Acknowledgement: The course slides are adapted from the slides prepared by R. Sedgewick and K. Wayne of Princeton University.
**Quicksort**

**Basic plan.**
- **Shuffle** the array.
- **Partition** so that, for some $j$
  - entry $a[j]$ is in place
  - no larger entry to the left of $j$
  - no smaller entry to the right of $j$
- **Sort** each piece recursively.

### Example

<table>
<thead>
<tr>
<th>input</th>
<th>QUICKSORTEXAMPLE</th>
<th>shuffle</th>
<th>partitioning item</th>
<th>partition</th>
<th>sort left</th>
<th>sort right</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick</td>
<td></td>
<td>KRATELEPUIMQCSXOS</td>
<td></td>
<td>ECAIEK</td>
<td>ACEEIKLPUTMQRXOS</td>
<td>ACEEIKLMOPQRSTUX</td>
<td>ACEEIKLMOPQRSTUX</td>
</tr>
</tbody>
</table>

**Sir Charles Antony Richard Hoare**

1980 Turing Award
Quicksort partitioning

Repeat until $i$ and $j$ pointers cross.

- Scan $i$ from left to right so long as $a[i] < a[lo]$.
- Scan $j$ from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.

stop $i$ scan because $a[i] \geq a[lo]$
Quicksort partitioning

Repeat until i and j pointers cross.

• Scan i from left to right so long as \( a[i] < a[lo] \).
• Scan j from right to left so long as \( a[j] > a[lo] \).
• Exchange \( a[i] \) with \( a[j] \).
Quicksort partitioning

Repeat until i and j pointers cross.
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$. 
Quicksort partitioning

**Repeat until i and j pointers cross.**

- Scan i from left to right so long as \( a[i] < a[\text{lo}] \).
- Scan j from right to left so long as \( a[j] > a[\text{lo}] \).
- Exchange \( a[i] \) with \( a[j] \).

---

\[
\begin{array}{cccccccccccccccc}
K & R & A & T & E & L & E & P & U & I & M & Q & C & X & O & S \\
\uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\
\text{lo} & i & j & & & & & & & & & & & & \\
\end{array}
\]

*stop j scan and exchange \( a[i] \) with \( a[j] \)*
Quicksort partitioning

Repeat until $i$ and $j$ pointers cross.

- Scan $i$ from left to right so long as $a[i] < a[lo]$.
- Scan $j$ from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$. 
Quicksort partitioning

Repeat until i and j pointers cross.
• Scan i from left to right so long as \(a[i] < a[lo]\).
• Scan j from right to left so long as \(a[j] > a[lo]\).
• Exchange \(a[i]\) with \(a[j]\).
Quicksort partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as \(a[i] < a[lo]\).
- Scan j from right to left so long as \(a[j] > a[lo]\).
- Exchange \(a[i]\) with \(a[j]\).

\[K\ C\ A\ T\ E\ L\ E\ P\ U\ I\ M\ Q\ R\ X\ O\ S\]

\[\uparrow\ lo\ \uparrow\ i\ \uparrow\ j\]

stop i scan because \(a[i] \geq a[lo]\)
Quicksort partitioning

Repeat until $i$ and $j$ pointers cross.

- Scan $i$ from left to right so long as $a[i] < a[lo]$.  
- Scan $j$ from right to left so long as $a[j] > a[lo]$.  
- Exchange $a[i]$ with $a[j]$.
Quicksort partitioning

Repeat until $i$ and $j$ pointers cross.

- Scan $i$ from left to right so long as $a[i] < a[lo]$.
- Scan $j$ from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
Quicksort partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as a[i] < a[lo].
- Scan j from right to left so long as a[j] > a[lo].
- Exchange a[i] with a[j].

\[\begin{array}{cccccccccccccccc}
K & C & A & T & E & L & E & P & U & I & M & Q & R & X & O & S \\
\hline
\uparrow & \uparrow & \uparrow & i & & & & & & & & & & j & \\
lo & i & & & & & & & & & & & & \\
\end{array}\]

stop j scan and exchange a[i] with a[j]
Quicksort partitioning

Repeat until $i$ and $j$ pointers cross.

