

## Recap: basic data structures

Data Structures and Algorithms for Computational Linguistics III  
ISCL-BA-07Çağrı Çöltekin  
ccolt@infsys.uni-tuebingen.deUniversity of Tübingen  
Seminar für Sprachwissenschaft

Winter Semester 2021-2022

© C. Çöltekin, 2021-2022 2 / 14

## Overview

- Some basic data structures
  - Arrays
  - Lists
  - Stacks
  - Queues
- Revisiting searching a sequence

© C. Çöltekin, 2021-2022 3 / 14

Winter Semester 2021-2022 3 / 14

## Abstract data types and data structures

Introduction Abstract data types Algorithms

- An *abstract data type* (ADT), or abstract data structure, is an object with well-defined operations. For example a *stack* supports *push()* and *pop()* operations
- An abstract data structure can be implemented using different *data structures*. For example a stack can be implemented using a linked list, or an array
- Sometimes names, usage is confusingly similar

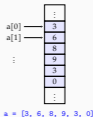
© C. Çöltekin, 2021-2022 4 / 14

Winter Semester 2021-2022 4 / 14

## Arrays

Introduction Abstract data types Algorithms

- An array is simply a contiguous sequence of objects with the same size
- Arrays are very close to how computers store data in their memory
- Arrays can also be multi-dimensional. For example, matrices can be represented with 2-dimensional arrays
- Arrays support fast access to their elements through indexing
- On the downside, resizing and inserting values in arbitrary locations are expensive



© C. Çöltekin, 2021-2022 5 / 14

Winter Semester 2021-2022 5 / 14

## Arrays in Python

Introduction Abstract data types Algorithms

- No built-in array data structure in Python
- Lists are indexable
- For proper/faster arrays, use the *numpy* library

```

List indexing in Python

a = [3, 6, 8, 9, 3, 0]
a[0] = 3
a[-1] = 0
a[1:4] = [6, 8, 9]
a2d = [[3, 6, 8], [9, 3, 0]]
a2d[0][1] = 6
  
```

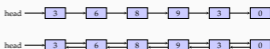
© C. Çöltekin, 2021-2022 6 / 14

Winter Semester 2021-2022 6 / 14

## Lists

Introduction Abstract data types Algorithms

- Main operations for list ADT are
  - append
  - prepend
  - head (and tail)
- Lists are typically implemented using linked lists (but array-based lists are also common)
- Python lists are array-based



© C. Çöltekin, 2021-2022 7 / 14

Winter Semester 2021-2022 7 / 14

## Stacks

Introduction Abstract data types Algorithms

- A stack is a last-in-first (LIFO) out data structure
- Two basic operations:
  - push
  - pop
- Stacks can be implemented using linked lists (or arrays)



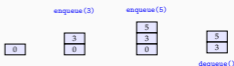
© C. Çöltekin, 2021-2022 8 / 14

Winter Semester 2021-2022 8 / 14

## Queues

Introduction Abstract data types Algorithms

- A queue is a first-in-first (FIFO) out data structure
- Two basic operations:
  - enqueue
  - dequeue
- Queues can be implemented using linked lists (or maybe arrays)



© C. Çöltekin, 2021-2022 9 / 14

Winter Semester 2021-2022 9 / 14

## Other common ADT

Introduction Abstract data types Algorithms

- *Strings* are often implemented based on character arrays
- *Maps* or *dictionaries* are similar to arrays and lists, but allow indexing with (almost) arbitrary data types
  - Maps are generally implemented using hashing (later in this course)
- Sets implement the mathematical (finite) sets: a collection unique elements without order
- Trees are used in many algorithms we discuss later (we will revisit trees as data structures)

