CPE 101 slides modified from UW course slides

Lecture 16: Sorting

Overview

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

Sorting

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"

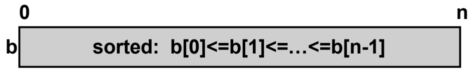
Other algorithms you might hear about in other courses include Bubble Sort, Insertion Sort, QuickSort, and MergeSort. And that's only the beginning!

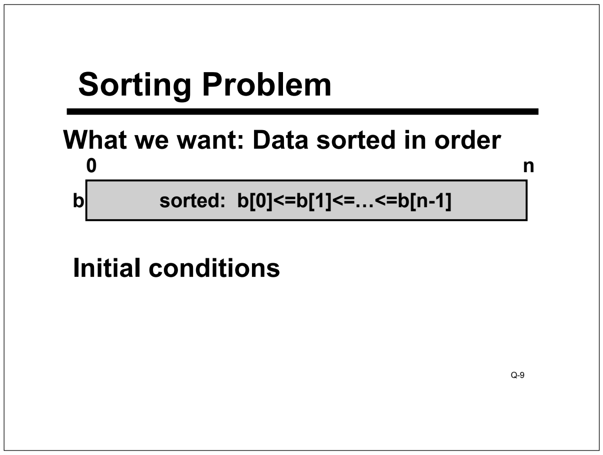
Sorting Problem

What we want: Data sorted in order

Sorting Problem

What we want: Data sorted in order





Sorting Problem

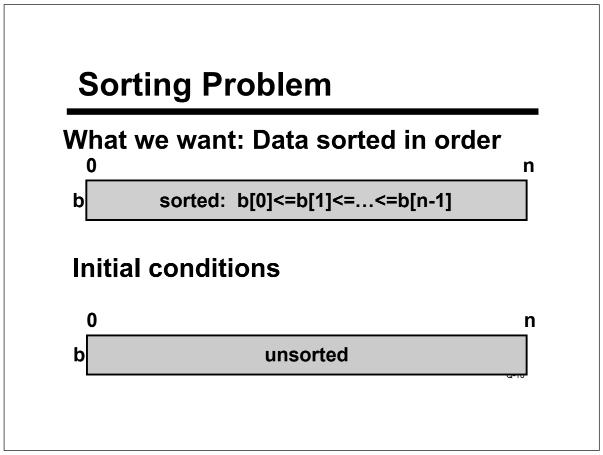
What we want: Data sorted in order

0 n

b sorted: b[0]<=b[1]<=…<=b[n-1]

Initial conditions

Q-9



Sorting Problem

What we want: Data sorted in order

0 n

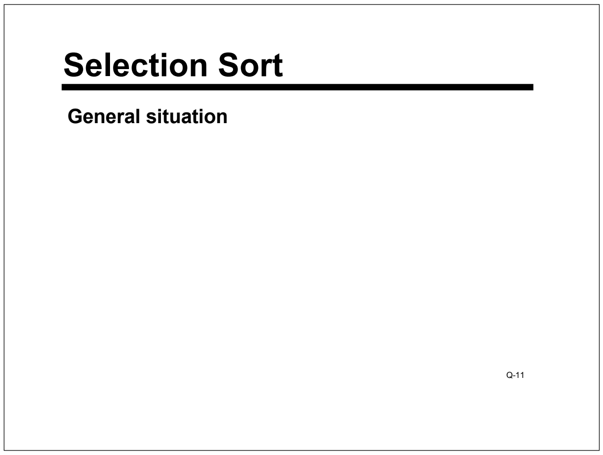
b sorted: b[0]<=b[1]<=…<=b[n-1]

