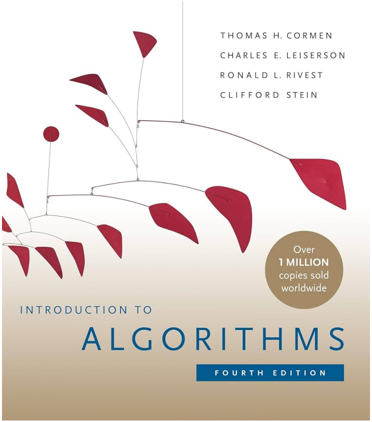
Introduction to Algorithms
Lecture 4: Quicksort

Prof. Charles E. Leiserson and Prof. Erik Demaine
Massachusetts Institute of Technology

March 4, 2025

Introduction to Algorithms



Content has been extracted from *Introduction to Algorithms*, Fourth Edition, by Cormen, Leiserson, Rivest, and Stein. MIT Press. 2022.
Visit `https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/`.
Original slides from *Introduction to Algorithms 6.046J/18.401J*, Fall 2005 Class by Prof. Charles Leiserson and Prof. Erik Demaine. MIT OpenCourseWare Initiative available at
`https://ocw.mit.edu/courses/6-046j-introduction-to-algorithms-sma-5503-fall-2005/`.

Plan

Quicksort

Divide & Conquer

Partitioning

Worst-case Analysis

Intuition

Randomized Quicksort

Analysis

Quicksort

- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- Sorts ‘in place’ (like insertion sort, but not like merge sort).
- Very practical (with tuning).

Plan

Quicksort

Divide & Conquer

Partitioning

Worst-case Analysis

Intuition

Randomized Quicksort

Analysis

Divide and Conquer

Quicksort an n-element array:

1. **Divide:** Partition the array into two subarrays around a **pivot** x such that elements in lower subarray $\leq x \leq$ elements in upper subarray.

2. **Conquer:** Recursively sort the two subarrays.
3. **Combine:** Trivial.

Key:

Linear-time partitioning subroutine.

Plan

Quicksort

Divide & Conquer

Partitioning

Worst-case Analysis

Intuition

Randomized Quicksort

Analysis

Partitioning Subroutine

```
procedure PARTITION(A, p, q)
    x ← A[p]
    i ← p
    for j ← p + 1 to q do
        if A[j] ≤ x then
            i ← i + 1
            exchange A[i] ↔ A[j]
        end if
    end for
    exchange A[p] ↔ A[i]
    return i
end procedure
```

Running time:

$O(n)$ for n elements.

Invariant:

x	$\leq x$	$\geq x$	?
p	i	j	q

Example of partitioning

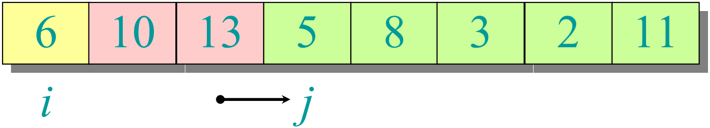
6	10	13	5	8	3	2	11
i	j						

Example of partitioning

6	10	13	5	8	3	2	11

i j

Example of partitioning



Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11

i j

Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11
	i			j			

Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

i j

Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
		i			j		

Example of partitioning

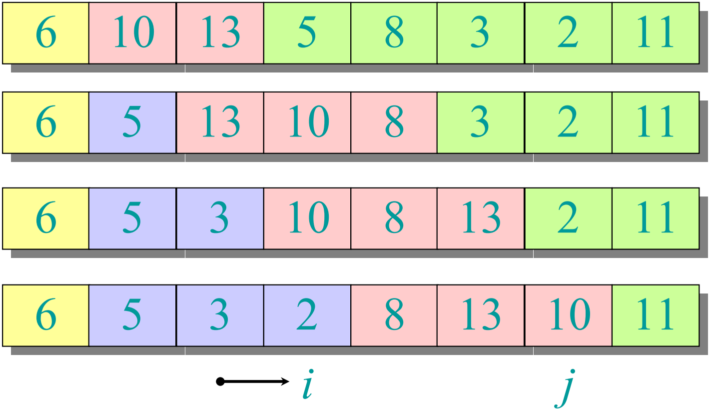
6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

