From practice to theory and back again

In theory there is no difference between theory and practice, but not in practice

- We’ve seen binary search, but that requires a sorted list
  - Much faster than sequential search (how much)
  - Add elements in sorted order or sort vector after adding

- Many sorting algorithms have been well-studied
  - Slower ones are often “good enough” simple to implement
  - Some fast algorithms are better than others
    * Always fast, fast most-of-the-time
    * Good in practice even if flawed theoretically?

- New algorithms still discovered
  - Quick sort in 1960, revised and updated in 1997
We can time different methods, but how to compare timings?
- Different on different machines, what about "workload"?
- Mathematical tools can help analyze/discuss algorithms

We often want to sort by different criteria
- Sort list of stocks by price, shares traded, volume traded
- Sort directories/files by size, alphabetically, or by date
- Object-oriented concepts can help in implementing sorts
Sorting: From Theory to Practice

- **Why do we study sorting?**
  - Because we have to
  - Because sorting is beautiful
  - Because ... and ...

- **There are many sorting algorithms, how many should we study?**
  - Why do we study more than one algorithm?
    - Paradigms of trade-offs and algorithm design
    - Because we can! And they’re beautiful!
  - Which sorting algorithm is best?
  - Which sort should you call from code you write?
On to sorting: Selection Sort

- Find smallest element, move into first array location
- Find next smallest element, move into second location
  - Generalize and repeat

- How many elements examined to find smallest?
  - How many elements examined to find next smallest?
  - Total number of elements examined? $N + (N-1) + \ldots + 1$
  - How many elements swapped?

- Simple to code, reasonable in practice for small vectors
  - What’s small? What’s reasonable? What’s simple?
Selection sort

- **Stable, simple to code $n^2$ sort: $n^2$ comparisons, $n$ swaps**

```java
public void selectSort (ArrayList a)
{
    for(int k=0; k < a.size(); k++)
    {
        int minIndex = findMin(a,k,a.size());
        swap(a, k, minIndex);
    }
}
```

- **# comparisons:** $\sum_{k=1}^{n} k = 1 + 2 + \ldots + n = n(n+1)/2 = O(n^2)$
  - Swaps?
  - Invariant: **Sorted, won’t move final position**

| Sorted, won’t move final position | ???? |
Insertion Sort

- **Stable sort, \( O(n^2) \), good on nearly sorted vectors**
  - Stable sorts maintain order of equal keys
    - Good for sorting on two criteria: name, then age

```java
public void insertSort(ArrayList a) {
    for (int k = 1; k < a.size(); k++) {
        Comparable e = (Comparable)a.get(k);
        int loc = k;
        while (0 < loc && e.compareTo(a.get(loc-1)) < 0) {
            a.set(loc, a.get(loc-1)); // shift right
            loc--;
        }
        a.set(loc, e);
    }
}
```

```
Sorted relative to each other ????
```
Bubble sort

- For completeness you should know about this sort
  - Few (if any) redeeming features. Really slow, really, really
  - Can code to recognize already sorted vector (see insertion)
    * Not worth it for bubble sort, much slower than insertion

```java
void bubbleSort (ArrayList a) {
    for(int j=a.size()-1; j >= 0; j--) {
        for(int k=0; k < j; k++) {
            if (a.get(k).compareTo(a.get(k+1)) > 0)
                swap(a, k, k+1);
        }
    }
}
```

- “bubble” elements down the vector/array
From practical to theoretical

- We want a notation for discussing differences between algorithms, avoid empirical details at first
  - Empirical studies needed in addition to theoretical studies
  - As we’ll see, theory hides some details, but still works

- Binary search: roughly 10 entries in a 1,000 element vector
  - What is exact relationship? How to capture “roughly”?
  - Compared to sequential/linear search?

- Use O-notation, big-O, to capture properties but avoid details
  - $N^2$ is the same as 13$N^2$ is the same as 13$N^2 + 23N$
  - $O(N^2)$, in the limit everything is the same
### Running times @ $10^6$ instructions/sec

<table>
<thead>
<tr>
<th>$N$</th>
<th>$O(\log N)$</th>
<th>$O(N)$</th>
<th>$O(N \log N)$</th>
<th>$O(N^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.000003</td>
<td>0.00001</td>
<td>0.000033</td>
<td>0.0001</td>
</tr>
<tr>
<td>100</td>
<td>0.000007</td>
<td>0.00010</td>
<td>0.000664</td>
<td>0.1000</td>
</tr>
<tr>
<td>1,000</td>
<td>0.000010</td>
<td>0.00100</td>
<td>0.010000</td>
<td>1.0</td>
</tr>
<tr>
<td>10,000</td>
<td>0.000013</td>
<td>0.01000</td>
<td>0.132900</td>
<td>1.7 min</td>
</tr>
<tr>
<td>100,000</td>
<td>0.000017</td>
<td>0.10000</td>
<td>1.661000</td>
<td>2.78 hr</td>
</tr>
<tr>
<td>1,000,000</td>
<td>0.000020</td>
<td>1.0</td>
<td>19.9</td>
<td>11.6 day</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>0.000030</td>
<td>16.7 min</td>
<td>18.3 hr</td>
<td>318 centuries</td>
</tr>
</tbody>
</table>
What does table show? Hide?

