Quicksort

- Divide-and-conquer algorithm.
- Sorts “in place” (like insertion sort, but not like merge sort).
- Very practical (with tuning).
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.

   \[
   \begin{array}{ccc}
   \leq x & x & \geq x
   \end{array}
   \]

2. **Conquer:** Recursively sort the two subarrays.

3. **Combine:** Trivial.

   **Key:** Linear-time partitioning subroutine.
Partitioning subroutine

\textbf{PARTITION}(A, p, q) \triangleright A[p \ldots q]
\begin{align*}
x & \leftarrow A[p] \quad \triangleright \text{pivot } = A[p] \\
i & \leftarrow p \\
\text{for } j & \leftarrow p + 1 \text{ to } q \\
& \text{do if } A[j] \leq x \\
& \quad \text{then } i \leftarrow i + 1 \\
& \quad \text{exchange } A[i] \leftrightarrow A[j] \\
& \text{exchange } A[p] \leftrightarrow A[i] \\
\text{return } i
\end{align*}

**Invariant:**

\begin{tabular}{|c|c|c|c|}
\hline
x & \leq x & \geq x & ? \\
\hline
p & i & j & q \\
\hline
\end{tabular}

Running time

$= O(n)$ for $n$ elements.
Example of partitioning

\[ i \rightarrow j \]
Example of partitioning

\[ \text{6} \quad \text{10} \quad \text{13} \quad \text{5} \quad \text{8} \quad \text{3} \quad \text{2} \quad \text{11} \]

\[ i \quad \rightarrow \quad j \]
Example of partitioning

\[ \text{6} \quad \text{10} \quad \text{13} \quad \text{5} \quad \text{8} \quad \text{3} \quad \text{2} \quad \text{11} \]

\[ i \quad \quad \rightarrow \quad \quad j \]
Example of partitioning

\[
\begin{array}{llllllll}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
6 & 5 & 13 & 10 & 8 & 3 & 2 & 11
\end{array}
\]
Example of partitioning

\[
\begin{array}{cccccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
6 & 5 & 13 & 10 & 8 & 3 & 2 & 11 \\
\end{array}
\]
Example of partitioning

\[
\begin{array}{cccccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
6 & 5 & 13 & 10 & 8 & 3 & 2 & 11 \\
\end{array}
\]

\[i \quad \rightarrow \quad j\]
Example of partitioning

\[
\begin{array}{cccccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
6 & 5 & 13 & 10 & 8 & 3 & 2 & 11 \\
6 & 5 & 3 & 10 & 8 & 13 & 2 & 11 \\
\end{array}
\]
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

\[ \begin{array}{cccccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
6 & 5 & 13 & 10 & 8 & 3 & 2 & 11 \\
6 & 5 & 3 & 10 & 8 & 13 & 2 & 11 \\
6 & 5 & 3 & 2 & 8 & 13 & 10 & 11 \\
\end{array} \]

\[ i \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad j \]
Example of partitioning
Example of partitioning
Pseudocode for quicksort

\[
\text{QUICKSORT}(A, p, r)
\]
\[
\text{if } p < r \text{ then } q \leftarrow \text{PARTITION}(A, p, r)
\]
\[
\text{QUICKSORT}(A, p, q-1)
\]
\[
\text{QUICKSORT}(A, q+1, r)
\]

Initial call: \text{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.
Worst-case of quicksort

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

\[ T(n) = T(0) + T(n-1) + \Theta(n) \]

\[ = \Theta(1) + T(n-1) + \Theta(n) \]

\[ = T(n-1) + \Theta(n) \]

\[ = \Theta(n^2) \text{ (arithmetic series)} \]
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

\[ T(n) = T(0) + T(n-1) + cn \]

\[ \Theta(1) \]
Worst-case recursion tree

\[ T(n) = T(0) + T(n-1) + cn \]

\[ \Theta \left( \sum_{k=1}^{n} k \right) = \Theta (n^2) \]
Worst-case recursion tree

\[ T(n) = T(0) + T(n-1) + cn \]

\[ h = n \]

\[ \Theta(1) \quad c(n-1) \]

\[ \Theta(1) \quad c(n-2) \]

\[ \Theta(1) \quad \cdots \]

\[ \Theta(1) \]

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

\[ T(n) = \Theta(n) + \Theta(n^2) \]

\[ = \Theta(n^2) \]
Best-case analysis

*(For intuition only!)*

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

\[
T(n) = 2T(n/2) + \Theta(n)
\]

\[
= \Theta(n \log n) \quad \text{(same as merge sort)}
\]

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

\[ T\left(\frac{1}{10}n\right) \quad cn \quad T\left(\frac{9}{10}n\right) \]
Analysis of “almost-best” case

\[
\begin{align*}
    & cn \\
    \frac{1}{10} & \quad \frac{9}{10} \\
    T\left(\frac{1}{100} n\right) & \quad T\left(\frac{9}{100} n\right) & \quad T\left(\frac{9}{100} n\right) \quad T\left(\frac{81}{100} n\right)
\end{align*}
\]
Analysis of “almost-best” case

\[ O(n) \text{ leaves} \]

\[ \Theta(1) \]

\[ \log_{10/9} n \]
Analysis of “almost-best” case

\[ T(n) \leq cn \log_{10/9} n + O(n) \]

\( \Theta(1) \) leaves

\( \Theta(n \log n) \)

Lucky!
More intuition

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

\[ L(n) = 2U(n/2) + \Theta(n) \quad \text{ lucky} \]
\[ U(n) = L(n - 1) + \Theta(n) \quad \text{ unlucky} \]

Solving:

\[ L(n) = 2(L(n/2 - 1) + \Theta(n/2)) + \Theta(n) \]
\[ = 2L(n/2 - 1) + \Theta(n) \]
\[ = \Theta(n \log n) \quad \text{Lucky!} \]

How can we make sure we are usually lucky?
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.
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

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

$$E[X_k] = \Pr\{X_k = 1\} = 1/n,$$ since all splits are equally likely, assuming elements are distinct.
Analysis (continued)

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

\[ = \sum_{k=0}^{n-1} X_k (T(k) + T(n - k - 1) + \Theta(n)). \]
Calculating expectation

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

Take expectations of both sides.
Calculating expectation

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

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

Linearity of expectation.
Calculating expectation

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

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

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

Independence of \(X_k\) from other random choices.
Calculating expectation

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

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

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

\[ = \frac{1}{n} \sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n} \sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n) \]

Linearity of expectation; \( E[X_k] = 1/n \).
Calculating expectation

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

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

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

\[
= \frac{1}{n} \sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n} \sum_{k=0}^{n-1} E[T(n - k - 1)] + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n)
\]

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

Summations have identical terms.
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

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

Substitute inductive hypothesis.
Substitution method

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

\[ \leq 2a \left( \frac{1}{2} n^2 \log n - \frac{1}{8} n^2 \right) + \Theta(n) \]

Use fact.
Substitution method

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

\[
\leq \frac{2a}{n} \left( \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \right) + \Theta(n)
\]

\[
= an \lg n - \left( \frac{an}{4} - \Theta(n) \right)
\]

Express as *desired – residual*. 
Substitution method

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

\[
= \frac{2a}{n} \left( \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \right) + \Theta(n)
\]

\[
= an \lg n - \left( \frac{an}{4} - \Theta(n) \right)
\]

\[
\leq an \lg n ,
\]

if \( a \) is chosen large enough so that \( an/4 \) 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.