FACULTEIT ECONOMIE EN BEDRIJFSKUNDE

UGENT BUSINESS INFORMATICS

ALGORITHMS AND DATA STRUCTURES

Frederik Gailly

UNIVERSITEIT GENT

PRACTICAL STUFF

UNIVERSITEIT
GENT

TOPICS A&D

- Algorithms and Data Structures
- Files & Exception handling
- Tuples, Sets and Dictionaries
- Recursion
- Efficient Algorithms

- Web Application Development
  (only for group assignment)

PLANNING

Week	Date	Time	Content
Week 1	23rd September 2025	11.30 ➔12.45	On Campus: Intro + Practicalities
	24th September 2025	08.30 ➔09.45	Online: Dodona exercises
Week 2	30th September 2025	11.30 ➔12.45	On campus: Files & Exceptions / Tuples, Sets and Dicts
	1st October 2025	08.30 ➔9.45	Online: Dodona exercises
Week 3	7th October 2025	11.30 ➔12.45	On campus: Recursion
	8th October 2025	08.30 ➔9.45	Online: Dodona exercises
Week 4	14th October 2025	11.30 ➔12.45	On campus: Efficient Algorithms
	15th October 2025	08.30 ➔9.45	Online: Dodona exercises
Week 5	21st October 2025	11.00 ➔12.15	On campus: Insertion Sort + Linked List
	22nd October 2025	08.30 ➔9.45	Online: Dodona exercises
Week 6	28th October 2025	11.00 ➔12.15	Project time
	29th October 2025	08.30 ➔9.45	Project time
Week 7	4th November 2025	11.00 ➔12.15	On Campus: Flask Python Web Application
	5th November 2025	08.30 ➔9.45	Github

PLANNING

Week	Datum	Uur	Inhoud
Week 8	11th November 2025	11:30 ➔12.45	No lecture
	12th November 2025	08.30 ➔09.45	Project time
Week 9	18th November 2025	11:30 ➔12.45	No Lecture
	19th November 2025	08.30 ➔13.00	No Lecture
	19th November 2025	17.00 ➔ 20.00	**Exam**
Week 10	25th November 2025	11.30 ➔12.45	Project time
	26th November 2025	08.30 ➔09.45	Project time
Week 11	2nd December 2025	11.30 ➔12.45	Project time
	3rd December 2025	08.30 ➔13.00	On campus Feedback project
Week 12	9th December 2025	11.00 ➔12.15	Project time
	10th December 2025	08.30 ➔13.00	Project time
Week 12	**19th December 2025**	**11.00**	**Deadline Group Assignment**

ON CAMPUS LECTURE

- Tuesday 11.30 ➔ 12.45u
- Where?
  - Week 1 Aud D - Plateau
  - Week 2 - … : Aud Picard – Campus Tweekerken
- Lecture will be recorded
- Week 1 until week 5

ONLINE SESSIONS

- Wednesday 8.30 tot 09.45
- Online via Course MS Teams
- Goals:
  - Time to work on exercise
  - Ask questions
  - Prepare for the exam

FEEDBACK PROJECT

- Starts week 7
- Reserve a slot with Bookings with me
- Feedback Sessions Teaching Assistent

EVALUATION

- 50% project
- 50% PC exam

UNIVERSITEIT
GENT

EVALUATION

A&D	Database Systems	Evaluation	Remarls
Part of curriculum	Part of curriculum	• IS project • DBS exam • A&D exam	Grades seperate parts can be transfered if >9
Part of curriculum	Exemption	• A&D exam • Individual or group project	Grades seperate parts can be transfered if >9
Exemption	Part of curriculum	• DBS exam • Individual or group project	Grades seperate parts can be transfered if >9
Next year part of curriculum	Part of curriculum	• DBS exam	Grade DBS are transfered to next year if > 9
Part of curriculum	Next year part of curriculum	• A&D exam	Grade A&D are transfered to next year if > 9