- Scan $i$ from left to right so long as $a[i] < a[lo]$.
- Scan $j$ from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
Quicksort partitioning

Repeat until \( i \) and \( j \) pointers cross.

- Scan \( i \) from left to right so long as \( a[i] < a[lo] \).
- Scan \( j \) from right to left so long as \( a[j] > a[lo] \).
- Exchange \( a[i] \) with \( a[j] \).
Quicksort partitioning

Repeat until $i$ and $j$ pointers cross.

- Scan $i$ from left to right so long as $a[i] < a[lo]$.
- Scan $j$ from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.

stop $i$ scan because $a[i] \geq a[lo]$
Quicksort partitioning

Repeat until $i$ and $j$ pointers cross.

- Scan $i$ from left to right so long as $a[i] < a[lo]$.
- Scan $j$ from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
Quicksort partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
Quicksort partitioning

Repeat until \( i \) and \( j \) pointers cross.

- Scan \( i \) from left to right so long as \( a[i] < a[lo] \).
- Scan \( j \) from right to left so long as \( a[j] > a[lo] \).
- Exchange \( a[i] \) with \( a[j] \).

stop \( j \) scan and exchange \( a[i] \) with \( a[j] \)
Repeat until $i$ and $j$ pointers cross.

- Scan $i$ from left to right so long as $a[i] < a[lo]$.
- Scan $j$ from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
Quicksort partitioning

Repeat until i and j pointers cross.

• Scan i from left to right so long as a[i] < a[lo].
• Scan j from right to left so long as a[j] > a[lo].
• Exchange a[i] with a[j].

stop i scan because a[i] >= a[lo]
Quicksort partitioning

Repeat until $i$ and $j$ pointers cross.

- Scan $i$ from left to right so long as $a[i] < a[lo]$.
- Scan $j$ from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.

stop j scan because $a[j] \leq a[lo]$
Quicksort partitioning

Repeat until \( i \) and \( j \) pointers cross.

- Scan \( i \) from left to right so long as \( a[i] < a[lo] \).
- Scan \( j \) from right to left so long as \( a[j] > a[lo] \).
- Exchange \( a[i] \) with \( a[j] \).

When pointers cross.

- Exchange \( a[lo] \) with \( a[j] \).
Quicksort partitioning

Repeat until i and j pointers cross.

• Scan i from left to right so long as a[i] < a[lo].
• Scan j from right to left so long as a[j] > a[lo].
• Exchange a[i] with a[j].

When pointers cross.

• Exchange a[lo] with a[j].

partitioned!
Quicksort partitioning

Basic plan.
- Scan \(i\) from left for an item that belongs on the right.
- Scan \(j\) from right for an item that belongs on the left.
- Exchange \(a[i]\) and \(a[j]\).
- Repeat until pointers cross.

Partitioning trace (array contents before and after each exchange)
Quicksort: Java code for partitioning

```java
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo, j = hi + 1;
    while (true) {
        while (less(a[++i], a[lo]))
            if (i == hi) break;

        while (less(a[lo], a[--j]))
            if (j == lo) break;

        if (i >= j) break;
        exch(a, i, j);
    }
    exch(a, lo, j);
    return j;
}
```

Quicksort partitioning overview:

- **Before**:
  - `i` and `j` are not yet partitioned.
  - `lo` and `hi` define the range.

- **During**:
  - Items to the left of `i` are less than or equal to the pivot.
  - Items to the right of `j` are greater than or equal to the pivot.

- **After**:
  - The pivot is now in its correct position.
  - The range is correctly partitioned.

The code snippet above implements the partitioning function for quicksort. It iterates until `i` and `j` cross, swapping elements to move the pivot into its correct position.
Quicksort: Java implementation

```java
public class Quick {
    private static int partition(Comparable[] a, int lo, int hi) {
        /* see previous slide */
    }

    public static void sort(Comparable[] a) {
        StdRandom.shuffle(a);
        sort(a, 0, a.length - 1);
    }

    private static void sort(Comparable[] a, int lo, int hi) {
        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
}
```

