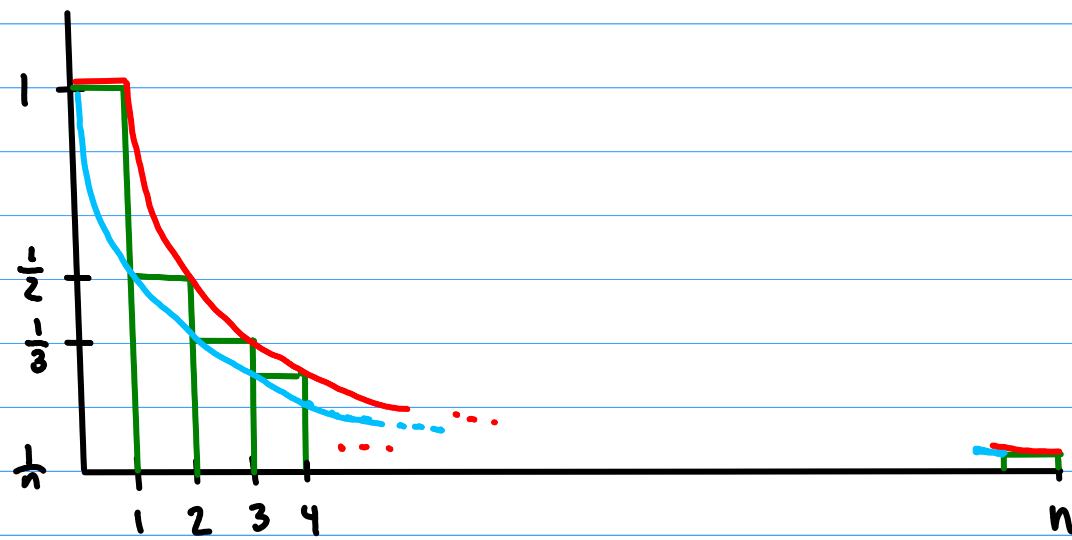
```
FindMin(A_1, ..., A_n)
m ← ∞
for i = 1 to n do
        if A_i < m then
                m ← A_i
return m
```

$$E\left[\sum_{i=1}^{n} I_i\right] = \sum_{i=1}^{n} E[I_i] = \sum_{i=1}^{n} 1/i \overset{\text{def}}{=} H_n$$

"nth harmonic number"

Harmonic Numbers

$$\int_1^n (1/x)\,dx \le H_n \le 1 + \int_1^n (1/x)\,dx$$



$$\ln n \le H_n \le \ln n + 1$$

n^{th} harmonic number is almost equal to $\ln n$

Basic Binary Search Trees

A sequence of keys $A_1, \ldots, A_n$ is given

Each key is inserted in a top-down fashion

Example: 5, 2, 1, 6, 3, 8, 7, 4, 9, 10

Problem: Some sequences lead to bad trees.

Random Binary Search Trees

A random binary search tree is a basic binary search tree in which $A_1, \ldots, A_n$ is a random permutation of $1, \ldots, n$

Question 1: What is the expected number of comparisons when building a RBST?

Question 2: What is the expected number of comparisons when searching a RBST?

Cost of Building RBST

$$I_{x,y} = \begin{cases} \text{if } x \text{ is compared to } y \text{ when building RBST} \\ 0 \text{ otherwise} \end{cases}$$

$$\boxed{1\ 2\ 3 \cdots\cdots \underbrace{(x\ \ x+1 \cdots\cdots\cdots y-1\ \ y)}_{y-x+1} \cdots\cdots\cdots\cdots n}$$

Which element of $\{x, x+1, \ldots\ldots, y\}$ appears first in $x_1, \ldots\ldots x_n$?

x:	y:	$i \in \{x+1, \ldots, y-1\}$:

Cost of Building RBST (Cont'd)

$$E[I_{x,y}] = \frac{2}{y-x+1}$$

The cost to build a RBST is

$\boxed{\text{Cost} = \#\text{comparisons}}$

$$E\left[\sum_{x=1}^{n-1} \sum_{y=x+1}^{n} I_{x,y}\right] = \sum_{x=1}^{n-1} \sum_{y=x+1}^{n} E[I_{x,y}]$$

$$= \sum_{x=1}^{n-1} \sum_{y=x+1}^{n} \frac{2}{y-x+1}$$

$$= 2\left(H_{n-1} + H_{n-2} + H_{n-3} + \cdots + H_1\right)$$

$$\leq 2 \cdot n \cdot H_n \leq 2 \cdot n \ln n + O(n)$$

Cost of Searching in RBST

The cost to search for an element K not in a RBST

$$I_x = \begin{cases} 1 & \text{if } K \text{ is compared to } x \\ 0 & \text{otherwise} \end{cases}$$

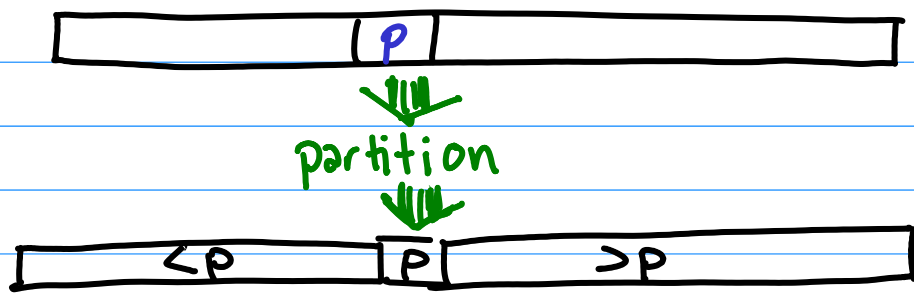
$$E\left[\sum_{x=1}^{n} I_x\right] = \sum_{x=1}^{\lfloor K \rfloor} \frac{1}{\lfloor K \rfloor - x + 1} + \sum_{x=\lceil K \rceil}^{n} \frac{1}{x - \lceil K \rceil + 1} \leq 2H_n \leq 2\ln n + O(1)$$

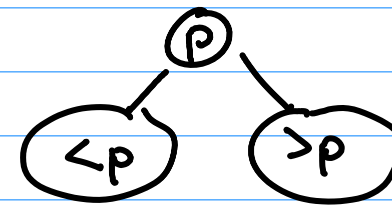
The expected cost of searching for a Key in the RBST is less.

Quicksort (Hoare 1961)

Constructing a RBST is an $O(n \log n)$ time algorithm for sorting n numbers.

This algorithm is called Quick Sort:





Theorem: The expected number of comparisons done by QuickSort when sorting an array of length n is at most $2n\ln n + O(n)$

Analysis of Random Binary Search Trees

The cost to search for an element K not in a RBST

$$I_x = \begin{cases} 1 & \text{if } K \text{ is compared to } x \\ 0 & \text{otherwise} \end{cases}$$

$$E\left[\sum_{x=1}^{n} I_x\right] = \sum_{x=1}^{\lfloor K \rfloor} \frac{1}{\lfloor K \rfloor - x + 1} + \sum_{x=\lceil K \rceil}^{n} \frac{1}{x - \lceil K \rceil + 1} \lesssim 2H_n \leq 2\ln n + O(1)$$

The expected cost of searching for a Key in the RBST is less.

$\therefore$ RBST is a fast searching data structure
But it's not dynamic

Treaps

A *treap* is a binary search tree where each key K has a *priority* $p(K)$.

A treap satisfies the heap property

$$p(K) \geq p(\text{parent}(K)) \text{ for all } K \text{ except the root.}$$

$\therefore$ A treap is uniquely defined by its priorities and its keys

Treap - Example

Keys	1	2	3	4	5	6	7	8	9	10
priorities	0.3	0.2	0.5	0.8	0.1	0.4	0.7	0.6	0.9	1.0

Equivalently:
- Sort Keys by priority values
- Insert into (basic) binary search tree

