QuickSort

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.

Sir Charles Antony Richard Hoare
1980 Turing Award
Shuffling

- Shuffling is the process of rearranging an array of elements randomly.
- A good shuffling algorithm is unbiased, where every ordering is equally likely.

- e.g. the Fisher–Yates shuffle (aka. the Knuth shuffle)

http://bl.ocks.org/mbostock/39566aca95eb03ddd526
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 $a[j]$.
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 $a[j]$.

<table>
<thead>
<tr>
<th>K</th>
<th>R</th>
<th>A</th>
<th>T</th>
<th>E</th>
<th>L</th>
<th>E</th>
<th>P</th>
<th>U</th>
<th>I</th>
<th>M</th>
<th>Q</th>
<th>C</th>
<th>X</th>
<th>O</th>
<th>S</th>
</tr>
</thead>
</table>

\[ \uparrow \quad \uparrow \quad \uparrow \]

\begin{align*}
\text{i} & \quad \text{lo} \\
\text{j} & \quad \end{align*}

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 \( a[j] \).

\[
\begin{array}{cccccccccccc}
K & C & A & T & E & L & E & P & U & I & M & Q \\
\uparrow & \ & \ & \ & \ & \ & \ & \ & \ & \ & \ \\
lo & \ & \ & \ & \ & \ & \ & \ & \ & \ & \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
R & X & O & S \\
\ & \ & \ & \ & \ & \ & \ & \ & \ & \\
\ & \ & \ & \ & \ & \ & \ & \ & \ & \\
\uparrow & \ & \ & \ & \ & \ & \ & \ & \ & \\
j & \ & \ & \ & \ & \ & \ & \ & \ & \\
\end{array}
\]

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 $a[j]$.

<table>
<thead>
<tr>
<th>K</th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>E</th>
<th>L</th>
<th>E</th>
<th>P</th>
<th>U</th>
<th>I</th>
<th>M</th>
<th>Q</th>
<th>R</th>
<th>X</th>
<th>O</th>
<th>S</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>
</tr>
</tbody>
</table>

$\uparrow$

$lo$

$\uparrow$

$i$

$\uparrow$

$j$
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]
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 \(a[j]\).

\[ \text{stop j scan and exchange } a[i] \text{ with } a[j] \]
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 \(a[j]\).

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

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 \( a[j] \).

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.


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)
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;
}
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);
    }
}
## Quicksort trace

<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>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Q</td>
<td>U</td>
<td>I</td>
<td>C</td>
<td>K</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>K</td>
<td>R</td>
<td>A</td>
<td>T</td>
<td>E</td>
<td>L</td>
<td>E</td>
<td>P</td>
<td>U</td>
<td>I</td>
<td>M</td>
<td>Q</td>
<td>C</td>
<td>X</td>
<td>O</td>
<td>S</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>15</td>
<td>E</td>
<td>C</td>
<td>A</td>
<td>I</td>
<td>E</td>
<td>K</td>
<td>L</td>
<td>P</td>
<td>U</td>
<td>T</td>
<td>M</td>
<td>Q</td>
<td>R</td>
<td>X</td>
<td>O</td>
<td>S</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>4</td>
<td>E</td>
<td>C</td>
<td>A</td>
<td>E</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>P</td>
<td>U</td>
<td>T</td>
<td>M</td>
<td>Q</td>
<td>R</td>
<td>X</td>
<td>O</td>
<td>S</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>P</td>
<td>U</td>
<td>T</td>
<td>M</td>
<td>Q</td>
<td>R</td>
<td>X</td>
<td>O</td>
<td>S</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>P</td>
<td>U</td>
<td>T</td>
<td>M</td>
<td>Q</td>
<td>R</td>
<td>X</td>
<td>O</td>
<td>S</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>P</td>
<td>U</td>
<td>T</td>
<td>M</td>
<td>Q</td>
<td>R</td>
<td>X</td>
<td>O</td>
<td>S</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>P</td>
<td>U</td>
<td>T</td>
<td>M</td>
<td>Q</td>
<td>R</td>
<td>X</td>
<td>O</td>
<td>S</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>15</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>P</td>
<td>U</td>
<td>T</td>
<td>M</td>
<td>Q</td>
<td>R</td>
<td>X</td>
<td>O</td>
<td>S</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>15</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>T</td>
<td>Q</td>
<td>R</td>
<td>X</td>
<td>U</td>
<td>S</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>T</td>
<td>Q</td>
<td>R</td>
<td>X</td>
<td>U</td>
<td>S</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td></td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>T</td>
<td>Q</td>
<td>R</td>
<td>X</td>
<td>U</td>
<td>S</td>
</tr>
<tr>
<td>10</td>
<td>13</td>
<td>15</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>S</td>
<td>Q</td>
<td>R</td>
<td>T</td>
<td>U</td>
<td>X</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>12</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>Q</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>X</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>11</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>X</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td></td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>X</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>15</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>X</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td></td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>X</td>
</tr>
</tbody>
</table>

| result | 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
Quicksort: 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 \((j == lo)\) test is redundant (why?), but the \((i == hi)\) 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 ($N^2$)</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>
</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 $\sim N \lg N$.

