Quadratic Sorting

Dr. Mattox Beckman

Illinois Institute of Technology
Department of Computer Science

Objectives

This lecture covers the $\mathcal{O}(n^2)$ sorting algorithms.

- Your objective: be able to describe and implement:
  - selection sort
  - insertion sort
  - bubble sort

Complexity Measures

Input Size	Algorithm Speed					
	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
1	1s	1s	1s	1s	1s	1s
2	1.1s	2s	2.2s	4s	2s	2s
4	2s	4s	8s	16s	16s	24s
8	3s	8s	24s	64s	256s	40320s ($>$ 10h)
16	4s	16s	64s	256s	65536s ($>$18h)	20,922,789,888,000s ($>$663,457 years)

Selection Sort

Suppose you start with a pile of 1000 papers to sort by name, and an empty box marked “already sorted”.

1. Go thru all 1000 papers, pick (i.e., *select*) the smallest.
2. Put it in the “already sorted” box.
3. Go through the remaining 999 papers, pick out the smallest.
4. Put it underneath the other paper in the “already sorted” box.
5. Repeat...

How long will this take? This is $\mathcal{O}(n^2)$.
$1,000 + 999 + 998 + \cdots + 1 = 500,500$ steps.

Selection Sorting Example

If we are using an array, we can sort "in-place" by exchanging the smallest element with the top-most unsorted element.

40
23
67
10
36
52

- Given an array, select the smallest element.
- We select the 10.
- Swap upward..
- Pass 1 done.

Selection Sorting Example

If we are using an array, we can sort "in-place" by exchanging the smallest element with the top-most unsorted element.

40	40
23	23
67	67
10	10
36	36
52	52

- Given an array, select the smallest element.
- We select the 10.
- Swap upward..
- Pass 1 done.

Selection Sorting Example

If we are using an array, we can sort "in-place" by exchanging the smallest element with the top-most unsorted element.

40	40	40
23	23	23
67	67	10
10	10	67
36	36	36
52	52	52

- Given an array, select the smallest element.
- We select the 10.
- Swap upward..
- Pass 1 done.

Selection Sorting Example

If we are using an array, we can sort "in-place" by exchanging the smallest element with the top-most unsorted element.

40	40	40	40
23	23	23	10
67	67	10	23
10	10	67	67
36	36	36	36
52	52	52	52

- Given an array, select the smallest element.
- We select the 10.
- Swap upward..
- Pass 1 done.

Selection Sorting Example

If we are using an array, we can sort "in-place" by exchanging the smallest element with the top-most unsorted element.

40	40	40	40	10
23	23	23	10	40
67	67	10	23	23
10	10	67	67	67
36	36	36	36	36
52	52	52	52	52

- Given an array, select the smallest element.
- We select the 10.
- Swap upward..
- Pass 1 done.

Selection Sorting Example

If we are using an array, we can sort "in-place" by exchanging the smallest element with the top-most unsorted element.

40	40	40	40	10	10
23	23	23	10	40	40
67	67	10	23	23	23
10	10	67	67	67	67
36	36	36	36	36	36
52	52	52	52	52	52

- Given an array, select the smallest element.
- We select the 10.
- Swap upward..
- Pass 1 done.

More steps

10
40
23
67
36
52

Notice the pattern made by the black, blue, and green elements!

More steps

10	10
40	23
23	67
67	40
36	36
52	52

Notice the pattern made by the black, blue, and green elements!

More steps

10	10	10
40	23	23
23	67	67
67	40	40
36	36	36
52	52	52

Notice the pattern made by the black, blue, and green elements!

More steps

10	10	10	10
40	23	23	23
23	67	67	36
67	40	40	40
36	36	36	67
52	52	52	52

Notice the pattern made by the black, blue, and green elements!

More steps

10	10	10	10	10
40	23	23	23	23
23	67	67	36	36
67	40	40	40	40
36	36	36	67	67
52	52	52	52	52

Notice the pattern made by the black, blue, and green elements!

More steps

10	10	10	10	10	10
40	23	23	23	23	23
23	67	67	36	36	36
67	40	40	40	40	40
36	36	36	67	67	52
52	52	52	52	52	67

Notice the pattern made by the black, blue, and green elements!

