Lecture 13 – 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/

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

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

Divide and Conquer

- **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
Merge sort – Merging 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

Recurrence Equations

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

$$T(n) = \begin{cases} 
  c & \text{if } n=1 \\
  2^\frac{n}{2} + cn & \text{if } n > 1 
\end{cases}$$

Solution. $T(n)$ is $O(n \log n)$. 
## Sort Efficiencies

### Array Sorting Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Best</th>
<th>Average</th>
<th>Worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quicksort</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Mergesort</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Timersort</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Heapsort</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Bubble Sort</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Tree Sort</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Shell Sort</td>
<td>$O(n \log^2 n)$</td>
<td>$O(n \log^2 n)$</td>
<td>$O(n \log^2 n)$</td>
</tr>
<tr>
<td>Bucket Sort</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Rada Sort</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Counting Sort</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Cubeord</td>
<td>$O(n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
</tr>
</tbody>
</table>