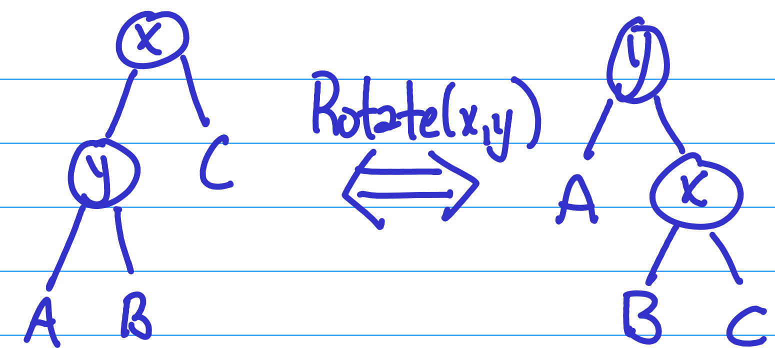
Random Treap

In a random treap, each priority is a random real number in $[0,1]$.

∴ A random treap is equivalent to a RBST

∴ The expected cost to search for an element using a random treap is at most $2\ln n + O(1)$.

Inserting into a Random Treap



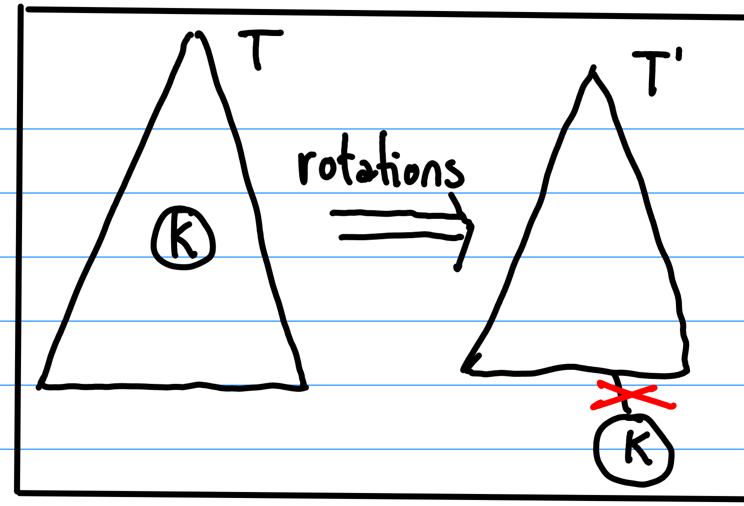
Insert(K)

1: · BasicInsert(K)
2: $p(K) \leftarrow$ random$(0,1)$
3: while K is not the root and $p(K) < p(\text{parent}(K))$ do
4: Rotate(K, parent(K))

Note: $\#\text{rotations} \leq \text{depth}(K) \Rightarrow E[\#\text{rotations} \leq 2\ln n + O(1)]^{*}$

$\therefore$ The expected time to insert into a random treap is $O(\log n)$

Deleting From a Random Treap



Delete(K)

```
while K has some child do
    ℓ ← child of K with minimum p(ℓ)
    Rotate(K, ℓ)
Snip(K)
```

Notice: Deleting K from T the reverse of inserting K into T'.

$\therefore$ The expected time to delete a Key from a random treap is $O(\log n)$

Summary

Theorem: The expected number of inversions in a random permutation is $\left(\frac{1}{2}\right)\cdot\binom{n}{2}$

Theorem: The expected number of records in a random permutation is $H_n \leq \ln n + 1$.

Theorem: The expected running time of QuickSort is $O(n \log n)$. The expected number of comparisons is $2n \ln n + O(n)$.

Theorem: The expected time to search, insert, or delete from a treap is $O(\log n)$.
The expected number of comparisons for each operation is $2 \ln n + O(1)$.

Summary

Theorem: The expected number of inversions in a random permutation is $\left(\frac{1}{2}\right)\cdot\binom{n}{2}$

Theorem: The expected number of records in a random permutation is $H_n \leq \ln n + 1$.

Theorem: The expected running time of QuickSort is $O(n \log n)$.
The expected number of comparisons is $2n\ln n + O(n)$.

Theorem: The expected time to search, insert, or delete from a treap is $O(\log n)$.
The expected number of comparisons for each operation is $2\ln n + O(1)$
The expected number of rotations is $O(1)$
The expected distance between the element of rank i and the element of rank j is $O(\log |j-i|)$