ETH zürich

Departement Informatik

Programming and Problem-Solving

Reading in data and Sorting 1

Dennis Komm

Spring 2021 – April 1, 2021

Lists

Advanced Concepts

2-Dimensional Lists

So far lists contain numbers or characters

2-Dimensional Lists

So far lists contain numbers or characters

- Lists can contain lists
- Such **2-dimensional lists** store, e.g., tables and matrices

2-Dimensional Lists

So far lists contain numbers or characters

- Lists can contain lists
- Such **2-dimensional lists** store, e.g., tables and matrices

$$M = \begin{pmatrix} 2 & 0 & 3 & 0 & 6 \\ 3 & 9 & 5 & 1 & 1 \\ 0 & 0 & 7 & 2 & 7 \\ 3 & 9 & 5 & 8 & 0 \\ 8 & 2 & 0 & 3 & 2 \\ 1 & 6 & 5 & 9 & 6 \end{pmatrix}$$

```
M = [ [2, 0, 3, 0, 6],
      [3, 9, 5, 1, 1],
      [0, 0, 7, 2, 7],
      [3, 9, 5, 8, 0],
      [8, 2, 0, 3, 2],
      [1, 6, 5, 9, 6] ]
```

2-Dimensional Lists

So far lists contain numbers or characters

- Lists can contain lists
- Such **2-dimensional lists** store, e.g., tables and matrices

$$M = \begin{pmatrix} 2 & 0 & 3 & 0 & 6 \\ 3 & 9 & 5 & 1 & 1 \\ 0 & 0 & 7 & 2 & 7 \\ 3 & 9 & 5 & 8 & 0 \\ 8 & 2 & 0 & 3 & 2 \\ 1 & 6 & 5 & 9 & 6 \end{pmatrix}$$

```
M = [ [2, 0, 3, 0, 6],
      [3, 9, 5, 1, 1],
      [0, 0, 7, 2, 7],
      [3, 9, 5, 8, 0],
      [8, 2, 0, 3, 2],
      [1, 6, 5, 9, 6] ]
```

- Accessing line `i` and column `j` with `M[i][j]`

Lists

Reading in Data und Saving it to Lists

Reading in Data

Example: Matrix given in file

- Content of the file is a text
- Matrix stored line by line
- Entries in each line separated by commas
- Entries are to be interpreted as numbers

Reading in Data

Example: Matrix given in file

- Content of the file is a text
- Matrix stored line by line
- Entries in each line separated by commas
- Entries are to be interpreted as numbers

Three Steps

1. Read in file line by line
2. Extract entries from the lines (separator symbol: comma)
3. Convert each entry to a number

Reading in Data

1. Read in file line by line

```
with open("data.txt") as file:
```

Reading in Data

1. Read in file line by line

```
with open("data.txt") as file:
```

- File `data.txt` is opened for the following block of instructions
- Accessible under the name `file`

Reading in Data

1. Read in file line by line

```
with open("data.txt") as file:
    lines = file.read().splitlines()
```

- File `data.txt` is opened for the following block of instructions
- Accessible under the name `file`
- `lines = file.read()` stores the whole content of `data.txt` in the variable `lines`

Reading in Data

1. Read in file line by line

```
with open("data.txt") as file:
    lines = file.read().splitlines()
```

- File `data.txt` is opened for the following block of instructions
- Accessible under the name `file`
- `lines = file.read()` stores the whole content of `data.txt` in the variable `lines`
- `lines = file.read().splitlines()` stores the individual lines of `daten.txt` in the list `lines`

Reading in Data: Example

1. Read in file line by line

```
with open("data.txt") as file:
    lines = file.read().splitlines()
```

data.txt

```
2, 0, 3, 0, 6
3, 9, 5, 1, 1
0, 0, 7, 2, 7
3, 9, 5, 8, 0
8, 2, 0, 3, 2
1, 6, 5, 9, 6
```

```
lines = [ "2, 0, 3, 0, 6",
          "3, 9, 5, 1, 1",
          "0, 0, 7, 2, 7",
          "3, 9, 5, 8, 0",
          "8, 2, 0, 3, 2",
          "1, 6, 5, 9, 6" ]
```

