Lecture 11: Runtime of Quicksort
COMS10007 - Algorithms

Dr. Christian Konrad

26.02.2020
Quicksort

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

if $n \leq 10$ then
  Sort $A$ using your favourite sorting algorithm
else
  $i \leftarrow \text{Partition}(A)$
  QUICKSORT($A[0, i-1]$)
  QUICKSORT($A[i+1, n-1]$)

Algorithm QUICKSORT
Algorithm Quicksort

\textbf{Require: } array $A$ of length $n$

\textbf{if} $n \leq 1$ \textbf{then}
\textbf{return } $A$

\textbf{else}
\begin{itemize}
  \item $i \leftarrow \text{Partition}(A)$
  \item \textsc{Quicksort}(\(A[0, i - 1]\))
  \item \textsc{Quicksort}(\(A[i + 1, n - 1]\))
\end{itemize}
Quicksort

Require: array $A$ of length $n$

if $n \leq 1$ then
    return $A$
else
    $i \leftarrow$ Partition($A$)
    QUICKSORT($A[0, i - 1]$)
    QUICKSORT($A[i + 1, n - 1]$)

Algorithm QUICKSORT

Partition $A$ around a Pivot:
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:

$14\ 3\ 9\ 8\ 16\ 2\ 1\ 7\ 11\ 12\ 5$
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

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

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

Algorithm QUICKSORT

Partition $A$ around a Pivot:

\[
\begin{array}{cccccccc}
14 & 3 & 9 & 8 & 16 & 2 & 1 & 7 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 5 & 7 & 8 & 9 & 11 & 12 & 14 & 16 \\
\end{array}
\]
Runtime of Quicksort

Runtime:

\[ T(n) := \begin{cases} O(1) & \text{(termination condition)} \\ O(n) + T(n_1) + T(n_2) & \text{where } n_1, n_2 \text{ are the lengths of the two resulting subproblems.} \end{cases} \]

Observe:
\[ n_1 + n_2 = n - 1 \]

Worst-case:
Suppose that pivot is always the largest element.
Then, \[ n_1 = n - 1, \quad n_2 = 0 \]

Best-case:
Suppose pivot splits array evenly, i.e., pivot is the median.
Then, \[ n_1 = \lfloor \frac{n - 1}{2} \rfloor, \quad n_2 = \lceil \frac{n - 1}{2} \rceil, \]
Runtime: $T(n)$: worst-case runtime on input of length $n$
Runtime: $T(n)$: worst-case runtime on input of length $n$

$T(1) = O(1)$ (termination condition)
Runtime: \( T(n) \): worst-case runtime on input of length \( n \)

\[
T(1) = O(1) \quad \text{(termination condition)}
\]

\[
T(n) = O(n) + T(n_1) + T(n_2),
\]

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

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

\[
T(1) = O(1) \quad \text{(termination condition)}
\]

\[
T(n) = O(n) + T(n_1) + T(n_2)
\]

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

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

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

$$
T(1) = O(1) \quad \text{(termination condition)}
$$

$$
T(n) = O(n) + T(n_1) + T(n_2),
$$

where $n_1, n_2$ are the lengths of the two resulting subproblems.

**Observe:** $n_1 + n_2 = n - 1$

**Worst-case:**
Runtime of Quicksort

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

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

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
Runtime of Quicksort

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

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

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, $n_1 = n - 1, n_2 = 0$
Runtime of Quicksort

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

$$T(1) = O(1) \quad \text{(termination condition)}$$
$$T(n) = O(n) + T(n_1) + T(n_2),$$

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:
Runtime of Quicksort

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

$$
T(1) = O(1) \quad \text{(termination condition)}
$$
$$
T(n) = O(n) + T(n_1) + T(n_2),
$$

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

\[
T(1) = O(1) \quad \text{(termination condition)}
\]
\[
T(n) = O(n) + T(n_1) + T(n_2),
\]
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, $n_1 = \lceil \frac{n-1}{2} \rceil, n_2 = \lfloor \frac{n-1}{2} \rfloor$
Partition:

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

Recurrence:

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

Total Runtime:

$$T(n) \leq n \sum_{i=1}^{n} C_i = Cn \sum_{i=1}^{n} i = C(n+1)n = C(2(n^2) + n) = \Theta(n^2).$$
**Partition**: Suppose Partition() runs in time at most $Cn$, for a constant $C$.
**Partition:** Suppose Partition() runs in time at most $Cn$, for a constant $C$. 