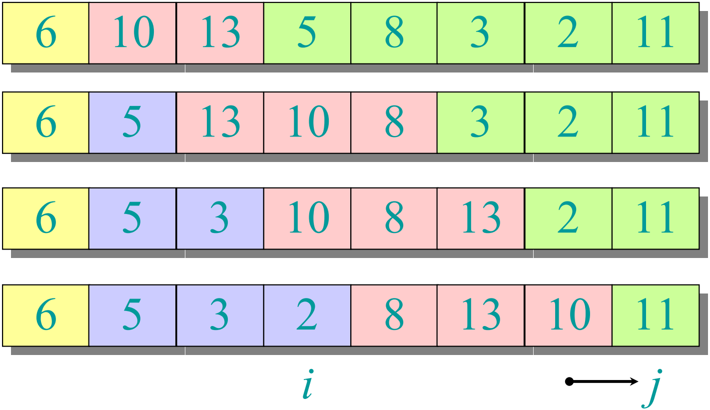
6	5	3	10	8	13	2	11

i j

Example of partitioning



Example of partitioning



Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11

i $\qquad\qquad\qquad \bullet\!\longrightarrow j$

Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11
2	5	3	6	8	13	10	11
			i				

Pseudocode for Quicksort

```
procedure QUICKSORT(A, p, r)
    if p < r then
        q ← PARTITION(A,p, r)
        QUICKSORT(A, p, q − 1)
        QUICKSORT(A, q + 1, r)
    end if
end procedure
```

Initial call:

QUICKSORT(A, 1, n)

Analysis of Quicksort

- Assume all input elements are distinct.
- In practice, there are better partitioning algorithms for when duplicate input elements may exist.
- Let $T(n)$ = worst-case running time on an array of n elements.

Plan

Quicksort

Divide & Conquer

Partitioning

Worst-case Analysis

Intuition

Randomized Quicksort

Analysis

Worst-case of Quicksort

- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.

$$
\begin{aligned}
T(n) =& T(0) + T(n-1) + \Theta(n) \\
=& \Theta(1) + T(n-1) + \Theta(n) \\
=& T(n-1) + \Theta(n) \\
=& \Theta(n^2)
\end{aligned}
$$

Worst-case of Quicksort

- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.

$$
\begin{aligned}
T(n) =& T(0) + T(n-1) + \Theta(n) \\
=& \Theta(1) + T(n-1) + \Theta(n) \\
=& T(n-1) + \Theta(n) \\
=& \Theta(n^2)
\end{aligned}
$$

Arithmetic Series!

