Feedback

Self-Testing

How to:

1. Download the self-testing script in the Assignment files.
2. Adjust the import statement (if needed) at the test script to import your solutions.
3. Run the test-script from the terminal or from Spyder and check which tests you passed.

Self-assessment

Problem categories:

After checking the feedback from the automated tests, count how many problems you could correctly solve in each of the categories below.

☐☐☐☐ (Head and Tail)
☐☐ (Array)
☐ (List)

Depending on how you did, look at the **study recommendations** in the margin notes.

Learning checklist:

After completing this week's assignment, you should be able to:

- ☐ Understand what the head and tail technique is. ●○○
- ☐ Implement recursive functions using head and tail. ●●●

Problem categories:

`move_max_rec`: ■■
`is_substring`: ■
`find_combinations`: ■■
`sort_array`: ■■

Study recommendations:

■■□□

Recursion is hard. Most of the problems in this assignment require you to have a good understanding of the head and tail technique and apply that to arrays, lists or strings. Read again about how to use head and tail in recursion and look for the other resources given in the lecture materials (the Recursive book of recursion has quite good explanations!).

CHAPTER

VIII

Weeks 6-8 - Programming Project

In this chapter:

General Information
Part 1
Part 2
Additional recommendations
Project description
Project description template
Datasets
Submission instructions

General information

In the previous Assignments, you worked on the concepts seen in the classroom through tailored and focused questions. Now it's the time to change that and start a more hands-on, longer project.
This assignment consists of three different parts:

- **Part 1 - Understanding sorting methods:** In the first part of the assignment, we will follow a flipped-classroom approach. You will first study **sorting methods** on your own, and record a small video about it, **to be posted on Brightpsace before our last lecture (Dec 16)**. After the lecture, you will watch some of the videos from your peers, and react to them, judging their explanation and your understanding of the content.
- **Part 2 - Algorithm complexity:** In the second part of the project, you will implement and use some predefined functions related to recursion, sorting and plotting to visualize some aspects of algorithm complexity (Lecture 6-7).
- **Part 3 - Working with different data structures:** In the third part, you will work with a **dataset** and implement code that manipulates dictionaries, tuples, and arrays. You have more freedom in this last part and may choose to create or find your own dataset, tailoring your program to anything you find interesting.

The **project** is individual, but you can discuss your **ideas** from part 2 and part 3 with others.

Dataset: there are three datasets provided in the assignment from which you can choose (you can find their description at the end of this chapter). Or you can create your own datasets. if you come up with your own dataset, make sure to have enough samples on it (30), and variables of at least 2 different types (ex: string and numerical).

> ⚠ Warning
> In the next three weeks you will have time to work on your project, connecting the content of the last lectures to it.

Part 1

Research about the three following sorting algorithms: Mergersort, Bubblesort and Quicksort. There are some materials available in Brightspace, including the code implementation provided with this assignment. You should also do your own research to find extra resources.

After understanding the way the algorithms work, your task is to record a **short video (3-7 min)** in which you perform the steps of one of the algorithms, using your S-number as an input array/list. **In your video, use the implementation provided in Part 2.**

You should start with your s-number and follow the steps of the algorithm until you reach a **sorted sequence**. In the beginning of the video, **explain the steps** you are following/performing, trying to relate to how the algorithm works.

> ⚠ Warning
> There are often different implementations of those algorithms. In your video you should **follow the algorithm provided in the python files of the assignment**.
>
> In addition to that, you should record your video yourself. Needless to say that the use of generative AI to do that is not allowed.

You will only need to record one of the algorithms, depending on the first letter of your name (see the table below).

First letter	Algorithm
A-I	Bubblesort
J-N	Mergesort
O-Z	Quicksort

Submit your video in Brightspace, (in the Pitch2Peer component of Module 3) **before the lecture on Dec 16**.

> Tip
> You can get inspiration from the following videos that explain Bubblesort, Mergesort and Quicksort.
>
> You don't need to necessarily appear on the video. You can use post-its with numbers; playing cards to represent the numbers; or ask your friends to help and participate!.
>
> Be creative (the reviewers and other students will be able to give likes to your video).

Remember that this is an academic activity and should follow the code of conduct of the university/programme/course. Be respectful with what you show/say in your video.

Project requirements (Part 1) **(4 points)**

- Before Dec 16: Submit your video in Brightspace (Assignment 3 - Part 1 (Pitch2Peer)) .
- After Dec 16: Watch and peer-review to two videos in a different category. - Optional: Watch other videos and give likes to the most creative ones.

Part 2

Download the file part2.py. There you can find an example implementation of the three sorting algorithms seen in part 1: `bubblesort()`, `mergesort()` and `quicksort()`. In addition to that, there are two functions that can be used to generate a random array: `generate_rnd_array1()` and `generate_rnd_array2()`.

Your goal in this part is to compare the performance of **two of the three sorting algorithms** (you can choose) when sorting an array/list generated by the provided functions. In order to do that, you will have to implement two other functions (see a detailed description below):

- `average_time(data: ndarray(int/float), n: int, f)`: receives the array `data`, an integer `n` and a function `f`. The function computes the average time (out of `n` repetitions) your computer takes to sort the array `data` using the function f.

