Sorting

Chapter 7 in Weiss

Sorting: The Big Picture

Given \( n \) comparable elements in an array, sort them in an increasing (or decreasing) order.

<table>
<thead>
<tr>
<th>Simple algorithms: ( O(n^2) )</th>
<th>Fancier algorithms: ( O(n \log n) )</th>
<th>Comparison lower bound: ( \Omega(n \log n) )</th>
<th>Specialized algorithms: ( O(n) )</th>
<th>Handling huge data sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion sort</td>
<td>Heap sort</td>
<td>Bucket sort</td>
<td>External sorting</td>
<td></td>
</tr>
<tr>
<td>Selection sort</td>
<td>Merge sort</td>
<td>Radix sort</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bubble sort</td>
<td>Quick sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shell sort</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Simple algorithms:
- Insertion sort
- Selection sort
- Bubble sort
- Shell sort
- ...

Fancier algorithms:
- Heap sort
- Merge sort
- Quick sort
- ...

Comparison lower bound:
- \( \Omega(n \log n) \)

Specialized algorithms:
- Insertion sort
- Selection sort
- Bubble sort
- Heap sort
- Merge sort
- Quick sort
- Shell sort
- Radix sort
- External sorting

Handling huge data sets

Insertion Sort: Idea

\- At the \( k \)th step, put the \( k \)th input element in the correct place among the first \( k \) elements
\- Result: After the \( k \)th step, the first \( k \) elements are sorted.

Runtime:
- worst case :
- best case :
- average case :

Selection Sort: idea

\- Find the smallest element, put it 1\textsuperscript{st}
\- Find the next smallest element, put it 2\textsuperscript{nd}
\- Find the next smallest, put it 3\textsuperscript{rd}
\- And so on …

Selection Sort: Code

```c
void SelectionSort (Array a[0..n-1]) {
   for (i=0, i<n; ++i) {
      j = Find index of smallest entry in a[i..n-1]
      Swap(a[i],a[j])
   }
}
```

Runtime:
- worst case :
- best case :
- average case :

HeapSort:
Using Priority Queue ADT (heap)

Shove all elements into a priority queue, take them out smallest to largest.

Runtime:
Merge Sort

- **MergeSort**(Array \[1..n\])
  1. Split Array in half
  2. Recursively sort each half
  3. Merge two halves together

```
Merge(a1[1..n], a2[1..n])
  i1=1, i2=1
  While (i1<n, i2<n) {
    if (a1[i1] < a2[i2]) {
      Next is a1[i1]
      i1++
    } else {
      Next is a2[i2]
      i2++
    }
  }
```

“Merge Sort: Complexity”

Quick Sort

1. Pick a “pivot”
2. Divide into less-than & greater-than pivot
3. Sort each side recursively

```
QuickSort(S)
  \[ S \rightarrow S_1, S_2 \]
```

QuickSort Example

- Choose the pivot as the median of three.
- Place the pivot and the largest at the right and the smallest at the left
  
- Move \(i\) to the right to be larger than pivot.
- Move \(j\) to the left to be smaller than pivot.
- Swap
QuickSort Example

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>9</td>
<td>1</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Recursive Quicksort

QuickSort(A[]): integer array, left, right : integer: {
  pivotindex := Partition(A,left,right); 
  if left + CUTOFF ≤ right then
    pivot := median3(A,left,right); 
    pivotindex := Partition(A,left,right-1,pivot); 
    QuickSort(A, left, pivotindex - 1); 
    QuickSort(A, pivotindex + 1, right); 
  else
    Insertionsort(A,left,right); 
}

Don't use quicksort for small arrays. CUTOFF = 10 is reasonable.

QuickSort: Best case complexity

Turns out to be O(n log n)

See Section 7.7.5 for an idea of the proof.
Don't need to know proof details for this course.

QuickSort: Worst case complexity

QuickSort: Average case complexity

Features of Sorting Algorithms

- In-place
  - Sorted items occupy the same space as the original items. (No copying required, only O(1) extra space if any.)
- Stable
  - Items in input with the same value end up in the same order as when they began.
## Sort Properties

<table>
<thead>
<tr>
<th>Sort Properties</th>
<th>stable?</th>
<th>in-place?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion Sort?</td>
<td>No</td>
<td>Can Be No</td>
</tr>
<tr>
<td>Selection Sort?</td>
<td>No</td>
<td>Can Be No</td>
</tr>
<tr>
<td>MergeSort?</td>
<td>No</td>
<td>Can Be No</td>
</tr>
<tr>
<td>QuickSort?</td>
<td>No</td>
<td>Can Be No</td>
</tr>
</tbody>
</table>