Worst-case Recursion Tree

$$T(n) = T(0) + T(n-1) + cn$$

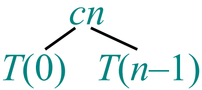
Worst-case Recursion Tree

$$T(n) = T(0) + T(n-1) + cn$$

$T(n)$

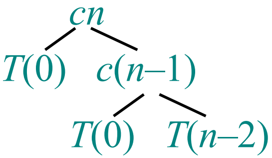
Worst-case Recursion Tree

$$T(n) = T(0) + T(n-1) + cn$$



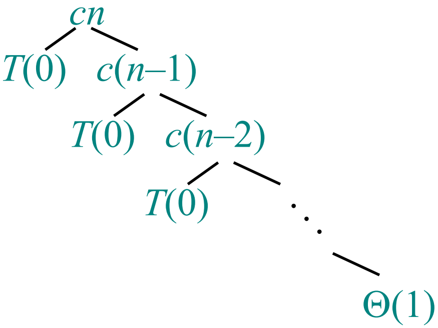
Worst-case Recursion Tree

$$T(n) = T(0) + T(n-1) + cn$$



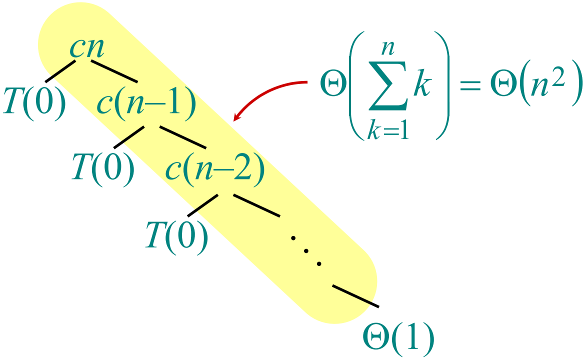
Worst-case Recursion Tree

$$T(n) = T(0) + T(n-1) + cn$$



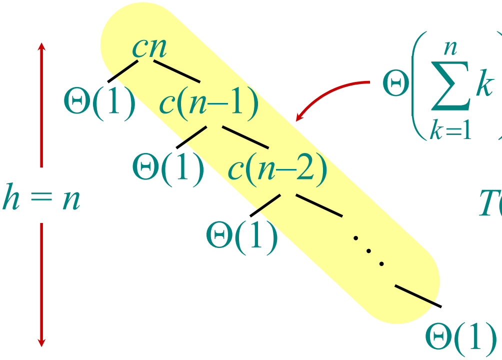
Worst-case Recursion Tree

$$T(n) = T(0) + T(n-1) + cn$$



Worst-case Recursion Tree

$$T(n) = T(0) + T(n-1) + cn$$



$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right)$$

$$\begin{aligned} T(n) &= \Theta(n) + \Theta(n^2) \\ &= \Theta(n^2) \end{aligned}$$

Plan

Quicksort

Divide & Conquer

Partitioning

Worst-case Analysis

Intuition

Randomized Quicksort

Analysis

Best-case Analysis

For intuition only!

If we're lucky, PARTITION splits the array evenly:

$$\begin{aligned} T(n) =& 2T\left(\frac{n}{2}\right) + \Theta(n) \\ =& \Theta(n \lg n) \qquad\qquad \text{(same as merge-sort)} \end{aligned}$$

What if the split is always $\frac{1}{10} : \frac{9}{10}$?

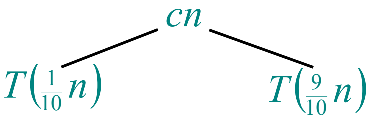
$$T(n) = T\left(\frac{1}{10}n\right) + T\left(\frac{9}{10}n\right) + \Theta(n)$$

What is the solution to this recurrence?