shuffle needed for performance guarantee (stay tuned)
| lo | j | hi | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
|----|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0  | 5 | 15 | E C A I E K L P U T M Q R X O S |
| 0  | 3 | 4  | E C A E I K L P U T M Q R X O S |
| 0  | 2 | 2  | A C E E I K L P U T M Q R X O S |
| 0  | 0 | 1  | A C E E I K L P U T M Q R X O S |
| 1  | 1 | 1  | A C E E I K L P U T M Q R X O S |
| 4  | 4 | 4  | A C E E I K L P U T M Q R X O S |
| 6  | 6 | 15 | A C E E I K L P U T M Q R X O S |
| 7  | 9 | 15 | A C E E I K L M O P T Q R X U S |
| 7  | 7 | 8  | A C E E I K L M O P T Q R X U S |
| 8  | 8 | 8  | A C E E I K L M O P T Q R X U S |
| 10 | 13| 15 | A C E E I K L M O P S Q R T U X |
| 10 | 12| 12 | A C E E I K L M O P R Q S T U X |
| 10 | 11| 11 | A C E E I K L M O P Q R S T U X |
| 10 | 10| 10 | A C E E I K L M O P Q R S T U X |
| 14 | 14| 15 | A C E E I K L M O P Q R S T U X |
| 15 | 15| 15 | A C E E I K L M O P Q R S T U X |

Quicksort trace (array contents after each partition)
Quicksort animation

50 random items

http://www.sorting-algorithms.com/quick-sort
Quickstart: implementation details

**Partitioning in-place.** Using an extra array makes partitioning easier (and stable), but is not worth the cost.

**Terminating the loop.** Testing whether the pointers cross is a bit trickier than it might seem.

**Staying in bounds.** The \((j == \ell_0)\) test is redundant (why?), but the \((i == h_1)\) test is not.

**Preserving randomness.** Shuffling is needed for performance guarantee.

**Equal keys.** When duplicates are present, it is (counter-intuitively) better to stop on keys equal to the partitioning item's key.
Quicksort: empirical analysis

Running time estimates:
• Home PC executes $10^8$ compares/second.
• Supercomputer executes $10^{12}$ compares/second.

<table>
<thead>
<tr>
<th></th>
<th>insertion sort (N)</th>
<th>mergesort (N log N)</th>
<th>quicksort (N log N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>computer</td>
<td>thousand</td>
<td>million</td>
<td>billion</td>
</tr>
<tr>
<td>home</td>
<td>instant</td>
<td>2.8 hours</td>
<td>317 years</td>
</tr>
<tr>
<td>super</td>
<td>instant</td>
<td>1 second</td>
<td>1 week</td>
</tr>
<tr>
<td></td>
<td>thousand</td>
<td>million</td>
<td>billion</td>
</tr>
<tr>
<td>home</td>
<td>instant</td>
<td>1 second</td>
<td>18 min</td>
</tr>
<tr>
<td>super</td>
<td>instant</td>
<td>instant</td>
<td>instant</td>
</tr>
<tr>
<td></td>
<td>thousand</td>
<td>million</td>
<td>billion</td>
</tr>
<tr>
<td>home</td>
<td>instant</td>
<td>1 second</td>
<td>12 min</td>
</tr>
<tr>
<td>super</td>
<td>instant</td>
<td>instant</td>
<td>instant</td>
</tr>
</tbody>
</table>

Lesson 1. Good algorithms are better than supercomputers.
Lesson 2. Great algorithms are better than good ones.
**Quicksort: best-case analysis**

**Best case.** Number of compares is $\sim N \lg N$. 

<table>
<thead>
<tr>
<th>lo</th>
<th>j</th>
<th>hi</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>H</td>
<td>A</td>
<td>C</td>
<td>B</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>D</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>initial values</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>H</td>
<td>A</td>
<td>C</td>
<td>B</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>D</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>random shuffle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>J</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>10</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
</tbody>
</table>
Worst case. Number of compares is $\sim \frac{1}{2} N^2$. 

### Quicksort: worst-case analysis

<table>
<thead>
<tr>
<th>lo</th>
<th>j</th>
<th>hi</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initial values</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>random shuffle</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
</tbody>
</table>
**Proposition.** The average number of compares $C_N$ to quicksort an array of $N$ distinct keys is $\sim 2N \ln N$ (and the number of exchanges is $\sim \frac{1}{3} N \ln N$).