Reading in Data

2. Extract entries from first line (separator symbol: comma)

```
tmp = lines[0].split(",")
```

Reading in Data

2. Extract entries from first line (separator symbol: comma)

```
tmp = lines[0].split(",")
```

```
lines = [ "2, 0, 3, 0, 6",
          "3, 9, 5, 1, 1",
          "0, 0, 7, 2, 7",
          "3, 9, 5, 8, 0",
          "8, 2, 0, 3, 2",
          "1, 6, 5, 9, 6" ]
```

```
tmp = ["2", "0", "3", "0", "6"]
```

Reading in Data

3. Convert each entry to a number

```
data = [0] * len(tmp)
for i in range(0, len(tmp)):
    data[i] = int(tmp[i])
```

Reading in Data

3. Convert each entry to a number

```
data = [0] * len(tmp)
for i in range(0, len(tmp)):
    data[i] = int(tmp[i])
```

```
tmp = ["2", "0", "3", "0", "6"]
```

```
data = [2, 0, 3, 0, 6]
```

Reading in Data: Summary

```
def readfile(filename):

    # Read in file line by line
    with open(filename) as file:
        lines = file.read().splitlines()

    # Extract entries from first line (separator symbol:  comma)
    tmp = lines[0].split(",")

    # Convert each entry to a number
    data = [0] * len(tmp)
    for i in range(0, len(tmp)):
        data[i] = int(tmp[i])

    return data
```

Exercise – Reading in Data

Extend the function so that

- all lines of the file are read and converted
- the content is stored in a 2-dimensional list

```
def readfile(filename):
    with open(filename) as file:
        lines = file.read().splitlines()
    tmp = lines[0].split(",")
    data = [0] * len(tmp)
    for i in range(0, len(tmp)):
        data[i] = int(tmp[i])
    return data
```

Reading in Data

```
def readfile2(filename):
    # Read in file line by line
    with open(filename) as file:
        lines = file.read().splitlines()
    data = []

    # Process all lines successively
    for i in range(0, len(lines)):
        tmp = lines[i].split(",")
        dataline = [0] * len(tmp)
        for j in range(0, len(tmp)):
            dataline[j] = int(tmp[j])
        data.append(dataline)
    return data
```

Reading in Data

```
def readfile2(filename):
    # Read in file line by line
    with open(filename) as file:
        lines = file.read().splitlines()
    data = []

    # Process all lines successively
    for i in range(0, len(lines)):
        tmp = lines[i].split(",")
        dataline = [0] * len(tmp)
        for j in range(0, len(tmp)):
            dataline[j] = int(tmp[j])
        data.append(dataline)
    return data
```

Data is often supplied as such `csv` files (comma separated values)

Sorting 1

Sorting and Searching

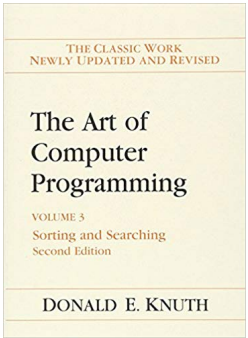
Sorting and Searching

Sorting and searching data are two of the fundamental tasks of computer scientists

Sorting and Searching

Sorting and searching data are two of the fundamental tasks of computer scientists

Standard reference only deals with these topics

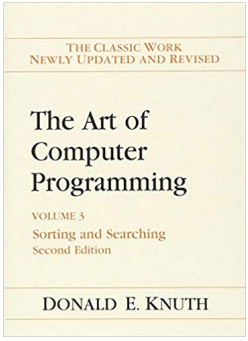


Sorting and Searching

Sorting and searching data are two of the fundamental tasks of computer scientists

Standard reference only deals with these topics

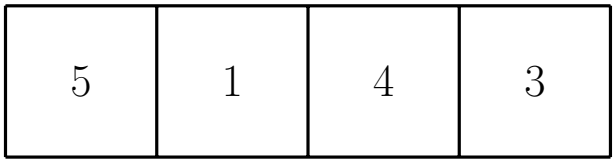
- Given n positive integers
- Specifically, **unsorted** list `data` with `n = len(data)`
- We consider n as input length
- Numbers may appear multiple times
- **Sort** numbers in as little time as possible



