**Quicksort**

We will discuss Quicksort, a sorting algorithm that, on average, is the fastest known comparison sort.

Comparison sort sorts sets of Comparable class, where we can perform only two kinds of operations on the elements:

- for two elements, $a$ and $b$, we can find the result of the test $a < b$ or $a \leq b$;
- we can copy content of a memory location containing an element to another location.

Clearly, usually we can perform other operations as well. For this reason we may have faster sorting algorithms for more specific classes of elements. But whenever we can sort at all, we can apply a comparison sorting, so the program libraries usually contain Quicksort.

Quicksort is faster than Heapsort, and in practice it is also faster than Mergesort. Moreover, unlike MergeSort, it uses very little additional memory.
The concept of Quicksort

We will discuss the concept of Randomized Quicksort.

To sort array fragment \(<A, p, r>\),
\[
\text{Qsort}(A, p, r) \\
\{ \text{ if } (r-p > 1) \} \\
\quad \text{pick at random } q \text{ from the set } \{p, p+1, \ldots, r\} \\
\quad x = A[q] \\
\quad \text{rearrange the content of } <A, p, r> \text{ so that for some } i \\
\quad \quad y \in <A, p, i> \rightarrow y \leq x \\
\quad \quad A[i] = x \\
\quad \quad y \in <A, p, i+1, q> \rightarrow x \leq y \\
\text{Qsort}(A, p, i) \\
\text{Qsort}(A, i+1, q) \\
\}
\]
Example:

Qsort(A, 0, 15)
pick 5, $x = 29$

<table>
<thead>
<tr>
<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>13</td>
<td>4</td>
<td>17</td>
<td>23</td>
<td>92</td>
<td>29</td>
<td>35</td>
<td>15</td>
<td>51</td>
<td>32</td>
<td>74</td>
<td>19</td>
<td>22</td>
<td>91</td>
<td>5</td>
</tr>
</tbody>
</table>

Qsort(A, 0, 8) Qsort(A, 9, 15)
pick 2, $x = 17$ pick 13, $x = 74$

<table>
<thead>
<tr>
<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>13</td>
<td>4</td>
<td>17</td>
<td>23</td>
<td>15</td>
<td>19</td>
<td>22</td>
<td>5</td>
<td>29</td>
<td>92</td>
<td>35</td>
<td>51</td>
<td>32</td>
<td>74</td>
<td>91</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<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>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>4</td>
<td>15</td>
<td>5</td>
<td>17</td>
<td>23</td>
<td>19</td>
<td>22</td>
<td>35</td>
<td>51</td>
<td>32</td>
<td>74</td>
<td>92</td>
<td>91</td>
</tr>
</tbody>
</table>
Worst case analysis

Let $T(n)$ be the worst case number of comparisons if $r - p = n$.

The key $x = A[q]$ is called the pivot.

To perform the rearrangement, we compare all keys other than the original $A[q]$ with the pivot, $n - 1$ comparisons.

The resulting subproblems do not include the pivot. Let $i$ be the final position of the pivot.

$$T(n) = n - 1 + T(i) + T(n - i - 1) \leq$$

inductive hypothesis: $T(m) = m(m - 1)/2$

$$n - 1 + i(i - 1)/2 + (n - i - 1)(n - i - 2)/2 \leq$$

assume $i = 0$

$$n - 1 + (n - 1)(n - 2)/2 =$$

$$n(n - 1)/2.$$  

It is indeed possible that we always pick the minimum or the maximum, in which case inequalities become equalities.

Indeed, it is very troublesome if all keys are equal.
Average case analysis

We assume that no two keys are equal. Then the eventual position of the pivot is uniformly random, so the average number of comparisons is the average of \( n \) equally probable cases: linearity of expectation.

We got the following recurrence relation:

\[
T(0) = T(1) = 0 \text{ and for } n > 1
\]

\[
T(n) = n - 1 + \frac{1}{n} \sum_{i=0}^{n-1} [T(i) + T(n - i - 1)] =
\]

\[
n - 1 + \frac{1}{n} \sum_{i=0}^{n-1} T(i) + \frac{1}{n} \sum_{i=0}^{n-1} T(n - i - 1) =
\]

\[
n - 1 + \frac{2}{n} \sum_{i=0}^{n-1} T(i)
\]

In the supplement at the end of these notes we show an exact solution to this recurrence.
In the proof of $T(n) = \Theta(n \log n)$, the easier inequality to prove is

$$T(n) < 2n \ln n.$$ 

Inductive basis: check for $n = 2$.

Inductive step:

$$n - 1 + \frac{2}{n} \sum_{i=2}^{n-1} T(i) <$$

$$n - 1 + \frac{4}{n} \sum_{i=2}^{n-1} n \ln n <$$

$$n - 1 + \frac{4}{n} \int_{i=2}^{n} x \ln x \, dx <$$

$$n - 1 + \frac{4}{n} \left[ \frac{1}{2} x^2 \ln x - \frac{1}{4} x^2 \right]^{n}_{2} <$$

$$n - 1 + \frac{4}{n} \left( \frac{1}{2} n^2 \ln n - \frac{1}{4} n^2 \right) =$$

$$n - 1 + 2n \ln n - n < 2n \ln n.$$
QuickSort, details

We need a function $\text{Partition}(A, p, q)$ that returns $i$ such that

\[ p \leq i < q \]

for $x$ randomly chosen from the fragment $< A. p. q >$ this fragment is rearranged so that

- for $y$ in $< A, p, i >$ we have $y \leq x$,
- $A[i] = x$,
- for $y$ in $< A, i + 1, q >$ we have $y \geq x$.

