Lecture 11: Runtime of Quicksort
COMS10007 - Algorithms

Dr. Christian Konrad

09.03.2020
Quicksort

**Require:** array $A$ of length $n$

if $n \leq 10$ then

Sort $A$ using your favourite sorting algorithm

else

$i \leftarrow \text{Partition}(A)$

$\text{QUICKSORT}(A[0, i - 1])$

$\text{QUICKSORT}(A[i + 1, n - 1])$


Algorithm $\text{QUICKSORT}$
Quicksort

**Require:** array $A$ of length $n$

if $n \leq 1$ then

return $A$

else

$i \leftarrow \text{Partition}(A)$

QUICKSORT($A[0, i - 1]$)

QUICKSORT($A[i + 1, n - 1]$)

Algorithm QUICKSORT
Quicksort

Require: array $A$ of length $n$
\[
\text{if } n \leq 1 \text{ then return } A
\]
else
\[
i \leftarrow \text{Partition}(A)
\]
QUICKSORT($A[0, i - 1]$)
QUICKSORT($A[i + 1, n - 1]$)

Algorithm QUICKSORT

Partition $A$ around a Pivot:

\begin{tabular}{cccccccccc}
14 & 3 & 9 & 8 & 16 & 2 & 1 & 7 & 11 & 12 & 5
\end{tabular}

\begin{tabular}{cccccccc}
\hline
\hline
\hline
\hline
\hline
\hline
\hline
\hline
\hline
\hline
\hline
\hline
\end{tabular}

\begin{tabular}{cccccccc}
7
\end{tabular}
Quicksort

Require: array $A$ of length $n$

if $n \leq 1$ then

return $A$

else

$i \leftarrow \text{Partition}(A)$

$\text{QUICKSORT}(A[0, i - 1])$

$\text{QUICKSORT}(A[i + 1, n - 1])$

Algorithm \text{QUICKSORT}

Partition $A$ around a Pivot:

\begin{center}
\begin{tabular}{cccccccc}
14 & 3 & 9 & 8 & 16 & 2 & 1 & 7 \\
3 & 2 & 1 & 5 & 7 & 14 & 9 & 8 & 16 & 11 & 12 & 5
\end{tabular}
\end{center}
**Quicksort**

**Require:** array $A$ of length $n$

- if $n \leq 1$ then
  - return $A$

- else
  - $i \leftarrow \text{Partition}(A)$
  - $\text{QUICKSORT}(A[0, i - 1])$
  - $\text{QUICKSORT}(A[i + 1, n - 1])$

*Algorithm QUICKSORT*

**Partition $A$ around a Pivot:**

```
14  3  9  8  16  2  1  7  11  12  5
```

```
 1  2  3  5  7  8  9 11 12 14 16
```
Runtime of Quicksort

**Runtime:** \( T(n) \): worst-case runtime on input of length \( n \)

\[
\begin{align*}
T(1) &= O(1) \quad \text{(termination condition)} \\
T(n) &= O(n) + T(n_1) + T(n_2),
\end{align*}
\]

where \( n_1, n_2 \) are the lengths of the two resulting subproblems.

**Observe:** \( n_1 + n_2 = n - 1 \)

**Worst-case:**
- Suppose that pivot is always the largest element
- Then, \( n_1 = n - 1, \ n_2 = 0 \)

**Best-case:**
- Suppose pivot splits array evenly, i.e., pivot is the median
- Then, \( n_1 = \lceil \frac{n-1}{2} \rceil, \ n_2 = \lfloor \frac{n-1}{2} \rfloor \)
Quicksort: Worst case

**Partition:** Suppose Partition() runs in time at most $Cn$, for a constant $C$

**Recurrence:**

$$T(n) \leq Cn + T(n-1)$$

**Total Runtime:**

$$T(n) \leq \sum_{i=1}^{n} Ci = C \sum_{i=1}^{n} i$$

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

$$= C \frac{n^2}{2} + n = O(n^2)$$
Quicksort: Best case

Best Case: \( n_1, n_2 \leq \frac{n}{2} \)

Number of Levels: \( \ell \)
- Last level: \( n = 1 \)
  \[
  \frac{n}{2^{\ell-1}} \leq 1
  \]
  \[
  \log(n) + 1 \leq \ell
  \]
- Last but one level: \( n = 2 \)
  \[
  \frac{n}{2^{\ell-2}} > 1 \text{ which implies } \log(n) + 2 > \ell
  \]
- Hence, there are \( \ell = \lceil \log(n) \rceil + 1 \) levels

Total Runtime:
- Observe: Total runtime of Partition() in a level: \( O(n) \)
- Total runtime: \( \ell \cdot O(n) = O(n \log n) \) .
Good versus Bad Splits:

- It is crucial that subproblems are *roughly* balanced.
- In fact, enough if \( n_1 = \frac{1}{1000} n \) and \( n_2 = n - 1 - n_1 \) to get a runtime of \( O(n \log n) \).
- Even enough if subproblems roughly balanced *most of the time*.
- In practice, this happens most of the time, *Quicksort* is therefore usually very fast.
Only good splits: Recursion tree depth $\lceil \log n \rceil + 1$
Good & bad splits alternate: Recursion tree depth $2 \cdot (\lceil \log n \rceil + 1)$
Selecting good Pivots

Ideal Pivot: Median

Pivot Selection

- To obtain runtime of $O(n \log n)$, we can spend $O(n)$ time to select a good pivot.
- There are $O(n)$ time algorithms for finding the median.
- They are complicated and not efficient in practice.
- However, using such an algorithm gives $O(n \log n)$ worst case runtime!

Random Pivot Selection

Randomized Algorithm

- Randomized pivot selection turns Quicksort into a *Randomized Algorithm*
- Worst-case runtime: still $O(n^2)$ (we may be unlucky!)
- *Expected runtime*: Since we introduce randomness, the runtime of the algorithm becomes a random variable

**Definition** (Bad Split)
A split is *bad* if $\min\{n_1, n_2\} \leq \frac{1}{10} n$.

If we select the pivot randomly, how likely is it to have a bad split?
Probability of a Bad Split

- Bad split if element chosen as pivot is either among smallest 0.1 fraction of elements or among largest 0.1 fraction.
- Since our choice is random, this happens with probability 0.2.
- Hence, in average only 1 out of 5 splits is bad.
- Hence, 4 out of 5 times the algorithm makes enough progress.

Random Pivot Selection: **Quicksort** runs in expected time $O(n \log n)$ if the pivot is chosen uniformly at random.