EXAM ON COMPUTER

- 19 november 2025
- Open book
- Programming assignment
  - Design partly given
  - Focus is on implementation algorithms and data structures

PROJECT

- Joint project A&D and Database systemen
- Two parts
  - Part 1:
    - Analysis requierments IS
    - Model, design and implementation of relational database
  - Part 2: Design, programming and testing IS

PROJECT

- Important dates:
  - 12/10/2025: deadline creation groups
  - 13/10/2025: publication assignment
  - 19/12/2025: Deadline
  - Last week exams: Project defense
- Every group 6 students
  - Create groups in ufora
  - No exceptions

PROJECT

- Evaluation
  - 30% joint evaluation
    - een peerassessment
    - Presentation slides
    - Oral presentation + defense
  - 70% linked to course content:
    - design
    - code
    - testing

INTRO ALGORITHMS AND DATA STRUCTURES

UNIVERSITEIT
GENT

CONTENT

- Algorithms
  - Introduction
  - Binary Search
  - Sort algorithms
- Data structures
  - Introduction
  - Stack class

UNIVERSITEIT
GENT

ALGORITHMS

- Examples:
  - Search and Sort algorithms
  - Shortest Path problem: Dijkstra's algorithm
  - Travelling salesman problem
- Definition:

  An algorithm is a finite, step-by-step sequence of instructions or a set of rules designed to perform a specific task or solve a problem. It takes an input, processes it, and produces an output in a defined manner.

BINARY SEARCH

- For binary search to work, the elements in the list must already be ordered. Without loss of generality, assume that the list is in ascending order.
  - e.g., 2 4 7 10 11 45 50 59 60 66 69 70 79
- The binary search first compares the key with the element in the middle of the list.

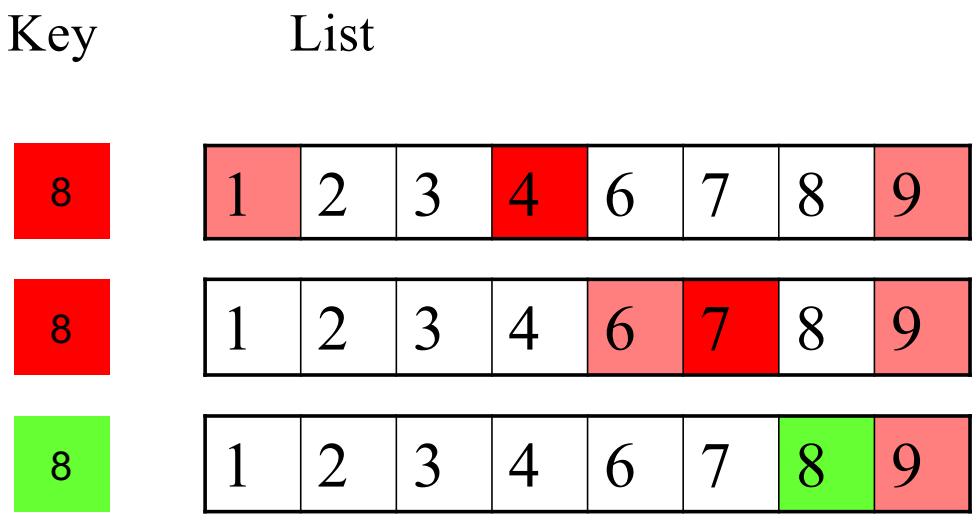
BINARY SEARCH, CONT.

Consider the following three cases:

- If the key is less than the middle element, you only need to search the key in the first half of the list.
- If the key is equal to the middle element, the search ends with a match.
- If the key is greater than the middle element, you only need to search the key in the second half of the list.

