Overview

Sorting defined
Algorithms for sorting
Selection Sort algorithm
Efficiency of Selection Sort

Sorting

The problem: put things in order
Usually smallest to largest: “ascending”
Could also be largest to smallest: “descending”

Lots of applications!
ordering hits in web search engine
preparing lists of output
merging data from multiple sources
to help solve other problems
faster search (allows binary search)
too many to mention!

Sorting: More Formally

Given an array \( b[0], b[1], \ldots, b[n-1] \),
reorder entries so that
\( b[0] \leq b[1] \leq \ldots \leq b[n-1] \)

Shorthand for these slides: the notation \( \text{array}[i..k] \)
means all of the elements
\( \text{array}[i], \text{array}[i+1], \ldots, \text{array}[k] \)
Using this notation, the entire array would be:
\( \text{b}[0..n-1] \)

P.S.: This is not C syntax!

Sorting Algorithms

Sorting has been intensively studied for decades
Many different ways to do it!
We’ll look at only one algorithm, called
“Selection Sort”
Other algorithms you might hear about in
other courses include Bubble Sort, Insertion Sort, QuickSort, and MergeSort. And that’s only the beginning!

Sorting Problem

What we want: Data sorted in order

\( b[0] \leq b[1] \leq \ldots \leq b[n-1] \)

Initial conditions

unsorted
Selection Sort

General situation

\[
\begin{array}{c|c|c|c}
0 & k & n \\
\hline
\text{smallest elements, sorted} & \text{remainder, unsorted} & \\
\end{array}
\]

Step:

Find smallest element \( x \) in \( b[k..n-1] \)
Swap smallest element with \( b[k] \), then increase \( k \)

\[
\begin{array}{c|c|c|c}
0 & k & n \\
\hline
\text{smallest elements, sorted} & \text{remainder, unsorted} & \\
\end{array}
\]

Subproblem: Find Smallest

\[\text{Find location of smallest element in } b[k..n-1] \]
\[\text{Assumption: } k < n \]
\[\text{Returns index of smallest, does not return the smallest value itself} \]

```c
int min_loc (int b[], int k, int n) {
    int j, pos; /* b[pos] is smallest element */
    pos = k;
    for (j = k + 1; j < n; j = j + 1)
        if (b[j] < b[pos])
            pos = j;
    return pos;
}
```

Code for Selection Sort

\[\text{Sort } b[0..n-1] \text{ in non-decreasing order} \]
\[\text{Assumption: } k < n \]

```c
void sel_sort (int b[], int n) {
    int k, m;
    for (k = 0; k < n - 1; k = k + 1) {
        m = min_loc(b,k,n);
        swap(&a[k], &b[m]);
    }
}
```

Example

\[
\begin{array}{rrrrrrrr}
3 & 12 & -5 & 6 & 142 & 21 & -17 & 45 \\
\end{array}
\]

Example (cont.)

\[
\begin{array}{rrrrrrrr}
-17 & -5 & 3 & 6 & 142 & 21 & 12 & 45 \\
\end{array}
\]

Example (concluded)

\[
\begin{array}{rrrrrrrr}
-17 & -5 & 3 & 6 & 12 & 21 & 45 & 142 \\
\end{array}
\]
Sorting Analysis

How many steps are needed to sort $n$ things?

For each swap, we have to search the remaining array

- length is proportional to original array length $n$
- Need about $n$ search/swap passes
- Total number of steps proportional to $n^2$

Conclusion: selection sort is pretty expensive (slow) for large $n$

Can We Do Better Than $n^2$?

Sure we can!
- Selection, insertion, bubble sorts are all proportional to $n^2$
- Other sorts are proportional to $n \log n$
  - Mergesort
  - Quicksort
  - etc.

$\log n$ is considerably smaller than $n$, especially as $n$ gets larger

As the size of our problem grows, the time to run a $n^2$ sort will grow much faster than an $n \log n$ one.

Any better than $n \log n$?

In general, no. But in special cases, we can do better

Example: Sort exams by score: drop each exam in one of 101 piles; work is proportional to $n$

Curious fact: efficiency can be studied and predicted mathematically, without using a computer at all!

Comments about Efficiency

Efficiency means doing things in a way that saves resources

- Usually measured by time or memory used
- Many small programming details have little or no measurable effect on efficiency
- The big differences comes with the right choice of algorithm and/or data structure

Summary

Sort means placing things in order
Selection sort is one of many algorithms

- At each step, finds the smallest remaining value
Selection sort requires on the order of $n^2$ steps

- There are sorting algorithms which are greatly more efficient
- It’s the algorithm that makes the difference, not the coding details