**Pf 1.** $C_N$ satisfies the recurrence $C_0 = C_1 = 0$ and for $N \geq 2$:

\[
C_N = (N + 1) + \left( \frac{C_0 + C_{N-1}}{N} \right) + \left( \frac{C_1 + C_{N-2}}{N} \right) + \ldots + \left( \frac{C_{N-1} + C_0}{N} \right)
\]

- Multiply both sides by $N$ and collect terms:

\[
NC_N = N(N + 1) + 2(C_0 + C_1 + \ldots + C_{N-1})
\]

- Subtract this from the same equation for $N - 1$:

\[
NC_N - (N - 1)C_{N-1} = 2N + 2C_{N-1}
\]

- Rearrange terms and divide by $N(N + 1)$:

\[
\frac{C_N}{N + 1} = \frac{C_{N-1}}{N} + \frac{2}{N + 1}
\]
QuickSort: average-case analysis

- Repeatedly apply above equation:

\[
\frac{C_N}{N+1} = \frac{C_{N-1}}{N} + \frac{2}{N+1}
\]

\[
= \frac{C_{N-2}}{N-1} + \frac{2}{N} + \frac{2}{N+1}
\]

\[
= \frac{C_{N-3}}{N-2} + \frac{2}{N-1} + \frac{2}{N} + \frac{2}{N+1}
\]

\[
= \frac{2}{3} + \frac{2}{4} + \frac{2}{5} + \cdots + \frac{2}{N+1}
\]

- Approximate sum by an integral:

\[
C_N = 2(N+1) \left( \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \cdots + \frac{1}{N+1} \right)
\]

\[
\sim 2(N+1) \int_3^{N+1} \frac{1}{x} \, dx
\]

- Finally, the desired result:

\[
C_N \sim 2(N+1) \ln N \approx 1.39N \ln N
\]
Quicksort: summary of performance characteristics

**Worst case.** Number of compares is quadratic.
- \(N + (N - 1) + (N - 2) + \ldots + 1 \sim \frac{1}{2} N^2.\)
- More likely that your computer is struck by lightning bolt.

**Average case.** Number of compares is \(\sim 1.39 N \lg N.\)
- 39% more compares than mergesort.
- But faster than mergesort in practice because of less data movement.

**Random shuffle.**
- Probabilistic guarantee against worst case.
- Basis for math model that can be validated with experiments.

**Caveat emptor.** Many textbook implementations go quadratic if array
- Is sorted or reverse sorted.
- Has many duplicates (even if randomized!)
Insertion sort small subarrays.

- Even quicksort has too much overhead for tiny subarrays.
- Cutoff to insertion sort for \( \approx 10 \) items.
- Note: could delay insertion sort until one pass at end.

```java
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo + CUTOFF - 1) {
        Insertion.sort(a, lo, hi);
        return;
    }
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```
Quicksort: practical improvements

Median of sample.
- Best choice of pivot item = median.
- Estimate true median by taking median of sample.
- Median-of-3 (random) items.

~ 12/7  N ln N compares (slightly fewer)
~ 12/35 N ln N exchanges (slightly more)

```java
private static void sort(Comparable[] a, int lo, int hi)
{
    if (hi <= lo) return;

    int m = medianOf3(a, lo, lo + (hi - lo)/2, hi);
    swap(a, lo, m);

    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```
Quicksort with median-of-3 and cutoff to insertion sort: visualization
Duplicate keys

Often, purpose of sort is to bring items with equal keys together.

- Sort population by age.
- Find collinear points.
- Remove duplicates from mailing list.
- Sort job applicants by college attended.

Typical characteristics of such applications.

- Huge array.
- Small number of key values.
Duplicate keys

Mergesort with duplicate keys.
Always between $\frac{1}{2} N \lg N$ and $N \lg N$ compares.

Quicksort with duplicate keys.
- Algorithm goes **quadratic** unless partitioning stops on equal keys!
- 1990s C user found this defect in `qsort()`.

