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

Quicksort example

<table>
<thead>
<tr>
<th>input</th>
<th>Q U I C K S O R T E X A M P L E</th>
</tr>
</thead>
<tbody>
<tr>
<td>shuffle</td>
<td>K R A T E L E P U I M Q C X O S</td>
</tr>
<tr>
<td>partition</td>
<td>E C A I E K L P U T M Q R X O S</td>
</tr>
<tr>
<td>sort left</td>
<td>A C E E I K L P U T M Q R X O S</td>
</tr>
<tr>
<td>sort right</td>
<td>A C E E I K L M O P Q R S T U X</td>
</tr>
<tr>
<td>result</td>
<td>A C E E I K L M O P Q R S T U X</td>
</tr>
</tbody>
</table>
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 \( a[i] \geq a[lo] \)
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]
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] >= 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 $a[j]$.

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

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

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

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

\[ text\]

\[ \text{stop i scan because } a[i] \geq a[lo] \]
Quick sort 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:**
  - Array: \( \leq V \) \( V \) \( \geq V \)
  - Indices: \( lo \), \( hi \)

- **During:**
  - Array: \( \leq V \) \( \leq V \) \( V \) \( \geq V \)
  - Indices: \( i \), \( j \), \( lo \), \( hi \)

- **After:**
  - Array: \( \leq V \) \( V \) \( \geq V \)
  - Indices: \( lo \), \( j \), \( hi \)
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 (array contents after each partition)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
QUICKSORTEXAMPLE
KRATELEPUIMQCXOS
0 5 15 ECAIEKLPUMQRXOS
0 3 4 ECAEIKLPUTMQRXOS
0 2 2 ACEEIKLPUTMQRXOS
0 0 1 ACEEIKLPUTMQRXOS
1 1 1 ACEEIKLPUTMQRXOS
4 4 ACEEIKLPUTMQRXOS
6 6 15 ACEEIKLPUTMQRXOS
7 9 15 ACEEIKLMOPTQRXUS
7 7 8 ACEEIKLMOPTQRXUS
8 8 ACEEIKLMOPTQRXUS
10 13 15 ACEEIKLMOPSQRTUX
10 12 12 ACEEIKLMOPRQSTUX
10 11 11 ACEEIKLMOQPQRSTUX
10 10 ACEEIKLMOQPQRSTUX
14 14 15 ACEEIKLMOQPQRSTUX
15 15 ACEEIKLMOQPQRSTUX

Quicksort trace (array contents after each partition)
Quicksort animation

50 random items

http://www.sorting-algorithms.com/quick-sort
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>computer</th>
<th>insertion sort ($N^2$)</th>
<th>mergesort ($N \log N$)</th>
<th>quicksort ($N \log N$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></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.
### Best case. Number of compares is $\sim N \log N$.

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>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>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>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>
</tbody>
</table>
**Quicksort: worst-case analysis**

**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>initial values</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
<th>O</th>
</tr>
</thead>
<tbody>
<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>
</tr>
<tr>
<td>0   0   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   1   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   2   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   3   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   4   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   5   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   6   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   7   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   8   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   9   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  10  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  11  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  12  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  13  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  14  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>
</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:

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

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

- Finally, the desired result:

\[
C_N \sim 2(N+1) \ln N \approx 1.39N \lg N
\]
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 \textit{quadratic} if array
- Is sorted or reverse sorted.
- Has many duplicates (even if randomized!)
Quicksort: practical improvements

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);
}
```
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
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.

```
B A A B A B B B C C C C A A A A A A A A A A A A A
```

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

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

```
B A A B A B B B C C B C B B A A A A A A A A A A A A A
```

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

```
A A A B B B B B B B C C C C A A A A A A A A A A A A A
```
**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
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 \( i \)

\[
\begin{array}{cccccccccccccccc}
\text{P} & A & B & X & W & \text{P} & \text{P} & V & \text{P} & D & P & C & Y & Z \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
\text{lt} & i & \text{gt} \\
\end{array}
\]

**Invariant**

\[
\begin{array}{cccc}
<\text{v} & =\text{v} & \text{P} & >\text{v} \\
\uparrow & \uparrow & \uparrow \\
\text{lt} & i & \text{gt} \\
\end{array}
\]
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 \( i \)

\[<v \quad =v \quad \quad \quad \quad \quad \quad >v\]

\(\downarrow \quad \downarrow \quad \quad \quad \quad \quad \quad \downarrow\)

\(\downarrow \quad \downarrow \quad \quad \quad \quad \quad \quad \downarrow\)

\(A \quad P \quad B \quad X \quad W \quad P \quad P \quad V \quad P \quad D \quad P \quad C \quad Y \quad Z\)
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

```
< v  = v  [ ]  > v
  ↓    ↓    ↓ 
 lt   i   gt
```
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 \( i \)

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

\textit{Invariant}

\[
\begin{array}{cccc}
< v & = v & \text{Gray} & > v \\
\text{lt} & i & \text{gt} \\
\end{array}
\]
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 $i$

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

\[\begin{array}{llll}
\text{lt} & i & \text{gt}
\end{array}\]

\[\begin{array}{c}
< v \\
= v
\end{array}\]

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

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

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

\[\begin{array}{c}
> v
\end{array}\]

\[\begin{array}{c}
\text{invariant}
\end{array}\]
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 $i$
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 partitioning](image)

**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></td>
<td>$gt$</td>
</tr>
</tbody>
</table>
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 diagram]

**Invariant**

- $a[lt] \leq \ldots \leq a[i] \leq \ldots \leq a[gt]$
- $\text{lt} < i < \text{gt}$
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 \( i \)

\[
\begin{array}{cccccccccccccccc}
\text{lt} & i & \text{gt} \\
\end{array}
\]

**Invariant**

\[
\begin{array}{cccc}
< v & = v & \_ & > v \\
\text{lt} & i & \text{gt} \\
\end{array}
\]
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 \( i \)

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c} 
\text{lt} & \downarrow & \text{i} & \downarrow & \text{gt} & \downarrow \\
\end{array} \]

\[ \begin{array}{c|c|c|c} 
\text{invatant} \\
< v & = v & \text{mark} & > v \\
\text{lt} & \uparrow & \text{i} & \uparrow & \text{gt} & \uparrow \\
\end{array} \]
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 $a[10]$.
• 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 $i$

\[<v \quad =v \quad \underline{\text{lo}} \quad \underline{\text{hi}} \quad >v\]

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

\[\text{lt} \quad \downarrow \quad i \quad \downarrow \downarrow \quad \text{gt}\]
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 $i$
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></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></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>✔️</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>?</td>
<td>(N)</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></td>
<td>(N^{2}/2)</td>
<td>(2N\ln N)</td>
<td>(N\lg N)</td>
<td>(N\log N) probabilistic guarantee</td>
</tr>
<tr>
<td>3-way quick</td>
<td>✔️</td>
<td></td>
<td>(N^{2}/2)</td>
<td>(2N\ln N)</td>
<td>(N)</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>