Sorting algorithms

Outline

In this topic, we will introduce sorting, including:

- Definitions
- Assumptions
- *In-place* sorting
- Sorting techniques and strategies
- Overview of run times

Lower bound on run times

Define inversions and use this as a measure of *unsortedness*

Definition

Sorting is the process of:

- Taking a list of objects which could be stored in a linear order

$$(a_0, a_1, \ldots, a_{n-1})$$

e.g., numbers, and returning an reordering

$$(a'_0, a'_1, \ldots, a'_{n-1})$$

such that

$$a'_0 \leq a'_1 \leq \cdots \leq a'_{n-1}$$

The conversion of an Abstract List into an Abstract Sorted List

Definition

Seldom will we sort isolated values

- Usually we will sort a number of records containing a number of fields based on a *key*:

19991532	Stevenson	Monica	3 Glendridge Ave.
19990253	Redpath	Ruth	53 Belton Blvd.
19985832	Kilji	Islam	37 Masterson Ave.
20003541	Groskurth	Ken	12 Marsdale Ave.
19981932	Carol	Ann	81 Oakridge Ave.
20003287	Redpath	David	5 Glendale Ave.

Numerically by ID Number

19981932	Carol	Ann	81 Oakridge Ave.
19985832	Khilji	Islam	37 Masterson Ave.
19990253	Redpath	Ruth	53 Belton Blvd.
19991532	Stevenson	Monica	3 Glendridge Ave.
20003287	Redpath	David	5 Glendale Ave.
20003541	Groskurth	Ken	12 Marsdale Ave.

Lexicographically by surname, then given name

19981932	Carol	Ann	81 Oakridge Ave.
20003541	Groskurth	Ken	12 Marsdale Ave.
19985832	Kilji	Islam	37 Masterson Ave.
20003287	Redpath	David	5 Glendale Ave.
19990253	Redpath	Ruth	53 Belton Blvd.
19991532	Stevenson	Monica	3 Glendridge Ave.

Definition

In these topics, we will assume that:

- Arrays are to be used for both input and output,
- We will focus on sorting objects and leave the more general case of sorting records based on one or more fields as an implementation detail

In-place Sorting

Sorting algorithms may be performed *in-place*, that is, with the allocation of at most $\Theta(1)$ additional memory (*e.g.*, fixed number of local variables)

- Some definitions of *in place* as using $\mathbf{o}(n)$ memory

Other sorting algorithms require the allocation of second array of equal size

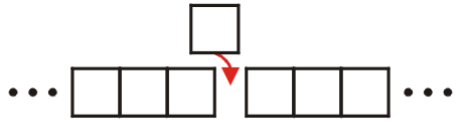
- Requires $\Theta(n)$ additional memory

We will prefer in-place sorting algorithms

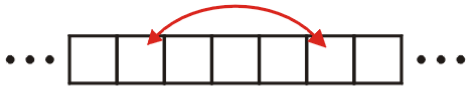
Classifications

The operations of a sorting algorithm are based on the actions performed:

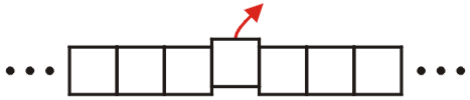
- Insertion



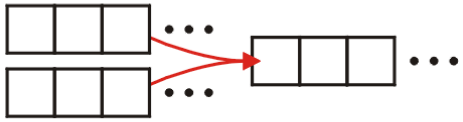
- Exchanging



- Selection



- Merging



- Distribution

Run-time

The run time of the sorting algorithms we will look at fall into one of three categories:

$$\Theta(n) \quad \Theta(n \ln(n)) \quad \mathbf{O}(n^2)$$

We will examine average- and worst-case scenarios for each algorithm

The run-time may change significantly based on the scenario

Run-time

We will review the more traditional $\mathbf{O}(n^2)$ sorting algorithms:

- Insertion sort, Bubble sort

Some of the faster $\mathbf{\Theta}(n \ln(n))$ sorting algorithms:

- Heap sort, Quicksort, and Merge sort

And linear-time sorting algorithms

- Bucket sort and Radix sort
- We must make assumptions about the data

Lower-bound Run-time

Any sorting algorithm must examine each entry in the array at least once

- Consequently, all sorting algorithms must be $\Omega(n)$

We will not be able to achieve $\Theta(n)$ behaviour without additional assumptions

Optimal Sorting Algorithms

The next seven topics will cover seven common sorting algorithms

- There is no *optimal* sorting algorithm which can be used in all places
- Under various circumstances, different sorting algorithms will deliver optimal run-time and memory-allocation requirements

Inversions

Consider the following three lists:

1 16 12 26 25 35 33 58 45 42 56 67 83 75 74 86 81 88 99 95

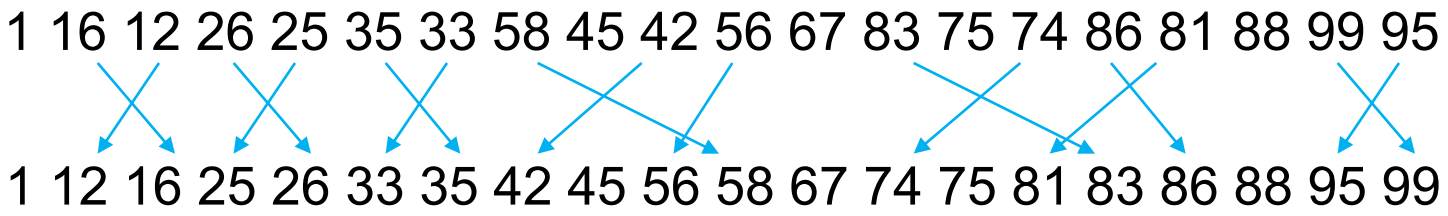
1 17 21 42 24 27 32 35 45 47 57 23 66 69 70 76 87 85 95 99

22 20 81 38 95 84 99 12 79 44 26 87 96 10 48 80 1 31 16 92

To what degree are these three lists unsorted?

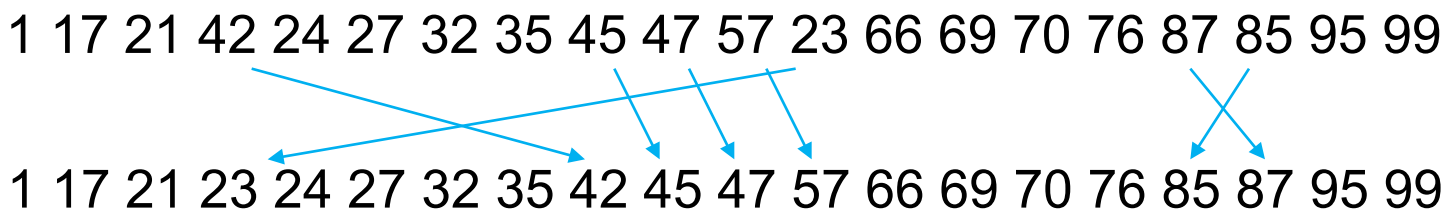
Inversions

The first list requires only a few exchanges to make it sorted



Inversions

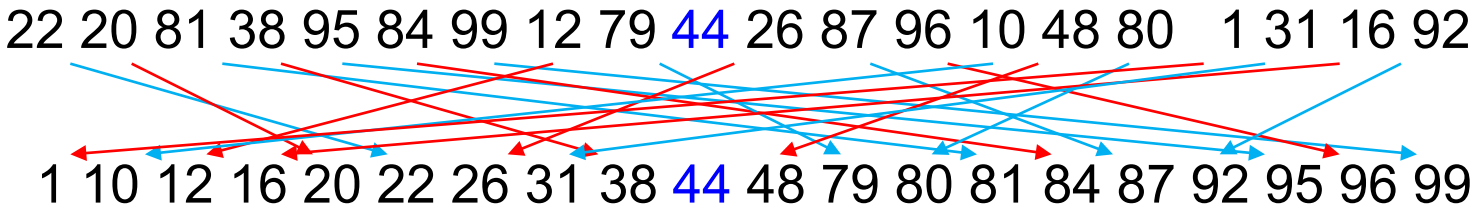
The second list has two entries significantly out of order



however, most entries (13) are in place

Inversions

The third list would, by any reasonable definition, be significantly unsorted



Inversions

Given any list of n numbers, there are

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

pairs of numbers

For example, the list (1, 3, 5, 4, 2, 6) contains the following 15 pairs:

(1, 3)	(1, 5)	(1, 4)	(1, 2)	(1, 6)
	(3, 5)	(3, 4)	(3, 2)	(3, 6)
		(5, 4)	(5, 2)	(5, 6)
			(4, 2)	(4, 6)
				(2, 6)

Inversions

You may note that 11 of these pairs of numbers are in order:

(1, 3)	**(1, 5)**	**(1, 4)**	**(1, 2)**	**(1, 6)**
	(3, 5)	**(3, 4)**	(3, 2)	**(3, 6)**
		(5, 4)	(5, 2)	**(5, 6)**
			(4, 2)	**(4, 6)**
				(2, 6)

Inversions

The remaining four pairs are *reversed*, or *inverted*

(1, 3)	(1, 5)	(1, 4)	(1, 2)	(1, 6)
	(3, 5)	(3, 4)	**(3, 2)**	(3, 6)
		(5, 4)	**(5, 2)**	(5, 6)
			(4, 2)	(4, 6)
				(2, 6)

Inversions

Given a permutation of n elements

$$a_0, a_1, \ldots, a_{n-1}$$

an inversion is defined as a pair of entries which are reversed

That is, (a_j, a_k) forms an inversion if

$$j < k \text{ but } a_j > a_k$$

Ref: Bruno Preiss, *Data Structures and Algorithms*

Inversions

Therefore, the permutation

1, 3, 5, 4, 2, 6

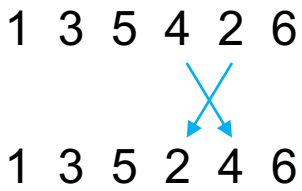
contains four inversions:

(3, 2) (5, 4) (5, 2) (4, 2)

Inversions

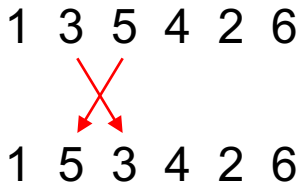
Exchanging (or swapping) two adjacent entries either:

– removes an inversion, *e.g.*,



removes the inversion (4, 2)

or introduces a new inversion, *e.g.*, (5, 3) with



Number of Inversions

There are $\binom{n}{2} = \frac{n(n-1)}{2}$ pairs of numbers in any set of n objects

Consequently, each pair contributes to

- the set of ordered pairs, or
- the set of inversions

For a random ordering, we would expect approximately half of all pairs, or

$$\frac{1}{2}\binom{n}{2} = \frac{n(n-1)}{4} = \mathbf{O}(n^2), \text{ inversions}$$

Number of Inversions

For example, the following unsorted list of 56 entries

61	548	3	923	195	973	289	237	57	299	594	928	515	55
507	351	262	797	788	442	97	798	227	127	474	825	7	182
929	852	504	485	45	98	538	476	175	374	523	800	19	901
349	947	613	265	844	811	636	859	81	270	697	563	976	539

has 655 inversions and 885 ordered pairs

The formula predicts $\frac{1}{2}\binom{56}{2} = \frac{56(56-1)}{4} = 770$ inversions

Number of Inversions

Let us consider the number of inversions in our first three lists:

1 16 12 26 25 35 33 58 45 42 56 67 83 75 74 86 81 88 99 95
1 17 21 42 24 27 32 35 45 47 57 23 66 69 70 76 87 85 95 99
22 20 81 38 95 84 99 12 79 44 26 87 96 10 48 80 1 31 16 92

Each list has 20 entries, and therefore:

- There are $\binom{20}{2} = \frac{20(20-1)}{2} = 190$ pairs
- On average, $190/2 = 95$ pairs would form inversions

Number of Inversions

The first list

1 16 12 26 25 35 33 58 45 42 56 67 83 75 74 86 81 88 99 95

has 13 inversions:

(16, 12) (26, 25) (35, 33) (58, 45) (58, 42) (58, 56) (45, 42)
(83, 75) (83, 74) (83, 81) (75, 74) (86, 81) (99, 95)

This is well below 95, the expected number of inversions

– Therefore, this is likely not to be a *random* list

Number of Inversions

The second list

1 17 21 42 24 27 32 35 45 47 57 23 66 69 70 76 87 85 95 99

also has 13 inversions:

(42, 24) (42, 27) (42, 32) (42, 35) (42, 23) (24, 23) (27, 23)
(32, 23) (35, 23) (45, 23) (47, 23) (57, 23) (87, 85)

This, too, is not a random list

Number of Inversions

The third list

22 20 81 38 95 84 **99** 12 79 44 26 87 96 10 48 80 **1** 31 16 92

has 100 inversions:

(22, 20) (22, 12) (22, 10) (22, **1**) (22, 16) (20, 12) (20, 10) (20, **1**) (20, 16) (81, 38)
(81, 12) (81, 79) (81, 44) (81, 26) (81, 10) (81, 48) (81, 80) (81, **1**) (81, 16) (81, 31)
(38, 12) (38, 26) (38, 10) (38, **1**) (38, 16) (38, 31) (95, 84) (95, 12) (95, 79) (95, 44)
(95, 26) (95, 87) (95, 10) (95, 48) (95, 80) (95, **1**) (95, 16) (95, 31) (95, 92) (84, 12)
(84, 79) (84, 44) (84, 26) (84, 10) (84, 48) (84, 80) (84, **1**) (84, 16) (84, 31) (**99**, 12)
(**99**, 79) (**99**, 44) (**99**, 26) (**99**, 87) (**99**, 96) (**99**, 10) (**99**, 48) (**99**, 80) (**99**, **1**) (**99**, 16)
(**99**, 31) (**99**, 92) (12, 10) (12, **1**) (79, 44) (79, 26) (79, 10) (79, 48) (79, **1**) (79, 16)
(79, 31) (44, 26) (44, 10) (44, **1**) (44, 16) (44, 31) (26, 10) (26, **1**) (26, 16) (87, 10)
(87, 48) (87, 80) (87, **1**) (87, 16) (87, 31) (96, 10) (96, 48) (96, 80) (96, **1**) (96, 16)
(96, 31) (96, 92) (10, **1**) (48, **1**) (48, 16) (48, 31) (80, **1**) (80, 16) (80, 31) (31, 16)

Summary

Introduction to sorting, including:

- Assumptions
- In-place sorting ($\mathbf{O}(1)$ additional memory)
- Sorting techniques
  - insertion, exchanging, selection, merging, distribution
- Run-time classification: $\mathbf{O}(n)$ $\mathbf{O}(n \ln(n))$ $\mathbf{O}(n^2)$

Overview of proof that a general sorting algorithm must be $\mathbf{\Omega}(n \ln(n))$

References

Wikipedia, http://en.wikipedia.org/wiki/Sorting_algorithm
http://en.wikipedia.org/wiki/Sorting_algorithm#Inefficient.2Fhumorous_sorts

[1] Donald E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd Ed., Addison Wesley, 1998, §5.1, 2, 3.

[2] Cormen, Leiserson, and Rivest, *Introduction to Algorithms*, McGraw Hill, 1990, p.137-9 and §9.1.

[3] Weiss, *Data Structures and Algorithm Analysis in C++, 3rd Ed.*, Addison Wesley, §7.1, p.261-2.

[4] Gruber, Holzer, and Ruepp, *Sorting the Slow Way: An Analysis of Perversely Awful Randomized Sorting Algorithms*, 4th International Conference on Fun with Algorithms, Castiglioncello, Italy, 2007.