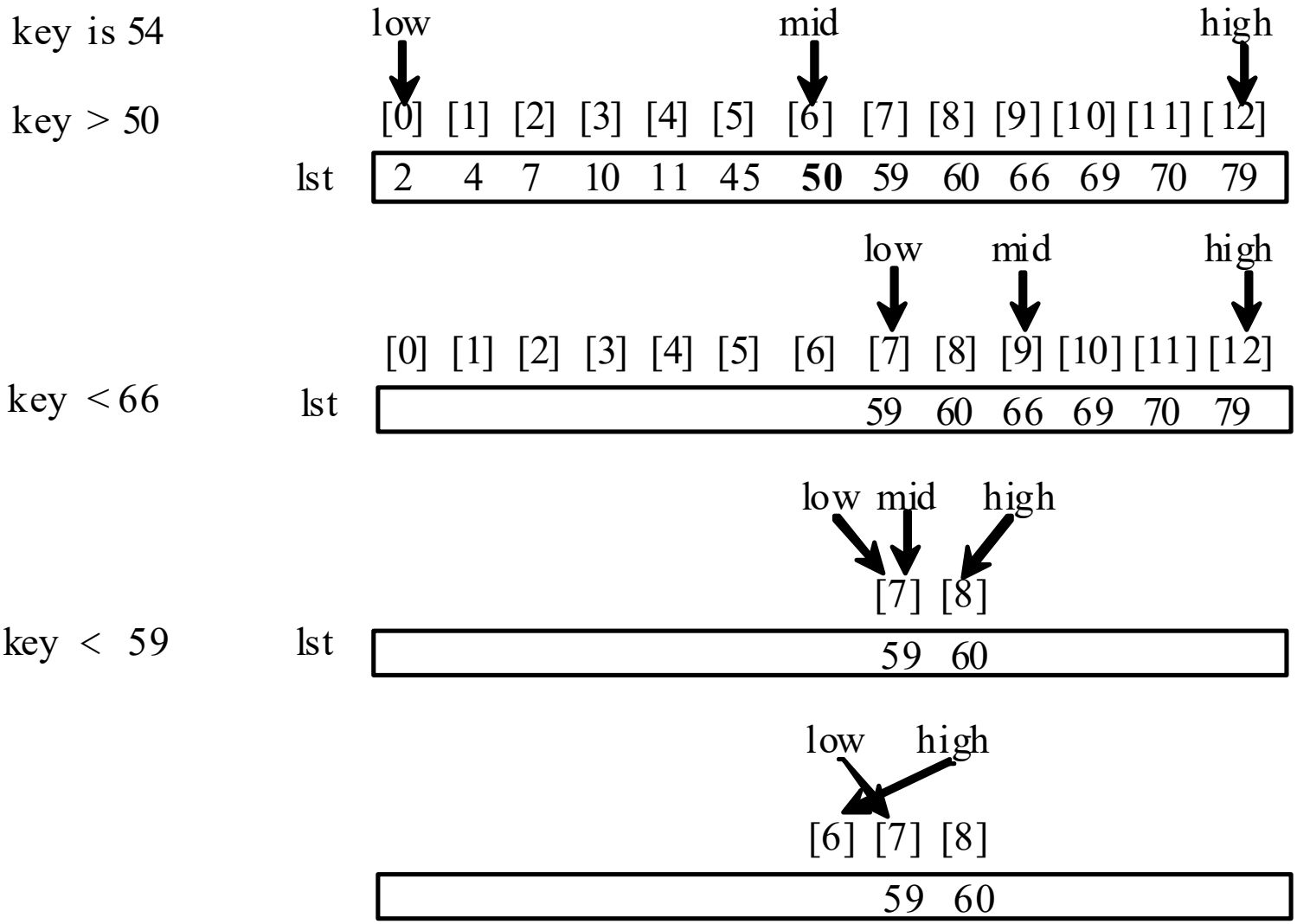
BINARY SEARCH

- https://liveexample.pearsoncmg.com/dsanimation/BinarySearcheBook.html



UNIVERSITEIT GENT

BINARY SEARCH



UNIVERSITEIT GENT

BINARY SEARCH

- The binarySearch method returns the index of the element in the list that matches the search key if it is contained in the list. Otherwise, it returns -1
- The binarySearch method can also return the insertion point
  - The insertion point is the point at which the key would be inserted into the list.