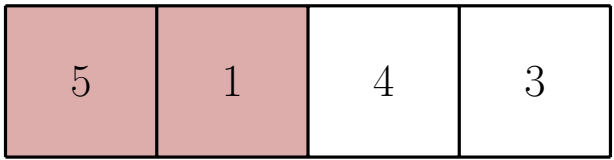
Sorting 1

Bubblesort

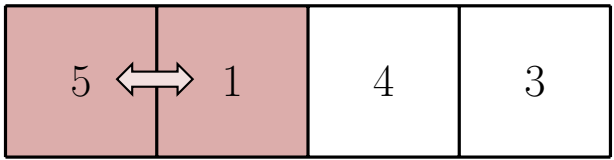
Bubblesort



Bubblesort

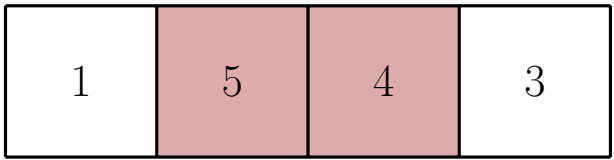


Bubblesort

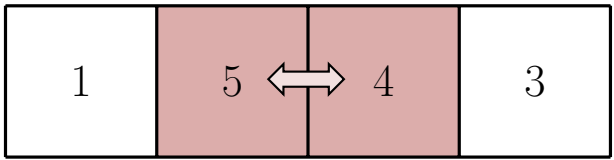


Bubblesort

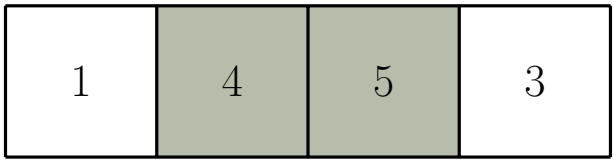
Bubblesort



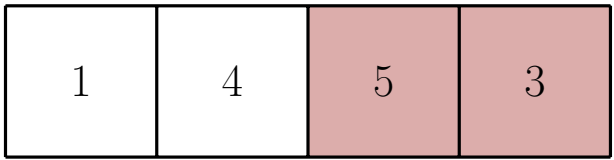
Bubblesort



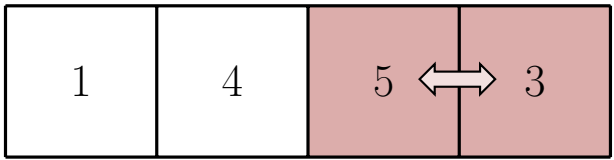
Bubblesort



Bubblesort

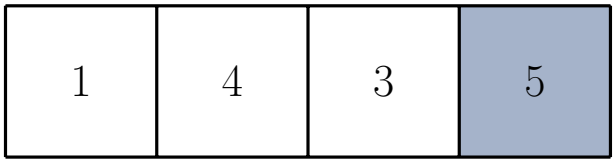


Bubblesort

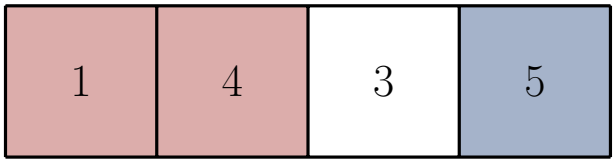


Bubblesort

Bubblesort

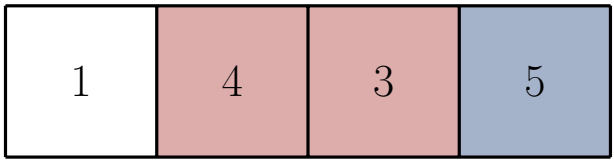


Bubblesort

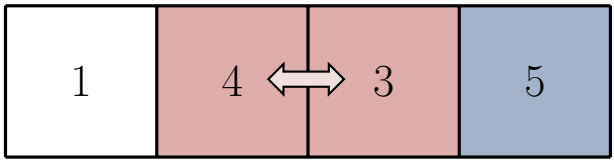


Bubblesort

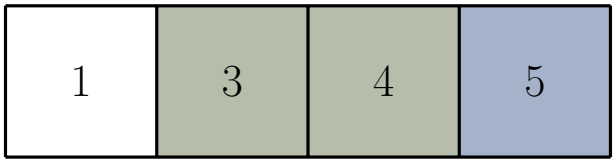
Bubblesort



Bubblesort

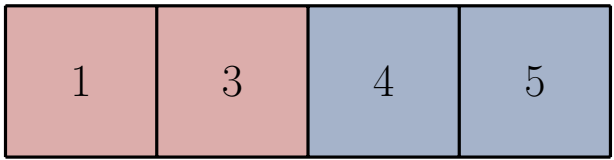


Bubblesort

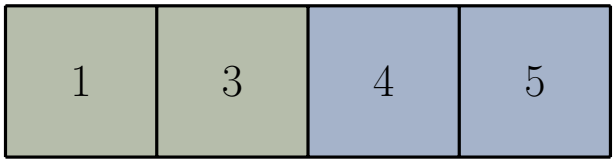


Bubblesort

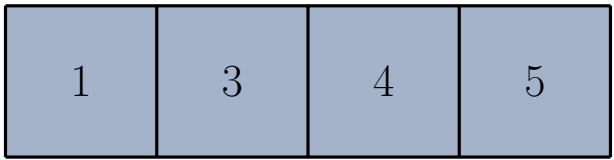
Bubblesort



Bubblesort



Bubblesort



Bubblesort

Idea

Sorting by repeatedly finding the maximum

Bubblesort

Idea

Sorting by repeatedly finding the maximum

Goal

Sort list `data` with `n` elements, i.e., range $0, \ldots, n-1$

Bubblesort

Idea

Sorting by repeatedly finding the maximum

Goal

Sort list `data` with `n` elements, i.e., range $0, \ldots, n-1$

- Find maximum and slide it to the last position
- To this end, iteratively compare neighboring elements

Bubblesort

Idea

Sorting by repeatedly finding the maximum

Goal

Sort list `data` with `n` elements, i.e., range $0, \ldots, n-1$

- Find maximum and slide it to the last position
- To this end, iteratively compare neighboring elements
- Maximum travels through list to the last position – like a bubble

Bubblesort

Idea

Sorting by repeatedly finding the maximum

Goal

Sort list `data` with `n` elements, i.e., range $0, \ldots, n-1$

- Find maximum and slide it to the last position
