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

Algorithm QUICKSORT

Partition $A$ around a Pivot:

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

$$
\begin{array}{ccccccc}
\hline
\hline
\hline
\hline
7
\end{array}
$$
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

Partition $A$ around a Pivot:
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:

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:

- Worst-case: Suppose that pivot is always the largest element. Then, $n_1 = n - 1$, $n_2 = 0$.
- Best-case: Suppose pivot splits the array evenly, i.e., pivot is the median. Then, $n_1 = \lfloor (n-1)/2 \rfloor$, $n_2 = \lceil (n-1)/2 \rceil$. 

Suppose that $T(n)$ is the runtime on input of length $n$. Then, we have:

\[
T(n) = \begin{cases} 
O(1) & \text{if } n = 1, \\
O(n) + T(n_1) + T(n_2) & \text{otherwise,}
\end{cases}
\]
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)

Observe:

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, $n_1 = \lfloor \frac{n - 1}{2} \rfloor$, $n_2 = \lceil \frac{n - 1}{2} \rceil$
Runtime of Quicksort

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

$T(1) = O(1)$ \hspace{1cm} (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:**

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,

\[
T(n) = O(n) + T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil),
\]

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

**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: $T(n)$: 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 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
- Then, $n_1 = \lfloor \frac{n-1}{2} \rfloor$, $n_2 = \lceil \frac{n-1}{2} \rceil$
Quicksort: Worst case

Partition:

\[ T(n) \leq Cn + T(n-1) \]

Total Runtime:

\[ T(n) \leq n \sum_{i=1}^{\frac{n}{2}} C_i = Cn \sum_{i=1}^{\frac{n}{2}} i = C \left( n + 1 \right) \frac{n}{2} = C \left( \frac{n^2}{2} + \frac{n}{2} \right) = O(n^2). \]
Quicksort: Worst case

**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 $C$.

![Diagram of Quicksort partitioning]
**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:**

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

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

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

$$= \frac{C}{2} (n^2 + n)$$
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: \( \ell \)

Last level: \( n = 1 \), \( n^{2 \ell - 1} \leq 1 \log(n) + 1 \leq \ell \)

Last but one level: \( n = 2^n \), \( n^{2 \ell - 2} > 1 \log(n) + 2 \) 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: \( \ell \cdot O(n) = O(n \log n) \)
**Quicksort: Best case**

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

![Quicksort Tree Diagram]

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

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

Number of Levels: $\ell$

Number of Levels:

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

Total Runtime:

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

Total runtime: $\ell \cdot O(n) = O(n \log n)$.
Best Case: \( n_1, n_2 \leq \frac{n}{2} \)

Number of Levels: \( \ell \)
- Last level: \( n = 1 \)

\[
\begin{align*}
\text{Number of Levels: } \ell &= \lceil \log(n) \rceil + 1 \\
\text{Total Runtime: } &= \ell \cdot O(n) = O(n \log n) 
\end{align*}
\]
Quicksort: Best case

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

Number of Levels: \( \ell \)

- Last level: \( n = 1 \)

\[ \frac{n}{2^{\ell-1}} \leq 1 \]
Best Case: \( n_1, n_2 \leq \frac{n}{2} \)

Number of Levels: \( \ell \)

- Last level: \( n = 1 \)
  
  \[ \frac{n}{2^{\ell-1}} \leq 1 \]
  
  \[ \log(n) + 1 \leq \ell \]
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: \( n = 2 \)
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: $n = 2$
  \[
  \frac{n}{2^{\ell-2}} > 1
  \]
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: $n = 2$
  \[\frac{n}{2^{\ell-2}} > 1\] which implies $\log(n) + 2 > \ell$
**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
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:
**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: $O(n)$
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 \( n_2 = n - n_1 - n_1 \) to get a runtime of \( O(n \log n) \). Even if subproblems are roughly balanced most of the time, 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 versus Bad Splits: Intuition and Rough Analysis

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!

Idea that works in Practice:

Selecting good Pivots

**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!

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:**
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!

Idea that works in Practice: Select Pivot at random!
Random Pivot Selection

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?
Randomized Pivot Selection

Randomized Algorithm

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

Randomized Algorithm

- Randomized pivot selection turns Quicksort into a *Randomized Algorithm*
- 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!)
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

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.
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.