The good sorts of sorts

Sorting – Keys and Records

Consider sorting a list of 1000 elements

- Engineer a solution
- Worse, best, and average times

Selection sort

- Always bring the smallest remaining to the front.
- Worse and best times are the same
  - About 500,000 comparisons on average
  - But only 1,000 moves

Insertion sort

- Increase the sorted first few
- Worse, best, and average times differ
  - About 250,000 comparisons on average
  - But the same number of moves

Bubble sort

- An old favorite with little merit
- Bring the bigger to the end of the line
  - While you “bubble up” the smaller
- Compute the probability that some element in last tenth belongs in the first tenth
- Increased complexity for lower performance

Measure the average move

- The average element needs to be moved though one-third of the list
  - Let’s use some calculus to figure this one out
  - About 333,333 total slots to move

Averages, in terms of N, size of list

Selection sort	$N^2/2$
Insertion sort	$N^2/4$
Total moves	$N^2/3$

A different sort of sort

Find the half-way point of each list

Time: ~1,500, if you really don't know the size of the list

Use insertion sort on each half of list

Time: 2*(500*500/4) or 125,000

Merge the two lists

Time: 1000

Total time: ~130,000

Two sorts in half the time!

Merge sort

Continue halving until you are sorting small lists

See the spreadsheet

Bad sorts

Time is proportional to the N^2

Good sorts

Time is proportional to N log N

In theory this is the best

Really good sorts when you really know your data

Time can be proportional to N

Example: Library Sort

Algorithm complexity

Finding the winner for large input sets

Ignores "constant" differences

Heavily used in graduate study in computer science

And becoming popular with mathematicians and engineers

Informally drawing and formally defining big-O

f(x) is O(g(x)) if

there exists K and B such that

for all $x > B$, $f(x) < K\, g(x)$

Examples:

$4*x^2 + 15000 *x + 3000000$ is $O(x^2)$

$4*x^2 + 15000 *x + 3000000$ is $O(x^{20})$

$4*x^2 + 15000 *x + 3000000$ is **not** $O(x^{1.999})$

Application of big-O to C-like programs

```
X = expression without function calls ;
    O(1)

if (test) {
    if-part ;
} else {
    else-part ;
}
    MAX(T(test), T(if-part), T(else-part)) ;

Statement1 ;
Statement2 ;
    MAX(T(Statement1), T(Statement2))

for(i=0; i<N; ++i) {
    Statement ;
}
    N * T(Statement)

for(i=1; i<N; i = 2*i) {
    Statement ;
}
    log N * T(Statement)
```

The running time doubles

Never, if O(1)
When input size multiplies by itself (N to N*N), if O(log N)
When input size doubles, if O(N)
When input size increases by factor of ~1.4, if $O(N^2)$
When input size increases by one, if $O(2^N)$