Algorithms and Data Structures

Sorting algorithms

dr Szymon Murawski

Comarch SA

March 14, 2019

Table of contents I

Course plan

Sorting

- Sorting is a process of transforming a sequence of numbers $< x_1, x_2, \ldots, x_n >$ into another sequence $< x'_1, x'_2, \ldots, x'_n >$ such that $x'_1 \leq x'_2 \leq \ldots x'_n$.
- In reality we rarely sort just numbers - usually we sort records of data consisting of a key and satellite data - key is the value to be sorted
- Key can either be a number or a string. Be aware that lexicography order might depend on alphabet used!
- A sorting algorithm describes a method, omitting the unimportant (from the point of sorting) parts, i.e. efficiently moving data, determining sort method and order
- Because of the above we will assume that we are sorting plain numbers in ascending order order - translating an algorithm for sorting numbers into a program for sorting records is conceptually straightforward, although in a given engineering situation other subtleties may make the actual programming task a challenge
- Keys are stored as a plain number/text, accessing key should not require calculating them

Data representation

Direct

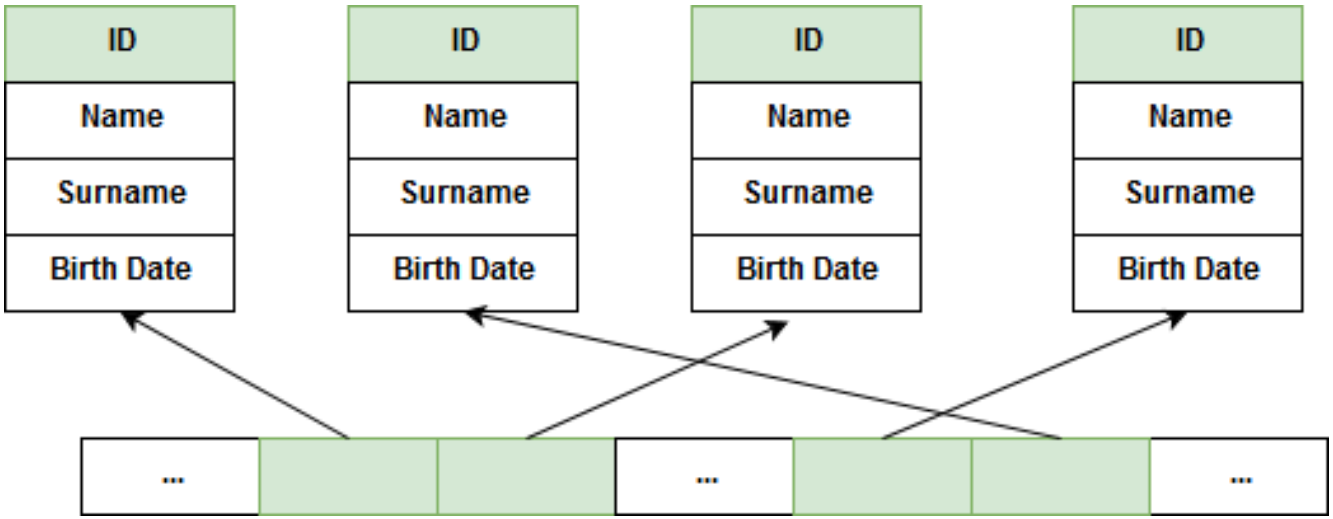
- Used when the record is quite small

ID	ID		ID	ID
Name	Name	...	Name	Name
Surname	Surname		Surname	Surname
Birth Date	Birth Date		Birth Date	Birth Date