© C. Çöltekin, 2021-2022 10 / 14

Winter Semester 2021-2022 10 / 14

## Studying algorithms

Introduction Abstract data types Algorithms

- In this course we will study a series of important algorithms, including
  - Sorting
  - Pattern matching
  - Graph traversal
- For any algorithm we design/use, there are a number of desirable properties
  - Correctness: an algorithm should do what it is supposed to do
  - Robustness: an algorithm should (correctly) handle all possible inputs it may receive
  - Efficiency: an algorithm should be light on resource usage
  - Simplicity: an algorithm should be as simple as possible
  - ...
- We will briefly touch upon a few of these issues with a simple case study

© C. Çöltekin, 2021-2022 11 / 14

Winter Semester 2021-2022 11 / 14

## A simple problem: searching a sequence for a value

Introduction Abstract data types Algorithms

```

def linear_search(seq, val):
    answer = None
    for i in range(len(seq)):
        if seq[i] == val:
            answer = i
    return answer
  
```

Is this a good algorithm? Can we improve it?

© C. Çöltekin, 2021-2022 12 / 14

Winter Semester 2021-2022 12 / 14

## Linear search: take 2

Introduction Abstract data types Algorithms

```

def linear_search(seq, val):
    for i in range(len(seq)):
        if seq[i] == val:
            return i
    return None
  
```

Can we do even better?

© C. Çöltekin, 2021-2022 13 / 14

Winter Semester 2021-2022 13 / 14

```

1 def linear_search(seq, val):
2     n = len(seq) - 1
3     last = seq[n]
4     seq[n] = val
5     i = 0
6     while seq[i] != val:
7         i += 1
8     seq[n] = last
9     if i < n or seq[n] == val:
10        return i
11    else:
12        return None

```

- Is this better?
- Any disadvantages?
- Can we do even better?

```

1 def binary_search(seq, val):
2     left, right = 0, len(seq)
3     while left <= right:
4         mid = (left + right) // 2
5         if seq[mid] == val:
6             return mid
7         if seq[mid] > val:
8             right = mid - 1
9         else:
10            left = mid + 1
11    return None

```

- We can do (much) better if the sequence is sorted.

## Binary search

recursive version

```

1 def binary_search_recursive(seq, val, left=None, right=None):
2     if left is None:
3         left = 0
4     if right is None:
5         right = len(seq)
6     if left > right:
7         return None
8     mid = (left + right) // 2
9     if seq[mid] == val:
10        return mid
11    if seq[mid] > val:
12        return binary_search_recursive(seq, val, left, mid - 1)
13    else:
14        return binary_search_recursive(seq, val, mid + 1, right)

```

## A note on recursion

- Some problems are much easier to solve recursively.
- Recursion is also a mathematical concept, properties of recursive algorithms are often easier to prove
- Reminder:
  - You have to define one or more base cases (e.g., if `left > right` for binary search)
  - Each recursive step should approach the base case (e.g., should run on a smaller portion of the data)
- We will see quite a few recursive algorithms, it is time for getting used to if you are not

Exercise: write a recursive function for linear search.

## Summary

- This lecture is a review of some basic data structure and algorithms
  - We will assume you know these concepts, please revise your earlier knowledge if needed
- Next:
- Analysis of algorithms (Reading: textbook (Goodrich, Tamassia, and Goldwasser 2013) chapter 3)
  - A few common patterns for designing (efficient) algorithms

## An interesting (but not-extremely-relevant) anecdote

How hard can binary search could be?

- It was first suggested in a lecture in 1946 (by John Mauchly)
- First fix to this version was suggested in 1960 (by Derrick Henry Lehmer)
- Another, fix/improvement over this was published in 1962 (by Hermann Bötenbruch)
- In 2006, a bug in Java's binary search implementation was discovered

## Acknowledgments, credits, references

- Some of the slides are based on the previous year's course by Corina Dima.

 Goodrich, Michael T., Roberto Tamassia, and Michael H. Goldwasser (2013). *Data Structures and Algorithms in Python*. John Wiley & Sons, Incorporated. isbn: 9781118476734.