Each partitioning process splits the array exactly in half.

<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></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>0</td>
<td>7</td>
<td>14</td>
<td>D</td>
<td>A</td>
<td>C</td>
<td>B</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>0</td>
<td>0</td>
<td>0</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>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>4</td>
<td>5</td>
<td>6</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>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>4</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>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</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>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</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>E</td>
<td>F</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>E</td>
<td>F</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>8</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>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</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>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>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>12</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>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>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>
<td></td>
</tr>
</tbody>
</table>
**Worst case. Number of compares is** \( \sim \frac{1}{2} N^2 \).

*One of the subarrays is empty for every partition.*

```
<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>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>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>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>
</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. $C_N$ 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:

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

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

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

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

- Approximate sum by an integral:

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

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

- Finally, the desired result:

\[
C_N \sim 2(N+1) \ln N \approx 1.39 \, N \, \ln N
\]
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 $\sim N \lg N$.
- 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!)
Proposition. Quicksort is an in-place sorting algorithm.
Pf.

- Partitioning: constant extra space.
- Depth of recursion: logarithmic extra space (with high probability).

Proposition. Quicksort is not stable.
Pf.

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>B1</td>
<td>C1</td>
<td>C2</td>
<td>A1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>B1</td>
<td>C1</td>
<td>C2</td>
<td>A1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>B1</td>
<td>A1</td>
<td>C2</td>
<td>C1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>A1</td>
<td>B1</td>
<td>C2</td>
<td>C1</td>
</tr>
</tbody>
</table>
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.

```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);
}
```

~ 12/7  \(N \ln N\) compares (slightly fewer)
~ 12/35 \(N \ln N\) exchanges (slightly more)
Quicksort with median-of-3 and cutoff to insertion sort: visualization
Selection

Goal. Given an array of $N$ items, find the $k^{th}$ largest.
Ex. Min ($k = 0$), max ($k = N - 1$), median ($k = N / 2$).

Applications.
• Order statistics.
• Find the "top $k$"

Use theory as a guide.
• Easy $N \log N$ upper bound. How?
• Easy $N$ upper bound for $k = 1, 2, 3$. How?
• Easy $N$ lower bound. Why?

Which is true?
• $N \log N$ lower bound? is selection as hard as sorting?
• $N$ upper bound? is there a linear-time algorithm for each $k$?
Quick-select

Partition array so that:

- Entry \( a[j] \) is in place.
- No larger entry to the left of \( j \).
- No smaller entry to the right of \( j \).

Repeat in one subarray, depending on \( j \); finished when \( j \) equals \( k \).

