

COMP2521 24T3

Sorting Algorithms (I)

Introduction to Sorting Algorithms

Sushmita Ruj

`cs2521@cse.unsw.edu.au`

sorting
properties of sorting algorithms

Motivation

Sorting

Analysis

Properties

Programming

- Sorting enables faster searching
 - Binary search
- Sorting arranges data in useful ways (for humans and computers)
 - For example, a list of students in a tutorial
- Sorting provides a useful intermediate for other algorithms
 - For example, duplicate detection/removal, merging two collections

- Sorting involves arranging a collection of items in order
 - **Arrays**, linked lists, files
- Items are sorted based on some property (called the **key**), using an ordering relation on that property
 - Numbers are sorted numerically
 - Strings are sorted alphabetically

We sort arrays of `Items`, which could be:

- Simple values: `int`, `char`, `double`
- Aggregate values: `strings`
- Structured values: `struct`

The items are sorted based on a **key**, which could be:

- The entire item, if the item is a single value
- One or more fields, if the item is a struct

Example: Each student has an ID and a name

5151515	5012345	3456789	5050505	5555555	5432109
John	Jane	Bob	Alice	John	Andrew

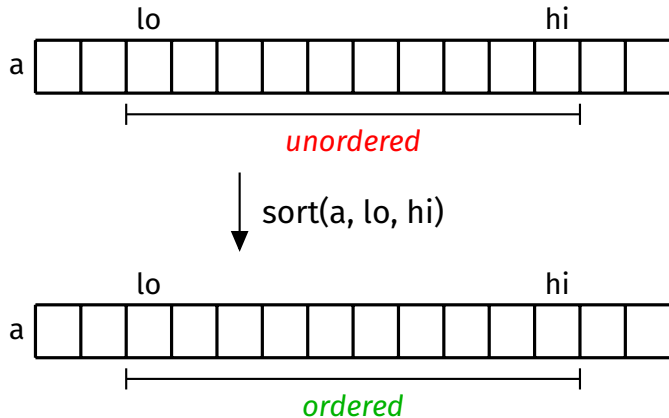
Sorting by ID (i.e., key is ID):

3456789	5012345	5050505	5151515	5432109	5555555
Bob	Jane	Alice	John	Andrew	John

Sorting by name (i.e., key is name):

5050505	5432109	3456789	5012345	5151515	5555555
Alice	Andrew	Bob	Jane	John	John

Arrange items in array slice $a[\text{lo}..hi]$ into sorted order:



To sort an entire array of size N , $lo == 0$ and $hi == N - 1$.

Motivation

Sorting

Analysis

Properties

Programming

Elementary sorting algorithms:

- Selection sort
- Bubble sort
- Insertion sort
- Shell sort

Divide-and-conquer sorting algorithms:

- Merge sort
- Quick sort

Non-comparison-based sorting algorithms:

- Radix sort
- Key-indexed counting sort

Motivation

Sorting

Analysis

Properties

Programming

Three main cases to consider for input order:

- Random order
- Sorted order
- Reverse-sorted order

When analysing sorting algorithms, we consider:

- n : the number of items ($h_i - l_o + 1$)
- C : the number of comparisons between items
- S : the number of times items are swapped

Motivation

Sorting

Analysis

Properties

Stability

Adaptability

In-place

Programming

Properties:

- Stability
- Adaptability
- In-place

Motivation

Sorting

Analysis

Properties

Stability

Adaptability

In-place

Programming

- A **stable** sort preserves the relative order of items with equal keys.
- **Formally:** For all pairs of items x and y where $\text{KEY}(x) \equiv \text{KEY}(y)$, if x precedes y in the original array, then x precedes y in the sorted array.

A stable sorting algorithm *always* performs a stable sort.