FROM IDEA TO SOLUTON

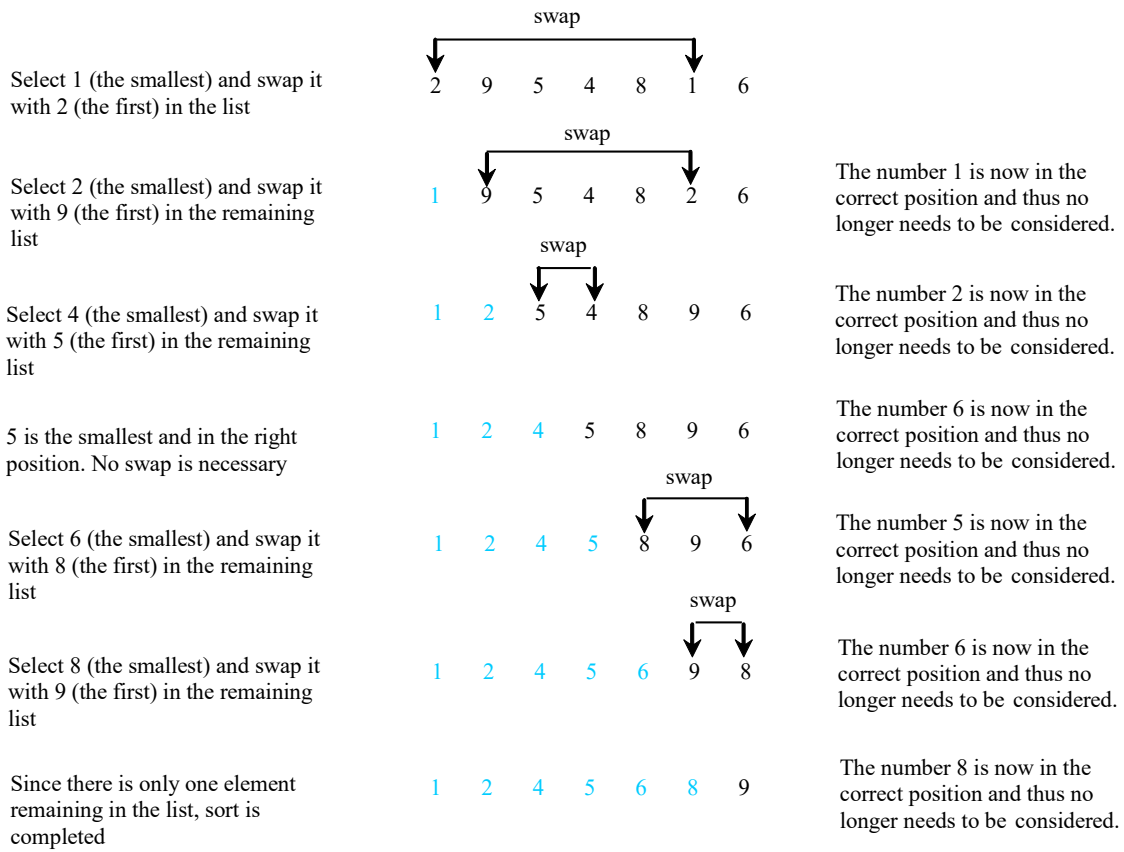
```
# Use binary search to find the key in the list
def binarySearch(lst, key):
    low = 0
    high = len(lst) - 1

    while high >= low:
        mid = (low + high) // 2
        if key < lst[mid]:
            high = mid - 1
        elif key == lst[mid]:
            return mid
        else:
            low = mid + 1

    return –low - 1 # Now high < low, key not found
```

SORTING LISTS

Sorting, like searching, is also a common task in computer programming. Many different algorithms have been developed for sorting. This section introduces two simple, intuitive sorting algorithms: selection sort and insertion sort.

SELECTION SORT

Selection sort finds the smallers number in the list and places it fist. It then finds the smallers number remaining and places it next to the first, and so on until the list contains only a single number.