Several textbook and system implementation also have this defect.
Duplicate keys: the problem

**Mistake.** Put all items equal to the partitioning item on one side.

**Consequence.** $\sim \frac{1}{2} N^2$ compares when all keys equal.

\[
\begin{array}{ccccccc}
B & C & C & C & & & \\
& & & & & & A \\
& & & & & & A
\end{array}
\quad
\begin{array}{cccccccc}
\end{array}
\]

**Recommended.** Stop scans on items equal to the partitioning item.

**Consequence.** $\sim N \lg N$ compares when all keys equal.

\[
\begin{array}{ccccccc}
B & C & C & B & C & B & \\
& & & & & & A \\
& & & & & & A
\end{array}
\quad
\begin{array}{cccccccc}
\end{array}
\]

**Desirable.** Put all items equal to the partitioning item in place.

\[
\begin{array}{ccccccc}
B & C & C & C & & & \\
& & & & & & A \\
& & & & & & A
\end{array}
\quad
\begin{array}{cccccccc}
\end{array}
\]
Goal. Partition array into 3 parts so that:
• Entries between \(\lt\) and \(\gt\) equal to partition item \(v\).
• No larger entries to left of \(\lt\).
• No smaller entries to right of \(\gt\).

Dutch national flag problem. [Edsger Dijkstra]
• Conventional wisdom until mid 1990s: not worth doing.
• New approach discovered when fixing mistake in C library \texttt{qsort()}.
• Now incorporated into \texttt{qsort()} and Java system sort.
Dijkstra 3-way partitioning

- Let $v$ be partitioning item $a[lo]$.
- Scan $i$ from left to right.
  - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both $lt$ and $i$
  - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement $gt$
  - $(a[i] == v)$: increment $i$

-invariant
Dijkstra 3-way partitioning

- Let $v$ be partitioning item $a[lo]$.
- Scan $i$ from left to right.
  - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both $lt$ and $i$
  - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement $gt$
  - $(a[i] == v)$: increment $i$

\[
\begin{array}{cccccccccccc}
\end{array}
\]

\[
\begin{array}{cccc}
  lt & i & gt \\
  \downarrow & \downarrow & \downarrow \\
\end{array}
\]

\[
\begin{array}{cccc}
  \lt & =V & \text{gray} & >V \\
  \uparrow & \uparrow & \uparrow \\
  lt & i & gt \\
\end{array}
\]

invariant
Dijkstra 3-way partitioning

- Let $v$ be partitioning item $a[lo]$. 
- Scan $i$ from left to right.
  - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both $lt$ and $i$
  - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement $gt$
  - $(a[i] == v)$: increment $i$
Dijkstra 3-way partitioning

- Let \( v \) be partitioning item \( a[lo] \).
- Scan \( i \) from left to right.
  - \((a[i] < v)\): exchange \( a[lt] \) with \( a[i] \) and increment both \( lt \) and \( i \)
  - \((a[i] > v)\): exchange \( a[gt] \) with \( a[i] \) and decrement \( gt \)
  - \((a[i] == v)\): increment \( i \)
Let $v$ be partitioning item $a[lo]$.

Scan $i$ from left to right.

- $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both $lt$ and $i$
- $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement $gt$
- $(a[i] == v)$: increment $i$

**Dijkstra 3-way partitioning**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>P</th>
<th>Z</th>
<th>W</th>
<th>P</th>
<th>P</th>
<th>V</th>
<th>P</th>
<th>D</th>
<th>P</th>
<th>C</th>
<th>Y</th>
<th>X</th>
</tr>
</thead>
</table>

**Invariant**

```
< v    = v  [ ]  > v

lt  i  gt
```
Dijkstra 3-way partitioning

- Let \( v \) be partitioning item \( a[lo] \).
- Scan \( i \) from left to right.
  - \((a[i] < v)\): exchange \( a[lt] \) with \( a[i] \) and increment both \( lt \) and \( i \)
  - \((a[i] > v)\): exchange \( a[gt] \) with \( a[i] \) and decrement \( gt \)
  - \((a[i] == v)\): increment \( i \)

\[\begin{array}{cccccccccccc}
\end{array}\]

\[\begin{array}{c}
l_t \\
\downarrow \\
< v \\
\uparrow \\
l_t
\end{array} \quad \begin{array}{c}
i \\
\downarrow \\
= v \\
\uparrow \\
i
\end{array} \quad \begin{array}{c}
\text{gray cell} \\
\end{array} \quad \begin{array}{c}
g_t \\
\downarrow \\
> v \\
\uparrow \\
g_t
\end{array}\]

\text{invariant}
Let \( v \) be partitioning item \( a[lo] \).

