COMP250: More Recursion examples. Merge sort.

Jérôme Waldispühl
School of Computer Science
McGill University

Based on slides from (Snoeyink, 2004)
Sorting problem

Problem: Given a list of $n$ elements from a totally ordered universe, rearrange them in ascending order.

<table>
<thead>
<tr>
<th>Songs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NAME</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

Classical problem in computer science with many different algorithms (bubble sort, merge sort, quick sort, etc.)
Insertion sort

6 3 1 5 2 4

3 6 1 5 2 4

1 3 6 5 2 4

1 3 5 6 2 4

1 2 3 5 6 4

1 2 3 4 5 6
Insertion sort

$n$ elements already sorted

New element to sort

$n+1$ elements sorted
Insertion sort

For $i \leftarrow 1$ to $\text{length}(A) - 1$
  $j \leftarrow i$
  while $j > 0$ and $A[j-1] > A[j]$
    swap $A[j]$ and $A[j-1]$
    $j \leftarrow j - 1$
  end while
end for

• Iterative method to sort objects.
• Relatively slow, we can do better using a recursive approach!
Divide and Conquer

Recursive in structure

- **Divide** the problem into sub-problems that are similar to the original but smaller in size

- **Conquer** the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.

- **Combine** the solutions to create a solution to the original problem
An Example: Merge Sort

**Sorting Problem:** Sort a sequence of $n$ elements into non-decreasing order.

- **Divide:** Divide the $n$-element sequence to be sorted into two subsequences of $n/2$ elements each.
- **Conquer:** Sort the two subsequences recursively using merge sort.
- **Combine:** Merge the two sorted subsequences to produce the sorted answer.
Idea: If we have 2 lists L and R already sorted, we can easily (i.e. quickly) build a sorted list A with all elements of L and R.
Merge sort (principle)

- Unsorted array A with \( n \) elements
- Split A in half \( \rightarrow \) 2 arrays L and R with \( n/2 \) elements
- Sort L and R
- Merge the two sorted arrays L and R

Base case: Stop the recursion when the array is of size 1.
Why? Because the array is already sorted!
Merge Sort – Example

Original Sequence

Sorted Sequence

18 26 32 6 43 15 9 1

1 6 9 15 18 26 32 43

18 26 32 6 43 15 9 1

6 18 26 32 1

18 26 32 6 43 15 9 1

18 26 32 6 43 15 9 1

18 26 32 6 43 15 9 1

18 26 32 6 43 15 9 1

18 26 32 6 43 15 9 1

18 26 32 6 43 15 9 1

18 26 32 6 43 15 9 1

18 26 32 6 43 15 9 1

18 26 32 6 43 15 9 1

18 26 32 6 43 15 9 1
Merge-Sort (A, p, r)

**INPUT:** a sequence of $n$ numbers stored in array $A$

**OUTPUT:** an ordered sequence of $n$ numbers

```
MergeSort (A, p, r)   // sort A[p..r] by divide & conquer
1   if p < r
2      then q ← ⌊(p+r)/2⌋
3      MergeSort (A, p, q)
4      MergeSort (A, q+1, r)
5      Merge (A, p, q, r) // merges A[p..q] with A[q+1..r]
```

Initial Call: MergeSort(A, 1, n)
Input: Array containing sorted subarrays $A[p..q]$ and $A[q+1..r]$.


**Sentinels**, to avoid having to check if either subarray is fully copied at each step.
Running time of Merge Sort

Running time $T(n)$ of Merge Sort:

- **Base case:** constant time $c$
- **Divide:** computing the middle takes constant time $c'$
- **Conquer:** solving 2 subproblems takes $2T(n/2)$
- **Combine:** merging $n$ elements takes $k \cdot n$ (i.e. time proportional to the number of elements to merge)

- **Total:**

  \[
  T(n) = \begin{cases} 
  c & \text{if } n = 1 \\
  2T(n/2) + k \cdot n + c' & \text{if } n > 1 
  \end{cases}
  \]

**Example:** Let $c=1$, $c'=1$ and $k=1$

<table>
<thead>
<tr>
<th>$n$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>...</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(n)$</td>
<td>1</td>
<td>5</td>
<td>15</td>
<td>39</td>
<td>95</td>
<td>223</td>
<td>511</td>
<td>...</td>
<td>?</td>
</tr>
</tbody>
</table>
Running time of Merge Sort