  You can choose how many repetitions are used to compute the **average time** (but make sure that you use at least 10).

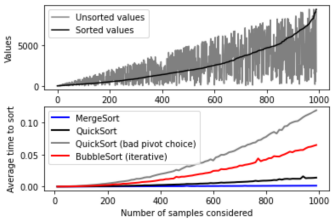
  > Tip
  > **Tip!** Yes! You can pass a function as an argument of another function!
  >
  > Use the perf_counter() function from the time module to compute the time it takes between two statements.

- `estimate_complexity(data: ndarray(int/float), n_samples: ndarray(int), f)`: For each integer `n` in `n_samples`, the function should use `average_time()` to com-

pute the mean time taken to sort the first `n` samples of the array data using the function f. Your function should return an array with the same shape of `n_samples` in which each element is the corresponding **average time**.

- `main()`: Your main function should use the functions `generate_rnd_array1()` and `generate_rnd_array2()` (already implemented in the `part2.py`) to create two random arrays. Then, your function should generate 3 different **plots**:
  - the two generated arrays in their original order
  - the two generated arrays sorted by two different sorting algorithms (ex: quicksort and mergesort()).
  - a plot showing the average time (y-axis of the plot) taken to sort each of the two arrays when considering different amount of samples (x-axis of the plot) and functions. Use the function estimate_complexity to make these plots.

Plots in main(): here is an example of a similar **plot** done for a specific random array:



In the top panel, the random (unsorted) data is shown (grey), with its respective sorted version (black). In the bottom panel the performance of three different algorithms are plotted for the dataset in the top panel. Note that the data statistics might influence the performance of some algorithms.

Include a discussion section in your project report where you talk about the results shown by your plots. Your discussion should include:

- Your results (include images of your plots)
- What you expected based on Big-O analysis of the code;
- A commentary on your results. (e.g., are they different than what you expected? if so, why is that the case? How does the sorted arrays relate to the average case of the algorithms?)

How

Project requirements (Part 2) (6 points)

You project should contain the following files:

- `part2.py`: a file with the functions related to the second part of the project.
  - `main()`
  - `bubblesort()`
  - `quicksort()`
  - `mergesort()`
  - `average_time()`
  - `estimate_complexity()`
- project_report.pdf with Discussion section. See more information below.

Part 3

Implement **four different functionalities** that operate on the dataset you choose. The functionalities should be encapsulated by (at least) four different functions.

Example. Plant Watering Tracker project.

You are using a dataset that contains different plant names, and the amount of water/light they require. You build the following four functionalities in your program:

- add_plant(): that adds a plant entry to the database. To do that, your function uses another module that you implemented called database.py to manage loading from and saving in the database.
- find_plant(): that looks for information about a specific plant in the database
- choose_plant(): that given the light and sun of a place, return good candidate plants for that place.
- choose_place(): that given a plant and two different set of conditions (light and

sun), returns the best place for that plant.

Those four functions should **necessarily** use one of the following data structures:

- Dictionary
- Tuple
- Array

> ⚠ **Warning**
> All those data structures should be used in your project at least once. In addition to that, you are free to use any data structure that we've learned before or that you research yourself.

To help defining it a bit better, see the list of requirements bellow.

Project requirements (Part 3) (6 points)

You project should contain the following files:

- `database.py`: a file that contains any function that deals with reading, writing or updating your database.
- `part3.py`: a file with the functions related to the first part of the project. This file requirements are the following:
  - A `main()` function that prompts the user with the possible choices of your program and is the function called upon running your code.
  - In total you should have at least **four** functions using dictionaries, arrays or tuples.
    - At least **one** of them should use arrays;
    - At least **one** of them should use tuples;
    - At least **one** of them should use dictionaries.
- project_report.pdf with a project description, example uses and authorship declaration.

main.py should import the other files as modules and manage the interface with the user choices. Depending on what the user asks, the main function will call the appropriate function in part3.py, database.py or any other module of your project.

Additional recommendations

- Your main function in part 3 should keep giving the user the option to execute one of the functionalities or to exit the program.
- Your program should not have *hardcoded* paths and work in other computer/operating systems (ex: you can use relative paths in the os module if you need to navigate folders).
- Comment your code with (at least) minimal descriptions of what the code is doing.
- If you want, you can create additional files than the ones described here. As long as your code runs and works.
- any module that does not require installation can be used (ex: comes with Spyder). Additionally, you can use any of the following modules:

  - numpy
  - scipy
  - pyplot
  - math
  - io
  - sys
  - random
  - datetime
  - time
  - csv
  - rich

  Beyond that, if you wish to use a module that does not fall into that category, ask your TA if that's possible.