Motivation

Sorting

Analysis

Properties

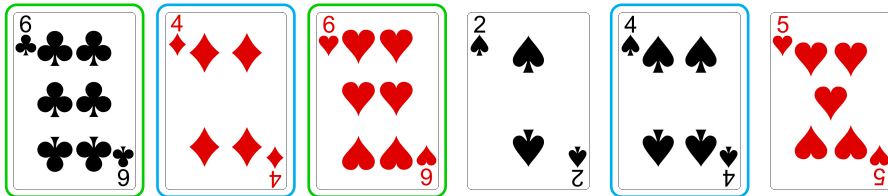
Stability

Adaptability

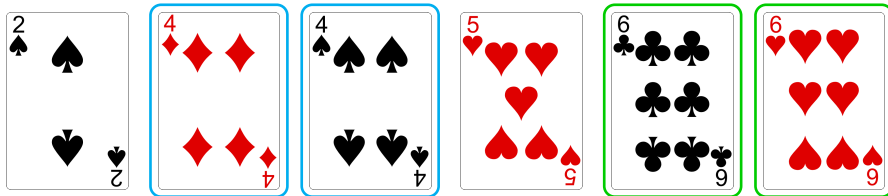
In-place

Programming

Example: Each card has a value and a suit



A stable sort on value:



Motivation

Sorting

Analysis

Properties

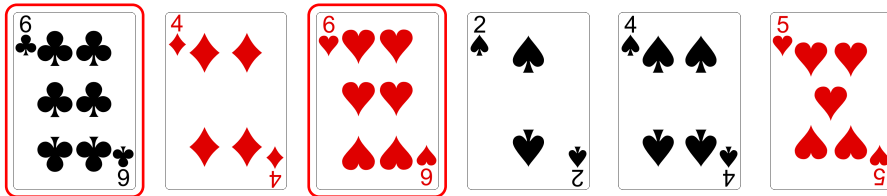
Stability

Adaptability

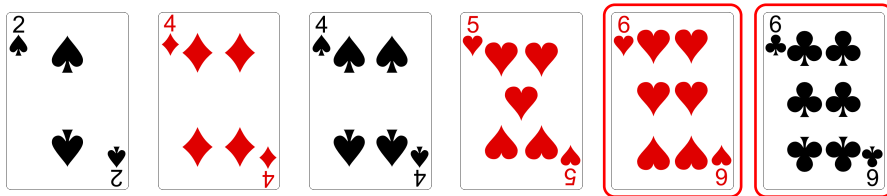
In-place

Programming

Example: Each card has a value and a suit



Example of an unstable sort on value:



Motivation

Sorting

Analysis

Properties

Stability

Adaptability

In-place

Programming

When is stability important?

- When sorting the same array multiple times on different keys
 - Some sorting algorithms rely on this, for example, radix sort

Motivation

Sorting

Analysis

Properties

Stability

Adaptability

In-place

Programming

Example: Array of first names and last names

Alice Wunder	Andrew Bennett	Jake Renzella	Alice Hatter	Andrew Taylor	John Shepherd
-----------------	-------------------	------------------	-----------------	------------------	------------------

Sort by last name:

Andrew Bennett	Alice Hatter	Jake Renzella	John Shepherd	Andrew Taylor	Alice Wunder
-------------------	-----------------	------------------	------------------	------------------	-----------------

Then sort by first name (using stable sort):

Alice Hatter	Alice Wunder	Andrew Bennett	Andrew Taylor	Jake Renzella	John Shepherd
-----------------	-----------------	-------------------	------------------	------------------	------------------

Motivation

Sorting

Analysis

Properties

Stability

Adaptability

In-place

Programming

Stability doesn't matter if...

- All items have unique keys
 - Example: Sorting students by ID
- The key is the entire item
 - Example: Sorting an array of integer values

Motivation

Sorting

Analysis

Properties

Stability

Adaptability

In-place

Programming

- An **adaptive** sorting algorithm takes advantage of existing order in its input
 - The nature of the algorithm allows sorted or nearly-sorted inputs to be sorted *much* quicker than other inputs

Motivation

Sorting

Analysis

Properties

Stability

Adaptability

In-place

Programming

Warning!

Just because a sorting algorithm
sorts sorted input faster than it sorts random input,
does not necessarily mean that it is adaptive.