- To this end, iteratively compare neighboring elements
- Maximum travels through list to the last position – like a bubble
- Repeat with range $0, \ldots, n-2$

Bubblesort

Idea

Sorting by repeatedly finding the maximum

Goal

Sort list `data` with `n` elements, i.e., range $0, \ldots, n-1$

- Find maximum and slide it to the last position
- To this end, iteratively compare neighboring elements
- Maximum travels through list to the last position – like a bubble
- Repeat with range $0, \ldots, n-2$
- Continue until `data` is sorted

Exercise – One Bubble Sequence

Implement one Bubble Sequence

- Run through `data` one time
- Compare neighboring elements
- Swap if the first element is larger
- Maximum bubbles to the right

One Bubble Sequence

One Bubble Sequence in Python

```
data = [6, 22, 61, 1, 89, 31, 9, 10, 76]
n = len(data)

for i in range(0, n-1):
    if data[i] > data[i+1]:
        tmp = data[i]
        data[i] = data[i+1]
        data[i+1] = tmp
```

Exercise – Bubblesort

Implement the complete algorithm

- Iterate bubble sequences
- After ith sequence, the last k elements of `data` are sorted
- Bubble sequences become shorter with each iteration
- To this end, use outer loop

Bubblesort

```
def bubblesort(data):
    n = len(data)
    for d in range(n, 1, -1):
        for i in range(0, d-1):
            if data[i] > data[i+1]:
                tmp = data[i]
                data[i] = data[i+1]
                data[i+1] = tmp
    return data


print(bubblesort([6, 22, 61, 1, 89, 31, 9, 10, 76]))
```

Sorting 1

Minsort

Minsort

Idea

Sorting by repeatedly finding the minimum

Minsort

Idea

Sorting by repeatedly finding the minimum

- Unlike Bubblesort, we do not compare neighboring elements
- Current minimum is stored
- Each element is compared to it
- If it is smaller, both are swapped

