CSE 142
Computer Programming I

Sorting

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], ... b[n-1],
reorder entries so that
b[0] <= b[1] <= ... <= b[n-1]

Shorthand for these slides: the notation *array[i..k]* means all of the elements
array[i],array[i+1]...array[k]
Using this notation, the entire array would be:
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

0 n
b | sorted: b[0]<=b[1]<=...<=b[n-1] |

Initial conditions

0 n
b | unsorted |

Selection Sort

General situation

	0	k	n
b	smallest elements, sorted	remainder, unsorted	

Strategy of the algorithm:

Grow the blue area, shrink the pink area

Q-7

Selection Sort

Step:

Find smallest element x in b[k..n-1]

Swap smallest element with b[k], then increase k

	0	k			n
b	smallest elements, sorted			x	

Q-8

Subproblem: Find Smallest

```
/* Find location of smallest element in b[k..n-1] */
/* Assumption: k < n */
/* Returns index of smallest, does not return the
   smallest value itself */
int min_loc (int b[ ], int k, int n) {
   int j, pos;  /* b[pos] is smallest element */
                /* found so far */
   pos = k;
   for ( j = k + 1; j < n; j = j + 1)
     if (b[j] < b[pos])
       pos = j;
   return pos;
}
```

Q-9

Code for Selection Sort

```
/* Sort b[0..n-1] in non-decreasing order
   (rearrange  elements in b so that
   b[0]<=b[1]<=…<=b[n-1] ) */

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(&b[k], &b[m]);
  }
}
```

Q-10

Example

b	3	12	-5	6	142	21	-17	45
							min_loc	

b	-17	12	-5	6	142	21	3	45
			min_loc					

b	-17	-5	12	6	142	21	3	45
							min_loc	

Q-11

Example (cont.)

b	-17	-5	3	6	142	21	12	45
				min_loc				

b	-17	-5	3	6	142	21	12	45
							min_loc	

b	-17	-5	3	6	12	21	142	45
						min_loc		

Q-12

Example (concluded)

b	-17	-5	3	6	12	21	142	45

min_loc

b	-17	-5	3	6	12	21	45	142

Q-13

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*

Q-14

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

(remember: linear search's time is proportional to n; binary search's is proportional to log n)

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

Q-15

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!

Q-16

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*

Q-17

Efficiency: Not Always What You Expect!

Myth: I should make everything efficient!

Imagine spending hours optimizing a binary search for an array you search only once. What's the problem?

Myth: It's the details that really matter… like using `chars` to represent small numbers instead of `ints`!

Imagine you need to send a huge amount of data from your headquarters in Denver to your printer 15 miles away. What's the FASTEST way?

(From Jon Bentley's Programming Pearls)

Q-18

Summary

Sorting 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

Q-19

QOTD: Sorting as you go

Sometimes arrays grow one element at a time. You could add each element so that the array is always in sorted order. Then you don't have to stop and sort the array later.

Example: the array of players in HW5

Write an addToPlayerArray function which inserts the new player in the array, maintaining alphabetical order by player name.

Q-20

What complication will there be to your search in this array?