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 \((A_1, \ldots, A_n)\)
1. \(\text{for } i < 1 \text{ to } n-1 \text{ do}\)
2. \(j \leftarrow i+1\)
3. \(\text{while } j > 1 \text{ and } A_{j-1} > A_j \)\)
4. \(\text{swap } A_{j-1} \leftrightarrow A_j\)
5. \(j \leftarrow j-1\)
Analysis of Insertion Sort

\[
\text{Sort}(A_1, \ldots, A_n)
\]

1. for \(i = 1\) to \(n-1\) do
2. \(j \gets i + 1\)
3. while \(j > 1\) and \(A_{j-1} > A_j\)
4. swap \(A_{j-1} \leftrightarrow A_j\)
5. \(j \gets j - 1\)

How often does each line execute?
Average Case Analysis of InsertionSort

```
Sort(A₁, ..., Aₙ)
1: for i ← 1 to n-1 do
2:   j ← i+1
3:   while j > 1 and Aⱼ₋₁ > Aⱼ do
4:     swap Aⱼ₋₁ ← Aⱼ
5:     j ← j-1
```

What if A₁, ..., Aₙ is a random permutation of 1, ..., n?

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

Then we are interested in \( \sum_{x=1}^{n-1} \sum_{y=x+1}^{n} I_{xy} \) (Inversions of A₁, ..., Aₙ)

\[ = \# \text{ times line 4 executes} \]
\[
E \left[ \sum_{x=1}^{n-1} \sum_{y=x+1}^{n} I_{x,y} \right] = \sum_{x=1}^{n-1} \sum_{y=x+1}^{n} E[I_{x,y}]
\]

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

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

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

**InsertionSort** does half as many swaps on a random input
Randomized Insertion Sort

\[
\text{Sort} \left( A_1, \ldots, A_n \right)
\]

\[
0: \quad \text{Permute} \left( A_1, \ldots, A_n \right)
\]

\[
1: \quad \text{for } i \leftarrow 1 \text{ to } n-1 \text{ do}
\]

\[
2: \quad j \leftarrow i+1
\]

\[
3: \quad \text{while } j > 1 \text{ and } A_{j-1} > A_j
\]

\[
4: \quad \text{swap } A_{j-1} \leftrightarrow A_j
\]

\[
5: \quad j \leftarrow j-1
\]

\[
\{ O(n) \}
\]

\[
\text{Permute} \left( A_1, \ldots, A_n \right)
\]

\[
0: \quad \text{for } i \leftarrow 1 \text{ to } n-1 \text{ do}
\]

\[
1: \quad j \leftarrow \text{random (} i, \ldots, n \text{)}
\]

\[
2: \quad \text{swap } A_i \leftrightarrow A_j
\]

\[
\{ O(n) \}
\]

\[
\frac{1}{2} \ln(n) = \frac{1}{4} n^2 - \frac{n}{4}
\]

The additional work is \( O(n) \) but eliminates \( \frac{n^2}{2} \) comparisons.
Finding the Minimum

\[
\text{Find Min}(A_1, \ldots, A_n)
\]

1: \( m \leftarrow \infty \)
2: \( \text{for } i = 1 \text{ to } n \text{ do} \)
3: \( \text{if } A_i < m \text{ then} \)
4: \( m \leftarrow A_i \)
5: \( \text{return } m \)

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

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

\[
E[I_i] = \frac{1}{i}
\]
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
```

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

“n-th harmonic number”
Harmonic Numbers

\[ \int_1^n \frac{1}{x} \, dx \leq H_n \leq 1 + \int_1^n \frac{1}{x} \, dx \]

\[ \ln n \leq H_n \leq \ln n + 1 \]

nth 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

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

\[ 1 \ 2 \ 3 \ \cdots \ (x \ x+1 \ \cdots \ y-1 \ y) \ \cdots \ \cdots \ n \]

\[ y-x+1 \]

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

\[
\begin{array}{c}
x: \\
y: \\
i \in \{x+1, \ldots, y-1\}: 
\end{array}
\]
Cost of Building RBST (Cont'd)

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

The cost to build a RBST is

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

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

\[
= 2 \left( H_{n-1} + H_{n-2} + H_{n-3} + \ldots + H_1 \right)
\]

\[
\leq 2 \cdot n \cdot H_n \leq 2 \cdot n \ln n + O(n)
\]
Cost of Searching in RBST

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

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

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

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 **QuickSort**:

```
> P
partition
< P | P | > P
```

**Theorem:** The expected number of comparisons done by Quicksort when sorting an array of length $n$ is at most $2n \log n + O(n)$
Analysis of Random Binary Search Trees

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

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

$$E \left[ \sum_{x=1}^{n} I_x \right] = \sum_{x=1}^{n} \frac{1}{x^{k_1-k_x+1}} + \sum_{x=k_1}^{n} \frac{1}{x^{k_1-k_1+1}} \leq 2H_n \leq 2\ln n + O(1)$$

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

\[ \therefore \text{RBST is a fast searching data structure} \]

\[ \text{But it's not dynamic} \]
Treaps

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

A treap satisfies the heap property

$$p(k) \geq p(\text{parent}(k))$$

for all $k$ except the root.

- A treap is uniquely defined by its priorities and its keys.
**Treap - Example**

<table>
<thead>
<tr>
<th>Keys</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priorities</td>
<td>0.3</td>
<td>0.2</td>
<td>0.5</td>
<td>0.8</td>
<td>0.1</td>
<td>0.4</td>
<td>0.7</td>
<td>0.6</td>
<td>0.9</td>
<td>1.0</td>
</tr>
</tbody>
</table>

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. Basic Insert(K)
2. \( p(K) \leftarrow \text{random}(0,1) \)
3. while \( K \) is not the root and \( p(K) < p(\text{parent}(K)) \) do
4. \( \text{Rotate}(K, \text{parent}(K)) \)

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

\(*\): The expected time to insert into a random treap is \( O(\log n) \)
Deleting from a Random Treap

Delete $(K)$
1. while $K$ has some child do
2. $\lambda \leftarrow$ child of $K$ with minimum $R(\lambda)$
3. Rotate $(K, \lambda)$
4. Snip $(K)$

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

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 $\binom{n}{2} \cdot \binom{1}{2}$.

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 $O(n \log m)$. The expected number of comparisons is $2n \ln n + O(m)$.

Theorem: The expected time to search, insert, or delete from a heap 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 \( O(n \ln n) \).
\[ \text{The expected number of comparisons is } 2n \ln n + O(n). \]

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