CSE 326: Data Structures  
Topic 12:  
Comparison-based Sorting  
Ashish Sabharwal  
Autumn, 2003

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: ( \Theta(n^2) )</th>
<th>Fancier algorithms: ( \Theta(n \log n) )</th>
<th>Comparison lower bound: ( \Omega(n \log n) )</th>
<th>Specialized algorithms: ( \Theta(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>Radix sort</td>
<td>External sorting</td>
</tr>
<tr>
<td>Selection sort</td>
<td>AVL sort</td>
<td>Merge sort</td>
<td>Quick sort</td>
<td></td>
</tr>
<tr>
<td>Bubble sort</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selection sort</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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\(^{st} \)
- Find the next smallest element, put it 2\(^{nd} \)
- Find the next smallest, put it 3\(^{rd} \)
- 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])
  }
  while (other people are coding QuickSort/MergeSort) {
    Twiddle thumbs
  }
}
```

Runtime:

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

Heap Sort:

Using Priority Q ADT (heap)

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

Runtime:
AVL Sort

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])
ii=1, i2=1
While (ii<n, i2<n) {
    if (a1[ii] < a2[i2]) {
        Next is a1[ii]
        ii++
    } else {
        Next is a2[i2]
        i2++
    }
}
Now throw in the dregs...

Merge Sort: Complexity

Quick Sort

Quick Sort Example

QuickSort: Best case complexity
QuickSort: Worst case complexity

QuickSort: Average case complexity

Dealing with Slow Quick Sorts

- Step 0: Randomly permute given input!!
  - Bad cases more common than simple probability would suggest. So, make it truly random.

- Pick pivot cleverly
  - "Median-of-3" rule: pivot = Median(first, middle, last)

- Pick pivot randomly!

  *With good choices, fastest in practice!!*

Quick Select

What if we want to find the $k^{th}$ smallest element in an array?

Say, $k = n/2$ (i.e., we want to find the median)?

QuickSelect (Array $A$, int $k$)

<table>
<thead>
<tr>
<th>Pick pivot:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>2</td>
<td>8</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Partition array:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>

- $k = \text{pindex}$?
- $k < \text{pindex}$?
- $k > \text{pindex}$?

Runtime:

To Do

- Work on Project 3
- Read Chapter 7