Initial conditions

0 n

b unsorted

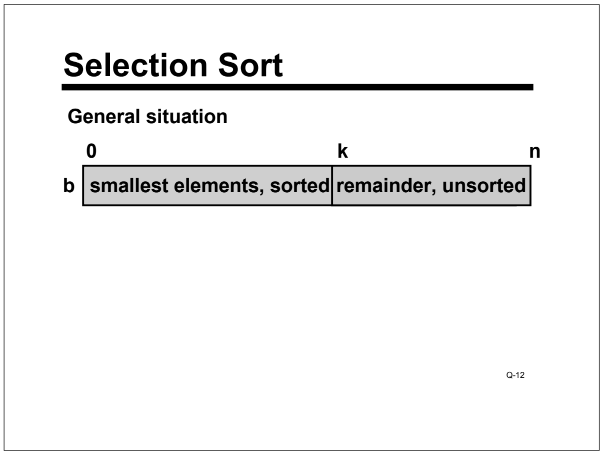
Q-10



Selection Sort

General situation

Q-11



Selection Sort

General situation

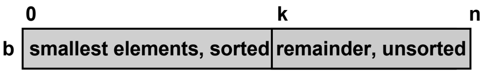
0 k n

b smallest elements, sorted | remainder, unsorted

Q-12

Selection Sort

General situation



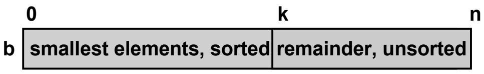
Step:

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

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

Selection Sort

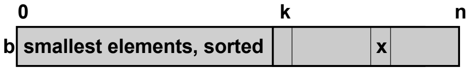
General situation



Step:

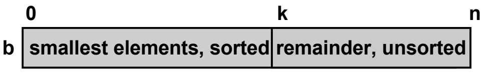
Find smallest element x in b[k..n-1]

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



Selection Sort

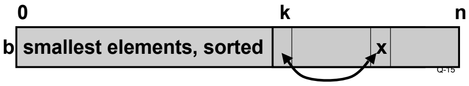
General situation



Step:

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

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



Subproblem: Find Smallest

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 */
```

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;
}
```