Project report

Beyond your codes, you should also submit a one/two page **project report** (as a pdf). This document should have the following information:

- A **description** of your project (ex: what is it about, what the user should do, how to use it, what the program does, etc)
- A **list of example uses** of arrays, tuples and dictionaries in the first part of your project

  Example.
  - **dictionaries:**
    In the function `add_plant` the program uses a dictionary to store information about the plant.

- **Discussion:** a short discussion on the results you get when you plot the run-time analysis in part 2. Include there what you expected based on Big-O analysis of the sorting algorithms that you tested. Is that what you found? Or are they different? Do the two datasets have the same Big-O complexity? Elaborate why do you think that is or not the case. Include images of the plot in your discussion.
- **Acknowledgements and references** of people that conceptually contributed to your project and **external source code** used, if any.

  If you use parts of code written by others in your project, that should be clearly recognizable in your code (use comments), and credited.

- **Authorship declaration** in the form of the following text:

  I declare that the work submitted here is from my authorship only. I haven't used any generative AI to help with any code/text included in my work. I have given credit for the help I had conceptualizing my project. My work respects the university and course code of conduct

See below an example of what your project description should look like (use that as a template).

Project report - Template

Discussion (Part 2):

Discuss the result you have when you plot the run-time analysis. Include the following points in your discussion:

- What you expected based on Big-O analysis of your code
- What you found (include images of your plots)
- Comment your results. (Ex: if they are different/equal, why is that the case? What would you expect if you have implemented a different sorting algorithm? etc).

Project and database (Part 3)

Brief project and dataset descriptions (1-2 paragraphs):

My project implements a Plant Watering Tracker...

Project functionalities:

Describe what are the functionalities implemented in your project, and which prerequisite connects to which functionality:

The user can chose among N options.

- *Option 1* ***(Dictionary)****: calls the function* ***add_plant****(plant_book: dict, plant_name: str, plant_data: tuple) that modifies the dictionary plant_book, adding plant_name as a new key with value defined by the tuple plant_data.*
- *Option 2* ***(Tupples and Dictionaries)****: calls the function* ***find_plant****(plant_name: str) which ... (description of the function)...*
- *Option 3: ...*
- *Option 4: ...*
  ...
- *Option N: Exit.*

Acknowledgements

Recognize the contribution of others:

I would like to thank X, Y and Z with the help on discussing the conceptual ideas for this project.

Authorship declaration

I certify that this assignment represents my own work. I have not used any unauthorized or unacknowledged assistance or sources in completing it including free or commercial systems or services offered on the internet or text generating systems embedded into software. I have given credit for the help I had conceptualizing my project. My work respects the university and course code of conduct.

Datasets

Below you can find the description of the three datasets, which you may use. If none of those datasets appeal to you, feel free to choose/make your own dataset! If you do that do not forget to examine the data before you work with it. Make sure it has no missing values, or otherwise make sure your code can account for those missing values. Last but not least, make sure the data is not too big, or too small, if in doubt ask your TA.

All provided **datasets** are in *.txt* format and as a first line have the titles of the columns (features).

choosing a dataset: regardless of the dataset you choose to use (either provided by us, found, or made by you), it might be worth it to familiarize yourself with it. To do that, you can write a short description of it. For example, which of the features are numeric? which ones are categorical? are there any edge cases that should be considered in your project? ...

Movie Dataset

The first dataset we provide is an adventure movie dataset. A link to the full dataset can be found here. We have cleaned and filtered the dataset. In the provided dataset, the entries are movies and the features are the following :

- name - Name of the Movie
- rating - Rating of the Movie
- duration - Duration of the Movie
- year - Year of movie release
- genre - Genre of the Movie
- gross_income - Gross Income of the Movie

There are 696 entries and 6 features.

Spotify Dataset

The second dataset is with the most streamed Spotify songs in 2023. A link to the full dataset can be found here. The features we have selected are the following :

- track_name - Name of the song
- artist(s)_name - Name of the artist(s) of the song
- released_year - Year when the song was released
- released_month - Month when the song was released
- in_spotify_charts - Presence and rank of the song on Spotify charts
- mode - Mode of the song (major or minor)
- danceability_%: - Percentage indicating how suitable the song is for dancing

There are 953 entries with 7 features each.

Goodreads Dataset

The third dataset is a book one gathered from Goodreads. A link to the full dataset can be found here. The features we have selected for you to work with are the following :

- title - The name under which the book was published.
- authors - Names of the authors of the book. Multiple authors are delimited with "-"
- average_rating - The average rating of the book received in total
- language_code - Helps understand what is the primary language of the book. For instance, eng is standard for English
- num_pages - Number of pages the book contains.
- ratings_count - Total number of ratings the book received
- publication_date - Date when the book was first published

There are 894 entries with 7 features each.

Submission Instructions - Final Project

Submitting your work

General submission:

Create a zip file called `Programming_Project` containing (at least) the files:

```
- project_report.pdf
- database.py
- part2.py
- part3.py
```

Any other file needed to run your project should be included in the zip folder. Moreover, your program should be able to run without error in other computers / operating systems.

> **Example.**
>
> ```
> ...\>python part3.py
> ```

Submissions that fail those specifications may not be considered.