Another Example

4
2
6
1
3
5
7

- Black = untouched
- Blue = checked for minimality
- Green = already sorted
- Red = selected

Another Example

4	4
2	2
6	6
1	1
3	3
5	5
7	7

- Black = untouched
- Blue = checked for minimality
- Green = already sorted
- Red = selected

Another Example

4	4	1
2	2	4
6	6	2
1	1	6
3	3	3
5	5	5
7	7	7

- Black = untouched
- Blue = checked for minimality
- Green = already sorted
- Red = selected

Another Example

4	4	1	1
2	2	4	2
6	6	2	4
1	1	6	6
3	3	3	3
5	5	5	5
7	7	7	7

- Black = untouched
- Blue = checked for minimality
- Green = already sorted
- Red = selected

Another Example

4	4	1	1	1
2	2	4	2	2
6	6	2	4	3
1	1	6	6	4
3	3	3	3	6
5	5	5	5	5
7	7	7	7	7

- Black = untouched
- Blue = checked for minimality
- Green = already sorted
- Red = selected

Another Example

4	4	1	1	1	1
2	2	4	2	2	2
6	6	2	4	3	3
1	1	6	6	4	4
3	3	3	3	6	6
5	5	5	5	5	5
7	7	7	7	7	7

- Black = untouched
- Blue = checked for minimality
- Green = already sorted
- Red = selected

Another Example

4	4	1	1	1	1	1
2	2	4	2	2	2	2
6	6	2	4	3	3	3
1	1	6	6	4	4	4
3	3	3	3	6	6	5
5	5	5	5	5	5	6
7	7	7	7	7	7	7

- Black = untouched
- Blue = checked for minimality
- Green = already sorted
- Red = selected

Another Example

4	4	1	1	1	1	1	1
2	2	4	2	2	2	2	2
6	6	2	4	3	3	3	3
1	1	6	6	4	4	4	4
3	3	3	3	6	6	5	5
5	5	5	5	5	5	6	6
7	7	7	7	7	7	7	7

- Black = untouched
- Blue = checked for minimality
- Green = already sorted
- Red = selected

Another Example

4	4	1	1	1	1	1	1	1
2	2	4	2	2	2	2	2	2
6	6	2	4	3	3	3	3	3
1	1	6	6	4	4	4	4	4
3	3	3	3	6	6	5	5	5
5	5	5	5	5	5	6	6	6
7	7	7	7	7	7	7	7	7

- Black = untouched
- Blue = checked for minimality
- Green = already sorted
- Red = selected

Other $\mathcal{O}(n^2)$ sorts

- Insertion Sort is similar to Selection Sort. The difference is when you do the sorting.
  1. Pick the first exam from the top of the pile.
  2. Go thru the exams in the "already sorted" pile, and *insert* it in the proper location.
  3. Repeat...
  4. This is used for linked lists.
- Bubble Sort uses only one box...
  1. Go through the exams in the box. When you are on exam n, check exam $n+1$. If they are out of order, swap them.
  2. Repeat...
  3. Never use this.

Insertion Sort Example

Arrays would look like this; linked lists are a bit more efficient.

Input	Out					
40	40	23	23	10	10	10
23	23	40	40	23	23	23
67	67	67	67	40	36	36
10	10	10	10	67	40	40
36	36	36	36	36	67	52
52	52	52	52	52	52	67

Red = "selected"
Blue = shifted
Green = not checked

How quickly would this run if the input list were already sorted?

Example 2 with insertion sort

4
2
6
1
3
5
7

Example 2 with insertion sort

4	4
2	2
6	6
1	1
3	3
5	5
7	7

Example 2 with insertion sort

4	4	2
2	2	4
6	6	6
1	1	1
3	3	3
5	5	5
7	7	7

Example 2 with insertion sort

4	4	2	2
2	2	4	4
6	6	6	6
1	1	1	1
3	3	3	3
5	5	5	5
7	7	7	7

Example 2 with insertion sort

4	4	2	2	1
2	2	4	4	2
6	6	6	6	4
1	1	1	1	6
3	3	3	3	3
5	5	5	5	5
7	7	7	7	7

Example 2 with insertion sort

4	4	2	2	1	1
2	2	4	4	2	2
6	6	6	6	4	3
1	1	1	1	6	4
3	3	3	3	3	6
5	5	5	5	5	5
7	7	7	7	7	7

Example 2 with insertion sort

4	4	2	2	1	1	1
2	2	4	4	2	2	2
6	6	6	6	4	3	3
1	1	1	1	6	4	4
3	3	3	3	3	6	5
5	5	5	5	5	5	6
7	7	7	7	7	7	7

Example 2 with insertion sort

4	4	2	2	1	1	1	1
2	2	4	4	2	2	2	2
6	6	6	6	4	3	3	3
1	1	1	1	6	4	4	4
3	3	3	3	3	6	5	5
5	5	5	5	5	5	6	6
7	7	7	7	7	7	7	7

Bubble Sort Example, Pass 1

Pass 1: the 67 will “sink” to the bottom.

40	23	23	23	23	23
23	40	40	40	40	40
67	67	67	10	10	10
10	10	10	67	36	36
36	36	36	36	67	52
52	52	52	52	52	67

Bubble Sort Example, Pass 2 and 3

On pass 3 we detect that the array is sorted.

23	23	23	23	23	10	10	10	10
40	40	10	10	10	23	23	23	23
10	10	40	36	36	36	36	36	36
36	36	36	40	40	40	40	40	40
52	52	52	52	52	52	52	52	52
67	67	67	67	67	67	67	67	67

This sort is inefficient because exchanges don't move very far.