Motivation

Sorting

Analysis

Properties

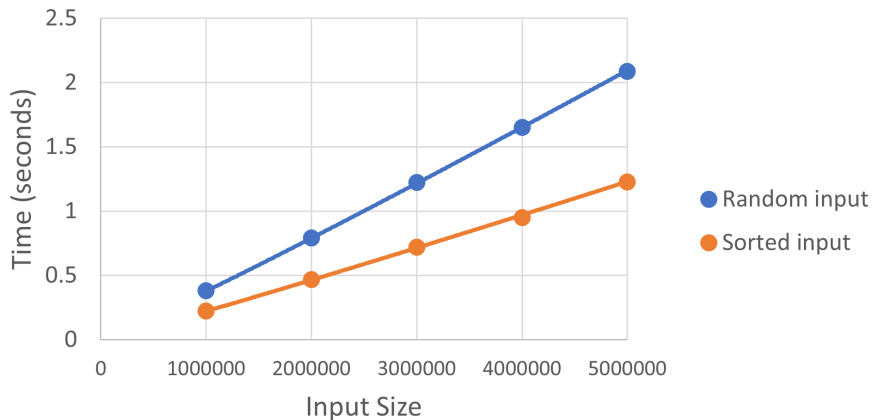
Stability

Adaptability

In-place

Programming

Example of data for non-adaptive sorting algorithm:



Motivation

Sorting

Analysis

Properties

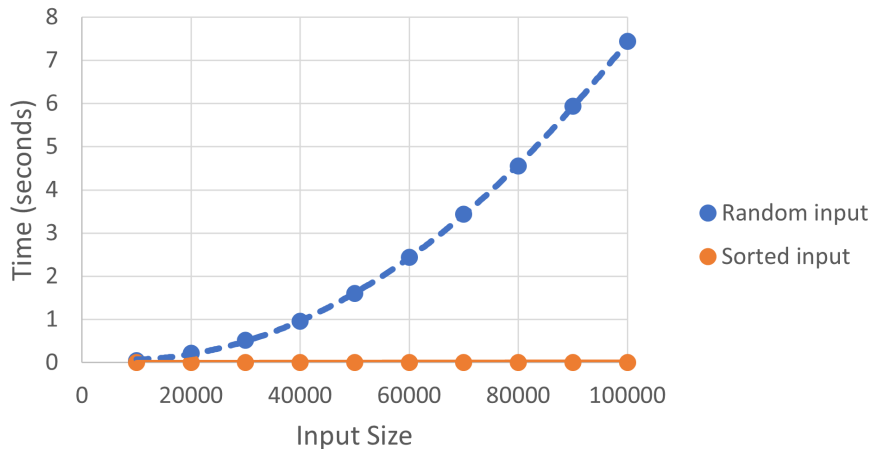
Stability

Adaptability

In-place

Programming

Example of data for adaptive sorting algorithm:



Motivation

Sorting

Analysis

Properties

Stability

Adaptability

In-place

Programming

- An **in-place** sorting algorithm sorts the data within the original structure, without using temporary arrays

Motivation

Sorting

Analysis

Properties

Programming

Generic sort function:

```
void sort(Item a[], int lo, int hi);
```

Helper function to swap elements at indices *i* and *j*:

```
void swap(Item a[], int i, int j);
```

Motivation

Sorting

Analysis

Properties

Programming

Item is a typedef,
which is a way to give a new name to a type.

For example, if we want to sort integers:

```
typedef int Item;
```

For example, if we want to sort strings:

```
typedef char *Item;
```

Motivation

Sorting

Analysis

Properties

Programming

We also define macros which indicate
(1) how to extract keys from an item, and
(2) how items should be compared.

For example, when sorting integers:

```
typedef int Item;

#define key(A) (A)
#define lt(A, B) (key(A) < key(B)) // less than
#define le(A, B) (key(A) <= key(B)) // less than or equal to
#define ge(A, B) (key(A) >= key(B)) // greater than or equal to
#define gt(A, B) (key(A) > key(B)) // greater than
```

Motivation

Sorting

Analysis

Properties

Programming

When sorting structs:

```
typedef struct {
    char *name;
    char *course;
} Item;

#define key(A) (A.name)
#define lt(A, B) (strcmp(key(A), key(B)) < 0)
#define le(A, B) (strcmp(key(A), key(B)) <= 0)
#define ge(A, B) (strcmp(key(A), key(B)) >= 0)
#define gt(A, B) (strcmp(key(A), key(B)) > 0)
```


Motivation

Sorting

Analysis

Properties

Programming

<https://forms.office.com/r/zEqxUXvmLR>