UNIVERSITEIT GENT

SELECTION SORT ANIMATION

- https://liveexample.pearsoncmg.com/dsanimation/SelectionSortNew.html

UNIVERSITEIT
GENT

FROM IDEA TO SOLUTION

```
for i in range(len(lst)):
    select the smallest element in lst[i.. len(lst)-1]
    swap the smallest with lst[i], if necessary
    # lst[i] is in its correct position.
    # The next iteration apply on lst[i+1..len(lst)-1]
```

```
lst[0] lst[1] lst[2] lst[3] ...                lst[10]

lst[0] lst[1] lst[2] lst[3] ...                lst[10]

lst[0] lst[1] lst[2] lst[3] ...                lst[10]

lst[0] lst[1] lst[2] lst[3] ...                lst[10]

lst[0] lst[1] lst[2] lst[3] ...                lst[10]

                                  ...

lst[0] lst[1] lst[2] lst[3] ...                lst[10]
```

UNIVERSITEIT GENT

EXPAND

```
for i in range(0, len(lst)):
    select the smallest element in lst[i.. len(lst)-1]
    swap the smallest with lst[i], if necessary
    # lst[i] is in its correct position.
    # The next iteration apply on lst[i+1..len(lst)-1]
```



```
    currentMin = min(lst[i : ])
```

Note: this is a new version much simpler than the one in the book.

UNIVERSITEIT GENT

EXPAND

```
for i in range(0, len(lst)):
    select the smallest element in lst[i.. len(lst)-1]
    swap the smallest with lst[i], if necessary
    # lst[i] is in its correct position.
    # The next iteration apply on lst[i+1..len(lst)-1]
```



```
    currentMin = min(lst[i : ])

    currentMinIndex = i + lst[i : ].index(currentMin)

    # Swap lst[i] with lst[currentMinIndex] if necessary
    if currentMinIndex != i:
        lst[currentMinIndex], lst[i] = lst[i], currentMin
```

UNIVERSITEIT GENT

WRAP IT IN A FUNCTION

```
# The function for sorting elements in ascending order
def selectionSort(lst):
    for i in range(len(lst) - 1):
        # Find the minimum in the lst[i : len(lst)]
        currentMin = min(lst[i : ])
        currentMinIndex = i + lst[i: ].index(currentMin)

        # Swap lst[i] with lst[currentMinIndex] if necessary
        if currentMinIndex != i:
            lst[currentMinIndex], lst[i] = lst[i], currentMin
```