- For $n = 16$, the array is partitioned into subarrays of size 1 and 15.
- For $n = 15$, the array is partitioned into subarrays of size 1 and 14.
- For $n = 14$, the array is partitioned into subarrays of size 1 and 13.
- This pattern continues until the base case is reached, where subarrays of size 1 are encountered.

The total runtime can be calculated as:

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

Solving this recurrence relation gives:

$$T(n) \leq Cn + C(n-1) + C(n-2) + \ldots + C2 + C1$$

$$\sum_{i=1}^{n} Ci = Cn + C(n-1) + C(n-2) + \ldots + C2 + C1$$

$$= C(n + (n-1) + (n-2) + \ldots + 2 + 1)$$

$$= C(n + (n-1) + (n-2) + \ldots + 2 + 1)$$

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

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

$$= O(n^2)$$
**Quicksort: Worst case**

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

**Recurrence:**

```
1 2 3 7 8 9 14 16  n
```
```
1 2 3 7 8 9 14  n-1
```
```
1 2 3 7 8 9  n-2
```
```
1 2 2  n
```
```
1 1 1
```

Total Runtime:

$$T(n) \leq Cn + T(n-1) + T(n-2)$$

$$T(n) \leq n \sum_{i=1}^{Cn} = Cn \cdot \frac{n+1}{2} = C \frac{n^2}{2} + Cn = O(n^2).$$
Quicksort: Worst case

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

**Recurrence:**

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

$$T(n) \leq Cn \sum_{i=1}^{n} i = C\left(\frac{n(n+1)}{2}\right) = O(n^2)$$
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:**
$$T(n) \leq Cn \sum_{i=1}^{n} i = C(n+1)n/2 = C(2n^2 + n) = O(n^2).$$
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:**
\[ T(n) \leq \sum_{i=1}^{n} Ci = C \sum_{i=1}^{n} i \]
**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:**
$$T(n) \leq \sum_{i=1}^{n} Ci = C \sum_{i=1}^{n} i$$
$$= C \frac{(n + 1)n}{2}$$
**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:**
\[
T(n) \leq \sum_{i=1}^{n} Ci = C \sum_{i=1}^{n} i = C \frac{(n+1)n}{2} = C \frac{n^2 + n}{2}
\]

\[
= C \frac{n^2 + n}{2}
\]

\[
= C \left( \frac{n^2}{2} + \frac{n}{2} \right)
\]

\[
= C \left( \frac{n^2}{2} + \frac{n}{2} \right)
\]
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:**

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

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

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

Best Case:

Number of Levels: $l$

Last level: $n = 1$

$\frac{n}{2} \leq n \leq n^2$ implies $l \geq \log(n) + 1$

Last but one level: $n = 2$

$\frac{n}{4} \leq n \leq \frac{n}{2}$ implies $l \geq \log(n) + 2$

Hence, there are $l = \lceil \log(n) \rceil + 1$ levels

Total Runtime:

Observe: Total runtime of Partition() in a level: $O(n)$

Total runtime: $l \cdot O(n) = O(n \log n)$
**Quicksort: Best case**

**Best Case:** $n_1, n_2 \leq \frac{n}{2}$

![Quicksort Tree Illustration](image)

- **Number of Levels:** $l$
- **Last level:** $n = 1$
- **Number of elements in last level:** $n^2 l - 1 \leq 1$
- **Logarithmic Bound:** $\log(n) + 1 \leq l$
- **Number of levels:** $l = \lceil \log(n) \rceil + 1$

**Total Runtime:**
- **Total runtime of Partition() in a level:** $O(n)$
- **Total runtime:** $l \cdot O(n) = O(n \log n)$
Quicksort: Best case

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

Number of Levels: \( l \)

- Last level: \( n = 1 \)
- \( n^2 - l - 1 \leq 1 \)
- \( \log(n) + 1 \leq l \)
- Hence, there are \( l = \lceil \log(n) \rceil + 1 \) levels

Total Runtime:
- Observe: Total runtime of Partition() in a level: \( O(n) \)
- Total runtime: \( l \cdot O(n) = O(n \log n) \).
QuickSort: Best case

