Introduction to Algorithms
Chap 05-07

Quicksort
• Divide and conquer
• Partitioning
• Worst-case analysis
• Intuition
• Randomized quicksort
• Analysis

Hsin-Lung Wu, CSIE, NTPU
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 $\leq x$ ≤ elements in upper subarray.

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

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

3. **Combine:** Trivial.

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

\text{\textsc{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 \\
\text{then } & i \leftarrow i + 1 \\
\text{exchange } & A[i] \leftrightarrow A[j] \\
\text{exchange } & A[p] \leftrightarrow A[i] \\
\text{return } & i
\end{align*}

\text{Invariant: } x \quad \leq x \quad \geq x \quad ?

\begin{array}{c}
p \\
i \\
j \\
q
\end{array}

Running time = \textit{O}(n) \text{ for } n \text{ elements.}
Example of partitioning

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

\( i \quad j \)
Example of partitioning

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

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

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

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

6 5 3 2 8 13 10 11
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

2  5  3  6  8  13  10  11

\( i \)
Pseudocode for quicksort

QUICKSORT(A, p, r)

if \( p < r \)

then \( q \leftarrow \text{PARTITION}(A, p, r) \)

QUICKSORT(A, p, q–1)

QUICKSORT(A, q+1, r)

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.
Worst-case of quicksort

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

\[ T(n) = T(0) + T(n-1) + \Theta(n) \]
\[ = \Theta(1) + T(n-1) + \Theta(n) \]
\[ = T(n-1) + \Theta(n) \]
\[ = \Theta(n^2) \quad \text{(arithmetic series)} \]
Worst-case recursion tree

\[ T(n) = T(0) + T(n-1) + cn \]
Worst-case recursion tree

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

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

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

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

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

*(For intuition only!)*

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

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

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

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

\[ cn \]

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

\[ \Theta(1) \quad O(n) \text{ leaves} \quad \Theta(1) \]

\[ c_n \quad \frac{1}{10} \quad \frac{1}{100} \quad \frac{9}{100} \quad \frac{9}{10} \quad \frac{81}{100} \quad \log_{10/9} n \]
Analysis of “almost-best” case

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

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

\[ \Theta(n \log n) \]

\[ \Theta(1) \]

\[ \log_{10} n \]

\[ cn \]

\[ \frac{1}{10} \]

\[ \frac{9}{100} \]

\[ \frac{81}{100} \]

\[ \log_{10/9} n \]

\[ \frac{9}{10} \]

\[ \frac{9}{100} \]

\[ \frac{81}{100} \]

\[ \Theta(1) \]

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

\[ \Theta(1) \]

\[ \Theta(n \log n) \]

\[ \text{Lucky!} \]
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:

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

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; \( E[X_k] = \frac{1}{n} \).
Calculating expectation

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

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

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

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

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

Summations have identical terms.
Hairy recurrence

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

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

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

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

**Use fact:** \( \sum_{k=2}^{n-1} k \log k \leq \frac{1}{2} n^2 \log n - \frac{1}{8} n^2 \) (exercise).
Substitution method

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

Substitute inductive hypothesis.
Substitution method

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

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

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 \textit{desired} – \textit{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.