5. Divide-and-Conquer
Not in book: Quicksort
How? Quicksort

Divide:
- Choose e from input A
- Partition A into $A^-$, e, and $A^+$

Conquer:
- Quicksort($A^-$)
- Quicksort($A^+$)

Combine:
- Output sorted $A^-$, then e, then sorted $A^+$

Mergesort: easy divide, hard combine
Quicksort: hard divide, easy combine
Quicksort pseudo-code

Precondition: A an array of elements

Quicksort(A)
if |A| <= 1
    return A
Choose a splitter \( a_i \) from A
\( A^- = A^+ = {} \)
foreach element \( a_j \) of A do
    LI: \( A^- \cup A^+ \cup a_j = A \) and \( e \) in \( A^- \leq a_j \) and \( e \) in \( A^+ \geq j \)
    if \( a_j < a_i \) then
        Add \( a_j \) to \( A^- \)
    else
        Add \( a_j \) to \( A^+ \)
end foreach
Assert: \( e \) in \( A^- \leq a_j; \ e \) in \( A^+ \geq j \)
Quicksort(\( A^- \))
Quicksort(\( A^+ \))
Return \( A^- \) followed by \( a_i \) followed by \( A^+ \)

Precondition: Returns A in sorted order
Quicksort in-action

ALGORITHMS
Quicksort in-place pseudo-code

Precondition: $A[l..r]$ is sub-array of items

procedure Quicksort($A, l, r$)
if $l + 1 \geq r$
    return $A$
Choose a splitter $a_i$ from $A[l]..A[r]$
k = Partition($A, l, r, j$)
Quicksort($A, l, k-1$) \hspace{1cm} $A- = A[l..k-1]$
Quicksort($A, k+1, r$) \hspace{1cm} $A+ = A[k+1..r]$

Postcondition: $A[l..r]$ contains original elements in sorted order

Index where splitter was put
Partition pseudo-code

Precondition: A[l..r] is sub-array of items. j is in range [l..r]

function Partition(A, l, r, j)
    Swap(A[j], A[r])
    ll = rr = i = l-1;
    loop
        exit when i = r
        i = i+1;
        if A[i] < A[r] then
            Swap(A[rr], A[i])
            rr = rr + 1;
            ll = ll + 1;
        endloop
    Assert: A[l..r] contains original elements of A.
    Swap(A[r], A[rr]):
    Return k = rr

Postcondition: A[l..r] contains original elements of A.
In-place quicksort in action
Runtime analysis of Quicksort

What is $T(n)$?

Divide: Partition: $O(n)$

Conquer: $\max(\quad$

\begin{align*}
   & T(1) + T(n-2) \\
   & T(2) + T(n-3) \\
   & \ldots \\
   & T(n-2) + T(1) \\
\end{align*}

\)

Combine: $O(1)$

Worst case: $O(n^2)$

Worst-case input?
Quicksort

How to choose the splitter

- First
- Last
- Middle
- Best of three
- Random
- Best of best of three
Average-case complexity of Quicksort

Observations:

- After an element $a_i$ is chosen as a splitter value and compared to other elements in its range, it is **never compared again**.

- If two elements are **separated into different partitions**, they will never be compared to each other again.

If we look at a range of sorted elements $z_i..z_j$, $z_i$ and $z_j$ are compared to each other only if **$z_i$ or $z_j$** chosen as a splitter before **any elements between $z_i$ and $z_j$**.
Average-case complexity of Quicksort

Let $x$ and $y$ be two elements with $x \leq y$

- If Quicksort picks any splitter $z$ with $x \leq y \leq z$ before it picks $x$ or $y$, then it never compares $x$ to $y$

- Assume that it picks the splitters randomly from all candidates in an unpartition group

- If $x$ and $y$ are $k$ places apart in the final sorted order, the chance of picking $x$ or $y$ before picking $z$ between them is $\frac{2}{k+1}$

The farther apart two elements in the final order, the less likely they are to be compared.
Average-case complexity for Quicksort

“If x and y are k places apart in the final sorted order, the chance of picking x or y before picking z between them is \(2/(k+1)\)

<table>
<thead>
<tr>
<th>1 pair</th>
<th>n-1 apart</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 pairs</td>
<td>n-2 apart</td>
</tr>
<tr>
<td>3 pairs</td>
<td>n-3 apart</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>n-2 pairs</td>
<td>2 apart</td>
</tr>
<tr>
<td>n-1 pairs</td>
<td>1 apart</td>
</tr>
</tbody>
</table>

Expected time

\[
\begin{align*}
1 \times \left(\frac{2}{n}\right) \\
2 \times \left(\frac{2}{n-1}\right) \\
(n-2) \times \left(\frac{2}{3}\right) \\
(n-1) \times \left(\frac{2}{2}\right) \\
\end{align*}
\]

\[\leq \sum_{i=1}^{n} \sum_{k=1}^{n-i} \frac{2}{k+1} \leq \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k} \leq 2(\log n + 1) = O(n \log n)\]

Averaged across all ____ inputs, Quicksort takes time \(\theta(n \log n)\)
Lower bound for comparison-based sorting

Decision tree

Compare \( A_1, A_2 \)

\[
\begin{array}{c}
A_1 \leq A_2 \\
> A_1 > A_2
\end{array}
\]

\[
\begin{array}{c}
2 : 3 \\
1 : 3
\end{array}
\]

\[
\begin{array}{c}
< 1, 2, 3 > \\
1 : 3 \\
< 2, 1, 3 > \\
2 : 3
\end{array}
\]

\[
\begin{array}{c}
< 1, 3, 2 > \\
< 3, 1, 2 >
\end{array}
\]

\[
\begin{array}{c}
< 2, 3, 1 > \\
< 3, 2, 1 >
\end{array}
\]

\( \geq n! \) nodes

Sorted order is \( A_1, A_3, A_2 \)

Model ANY sorting alg based on compares as decision tree
Lower-bound for comparison based sorting

Min height of tree with $k$ leaves $= \log k$

Min height of tree with $n!$ leaves $= \log(n!)$

$= \Theta(n \log n)$
Lower-bound for comparison-based sorting

Any algorithm based on comparisons between elements is in \( \Omega(n \log n) \) time.

For any algorithm, there’s an instance that takes at least \( n \log n \)

Can there be algorithms with instances that sort in \( O(n) \) time?

Can there be algorithms that sort in \( O(n \log n) \) time for every instance?

Can there be algorithms that sort in \( O(n \log n) \) time for most instances?