- Can we sort a million element list with selection sort?
  - How can we do this, what’s missing in the table?
  - What are hidden constants, low-order terms?

- Can we sort a billion-element list? Are there other sorts?
  - We’ll see quicksort, an efficient (most of the time) method
  - $O(N \log N)$, what does this mean?
From smarter code to algorithm

- We’ve seen selection sort, other $O(N^2)$ sorts include
  - Insertion sort: better on nearly sorted data, fewer comparisons, potentially more data movements (selection)
  - Bubble sort: dog, dog, dog, don’t use it

- Efficient sorts are trickier to code, but not too complicated
  - Avoids comparing each element to every other element
  - Often recursive as we’ll see, use divide and conquer
  - Quicksort and Mergesort are two standard examples

- Mergesort divide and conquer
  - Divide vector in two, sort both halves, merge together
  - Merging is easier because subvectors sorted, why?
Merge sort

- Divide and conquer
  - Divide list into two halves
    - Sort each half
    - Merge sorted halves together

void mergesort (ArrayList a, int left, int right)
{
  if (left < right) {
    int mid = (right+left)/2;
    mergesort(a, left, mid);
    mergesort(a, mid+1, right);
    merge(a, left, mid, right);
  }
}

- What is complexity of merging two sorted lists?
- Recursive because merge sort process can be applied to each half ...
Quicksort, an efficient sorting algorithm

- **Partition list, move smaller elements left, larger elements right**
  - Formally: choose a pivot element, all elements less than pivot moved to the left (of pivot), greater moved right
  - After partition/pivot, sort left half and sort right half

original         partition on 14         partition on 10

14 12 15 6 3 10 17 12 6 10 3 14 15 17 3 6 10 12 14 15 17
Quicksort: fast in practice

- Invented in 1962 by C.A.R. Hoare, didn’t understand recursion
  - Worst case is $O(n^2)$, but avoidable in nearly all cases
  - In 1997 Introsort published (Musser, introspective sort)
    - Like quicksort in practice, but recognizes when it will be bad

```java
void quick (ArrayList a, int left, int right) {
    if (left < right) {
        int pivot = partition(a, left, right);
        quick (a, left, pivot-1);
        quick (a, pivot+1, right);
    }
}
```

- Complexity?
- What’s a good pivot versus a bad pivot? What changes?
Partition code for quicksort

- Easy to develop partition based on loop invariant
  - statement true each time loop test is evaluated, used to verify correctness of loop

**what we want**

<table>
<thead>
<tr>
<th>( \leq ) pivot</th>
<th>( &gt; ) pivot</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>right</td>
</tr>
<tr>
<td>( pIndex )</td>
<td></td>
</tr>
</tbody>
</table>

**what we have**

\[ ??????????????? \]

| left | right |

**invariant**

<table>
<thead>
<tr>
<th>( \leq )</th>
<th>( &gt; )</th>
<th>???</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>( pIndex )</td>
<td>right</td>
</tr>
</tbody>
</table>

---

CPS 6
Summary of $O(n \log n)$ sorts

- **Quicksort** is relatively straightforward to code, very fast
  - Worst case is very unlikely, but possible, therefore ...
  - But, if lots of elements are equal, performance will be bad
    - One million integers from range 0 to 10,000
    - How can we change partition to handle this?

- **Merge sort** is stable, it’s fast, harder to code?
  - Worst case performance is $O(n \log n)$, compare quicksort
What is complexity?

- We've used O-notation, (big-Oh) to describe algorithms
  - Binary search is $O(\log n)$
  - Sequential search is $O(n)$
  - Selection sort is $O(n^2)$
  - Quicksort is $O(n \log n)$

- What do these measures tell us about “real” performance?
  - When is selection sort better than quicksort?
  - What are the advantages of sequential search?

- Describing the complexity of algorithms rather than implementations is important and essential
  - Empirical validation of theory is important too
Sorting out sort

- **Comparison based, \(O(n^2)\), sorts --- for sorting \(n\) elements**
  - Selection sort --- \(n^2\) comparisons, \(n\) swaps, easy to code
  - Insertion sort --- \(n^2\) comparisons, \(n^2\) moves, stable, fast
  - Bubble sort --- \(n^2\) everything, slow, slower, and ugly

- **Divide and conquer faster sorts: \(O(n \log n)\) for \(n\) elements**
  - Quick sort: fast in practice, though \(O(n^2)\) worst case
  - Merge sort: good worst case, great for linked lists, uses extra storage for vectors/arrays
Barbara Simons

- President of ACM: '98-'00
  - Founder and chair of branch that deals with public policy
- Awards from CPSR and EFF
  - Computer Professionals for Social Responsibility
  - Electronic Frontier Foundation

"... I urged the relaxation of export controls and the availability of better tools to protect online privacy and security. I have written and spoken of the need to increase the security of the Internet, both by encouraging the use of robust software and technology and by discouraging widespread surveillance and monitoring. "