Minsort

Idea

Sorting by repeatedly finding the minimum

- Unlike Bubblesort, we do not compare neighboring elements
- Current minimum is stored
- Each element is compared to it
- If it is smaller, both are swapped
- After one iteration, the minimum is copied to (current) first position
- Continue until `data` is sorted

Minsort

```
def minsort(data):
    n = len(data)
    for current in range(0, n-1):
        minimum = data[current]
        for i in range(current+1, n):
            if data[i] < minimum:
                tmp = data[i]
                data[i] = minimum
                minimum = tmp
        data[current] = minimum
    return data

print(minsort([6, 22, 61, 1, 89, 31, 9, 10, 76]))
```

Sorting 1

Time Complexity of Bubblesort

Time Complexity of Bubblesort

Count comparisons of two numbers

Time Complexity of Bubblesort

Count comparisons of two numbers

- $n - 1$ comparisons to find maximum

Time Complexity of Bubblesort

Count comparisons of two numbers

- $n - 1$ comparisons to find maximum
- $n - 2$ comparisons to find second largest element

Time Complexity of Bubblesort

Count comparisons of two numbers

- $n - 1$ comparisons to find maximum
- $n - 2$ comparisons to find second largest element
- ...

Time Complexity of Bubblesort

Count comparisons of two numbers

- $n - 1$ comparisons to find maximum
- $n - 2$ comparisons to find second largest element
- ...
- 1 comparison to find smallest element

Time Complexity of Bubblesort

Count comparisons of two numbers

- $n - 1$ comparisons to find maximum
- $n - 2$ comparisons to find second largest element
- $\ldots$
- 1 comparison to find smallest element

⇨ $\sum_{i=1}^{n-1} i = (n-1) \cdot n/2 = (n^2 - n)/2$ comparisons in total

⇨ Quadratic number of comparisons

Time Complexity of Bubblesort

Count comparisons of two numbers

- $n - 1$ comparisons to find maximum
- $n - 2$ comparisons to find second largest element
- ...
- 1 comparison to find smallest element

⇨ $\sum_{i=1}^{n-1} i = (n-1) \cdot n/2 = (n^2 - n)/2$ comparisons in total

⇨ Quadratic number of comparisons

The time complexity of Bubblesort is in $\mathcal{O}(n^2)$

Time Complexity of Bubblesort

Count comparisons of two numbers

- $n - 1$ comparisons to find maximum
- $n - 2$ comparisons to find second largest element
- $\ldots$
- 1 comparison to find smallest element

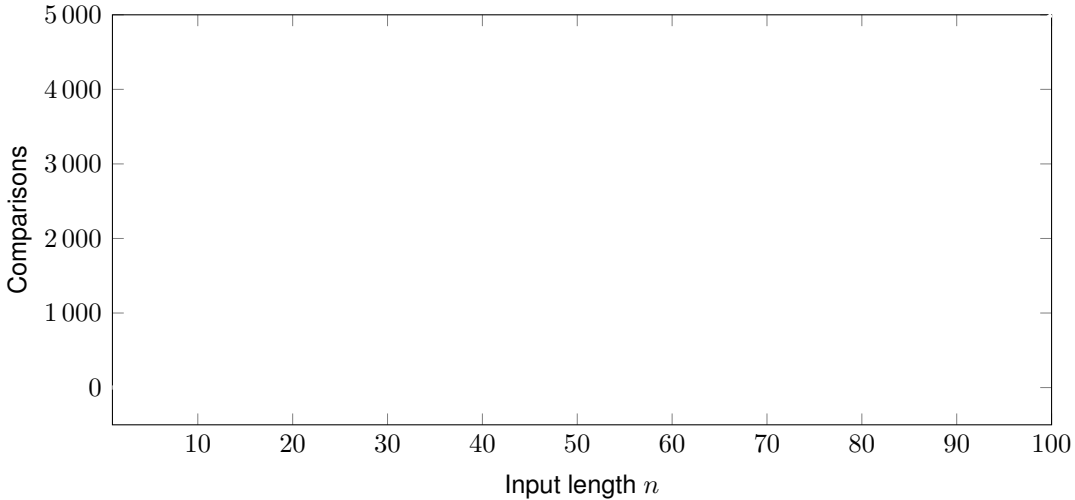
⇨ $\sum_{i=1}^{n-1} i = (n-1) \cdot n/2 = (n^2 - n)/2$ comparisons in total