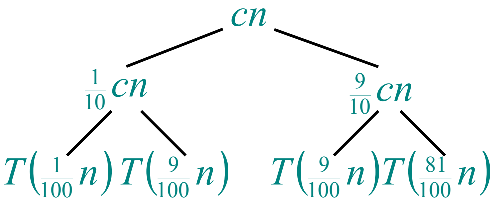
Analysis of "Almost-best" Case

$$T(n)$$

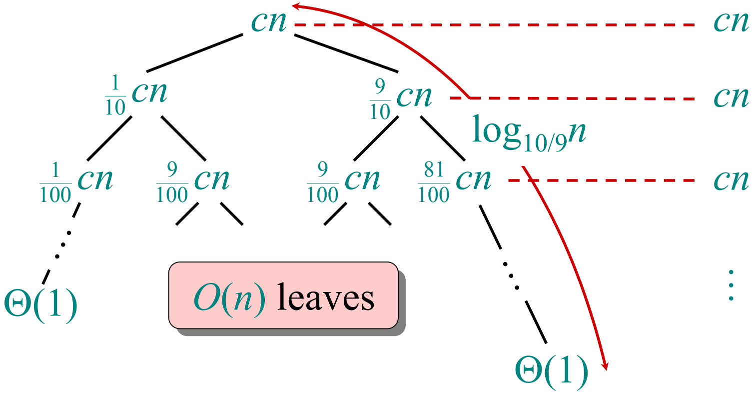
Analysis of “Almost-best” Case



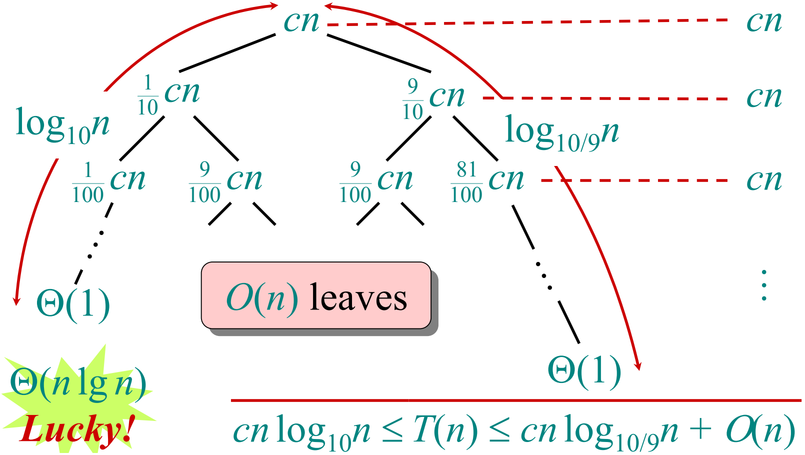
Analysis of “Almost-best” Case



Analysis of "Almost-best" Case



Analysis of "Almost-best" Case



More Intuition

Suppose we alternate lucky, unlucky, lucky, unlucky, lucky, ...

$$L(n) = 2U\left(\frac{n}{2}\right) + \Theta(n) \qquad \textit{luckcy}$$
$$U(n) = L(n-1) + \Theta(n) \qquad \textit{unluckcy}$$

Solving:

$$\begin{aligned} L(n) &= 2\left(L\left(\frac{n}{2} - 1\right) + \Theta(\frac{n}{2})\right) + \Theta(n) \\ &= 2L\left(\frac{n}{2} - 1\right) + \Theta(n) \\ &= \Theta(n \lg n) \qquad \textbf{Lucky!} \end{aligned}$$

How can we make sure we are usually lucky?

Plan

Randomized Quicksort

IDEA:
Partition around a **random** element.

- Running time is independent of the input order.
- No assumptions need to be made about the input distribution.
- No specific input elicits the worst-case behavior.
- The worst case is determined only by the output of a random-number generator.

Plan

Randomized Quicksort Analysis

Let $T(n)$ = the random variable for the running time of randomized quicksort on an input of size n, assuming random numbers are independent.

For $k = 0, 1, \ldots, n-1$, define the **indicator random variable**.

$$X_k = \begin{cases} 1 & \text{if } \textsc{Partition} \text{ generates a } k : n-k-1 split, \\ 0 & \text{otherwise.} \end{cases}$$

$E[X_k] = Pr\{X_k = 1\} = \frac{1}{n}$, since all splits are equally likely, assuming element are distinct.

Analysis (Cont.)

$$T(n) = \begin{cases} T(0) + T(n-1) + \Theta(n) & \text{if } 0 : n-1 \text{ split,} \\ T(1) + T(n-2) + \Theta(n) & \text{if } 1 : n-2 \text{ split,} \\ \quad \vdots & \\ T(n-1) + T(0) + \Theta(n) & \text{if } n-1 : 0 \text{ split.} \end{cases}$$

$$= \sum_{k=0}^{n-1} X_k (T(k) + T(n-k-1) + \Theta(n))$$

Calculating expectation

Take expectations of both sides.

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k(T(k) + T(n-k-1) + \Theta(n))\right]$$

Calculating expectation

Linearity of expectation.

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k(T(k) + T(n-k-1) + \Theta(n))\right] \\
&= \sum_{k=0}^{n-1} E\left[X_k(T(k) + T(n-k-1) + \Theta(n))\right]
\end{aligned}
$$

Calculating expectation

Independence of X_k from other random choices.

$$\begin{aligned} E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k(T(k) + T(n-k-1) + \Theta(n))\right] \\ &= \sum_{k=0}^{n-1} E\left[X_k(T(k) + T(n-k-1) + \Theta(n))\right] \\ &= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)] \end{aligned}$$

Calculating expectation

Linearity of expectation: $E[X_k] = \frac{1}{n}$.