```java
public static Comparable select(Comparable[] a, int k) {
    StdRandom.shuffle(a);
    int lo = 0, hi = a.length - 1;
    while (hi > lo) {
        int j = partition(a, lo, hi);
        if (j < k) lo = j + 1;
        else if (j > k) hi = j - 1;
        else return a[k];
    }
    return a[k];
}
```
Quick-select: mathematical analysis

Proposition. Quick-select takes linear time on average.

Pf sketch.

• Intuitively, each partitioning step splits array approximately in half:
  \[ N + N/2 + N/4 + \ldots + 1 \sim 2N \] compares.

• Formal analysis similar to quicksort analysis yields:
  \[
  C_N = 2N + k \ln (N/k) + (N-k) \ln (N/(N-k))
  \]

(2 + 2 ln 2) N to find the median

Remark. Quick-select uses \( \frac{1}{2} N^2 \) compares in the worst case, but (as with quicksort) the random shuffle provides a probabilistic guarantee.
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 \texttt{qsort()}. 

\begin{center}
\begin{tikzpicture}
  \node (stop) {STOP ONE EQUAL KEYS};
  \node (top) [very near start of stop] {TOP} edge[->] node {swap} (stop.west);
  \node (one) [very near start of stop] {ONE} edge[->] node {if we don't stop on equal keys} (stop.west);
  \node (qual) [very near start of stop] {EQUAL KEYS} edge[->] node {if we stop on equal keys} (stop.west);
\end{tikzpicture}
\end{center}

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.

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

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

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

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

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

\[
\begin{array}{cccccccc}
\end{array}
\]
3-way partitioning

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 `qsort()`.
- Now incorporated into `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 \( i \)

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c} l & t & i & & & & & & & & gt \\
\hline
\end{array} \]

\text{Invariant}

\[ \begin{array}{c|c|c|c|c} <v & =v & \_\_\_\_\_\_\_\_ & >v \\
\hline
l & t & i & gt
\end{array} \]
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 diagram]
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 \( i \)

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

invariant

\[
\begin{array}{cccc}
  \text{<v} & \text{=}v & \text{---} & \text{>v} \\
  \uparrow & \uparrow & \uparrow & \\
  \text{lt} & \text{i} & \text{gt} & \\
\end{array}
\]
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 $a[10]$.
- 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

<table>
<thead>
<tr>
<th>$&lt;v$</th>
<th>$=v$</th>
<th>$&gt;$v</th>
</tr>
</thead>
<tbody>
<tr>
<td>$lt$</td>
<td>$i$</td>
<td>$gt$</td>
</tr>
</tbody>
</table>
Dijkstra 3-way partitioning

- Let $v$ be partitioning item $a[lo]$.
- Scan $i$ from left to right.
  - $(a[i] < v)$: exchange $a[l.l]$ 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.

- \((a[i] \lt v)\): exchange \( a[lt] \) with \( a[i] \) and increment both \( lt \) and \( i \)
- \((a[i] \gt v)\): 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

\[
\begin{array}{cccccccccccccc}
\text{A} & \text{B} & \text{C} & \text{P} & \text{W} & \text{P} & \text{P} & \text{V} & \text{P} & \text{D} & \text{P} & \text{Y} & \text{Z} & \text{X}
\end{array}
\]

\begin{center}
\begin{tabular}{cccc}
\text{lt} & \text{i} & \text{gt}
\end{tabular}
\end{center}

\text{invariant}

\[
\begin{array}{cccc}
< V & = V & \text{\ldots} & > V
\end{array}
\]
Let $v$ be partitioning item $a[10]$.

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
Dijkstra 3-way partitioning

- Let $v$ be partitioning item $a[10]$.
- 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]

**Invariant**

<table>
<thead>
<tr>
<th>&lt;v</th>
<th>=v</th>
<th>___</th>
<th>&gt;v</th>
</tr>
</thead>
<tbody>
<tr>
<td>lt</td>
<td>i</td>
<td>gt</td>
<td></td>
</tr>
</tbody>
</table>
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 $a[10]$.
- 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 showing 3-way partitioning with indexes and elements]
• Let $v$ be partitioning item $a[10]$.
• 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 $i$

\[\begin{align*}
\text{A} & \quad \text{B} & \quad \text{C} & \quad \text{D} & \quad \text{P} & \quad \text{P} & \quad \text{P} & \quad \text{P} & \quad \text{P} & \quad \text{P} & \quad \text{V} & \quad \text{W} & \quad \text{Y} & \quad \text{Z} & \quad \text{X}
\end{align*}\]

invariant

\[\begin{array}{cccc}
<\text{v} & =\text{v} & \text{---} & >\text{v}
\end{array}\]

\[\begin{array}{c}
\text{lt} \\
\downarrow
\end{array}\]

\[\begin{array}{c}
\text{i} \\
\uparrow
\end{array}\]

\[\begin{array}{c}
\text{gt}
\end{array}\]

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**

![Diagram of Dijkstra 3-way partitioning](image-url)
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

3-way partitioning trace (array contents after each loop iteration)
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);
}
3-way quicksort: visual trace

equal to partitioning element
# Sorting summary

<table>
<thead>
<tr>
<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^2/2)</td>
<td>(N^2/2)</td>
<td>(N^2/2)</td>
<td>(N) exchanges</td>
</tr>
<tr>
<td>insertion</td>
<td>✔ ✔</td>
<td>(N^2/2)</td>
<td>(N^2/4)</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>tight code, subquadratic</td>
</tr>
<tr>
<td>merge</td>
<td>✔ ✔</td>
<td>(N \log N)</td>
<td>(N \log N)</td>
<td>(N \log N)</td>
<td>(N \log N) guarantee, stable</td>
</tr>
<tr>
<td>quick</td>
<td>✔</td>
<td>(N^2/2)</td>
<td>(N \log N)</td>
<td>(N \log N)</td>
<td>(N \log N) probabilistic guarantee</td>
</tr>
<tr>
<td>3-way quick</td>
<td>✔ ✔</td>
<td>(N^2/2)</td>
<td>(N \log N)</td>
<td>(N)</td>
<td>improves quicksort in presence of duplicate keys</td>
</tr>
<tr>
<td>???</td>
<td>✔ ✔</td>
<td>(N \log N)</td>
<td>(N \log N)</td>
<td>(N \log N)</td>
<td>holy sorting grail</td>
</tr>
</tbody>
</table>