Scan \( i \) from left to right.

- \( (a[i] < v) \): exchange \( a[lt] \) with \( a[i] \) and increment both \( lt \) and \( i \)
- \( (a[i] > v) \): exchange \( a[gt] \) with \( a[i] \) and decrement \( gt \)
- \( (a[i] == v) \): increment \( i \)
Dijkstra 3-way partitioning

- Let \( v \) be partitioning item \( a[lo] \).
- Scan \( i \) from left to right.
  - \( (a[i] < v) \): exchange \( a[lt] \) with \( a[i] \) and increment both \( lt \) and \( i \)
  - \( (a[i] > v) \): exchange \( a[gt] \) with \( a[i] \) and decrement \( gt \)
  - \( (a[i] == v) \): increment \( i \)

\[\begin{array}{cccccccccccccccc}
\end{array}\]

\[\text{invariant}\]

\[\begin{array}{cccc}
<&V & =V & \text{[grey]} & >V \\
\uparrow & \uparrow & \uparrow & \uparrow \\
lt & i & gt \\
\end{array}\]
Let \( v \) be partitioning item \( a[lo] \).

Scan \( i \) from left to right.

- \((a[i] < v)\): exchange \( a[lt] \) with \( a[i] \) and increment both \( lt \) and \( i \)
- \((a[i] > v)\): exchange \( a[gt] \) with \( a[i] \) and decrement \( gt \)
- \((a[i] == v)\): increment \( i \)
Dijkstra 3-way partitioning

- Let \( v \) be partitioning item \( a[lo] \).
- Scan \( i \) from left to right.
  - \( (a[i] < v) \): exchange \( a[lt] \) with \( a[i] \) and increment both \( lt \) and \( i \)
  - \( (a[i] > v) \): exchange \( a[gt] \) with \( a[i] \) and decrement \( gt \)
  - \( (a[i] == v) \): increment \( i \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>P</th>
<th>P</th>
<th>P</th>
<th>V</th>
<th>P</th>
<th>D</th>
<th>W</th>
<th>Y</th>
<th>Z</th>
<th>X</th>
</tr>
</thead>
</table>

**Invariant**

\[
\begin{array}{c|c|c|c|c|c}
< v & = v & \text{gray} & > v \\
\downarrow & \uparrow & \downarrow & \uparrow \\
lt & i & gt \\
\end{array}
\]
Dijkstra 3-way partitioning

- Let $v$ be partitioning item $a[lo]$.
- Scan $i$ from left to right.
  - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both $lt$ and $i$
  - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement $gt$
  - $(a[i] == v)$: increment $i$
Dijkstra 3-way partitioning

- Let $v$ be partitioning item $a[lo]$.
- Scan $i$ from left to right.
  - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both $lt$ and $i$
  - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement $gt$
  - $(a[i] == v)$: increment $i$
Dijkstra 3-way partitioning

- Let $v$ be partitioning item $a[lo]$.
- Scan $i$ from left to right.
  - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both $lt$ and $i$
  - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement $gt$
  - $(a[i] == v)$: increment $i$

![Diagram of 3-way partitioning](image)
Let $v$ be partitioning item $a[lo]$. 