Data representation

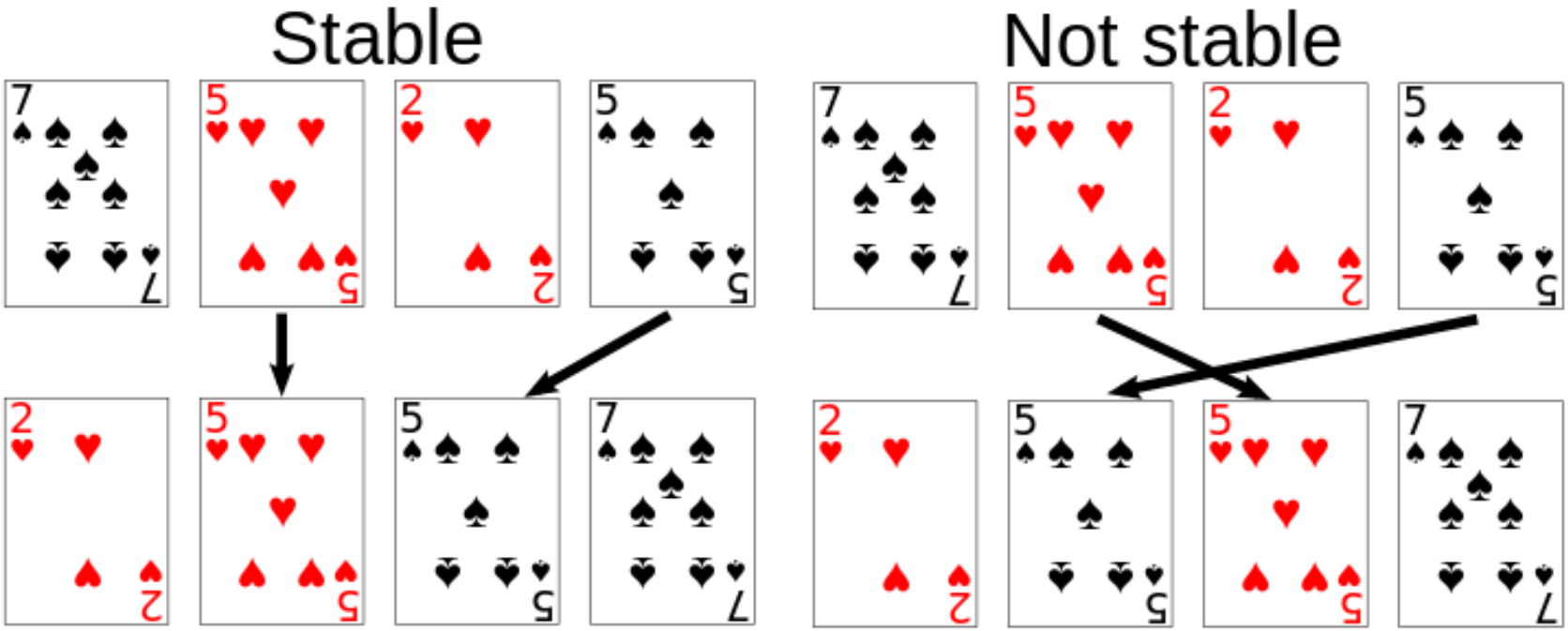
Indirect

- Used when the record is large, to minimize movement of data in memory
- Realized by pointers and references
- Using this technique we can store many sorting orders for one data structure!



Stability of sorting

Sorting algorithm is stable, if the relative position of records with same key is preserved before and after executing the algorithm



Sorting in or out of place

Depending on addition memory needed for sorting algorithm we can it is:

- In-place, when no additional memory is needed and output is produced in the same memory that contained data. In this types of algorithm output is produced by transforming input. Some extra space for variables is allowed however.
- Out-of-place, when the above constrains are broken. Usually when extra memory needed for an algorithm scales with size like $O(lgn)$ or worse we can say it is out-of-place.