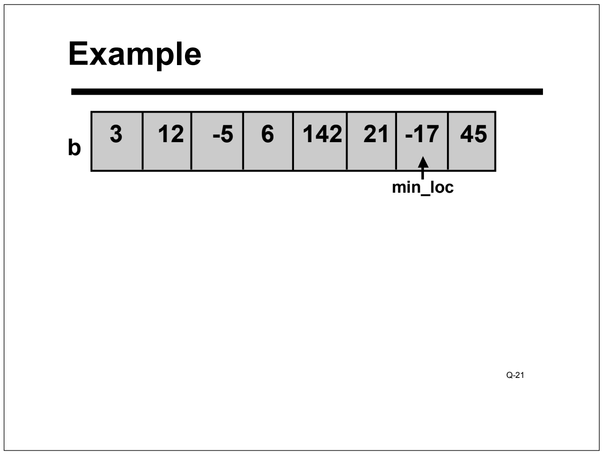
Code for Selection Sort

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

Example

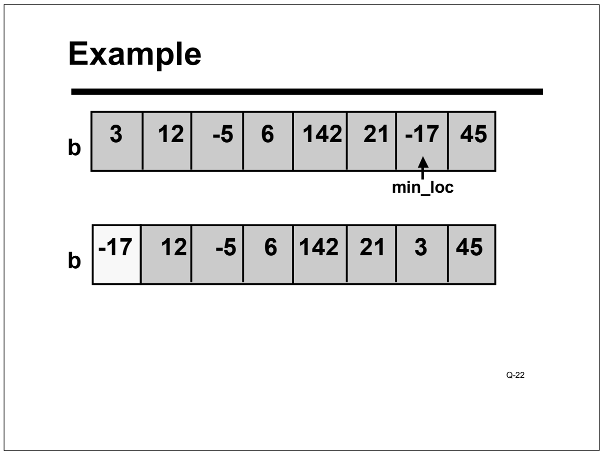
b	3	12	-5	6	142	21	-17	45



Example

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

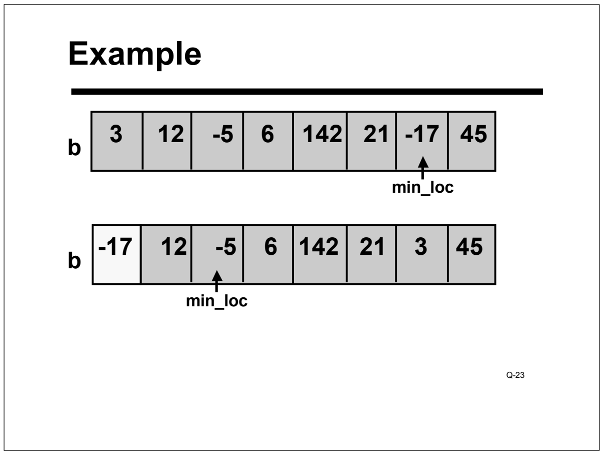
Q-21



Example

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

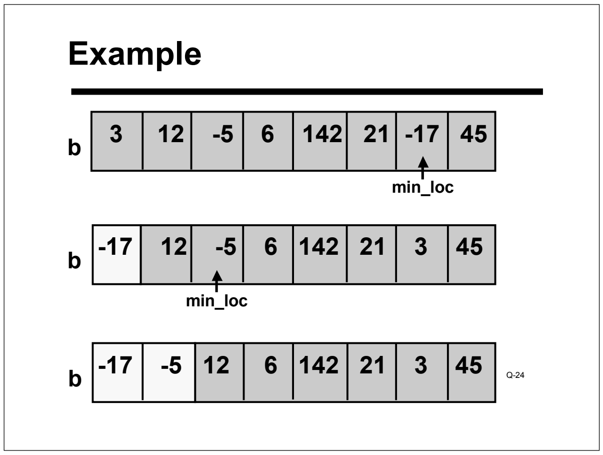
Q-22



Example

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

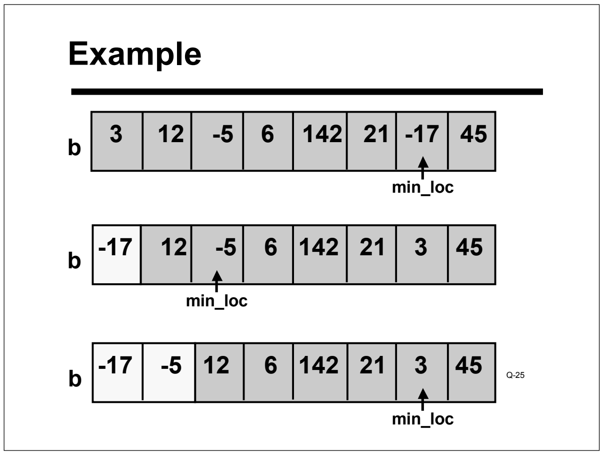
Q-23



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

Q-24



Example

b	3	12	-5	6	142	21	-17	45

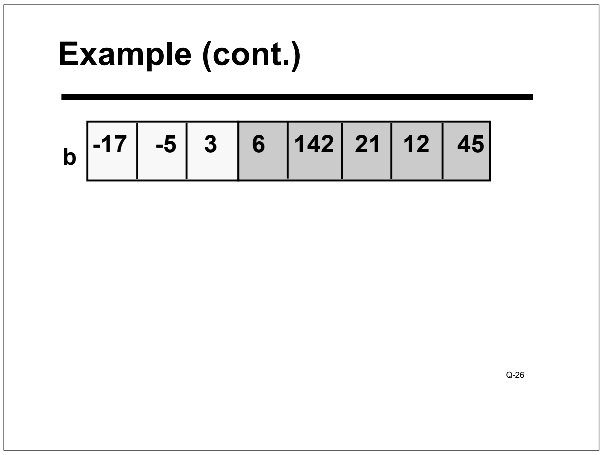
min_loc

b	-17	12	-5	6	142	21	3	45

min_loc

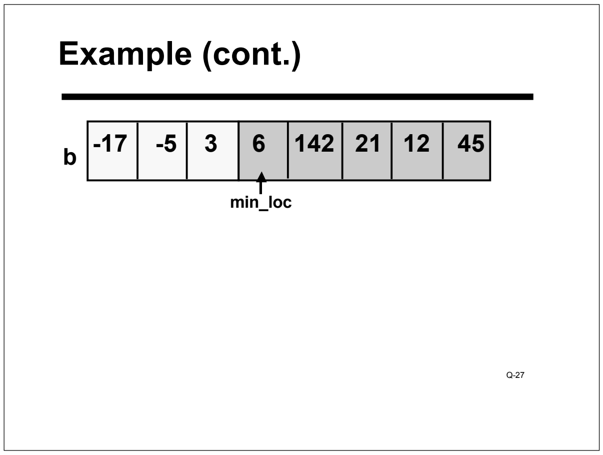
b	-17	-5	12	6	142	21	3	45

min_loc



Example (cont.)