Scan $i$ from left to right.
- $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both $lt$ and $i$
- $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement $gt$
- $(a[i] == v)$: increment $i$
Dijkstra 3-way partitioning

- Let \( v \) be partitioning item \( a[lo] \).
- Scan \( i \) from left to right.
  - \( (a[i] < v) \): exchange \( a[lt] \) with \( a[i] \) and increment both \( lt \) and \( i \)
  - \( (a[i] > v) \): exchange \( a[gt] \) with \( a[i] \) and decrement \( gt \)
  - \( (a[i] == v) \): increment \( i \)
Dijkstra 3-way partitioning algorithm

3-way partitioning.
• Let v be partitioning item a[lo].
• Scan i from left to right.
  - a[i] less than v: exchange a[lt] with a[i] and increment both lt and i
  - a[i] greater than v: exchange a[gt] with a[i] and decrement gt
  - a[i] equal to v: increment i

Most of the right properties.
• In-place.
• Not much code.
• Linear time if keys are all equal.
Dijkstra's 3-way partitioning: trace

<table>
<thead>
<tr>
<th>lt</th>
<th>i</th>
<th>gt</th>
<th>a[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>11</td>
<td>R B W W R W B R R W B R</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>11</td>
<td>R B W W R W B R R W B R</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>11</td>
<td>B R W W R W B R R W B R</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>10</td>
<td>B R R W R W B R R W B W</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>10</td>
<td>B R R W R W B R R W B W</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>9</td>
<td>B R R B R W B R R W W W</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>9</td>
<td>B B R R R W B R R W W W</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>9</td>
<td>B B R R R W B R R W W W</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>B B R R R W B R R W W W</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>7</td>
<td>B B R R R R R B R W W W W</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>7</td>
<td>B B R R R R B R W W W W</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>7</td>
<td>B B B R R R R R W W W W</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>7</td>
<td>B B B R R R R R R W W W W</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>7</td>
<td>B B B R R R R R R R W W W W</td>
</tr>
</tbody>
</table>

3-way partitioning trace (array contents after each loop iteration)
3-way quicksort: Java implementation

private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int lt = lo, gt = hi;
    Comparable v = a[lo];
    int i = lo;
    while (i <= gt) {
        int cmp = a[i].compareTo(v);
        if (cmp < 0) exch(a, lt++, i++);
        else if (cmp > 0) exch(a, i, gt--);
        else i++;
    }
    sort(a, lo, lt - 1);
    sort(a, gt + 1, hi);
}

before
<table>
<thead>
<tr>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>l0</td>
</tr>
</tbody>
</table>

during
<table>
<thead>
<tr>
<th>&lt;v</th>
<th>=v</th>
<th>&gt;v</th>
</tr>
</thead>
<tbody>
<tr>
<td>l1</td>
<td>i</td>
<td>g1</td>
</tr>
</tbody>
</table>

after
<table>
<thead>
<tr>
<th>&lt;v</th>
<th>=v</th>
<th>&gt;v</th>
</tr>
</thead>
<tbody>
<tr>
<td>l0</td>
<td>l1</td>
<td>g1</td>
</tr>
</tbody>
</table>
3-way quicksort: visual trace

equal to partitioning element
<table>
<thead>
<tr>
<th></th>
<th>inplace?</th>
<th>stable?</th>
<th>worst</th>
<th>average</th>
<th>best</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>selection</td>
<td>✔</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N exchanges</td>
</tr>
<tr>
<td>insertion</td>
<td>✔</td>
<td>✔</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>use for small $N$ or partially ordered</td>
</tr>
<tr>
<td>shell</td>
<td>✔</td>
<td>?</td>
<td>?</td>
<td>N</td>
<td></td>
<td>tight code, subquadratic</td>
</tr>
<tr>
<td>merge</td>
<td>✔</td>
<td>✔</td>
<td>N $lg , N$</td>
<td>N $lg , N$</td>
<td>N $lg , N$</td>
<td>$N \log , N$ guarantee, stable</td>
</tr>
<tr>
<td>quick</td>
<td>✔</td>
<td>N</td>
<td>2 $N \ln , N$</td>
<td>N $lg , N$</td>
<td></td>
<td>$N \log , N$ probabilistic guarantee</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>fastest in practice</td>
</tr>
<tr>
<td>3-way quick</td>
<td>✔</td>
<td>N</td>
<td>2 $N \ln , N$</td>
<td>N</td>
<td></td>
<td>improves quicksort in presence of duplicate keys</td>
</tr>
<tr>
<td>???</td>
<td>✔</td>
<td>✔</td>
<td>N $lg , N$</td>
<td>N $lg , N$</td>
<td>N $lg , N$</td>
<td>holy sorting grail</td>
</tr>
</tbody>
</table>