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k(T(k) + T(n-k-1) + \Theta(n))\right] \\
&= \sum_{k=0}^{n-1} E\left[X_k(T(k) + T(n-k-1) + \Theta(n))\right] \\
&= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)] \\
&= \frac{1}{n}\sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n}\sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n)
\end{aligned}
$$

Calculating expectation

Summations have identical terms.

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k(T(k) + T(n-k-1) + \Theta(n))\right] \\
&= \sum_{k=0}^{n-1} E\left[X_k(T(k) + T(n-k-1) + \Theta(n))\right] \\
&= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)] \\
&= \frac{1}{n}\sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n}\sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n) \\
&= \frac{2}{n}\sum_{k=0}^{n-1} E[T(k)] + \Theta(n)
\end{aligned}
$$

Hairy recurrence

$$E[T(n)] = \frac{2}{n} \sum_{k=2}^{n-1} E[T(k)] + \Theta(n)$$

(The $k = 0$, 1 terms can be absorbed in the $\Theta(n)$.)

Prove:
$E[T(n)] \leq an \lg n$ for constant $a > 0$.

- Choose a large enough so that $an \lg n$ dominates $E[T(n)]$ for sufficiently small $n \geq 2$.

Use fact:
$\sum_{k=2}^{n-1} k \lg k \leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$ (exercise).

Substitution method

Substitute inductive hypothesis.

$$E[T(n)] \leq \frac{2}{n} \sum_{k=2}^{n-1} ak \lg k + \Theta(n)$$

Substitution method

Use fact.

$$
\begin{aligned}
E[T(n)] &\leq \frac{2}{n}\sum_{k=2}^{n-1} ak\lg k + \Theta(n) \\
&\leq \frac{2a}{n}\left(\frac{1}{2}n^2\lg n - \frac{1}{8}n^2\right) + \Theta(n)
\end{aligned}
$$

Substitution method

$$
\begin{aligned}
E[T(n)] &\leq \frac{2}{n}\sum_{k=2}^{n-1} ak\lg k + \Theta(n) \\
&\leq \frac{2a}{n}\left(\frac{1}{2}n^2\lg n - \frac{1}{8}n^2\right) + \Theta(n) \\
&= an\lg n - \left(\frac{an}{4} - \Theta(n)\right) \\
&\leq an\lg n,
\end{aligned}
$$

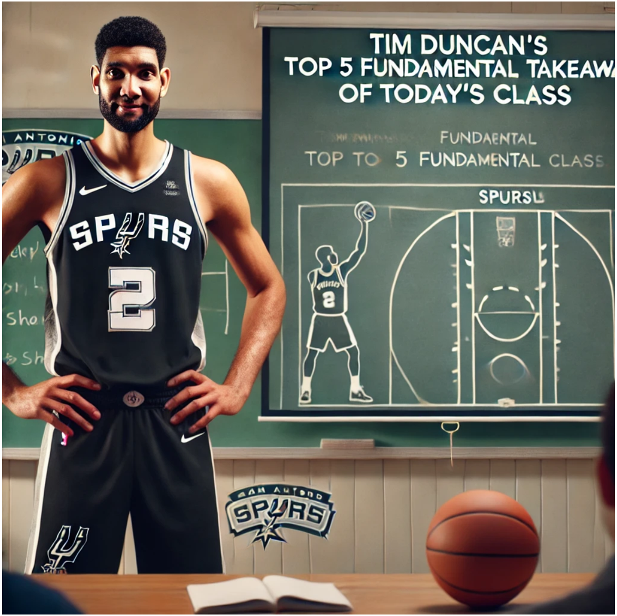
if a is chosen large enough so that $\frac{an}{4}$ dominates the $\Theta(n)$.