Given that, we can write QuickSort as follows:

```c
QuickSort(A, p, q)
{
    if (q-p < 2)
        return;
    i = Partition(A,p,q);
    QuickSort(A,p,i);
    QuickSort(A,i+1,q);
}
```
Before we describe *Partition*, we can address the following point: what is the worst case memory overhead of Quicksort and is it easy to improve it?

The memory overhead is basically the recurrence stack. In the worst case, the depth can be $n$. However, we can use tail recursion paradigm to reduce this worst case to $\log_2 n$.

First, we rewrite Quicksort and then we convert:

```plaintext
Quicksort(A,p,q)  
{   if (q-p < 2) 
    return; 
    i = Partition(A,p,q); 
    j = i+1; 
    if (p-i > q-i-1) 
      j = p, 
      p = i+1, 
      i = q, 
      q = p-1; 
    Quicksort(A,p,i); 
    Quicksort(A,j,q); 
} 
```

Each time we make an explicit recursive call, the size of the fragment to be sorted is less than half of fragment sorted by the parent call. Therefore the depth of the recurrence is below $\log_2 n$. 
One of the problems addressed by skillfully written *Partition* is the problem of duplicates. Suppose that all entries of the fragment to be sorted are equal. Our specification of the *Partition* allows it to make any split of the given fragment, in particular, the worst case split is possible. Hoare developed an elegant way of avoiding such possibility without slowing the execution down much.

This is Hoare partition:

\[
\text{Partition}(A,p,q) \\
\begin{array}{l}
\{ \\
\quad r = \text{random pick from } \{p, p+1, \ldots, q-1\}; \\ \\
\quad \text{exchange } A[p] \text{ and } A[r]; \\ \\
\quad x = A[p]; \\ \\
\quad i = p-1; \\ \\
\quad j = q; \\ \\
\quad \text{while } (1) \{ \\ \\
\qquad \text{do } \\
\qquad \quad j--; \\ \\
\qquad \text{while } (A[j] > x); \\ \\
\qquad \text{do } \\
\qquad \quad i++; \\ \\
\qquad \text{while } (A[i] < x); \\ \\
\qquad \text{if } (i < j) \\ \\
\qquad \quad \text{exchange } A[i] \text{ and } A[j]; \\ \\
\qquad \text{else} \\ \\
\qquad \quad \text{return } j+1; \\ \\
\quad \} \\\n\}
\]
This partition has several nice properties, but they come at a small price: the returned position does not have to contain the pivot, it is merely guaranteed to contain something that is at least as large as the pivot. Thus we cannot exclude this position from the further sorting.

However, this partition works properly in a slightly different algorithm:

```plaintext
Quicksort(A, p, q)
{
    if (q-p < 2)
        return;
    k = Partition(A, p, q);
    Quicksort(A, p, k);
    Quicksort(A, k, q);
}
```

Question: is it possible that `Partition` makes segmentation faults?

When the loop that modifies $i$ terminates, we have $A[i]$ at least $x$. Because such an entry exists, first time this loop must terminate with $i < q$. Before subsequent executions of this loop, we assure that $i < j$ and $A[j] ≥ x$, hence $i$ may increase at most to the current value of $j$.

A symmetric argument assures that $j$ is never smaller than $p$.

We can also show that we do divide $< A, p, q >$ properly.
Supplement: exact average case analysis

The recurrence relation for the average number of comparisons performed by Quicksort, \( T(0) = 0 \) and
\[
T(n) = n - 1 + \frac{2}{n} \sum_{i=0}^{n-1} T(i)
\]
can be changed into equivalent form that does not contain \( \sum \). In particular, if \( n > 1 \) then
\[
T(n) = n - 1 + \frac{2}{n} \sum_{i=0}^{n-2} T(i) + \frac{2}{n} T(n - 1) =
\]
\[
n - 1 + \frac{n-1}{n} \frac{2}{n-1} \sum_{i=0}^{n-2} T(i) + \frac{2}{n} T(n - 1) =
\]
\[
n - 1 + \frac{n-1}{n} [T(n - 1) - (n - 2)] + \frac{2}{n} T(n - 1) =
\]
\[
2 \frac{n-1}{n} + \frac{n+1}{n} T(n - 1).
\]
Harmonic numbers are defined with the recurrence: \( H_0 = 0 \), and \( H_n = H_{n-1} + \frac{1}{n} \) for \( n > 0 \). Equivalently, \( H_n = \sum_{i=1}^{n} \frac{1}{i} \).

We show by induction that \( T(n) = 2(n + 1)(H_n - 2) + 4 \). Inductive step:
\[
T(n) = 2 \frac{n-1}{n} + \frac{n+1}{n} T(n - 1) =
\]
\[
2 \frac{n-1}{n} + \frac{n+1}{n} [2n(H_{n-1} - 2) + 4] =
\]
\[
2 \frac{n-1}{n} + \frac{n+1}{n} [2n(H_n - \frac{1}{n} - 2) + 4] =
\]
\[
2(n + 1)(H_n - 2) + 2 \frac{n-1}{n} + \frac{n+1}{n} [-2n \frac{1}{n} + 4] =
\]
\[
2(n + 1)(H_n - 2) + 4.
\]