Sorting efficiency

- Efficiency depends on
  - count of key comparisons
  - count of data movement
  - input data distribution
- Data distributions used when analyzing efficiency:
  - sorted in ascending order

1	2	3	4	5	6	7

  - Sorted in descending order

7	6	5	4	3	2	1

  - V-shape distribution

7	5	3	1	2	4	6

  - A-shape distribution

1	3	5	7	6	4	2

  - Constant

5	5	5	5	5	5	5

  - Random

7	4	9	2	1	8	2

Why sorting

Sorting is considered to be one of the most fundamental problems in the study of algorithms:

- Many applications inherently needs to sort information - banks need to sort transactions by their date
- Algorithms often use sorting as a key subroutine - when rendering graphics objects must be sorted to determine what is displayed on top
- There are many sorting algorithms based on many techniques. Those techniques are widely used in other, more complicated, problems
- Choosing the right algorithm for specific problem can be quite challenging - it can depend on initial data distribution, prior knowledge of the data, OS used, hardware constraints
- We can prove nontrivial lower bound for sorting and we can use that to prove lower bounds for other problems

https://www.youtube.com/watch?v=kPRA0W1kECg

Course plan

Insertion sort

- Commonly used when sorting cards in hand
- We divide data into two sequences: left (sorted) and right (unsorted)
- At every step we take the first record from the right sequence and put it into proper place into the left sequence

Pseudocode

- Input: Array A
- Output: Sorted array A

```
1    for( i = 1; i < length(A); i++)
2      j = i
3      while j>1 AND A[j-1] > A[j]
4        swap A[j] and A[j-1]
5        j--
6    return A
7
```

Insertion sort example

$i = 2$	44	55	12	42	94	18	6	67

Insertion sort example

$i = 2$	44	55	12	42	94	18	6	67
$i = 3$	44	55	12	42	94	18	6	67

Insertion sort example

$i = 2$	44	55	12	42	94	18	6	67
$i = 3$	44	55	12	42	94	18	6	67
$i = 4$	12	44	55	42	94	18	6	67

Insertion sort example

$i = 2$	44	55	12	42	94	18	6	67
$i = 3$	44	55	12	42	94	18	6	67
$i = 4$	12	44	55	42	94	18	6	67
$i = 5$	12	42	44	55	94	18	6	67

Insertion sort example

$i = 2$	44	55	12	42	94	18	6	67
$i = 3$	44	55	12	42	94	18	6	67
$i = 4$	12	44	55	42	94	18	6	67
$i = 5$	12	42	44	55	94	18	6	67
$i = 6$	12	42	44	55	94	18	6	67

Insertion sort example

$i = 2$	44	55	12	42	94	18	6	67
$i = 3$	44	55	12	42	94	18	6	67
$i = 4$	12	44	55	42	94	18	6	67
$i = 5$	12	42	44	55	94	18	6	67
$i = 6$	12	42	44	55	94	18	6	67
$i = 7$	12	18	42	44	55	94	6	67

Insertion sort example

$i = 2$	44	55	12	42	94	18	6	67
$i = 3$	44	55	12	42	94	18	6	67
$i = 4$	12	44	55	42	94	18	6	67
$i = 5$	12	42	44	55	94	18	6	67
$i = 6$	12	42	44	55	94	18	6	67
$i = 7$	12	18	42	44	55	94	6	67
$i = 8$	6	12	18	42	44	55	94	67

Insertion sort example

$i = 2$	44	55	12	42	94	18	6	67
$i = 3$	44	55	12	42	94	18	6	67
$i = 4$	12	44	55	42	94	18	6	67
$i = 5$	12	42	44	55	94	18	6	67
$i = 6$	12	42	44	55	94	18	6	67
$i = 7$	12	18	42	44	55	94	6	67
$i = 8$	6	12	18	42	44	55	94	67
$i = 9$	6	12	18	42	44	55	67	94

Insertion sort analysis

- Easy to implement
- Complexity $O(n^2)$
- Natural behavior - optimistic case is data already sorted in ascending order, pessimistic case - data is sorted in reversed order.
- Stable sorting
- In-place sorting
- We can optimize algorithm by first finding a place to put new element and then moving large chunk of array at once
- For fastest finding place to put element we can use binary search!

