From practice to theory and back again

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

- How do we search an array or an ArrayList for a value?
  - I'm thinking of a number from 1 to 100
    - What if I tell you: low, high, correct?
    - What if I tell you: yes or no?
- Two kinds of array search
  - Binary search, like dictionary lookup, requires sorted list
  - Sequential search, old-fashioned phone book search for number
- Which algorithm is better?
  - Slower ones are often "good enough" simple to implement
  - Some fast algorithms are better than others

Tools for algorithms and programs

- 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 CDs by artist, title, genre, length, …
  - Sort directories/files by size, alphabetically, or by date
  - Object-oriented concepts can help in implementing sorts
- We often want to sort different kinds of arrays: String and int
  - Don't want to duplicate the code, that leads to errors
  - Generic programming helps, new in Java 5, now Objects

To code or not to code, that is the …

- Should you call an existing sorting routine or write your own?
  - If you can, don't rewrite code written and accessible
  - Sometimes you don't know what to call
  - Sometimes you can't call the existing library routine
- In Java there are standard sort functions that can be used with built-in arrays and with ArrayLists
  - Accessible via `java.util.Arrays/Collectoins`
  - These are robust and fast and code is readable
- Also code for searching and min/max finding
  - Divided between Arrays and Collections

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?
- We use O-notation, big-Oh, to capture properties but avoid details
  - N^2 is the same as $13N^2$ is the same as $13N^2 + 23N$
  - $O(N^2)$, in the limit everything is the same

Running times @ 10^6 instructions/sec

N	*O(log N)*	*O(N)*	*O(N log N)*	*O(N^2)*
10	0.000003	0.00001	0.000033	0.0001
100	0.000007	0.00010	0.000664	0.1000
1,000	0.000010	0.00100	0.010000	1.0
10,000	0.000013	0.01000	0.132900	1.7 min
100,000	0.000017	0.10000	1.661000	2.78 hr
1,000,000	0.000020	1.0	19.9	11.6 day
1,000,000,000	0.000030	16.7 min	18.3 hr	318 centuries

What does table show? Hide?

- Can we sort a million element array 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 array? Are there other sorts?
  - We'll see quicksort, an efficient (most of the time) method
  - O(N log N), what does this mean?
- Sorting code for different algorithms java.util
  - Collections and Object arrays use same algorithm/code
  - Primitive types: int, double, … use different algorithm

Who is Alan Perlis?

- It is easier to write an incorrect program than to understand a correct one
- Simplicity does not precede complexity, but follows it
- If you have a procedure with ten parameters you probably missed some
- If a listener nods his head when you're explaining your program, wake him up
- Programming is an unnatural act
- Won first Turing award

http://www.cs.yale.edu/homes/perlis-alan/quotes.html

Selection sort: summary

- Simple to code n^2 sort: n^2 comparisons, n swaps

```
void selectSort(String[] a)
{
    for(int k=0; k < a.length; k++){
        int minIndex = findMin(a,k);
        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	?????

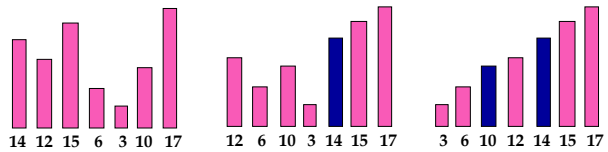
From smarter code to algorithm

- 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
  - 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 halfs, merge together
  - Merging is easier because sub-arrays sorted, why?

Quicksort, an efficient sorting algorithm

- Step one, partition the vector, moving 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*



Quicksort details

```
void quick(String[] a,int first,int last)
// pre: first <= last
// piv: a[first] <= ... <= a[list]
{
    int piv;
    if (first < last)
    {
        piv = pivot(a,first,last);
        quick(a,first,piv-1);
        quick(a,piv+1,last);
    }
}
// original call is Quick(a,0,a.length-1);
```

- How do we make progress towards base case? What's a good pivot versus a bad pivot? What changes?
  - What about the code for pivot?
  - What about other types of arrays?

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

Do it fast, do it slow, can we do it at all?

- Some problems can be solved quickly using a computer
  - Searching a sorted list
- Some problems can be solved, but it takes a long time
  - Towers of Hanoi
- Some problems can be solved, we don't know how quickly
  - Traveling salesperson, optimal class scheduling
- Some problems can't be solved at all using a computer
  - The halting problem, first shown by Alan Turing
- The halting problem: can we write one program used to determine if an arbitrary program (any program) stops?
  - One program that reads other programs, must work for every program being checked, *computability*

What is computer science?

- What is a computation?
  - Can formulate this precisely using mathematics
  - Can say "anything a computer can compute"
  - Study both theoretical and empirical formulations, build machines as well as theoretical models
- How do we build machines and the software that runs them?
  - Hardware: gates, circuits, chips, cache, memory, disk, …
  - Software: operating systems, applications, programs
- Art, Science, Engineering
  - How do we get better at programming and dealing with abstractions
  - What is hard about programming?

Shafi Goldwasser

- RCS professor of computer science at MIT
  - Co-inventor of zero-knowledge proof protocols

How do you convince someone that you know something without revealing "something"

- Consider card readers for dorms
  - Access without tracking

Work on what you like, what feels right, I now of no other way to end up doing creative work