We assume an array \( A \) of \( N \) integers to be sorted:

\( A[1], ..., A[N] \)

**Quick sort**

**Basic idea:**

- Pick pivot \( x \) (e.g., first element)
- Concatenate:
  - Elements smaller than \( x \)
  - Elements larger than \( x \)
- Recurse, recurse

**Better version:** inplace

Show for 7 8 5 2 1 9 5 4

**Analysis:**
- **Best case:** \( O(N \log(N)) \)
- **Worst case:** \( O(N^2) \)
- **Average case:**
  \( \approx 1.38 \frac{N}{\log_2(N)} + O(1) \)
  + Best among comparison-based algs
  + Locality (spatial and temporal)
  - Worst case \( O(N^2) \)
  - Not good for small sizes (empirical)

**Optimizations (Sedgwick 78):**
- Choose pivot = median (first, middle, last)
- Use other sorting algs for small sizes
- Choose multiple pivots (not worth it?)
**Merge sort**

Basic idea: cut in half

\[ \text{sort} \quad \text{sort} \quad \text{merge} \]

Show merging is \( O(N) \)

Analysis: \( C(N) = 2C(N/2) + O(N) \)

\[ \Rightarrow C(N) = O(N \log(N)) \]

+ locality (temporal and spatial)
- \( O(N) \) extra storage

Optimizations:
- other search for smaller problems
- divide into \( p \) chunks (\( N/p \) fits in cache, less overhead) \( \Rightarrow \) multi-way merge sort

**Multi-way Merge sort**

Basic idea: cut in multiple

\[ \text{sort} \quad \text{merge} \]

Merging:

\[ \begin{align*}
11 & \rightarrow [1] \\
5 & \rightarrow [5] \\
15 & \rightarrow [15] \\
20 & \rightarrow [20] \\
\text{sorted subtrees} & \rightarrow \\
& \text{propagate smallest element}
\end{align*} \]
Optimizations

- use $p$ as degree of freedom (e.g., $N/p$ and priority queue fit into cache)
- other search for smaller problems
- increase far-out of priority queue to match cache line size
  e.g.: cache line = 4 elements, then
  instead of
  do
  
  Next, two algorithms suitable for small sizes

Insertion Sort

Basic idea: move through $A$ and sort iteratively by swapping

Show code and an example

Analysis:

<table>
<thead>
<tr>
<th></th>
<th>best: $O(N)$ (already sorted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>worst:</td>
<td>$O(N^2)$ (reverse sorted)</td>
</tr>
<tr>
<td>average:</td>
<td>$O(N^2)$</td>
</tr>
<tr>
<td>general:</td>
<td>$O(N + d)$</td>
</tr>
<tr>
<td>$d = { (i, j) \mid i &lt; j$ and $A[i] &gt; A[j] }$</td>
<td></td>
</tr>
</tbody>
</table>

- fast on almost sorted lists
- bad average case
- simple, in place
Sorting Networks

Basic idea:

\[
\text{Inputs: } \{N_1, N_2, \ldots, N_N\} \quad \text{sorted}
\]

\[
\begin{array}{c}
\text{Comparators: } \begin{array}{c}
\begin{array}{c}
N_1 \\
N_2 \\
\vdots \\
N_N
\end{array}
\end{array}
\end{array}
\]

Show example

Constructions:
- Best known uses \( O(N \log N) \) comparators
- Useful constructions have \( O(N \log^2 N) \) comparators
  - Construction: (recursive)

\[
C(N) = 2C(N/2) + O(N \log N)
\]

\[
\Rightarrow C(N) = O(N \log^2 N)
\]

Analysis:
- Independent of input data, in place
- \( O(N \log^2 N) \) worst case
- \( \text{nil} \) best case

Optimizations:
- Schedule comparisons, e.g., for instruction level parallelism