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

Number of Levels: \( l \)
- Last level: \( n = 1 \)
**Quicksort: Best case**

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

**Number of Levels:** \( l \)
- Last level: \( n = 1 \)
  \[
  \frac{n}{2^{l-1}} \leq 1
  \]

**Total Runtime:**
- Observe: Total runtime of \( \text{Partition}() \) in a level: \( \mathcal{O}(n) \)
- Total runtime: \( l \cdot \mathcal{O}(n) = \mathcal{O}(n \log n) \)
Best Case: $n_1, n_2 \leq \frac{n}{2}$

Number of Levels: $l$
- Last level: $n = 1$
  \[ \frac{n}{2^{l-1}} \leq 1 \]
  \[ \log(n) + 1 \leq l \]
Quicksort: Best case

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

**Number of Levels:** \( l \)
- Last level: \( n = 1 \)
  \[ \frac{n}{2^{l-1}} \leq 1 \]
  \[ \log(n) + 1 \leq l \]
- Last but one level: \( n = 2 \)
Best Case: $n_1, n_2 \leq \frac{n}{2}$

Number of Levels: $l$

- Last level: $n = 1$
  \[ \frac{n}{2^{l-1}} \leq 1 \]
  \[ \log(n) + 1 \leq l \]

- Last but one level: $n = 2$
  \[ \frac{n}{2^{l-2}} > 1 \]
Quicksort: Best case

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

Number of Levels: \( l \)
- Last level: \( n = 1 \)
  \[ \frac{n}{2^{l-1}} \leq 1 \]
  \[ \log(n) + 1 \leq l \]
- Last but one level: \( n = 2 \)
  \[ \frac{n}{2^{l-2}} > 1 \text{ which implies } \log(n) + 2 > l \]
**Quicksort: Best case**

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

**Number of Levels:** \( l \)
- Last level: \( n = 1 \)
  
  \[
  \frac{n}{2^{l-1}} \leq 1
  \]
  
  \[
  \log(n) + 1 \leq l
  \]
- Last but one level: \( n = 2 \)

  \[
  \frac{n}{2^{l-2}} > 1 \text{ which implies } \log(n) + 2 > l
  \]
- Hence, there are \( l = \lceil \log(n) \rceil + 1 \) levels
Quicksort: Best case

**Best Case:** $n_1, n_2 \leq \frac{n}{2}$

**Number of Levels:** $l$

- **Last level:** $n = 1$
  
  \[
  \frac{n}{2^{l-1}} \leq 1
  \]
  
  \[
  \log(n) + 1 \leq l
  \]

- **Last but one level:** $n = 2$
  
  \[
  \frac{n}{2^{l-2}} > 1 \text{ which implies } \log(n) + 2 > l
  \]

- **Hence, there are** $l = \lceil \log(n) \rceil + 1$ levels

**Total Runtime:**

```
Observe: Total runtime of Partition() in a level: $O(n)$
Total runtime: $l \cdot O(n) = O(n \log n)$.
```
Quicksort: Best case

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

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

Total Runtime:
- Observe: Total runtime of Partition() in a level: \( O(n) \)
**Best Case:** \( n_1, n_2 \leq \frac{n}{2} \)

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

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

It is crucial that subproblems are roughly balanced. In fact, enough if
\[ n_1 = \frac{n}{1000} \quad \text{and} \quad n_2 = n - n_1 \]

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, so Quicksort is therefore usually very fast.
Good versus Bad Splits:

- It is crucial that subproblems are *roughly* balanced
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 \( O(n \log n) \).
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*. 
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.
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 $2 \cdot (\lceil \log n \rceil + 1)$
Ideal Pivot:

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!

Idea that works in practice:
Select Pivot at random!

Ideal Pivot: Median
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!


Dr. Christian Konrad
Lecture 11: Runtime of Quicksort 8 / 10
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
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
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
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!
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!

**Idea that works in Practice:**
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!

**Idea that works in Practice:** Select Pivot at random!
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. The worst-case runtime is 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?
Randomized Pivot Selection

Randomized Algorithm

- Randomized pivot selection turns Quicksort into a Randomized Algorithm
Random Pivot Selection

**Randomized Algorithm**
- Randomized pivot selection turns Quicksort into a *Randomized Algorithm*
- Worst-case runtime:
Randomized Algorithm

- Randomized pivot selection turns Quicksort into a **Randomized Algorithm**
- Worst-case runtime: still $O(n^2)$ (we may be unlucky!)
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
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)
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 $\min\{n_1, n_2\} \leq \frac{1}{10} n$. 
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.
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.
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
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.
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