Binary insertion sort

Pseudocode

- Input: Array A
- Output: Sorted array A

```
1    for( i = 1; i < length(A); i++)
2      left = 0, right = i-1, middle
3      while left <= right
4        middle = (left + right) / 2
5        if (A[i] < A[middle]) right = middle - 1
6        else left = middle + 1
7      buf = A[i]
8      Move Array from left to left + 1
9      A[left] = buf
10   return A
11
```

Course plan

Selection sort

- Inverse approach compared to the previous algorithm
- At each step we search for minimal value in unsorted sequence and place it at the end of the sorted sequence

Pseudocode

- Input: Array A
- Output: Sorted array A

```
1    for( i = 0; i < length(A); i++)
2      min_index = i
3      for ( j = i + 1; j < length(A); j++)
4        min_index = A[min_index] < A[j] ? min_index : j
5      swap A[i] with A[min_index]
6    return A
7
```

Selection sort example

$i = 1$	44	55	12	42	94	18	6	67

Selection sort example

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	6	55	12	42	94	18	44	67

Selection sort example

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	6	55	12	42	94	18	44	67
$i = 3$	6	12	55	42	94	18	44	67

Selection sort example

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	6	55	12	42	94	18	44	67
$i = 3$	6	12	55	42	94	18	44	67
$i = 4$	6	12	18	42	94	55	44	67

Selection sort example

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	6	55	12	42	94	18	44	67
$i = 3$	6	12	55	42	94	18	44	67
$i = 4$	6	12	18	42	94	55	44	67
$i = 5$	6	12	18	42	94	55	44	67

Selection sort example

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	6	55	12	42	94	18	44	67
$i = 3$	6	12	55	42	94	18	44	67
$i = 4$	6	12	18	42	94	55	44	67
$i = 5$	6	12	18	42	94	55	44	67
$i = 6$	6	12	18	42	44	55	94	67

Selection sort example

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	6	55	12	42	94	18	44	67
$i = 3$	6	12	55	42	94	18	44	67
$i = 4$	6	12	18	42	94	55	44	67
$i = 5$	6	12	18	42	94	55	44	67
$i = 6$	6	12	18	42	44	55	94	67
$i = 7$	6	12	18	42	44	55	94	67
$i = 8$	6	12	18	42	44	55	67	94

Selection sort analysis

- Very easy to implement
- Complexity $O(n^2)$
- Natural behavior - optimistic case is data already sorted in ascending order, pessimistic case - data is sorted in reversed order.
- Unstable sorting
- In-place sorting
- The worst time complexity of all sorting algorithms, but data movement is only $O(n)$ - this algorithm could be useful if records are very big, or read/write access is heavily constrained

Course plan

Bubble sort

- At each step we compare and swap pairs - bubbles
- At the end of each loop the smallest element ("lightest bubble") goes to the top of the table

Pseudocode

- Input: Array A
- Output: Sorted array A

```
1    for( i = 1; i < length(A); i++)
2      for ( j = length(A); j > 1; j - -)
3        if A[j] < A[j-1]
4            swap A[j] with A[j-1]
5    return A
```

Bubble sort example (one loop)

$i = 1$	44	55	12	42	94	18	6	67

Bubble sort example (one loop)

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	44	55	12	42	94	18	6	67

Bubble sort example (one loop)

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	44	55	12	42	94	18	6	67
$i = 3$	44	55	12	42	94	6	18	67

Bubble sort example (one loop)

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	44	55	12	42	94	18	6	67
$i = 3$	44	55	12	42	94	6	18	67
$i = 4$	44	55	12	42	6	94	18	67

Bubble sort example (one loop)

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	44	55	12	42	94	18	6	67
$i = 3$	44	55	12	42	94	6	18	67
$i = 4$	44	55	12	42	6	94	18	67
$i = 5$	44	55	12	6	42	94	18	67

Bubble sort example (one loop)

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	44	55	12	42	94	18	6	67
$i = 3$	44	55	12	42	94	6	18	67
$i = 4$	44	55	12	42	6	94	18	67
$i = 5$	44	55	12	6	42	94	18	67
$i = 6$	44	55	6	12	42	94	18	67