Invoke it

```
selectionSort(yourList)
```

UNIVERSITEIT GENT

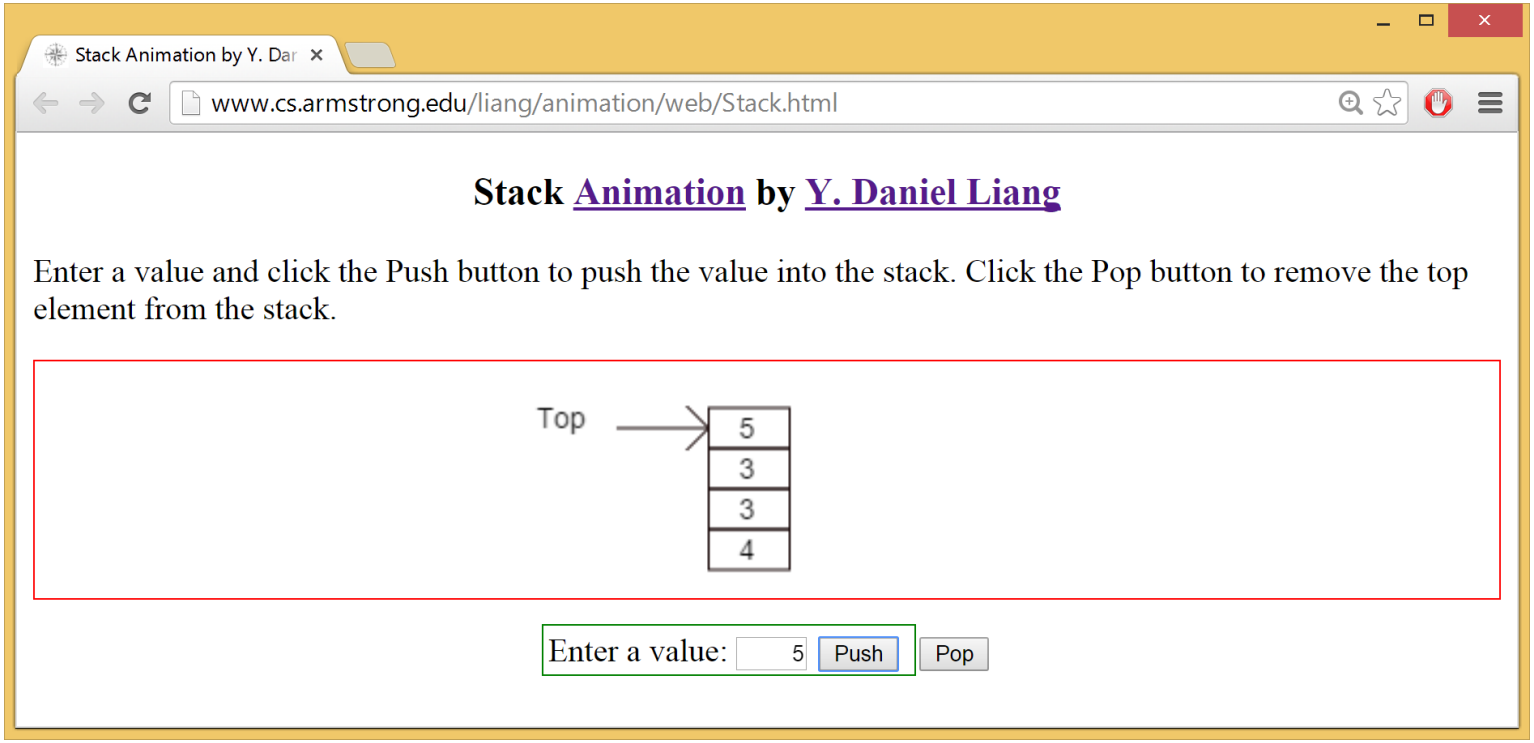
DATA STRUCTURES

- Example: List, Set, Stack, Row, Binary Tree, Hashing Table, Dictionary
- Definition:

  A data structure is a specialized format for organizing, managing, and storing data in a way that enables efficient access and modification. It defines the relationships between data elements and provides methods for performing operations such as searching, sorting, inserting, updating, and deleting data.

STACK ANIMATION

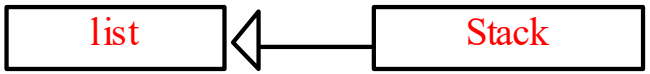
https://liveexample.pearsoncmg.com/dsanimation/StackeBook.html

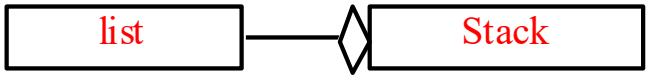


UNIVERSITEIT GENT

THE STACK CLASS

- You can define a class to model stacks. You can use a list to store the elements in a stack.
- There are two ways to design the stack and queue classes:
  - Using inheritance: You can define a stack class by extending list.
  - Using composition: You can create a list as a data field in the stack class.





UNIVERSITEIT GENT

THE STACK CLASS

Stack	
-elements: list	A list to store elements in the stack.
+Stack()	Constructs an empty stack.
+isEmpty(): bool	Returns True if the stack is empty.
+peek(): object	Returns the element at the top of the stack without removing it from the stack.
+push(value: object): None	Stores an element into the top of the stack.
+pop(): object	Removes the element at the top of the stack and returns it.
+getSize(): int	Returns the number of elements in the stack.

UNIVERSITEIT GENT

Stack TestStack