Quicksort in practice

- Quicksort is a great general-purpose sorting algorithm.
- Quicksort is typically over twice as fast as merge sort.
- Quicksort can benefit substantially from **code tuning**.
- Quicksort behaves well even with caching and virtual memory.

End of Lecture 4.

TDT5FTOTTC


TIM DUNCAN'S
TOP 5 FUNDAMENTAL TAKEAW
OF TODAY'S CLASS
FUNDAENTAL
TOP TO 5 FUNDAMENTAL CLASS
SPURSL
SPURS
2
SPURS

Top 5 Fundamental Takeaways

Top 5 Fundamental Takeaways

5. **Quicksort is a Divide-and-Conquer Algorithm** – It recursively partitions an array around a pivot and sorts the subarrays efficiently.

Top 5 Fundamental Takeaways

5 **Quicksort is a Divide-and-Conquer Algorithm** – It recursively partitions an array around a pivot and sorts the subarrays efficiently.

4 **Partitioning is the Core of Quicksort** – The partitioning step ensures elements are correctly placed around the pivot in $O(n)$ time.

Top 5 Fundamental Takeaways

5 **Quicksort is a Divide-and-Conquer Algorithm** – It recursively partitions an array around a pivot and sorts the subarrays efficiently.

4 **Partitioning is the Core of Quicksort** – The partitioning step ensures elements are correctly placed around the pivot in $O(n)$ time.

3 **Best-case and Worst-case Analysis** – Quicksort runs in $O(n \log n)$ in the best case but can degrade to $O(n^2)$ if poorly partitioned.

Top 5 Fundamental Takeaways

5 **Quicksort is a Divide-and-Conquer Algorithm** – It recursively partitions an array around a pivot and sorts the subarrays efficiently.

4 **Partitioning is the Core of Quicksort** – The partitioning step ensures elements are correctly placed around the pivot in $O(n)$ time.

3 **Best-case and Worst-case Analysis** – Quicksort runs in $O(n \log n)$ in the best case but can degrade to $O(n^2)$ if poorly partitioned.

2 **Randomized Quicksort Helps Avoid Worst-case Behavior** – Choosing a random pivot prevents consistently bad splits and ensures an expected $O(n \log n)$ runtime.

Top 5 Fundamental Takeaways

5 **Quicksort is a Divide-and-Conquer Algorithm** – It recursively partitions an array around a pivot and sorts the subarrays efficiently.

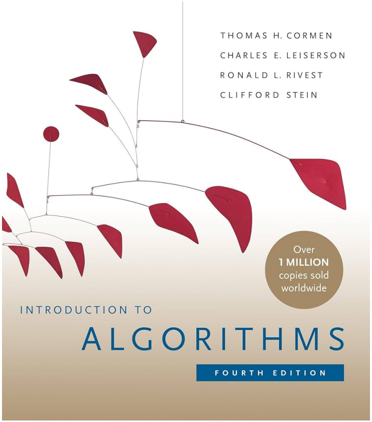
4 **Partitioning is the Core of Quicksort** – The partitioning step ensures elements are correctly placed around the pivot in $O(n)$ time.

3 **Best-case and Worst-case Analysis** – Quicksort runs in $O(n \log n)$ in the best case but can degrade to $O(n^2)$ if poorly partitioned.

2 **Randomized Quicksort Helps Avoid Worst-case Behavior** – Choosing a random pivot prevents consistently bad splits and ensures an expected $O(n \log n)$ runtime.

1 **Quicksort is Highly Efficient in Practice** – It outperforms merge sort in most cases and benefits from hardware optimizations.

Introduction to Algorithms



Content has been extracted from *Introduction to Algorithms*, Fourth Edition, by Cormen, Leiserson, Rivest, and Stein. MIT Press. 2022.
Visit `https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/`.
Original slides from *Introduction to Algorithms 6.046J/18.401J*, Fall 2005 Class by Prof. Charles Leiserson and Prof. Erik Demaine. MIT OpenCourseWare Initiative available at
`https://ocw.mit.edu/courses/6-046j-introduction-to-algorithms-sma-5503-fall-2005/`.