Bubble sort example (one loop)

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	44	55	12	42	94	18	6	67
$i = 3$	44	55	12	42	94	6	18	67
$i = 4$	44	55	12	42	6	94	18	67
$i = 5$	44	55	12	6	42	94	18	67
$i = 6$	44	55	6	12	42	94	18	67
$i = 7$	44	6	55	12	42	94	18	67

Bubble sort example (one loop)

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	44	55	12	42	94	18	6	67
$i = 3$	44	55	12	42	94	6	18	67
$i = 4$	44	55	12	42	6	94	18	67
$i = 5$	44	55	12	6	42	94	18	67
$i = 6$	44	55	6	12	42	94	18	67
$i = 7$	44	6	55	12	42	94	18	67
$i = 8$	6	44	55	12	42	94	18	67

Bubble sort example

$i = 1$	44	55	12	42	94	18	6	67

Bubble sort example

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	6	44	55	12	42	94	18	67

Bubble sort example

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	6	44	55	12	42	94	18	67
$i = 3$	6	12	44	55	18	42	94	67

Bubble sort example

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	6	44	55	12	42	94	18	67
$i = 3$	6	12	44	55	18	42	94	67
$i = 4$	6	12	18	44	55	42	67	94

Bubble sort example

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	6	44	55	12	42	94	18	67
$i = 3$	6	12	44	55	18	42	94	67
$i = 4$	6	12	18	44	55	42	67	94
$i = 5$	6	12	18	42	44	55	67	94

Bubble sort example

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	6	44	55	12	42	94	18	67
$i = 3$	6	12	44	55	18	42	94	67
$i = 4$	6	12	18	44	55	42	67	94
$i = 5$	6	12	18	42	44	55	67	94
$i = 6$	6	12	18	42	44	55	67	94

Bubble sort example

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	6	44	55	12	42	94	18	67
$i = 3$	6	12	44	55	18	42	94	67
$i = 4$	6	12	18	44	55	42	67	94
$i = 5$	6	12	18	42	44	55	67	94
$i = 6$	6	12	18	42	44	55	67	94
$i = 7$	6	12	18	42	44	55	67	94

Bubble sort example

$i = 1$	44	55	12	42	94	18	6	67
$i = 2$	6	44	55	12	42	94	18	67
$i = 3$	6	12	44	55	18	42	94	67
$i = 4$	6	12	18	44	55	42	67	94
$i = 5$	6	12	18	42	44	55	67	94
$i = 6$	6	12	18	42	44	55	67	94
$i = 7$	6	12	18	42	44	55	67	94
$i = 8$	6	12	18	42	44	55	67	94