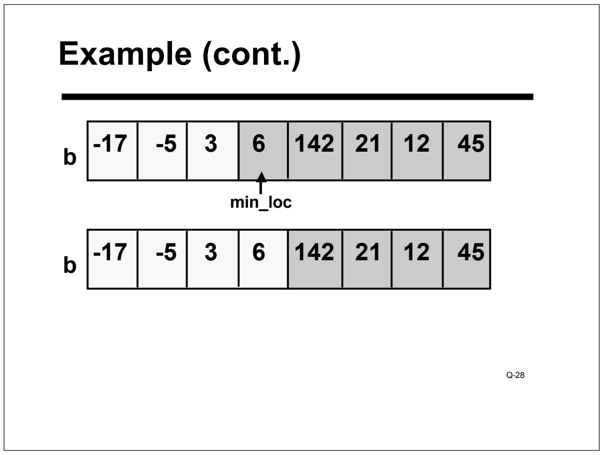
b	-17	-5	3	6	142	21	12	45



Example (cont.)

b	-17	-5	3	6	142	21	12	45

min_loc

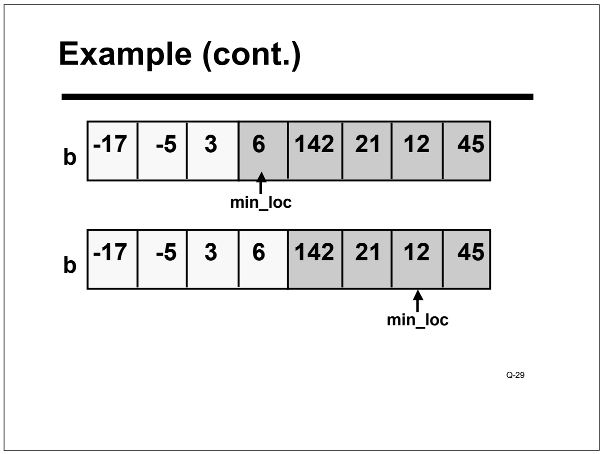


Example (cont.)

b	-17	-5	3	6	142	21	12	45

min_loc

b	-17	-5	3	6	142	21	12	45



Example (cont.)

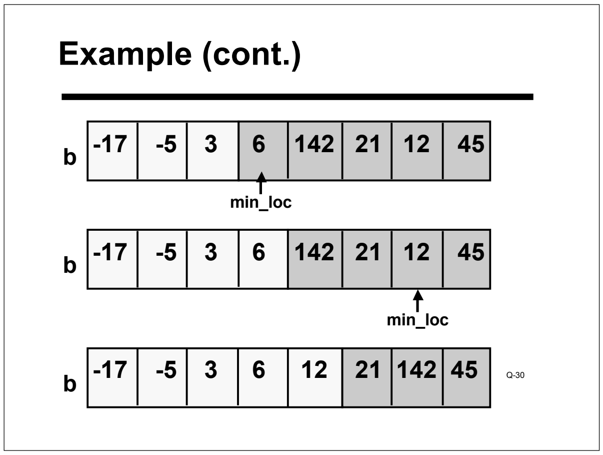
b	-17	-5	3	6	142	21	12	45

min_loc

b	-17	-5	3	6	142	21	12	45

min_loc

Q-29



Example (cont.)

b	-17	-5	3	6	142	21	12	45

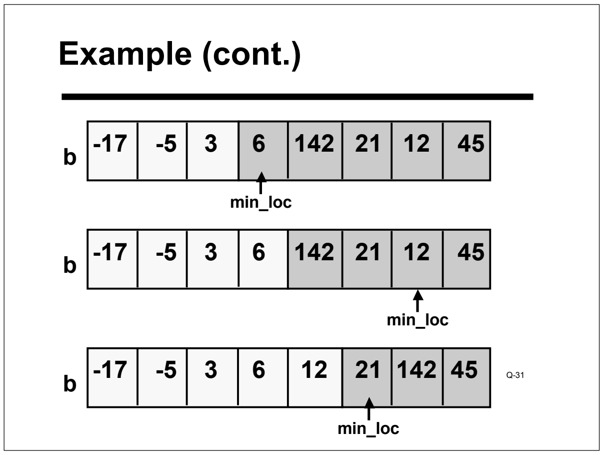
min_loc

b	-17	-5	3	6	142	21	12	45

min_loc

b	-17	-5	3	6	12	21	142	45

Q-30



Example (cont.)