⇨ Quadratic number of comparisons

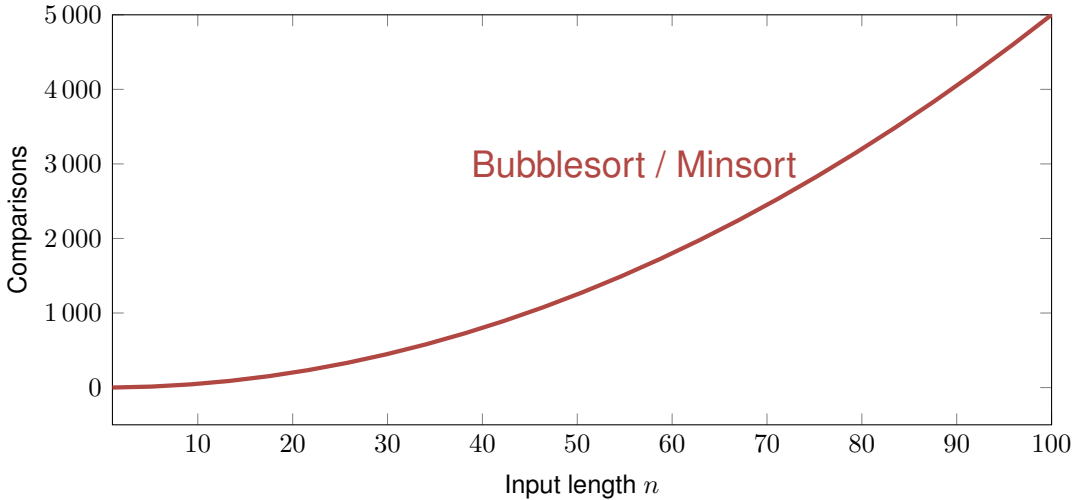
The time complexity of Bubblesort is in $\mathcal{O}(n^2)$

With similar arguments, the time complexity of Minsort is in $\mathcal{O}(n^2)$

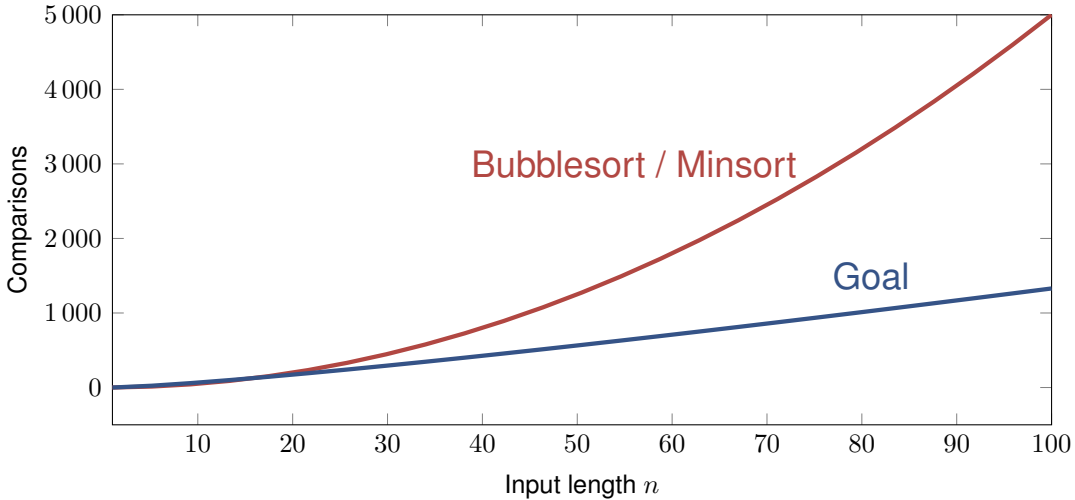
Time Complexity of Bubblesort



Time Complexity of Bubblesort



Time Complexity of Bubblesort



Thanks for your attention