Lecture 15 – Merge sort
Reading: KT Sections 5.1 and 5.2

Partial content of these slides have been obtained from the official lecture slides that accompany the textbook. A complete set of slides can be found at: http://www.cs.princeton.edu/~wayne/kleinberg-tardos/

Sorting

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

Selection Sort

• Selection Sort orders values by repetitively putting a particular value into its final position

• The algorithm
  1. find the smallest value in the list
  2. switch it with the value in the first position
  3. find the next smallest value in the list
  4. switch it with the value in the second position
  5. repeat until all values are in their proper places

Search Visualizer (https://visualgo.net/sorting)

Insertion Sort

• Insertion sort orders a list of values by repetitively inserting a particular value into a sorted subset of the list

• The algorithm
  1. consider the first item to be a sorted sublist of length 1
  2. insert the second item in the sorted sublist, shifting the first item if needed
  3. insert the third item into the sorted sublist, shifting the other items as needed
  4. repeat until all values inserted into their proper positions

(3) is sorted sublist. Consider 9.
Shift nothing, insert 9: (3, 9) are sorted. Consider 6.
Shift 9, insert 6: (3, 6, 9) are sorted. Consider 1.
Shift 3, 6, 9, insert 1: (1, 3, 6, 9) sorted. Consider 2.
Shift 3, 6, 9, insert 2: (1, 2, 3, 6, 9) sorted.
Bubble Sort

- **Bubble sort** orders a list of values by repetitively comparing neighboring elements and swapping their positions if necessary

- The algorithm
  1. scan the list, exchanging adjacent elements if they are not in relative order, bubbles highest value to the top
  2. scan the list again, bubbling up the second highest value
  3. repeat until all elements are in proper order

Merge Sort

- **Merge sort** orders a list of values by recursively dividing the list in half until each sublist has one element, then recombining

- The algorithm
  1. divide the list into two roughly equal parts
  2. recursively divide each part in half, continuing until a part contains only one element
  3. merge the two parts into one sorted list
  4. continue to merge parts as the recursion unfolds

Divide and Conquer

- **Divide-and-conquer.**
  - Divide up problem into several subproblems.
  - Solve each subproblem recursively.
  - Combine solutions to subproblems into overall solution.

Most common usage:
- Divide problem of size $n$ into two subproblems of the same kind of size $n/2$ in **linear time**.
- Solve two subproblems recursively.
- Combine two solutions into overall solution in **linear time**.

Consequence:
- **Brute force:** $O(n^2)$.
- **Divide-and-conquer:** $O(n \log n)$.

Merge sort – Decomposition step

![Diagram of merge sort decomposition step]
mergeSort

mergeSort(A, min, max)
{
   if (min < max) {
      mid = (min + max) / 2;
      mergeSort(data, min, mid);
      mergeSort(data, mid + 1, max);
      merge(data, min, mid, max);
   }
}

Merging

Coil. Combine two sorted lists, \( a \) and \( b \) into a sorted whole \( c \).
- Scan \( a \) and \( b \) from left to right.
- Compare \( a_i \) and \( b_j \).
- If \( a_i \leq b_j \), append \( a_i \) to \( c \) (no larger than any remaining element in \( a \)).
- If \( a_i > b_j \), append \( b_j \) to \( c \) (smaller than every remaining element in \( a \)).

Recurrence Equations

The worst-case running time of Merge-Sort on inputs of size \( n \), denoted \( T(n) \) satisfies:

\[
T(n) = \begin{cases} 
  c & \text{if } n=1 \\
  2T(n/2) + cn & \text{if } n > 1 
\end{cases}
\]

Solution. \( T(n) \) is \( \Theta(n \log_{2} n) \).
## Sort Efficiencies

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time Complexity</th>
<th>Space Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quicksort</td>
<td>O(n log n)</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Mergesort</td>
<td>O(n log n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>O(n^2)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>O(n^2)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Tree Sort</td>
<td>O(n log n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Shell Sort</td>
<td>O(n log^2 n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Bubble Sort</td>
<td>O(n^2)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Counting Sort</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Cutoff Sort</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
</tbody>
</table>

bigscheatsheet.com