Mergesort

(guest lecture by Michael Langer)

Recall "Selection Sort"

\[
i = n-1
\]

while \( i > 0 \) {

\[
\text{indexOfMax} = \text{findMax}(a[i])
\]

// finds the location

// of max in

// \( a[0], \ldots, a[i] \)

\[
\text{Swap}(a, i, \text{indexOfMax})
\]

\[
i = i - 1
\]

swap does nothing

The number of comparisons need to find the max

\[
i = (n-1) + (n-2) + (n-3) + \ldots + 1
\]

\[
= \frac{n(n-1)}{2}
\]

Typical computers today can perform \( \sim 10^9 \) comparisons per second (GHz)

<table>
<thead>
<tr>
<th>( n ) \text{ (size of problem)}</th>
<th>( n^2 ) \text{ (number of comparisons)}</th>
<th>time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 10^3 ) (( \sim \text{KB} ))</td>
<td>( 10^6 )</td>
<td>( 10^3 )</td>
</tr>
<tr>
<td>( 10^6 ) (( \sim \text{MB} ))</td>
<td>( 10^{12} )</td>
<td>( 10^3 ) (seconds)</td>
</tr>
<tr>
<td>( 10^9 ) (( \sim \text{GB} ))</td>
<td>( 10^{18} )</td>
<td>( 10^9 ) (centuries)</td>
</tr>
</tbody>
</table>

Today we will look at a sorting algorithm that is significantly faster: "merge sort."

It is a recursive algorithm.
mergeSort (list) 

if (list.size == 1) 
    return list 
else 
    partition list into two approximately equal size lists l1, l2 
    l1 ← mergeSort (l1) 
    l2 ← mergeSort (l2) 
    return merge (l1, l2)

// Partition list into two approximately equal size lists l1, l2. 
// getElements(low, high) returns a sublist 

mid ← list.size - 1 / 2 

l1 ← list.getElements(0, mid) 
l2 ← list.getElements(mid + 1, size - 1)

merge (l1, l2) ? // two sorted lists 

l ← empty list 
while l1 and l2 are not empty? 
    if l1.get(0) < l2.get(0)
        l.addLast (l1.remove(0))
    else
        l.addLast (l2.remove(0))

while l1 is not empty 
    l.addLast (l1.remove(0))
while l2 is not empty 
    l.addLast (l2.remove(0))

return l

Solution
Ordering of method calls

Call Stack (see numbers on previous slide)

How many operations are required to mergesort a list of \( n \) elements?

( SKETCH OF IDEA )

\[
\begin{array}{c|c|c|c}
 n & \log n & n\log n & n^2 \\
\hline
10^3 & \approx 10 & 10^4 & 10^6 \\
10^6 & \approx 20 & 2 \times 10^7 & 10^{12} \\
10^9 & \approx 30 & 3 \times 10^{10} & 10^{18} \\
\end{array}
\]

Wow! minutes centuries