Bubble sort analysis

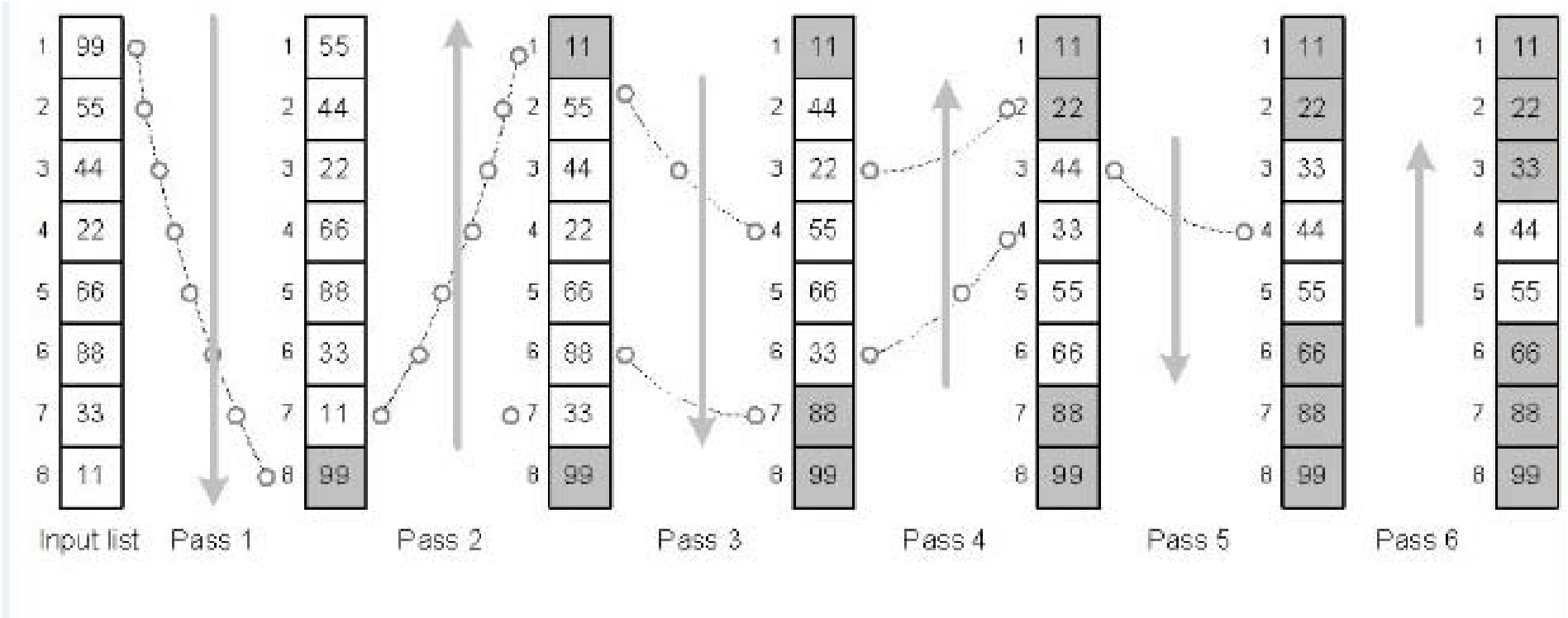
- Very easy to implement
- Complexity $O(n^2)$
- Natural behavior - optimistic case is data already sorted in ascending order, pessimistic case - data is sorted in reversed order.
- Unstable sorting
- In-place sorting
- We can reduce the running time by tracking whether swap happened and break the for loop if it did not
- There is asymmetry in the movement of light and heavy bubbles, to increase performance we can alternate between moving light and heavy bubbles

Course plan

Cocktail sort

- Improved bubble sort
- Also known as two-way bubble sort, shaker sort
- Array is traversed in both directions,
- If no swaps took place the algorithm terminates

Coctail sort example



Cocktail sort pseudocode

Pseudocode

- Input: Array A
- Output: Sorted array A

```
1   swapped = true, start = 0, end = length(A)
2   while (swapped)
3     swapped = false
4     for (i = start; i < end; i++)
5       if A[i] > A[i+1]
6         swap A[i] with A[i+1]
7         swapped = true
8     break if !swapped
9     end--
10    for (i = end -1; i >=start; --i)
11      if A[i] > A[i+1]
12        swap A[i] with A[i+1]
13        swapped = true
14    start++;
```

Cocktail sort analysis

- Quite easy to implement
- Complexity $O(n^2)$
- Natural behavior - optimistic case is data already sorted in ascending order, pessimistic case - data is sorted in reversed order.
- Stable sorting
- In-place sorting
- Approximately two times faster than bubble sort
- There is asymmetry in the movement of light and heavy bubbles, to increase performance we can alternate between moving light and heavy bubbles
- If data is initially mostly ordered such that every element is at a position that by at most k from the position it is going to end up in, the complexity becomes $O(kn)$

Course plan

Summary of simple sorting algorithms

Name	Best	Average	Worst	Memory	Stable	In-place
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Yes	Yes
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	No	Yes
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Yes	Yes
Cocktail sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Yes	Yes