b	-17	-5	3	6	142	21	12	45

min_loc

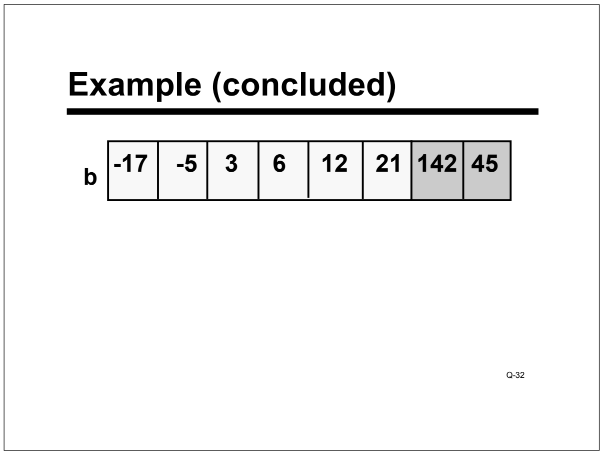
b	-17	-5	3	6	142	21	12	45

min_loc

b	-17	-5	3	6	12	21	142	45

min_loc

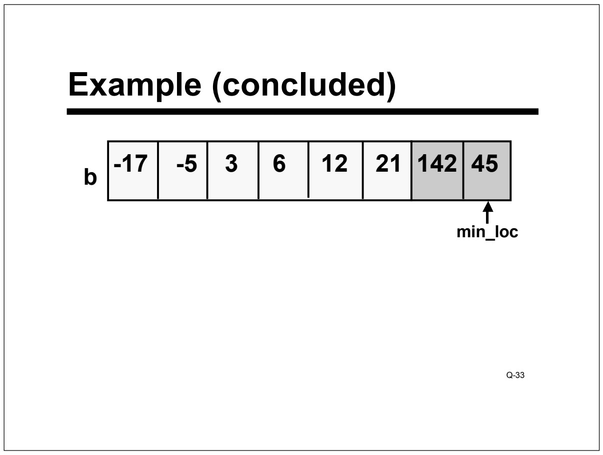
Q-31



Example (concluded)

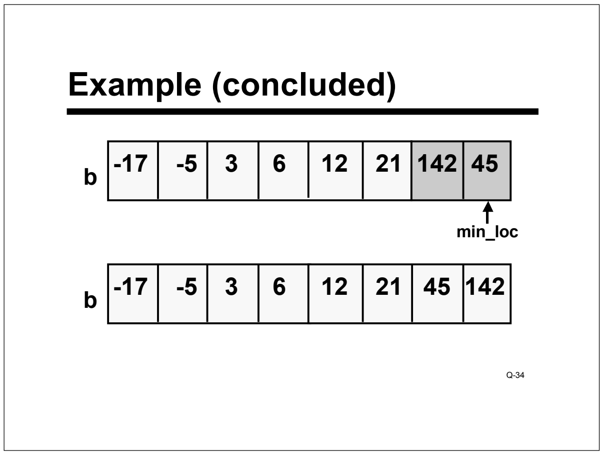
b	-17	-5	3	6	12	21	142	45

Q-32



Example (concluded)

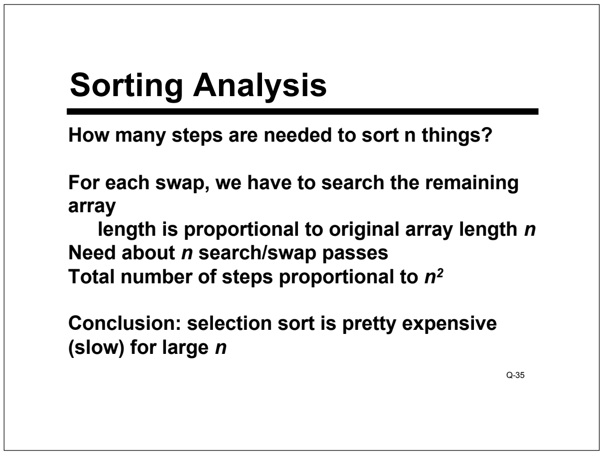
b	-17	-5	3	6	12	21	142	45
								min_loc



Example (concluded)

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

b	-17	-5	3	6	12	21	45	142



Sorting Analysis

How many steps are needed to sort n things?

