Randomized Quicksort

Viet Tung Hoang

The slides are loosely based on those of Prof. Mary Wootters, Stanford University.
Randomized Quick Sort

We want to sort this array.

7  6  3  5  1  2  4
Randomized Quick Sort

We want to sort this array.

First, pick a pivot
Do it at random

7 6 3 5 1 2 4
We want to sort this array.

First, pick a **pivot**

Do it at random
Randomized Quick Sort

First, pick a **pivot**
*Do it at random*

Next, **partition** the array into “bigger than 5” or “less than 5”

We want to sort this array.

```
7 6 3 5 1 2 4
```
Randomized Quick Sort

We want to sort this array.

First, pick a **pivot**
- **Do it at random**

Next, **partition** the array into
- “bigger than 5” or “less than 5”

\[
\begin{array}{cccc}
7 & 6 & 3 & 5 \\
1 & 2 & 4 &
\end{array}
\]

**L** = array with things smaller than pivot

**R** = array with things larger than pivot
Randomized Quick Sort

We want to sort this array.

First, pick a **pivot**
*Do it at random*

Next, **partition** the array into
“bigger than 5” or “less than 5”

$L = \text{array with things smaller than } A[\text{pivot}]$

$R = \text{array with things larger than } A[\text{pivot}]$
We want to sort this array.

First, pick a **pivot**

**Do it at random**

Next, **partition** the array into

“bigger than 5” or “less than 5”

$L = \text{array with things smaller than pivot}$

$R = \text{array with things larger than pivot}$

Recurse on $L$ and $R$: 
Randomized Quick Sort

First, pick a **pivot**

**Do it at random**

Next, **partition** the array into “bigger than 5” or “less than 5”

We want to sort this array.

```
7 6 3 5 1 2 4
```

```
random pivot
```

```
3 1 2 4
```

$L = \text{array with things smaller than pivot}$

```
5
```

$R = \text{array with things larger than pivot}$

```
1 2 3 4 5 6 7
```
Implementing Randomized Quick Sort

procedure QuickSort(A[1 : n])
if $n \leq 0$ then return $A$  // Base case
Pick $r \leftarrow \{1, \ldots, n\}$; pivot $\leftarrow A[r]$
$k \leftarrow \text{Partition}(A, \text{pivot})$
QuickSort($A[1 : k - 1]$) // Sort left half
QuickSort($A[k + 1 : n]$) // Sort right half
return $A$
How to Analyze the Running Time?

Scenario 1
1. Bad guy picks the input.
2. You run your randomized algorithm.

Scenario 2
1. Bad guy picks the input.
2. Bad guy chooses the randomness (fixes the dice)

Running time is a random variable $T$

Expected running time: $E[T]$

Worst-case running time

What’s the worst-case running time of Randomized QuickSort?
Expected Running Time of Quick Sort

$C_n$: expected cost for sorting $n$ elements

```plaintext
procedure QuickSort(A[1 : n])
if $n \leq 0$ then return $A$  // Base case
Pick $r \leftrightarrow \{1, \ldots, n\}$; pivot $\leftarrow A[r]$
$k \leftarrow \text{Partition}(A, \text{pivot})$
return $A$
```

Since $k \leftrightarrow \{1, \ldots, n\}$:  $C_n \leftarrow n + 1 + \frac{1}{n} \sum_{i=1}^{n} C_{i-1} + C_{n-i}$
Expected Running Time of Quick Sort

\[
C_0 \leftarrow 0, \text{ and } C_n \leftarrow n + 1 + \frac{1}{n} \sum_{i=1}^{n} C_{i-1} + C_{n-i} \text{ for } n > 0
\]

Simplify

\[
C_0 \leftarrow 0, \text{ and } nC_n \leftarrow (n + 1)C_{n-1} + 2n \text{ for } n > 0
\]

Summation factor

\[
C_n = 2(n + 1)H_{n+1} - 2(n + 1) = 2n \ln(n) + O(n)
\]
Question: If the array size is tiny (say $n \leq 8$) then which of the following algorithms is the fastest on an average input?

A. Merge Sort
B. Heap Sort
C. Quick Sort
D. Insertion Sort
Recap: Insertion Sort

Start by moving A[2] toward the beginning of the list until you find something smaller

\[
\begin{array}{cccccc}
6 & 4 & 3 & 8 & 5 \\
3 & 4 & 6 & 3 & 8 & 5 \\
3 & 4 & 6 & 8 & 5 \\
\end{array}
\]

Then move A[4]:

\[
\begin{array}{cccccc}
3 & 4 & 6 & 8 & 5 \\
3 & 4 & 6 & 8 & 5 \\
3 & 4 & 5 & 6 & 8 \\
\end{array}
\]

Then move A[5]:

\[
\begin{array}{cccccc}
3 & 4 & 5 & 6 & 8 \\
3 & 4 & 5 & 6 & 8 \\
3 & 4 & 5 & 6 & 8 \\
\end{array}
\]

Then we are done!

Cost \leq \frac{n(n - 1)}{2}.

Improving Quick Sort

Use Insertion Sort if the array size is small

```plaintext
procedure QuickSort(A[1 : n])
if n ≤ M then return InsertionSort(A) // Base case

Pick r ∈ {1, ..., n}; pivot ← A[r]
k ← Partition(A, pivot)
QuickSort(A[1 : k − 1]) // Sort left half
QuickSort(A[k + 1 : n]) // Sort right half
return A
```

Assuming that Insertion Sort is free, what’s the expected cost?
**Added Cost Due to Insertion Sort**

**Observation:** Subarrays sorted by Insertion Sort are disjoint.

\[
S_1 \leq M \quad S_2 \leq M \quad \ldots \quad S_t \leq M
\]

\[
\text{Subarray sorted by Insertion Sort}
\]

\[
\text{Cost} \leq \sum_{i=1}^{t} \frac{S_i(S_i - 1)}{2} \leq \frac{M - 1}{2} \cdot \sum_{i=1}^{t} S_i \leq \frac{(M - 1)n}{2}
\]