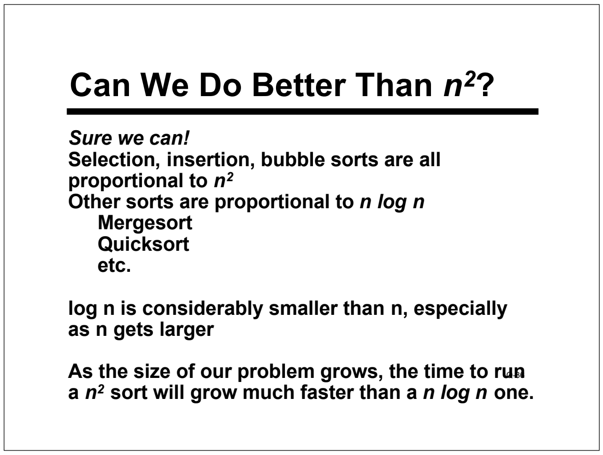
For each swap, we have to search the remaining array

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!

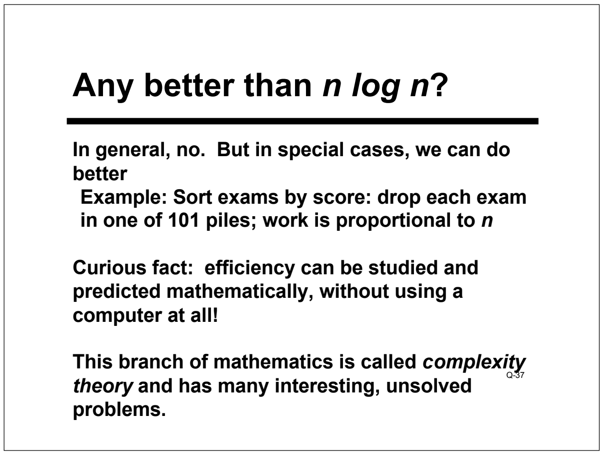
Selection, insertion, bubble sorts are all proportional to n^2

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 a *n log n* one.



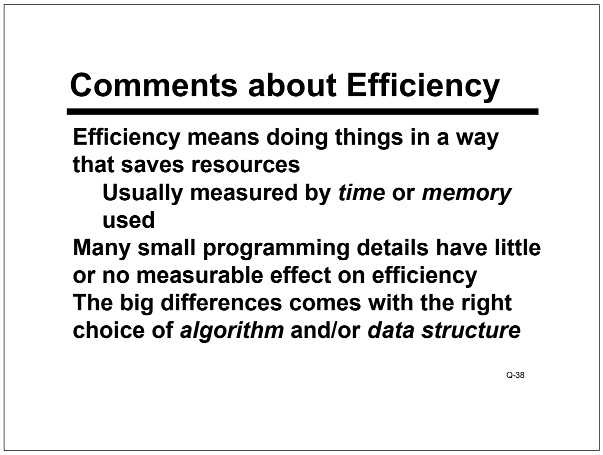
Any better than *n log n*?

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!

This branch of mathematics is called *complexity theory* and has many interesting, unsolved problems.



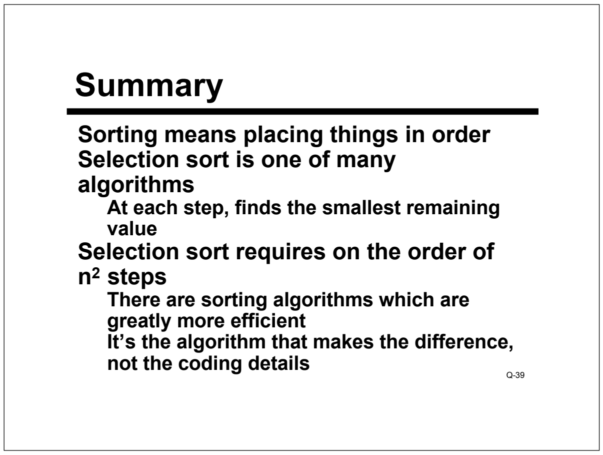
Comments about Efficiency

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

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