# 1DV505/1DT910 - Assignment 1

Jonas Lundberg

October 2024

## Instructions

- Please contact your TA if you have problems when trying to install Python or VS Code.
- Do not hesitate to ask your supervisor at the tutoring sessions (or Jonas at the lectures) if you have any problems with the Lecture 1- 3 exercises. You can also post a question in the Slack channel.
- Exercises are graded Fail, Pass or Pass with distinction (or Swedish U, G or VG).
- Pass (or G) exercises are mandatory. Pass with distinction (or VG) are only needed if you aspire for A or B grades.
- For this course we recommend a file structure that looks like this:

```
1DV505 (or 1DT910)
    Assignment1
        Lecture1
            palindrome.py
            pascal.py
            ...
        Lecture2
            ...
        Lecture3
    Assignment2
        Lecture4
        ...
    ...
```

  The topmost folder (1DV501 or 1DT910) is created using your operating system. The other files and folders (e.g. `Assignment1` and `pascal.py`) are created from within VS Code.

- Assignment 1 should be submitted using the Moodle submission system before the given deadline. Using the file setup described above, just zip the directory `Assignment1`, and submit the zip-file.
- Any use of Python features (e.g. libraries) not presented in this course, or the previous course (1DV510/1DT9001), will result in grade F. Exceptions can be given if you in advance (before submission) contact your TA.

## Lecture 0: Install Software

The first task below is only to get you started and you do not need to show it as it only guides you through how to install and test out Python. You show that this works by showing the other tasks in Assignment 1.

1. Install Python via Anaconda Distribution. Verify that it works by opening a Console (Windows) or Terminal (mac) window and type python. Example from Jonas' Mac:

   ```
   jlnmsi % python
   Python 3.8.3 (default, Jul  2 2020, 11:26:31)
   [Clang 10.0.0 ] :: Anaconda, Inc. on darwin
   Type "help", "copyright", "credits" or "license" for more information.
   >>> exit()  # terminate Python
   ```

   Here is an Anaconda installation video from Youtube for Windows user. Make sure to use the option "Add Anaconda to my PATH environmental variable" during the installation process.

2. Install Visual Studio Code and extensions
   (a) Download and install Visual Studio Code (VSC)
   (b) Download and install Python Extension for VSC

3. Set up Visual Studio Code: Before you start programming, do the following.
   (a) Create a folder with the name `python_courses` in some location in your home directory.
   (b) Create a folder named 1DV505 (or 1DT910) inside `python_courses`.
   (c) Use Visual Studio Code to open the folder 1DV505 (or 1DV910)
   (d) Use Visual Studio Code to create a new folder named
   (e) `YourLnuUserName_assign1` inside folder 1DV501. For example, it might look something like `wo222ab_assign1`.
   (f) Save all program files from the exercises in this assignment inside the folder `YourLnuUserName_assign1`.
   (g) In the future: create a new folder (`YourLnuUserName_assignX`) for each assignment and a new folder (with the course code as name) for each new course using Python.

4. Prepare (and install) support for linting by watching the video *Getting started with Linting* available in Moodle.

# Lecture 1: Recursion

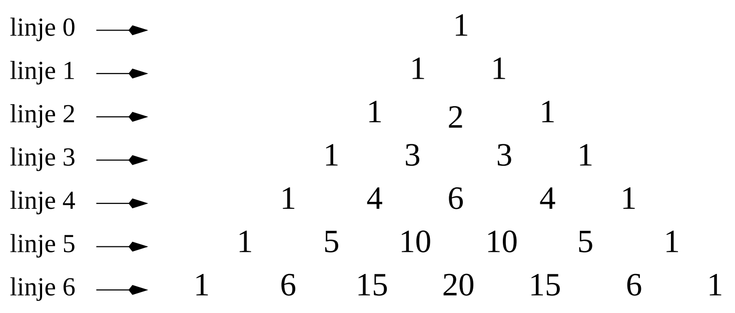
1. Simple Palindrome

   - A string is a *simple palindrome* if it has the same text in reverse.
   - Examples: x, anna, madam, abcdefedcba, yyyyyyyy
   - A simple palindrome can be defined as:
     (a) An empty string is a palindrome
     (b) A string with the length 1 is a palindrome.
     (c) A string is a palindrome if the first and last characters are equal, and all characters in between is a palindrome.
   - (a) and (b) are our base cases, (c) is our recursive step

   **Exercise:** Write a program `palindrom.py` that uses a function `check_palindrome(s)` to decide (`True` or `False`) whether a given string `s` is a simple palindrome or not.
   **Hint:** Let `check_palindrome(s)` be a help function that calls a recursive function `is_palindrome(s, first, last)` taking two indices `first` and `last`.

2. Pascal's Triangle

   - **Exercise:** Write a program `pascal.py` that recursively calculates and returns the n:th line in Pascal's triangle.



   - **Hint:** Use a help function `pascal_line(n)` that initializes a call to the recursive function `pascal_rec(lst, n)`

3. Print Directories and Subdirectories

   - **Exercise:** Write a Python program named `directories.py` containing a recursive function `print_files(dir_path)` that prints the names of all non-hidden files in directory `dir_path` *and all its subdrectories.*

4. Count Python Lines (**VG Exercise**)

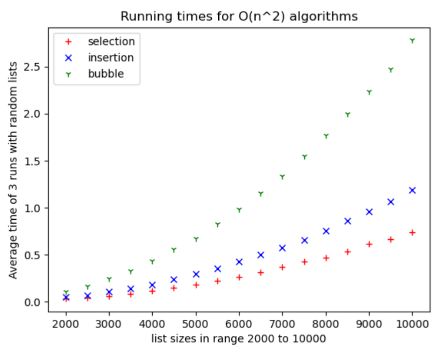
   - **Exercise:** Write a Python program named `count_python.py` containing a recursive function `count_lines(path, depth)` that:
     (a) Makes and indented print of directory `path` all the its subdirectories
     (b) Computes the total number of non-empty lines of Python code found in .py-files in directory `path` all the its subdirectories.

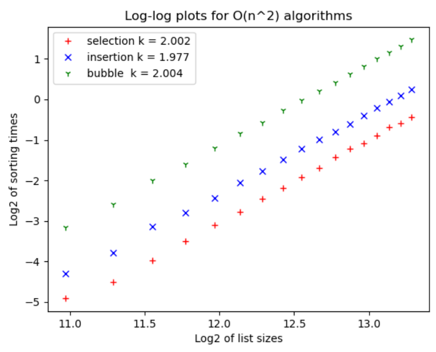
## Lecture 2: Evaluation Algorithms

1. **The 3-sum Report**
   The end product of this exercise is a written report entitled *The 3-sum Problem.* It should be submitted together with the code you used when implementing and evaluating your algorithms.

   - We have put together a report entitled *The 2-sum Problem* (available in Assignment 1). Read that report to get an idea of what type of report we are looking for.
   - When writing the report, feel free to use any word processing tool (e.g. Microsoft Word or Google Doc) you like. In the end we expect you to submit a good looking report in the PDF format.
   - We expect you to use `matplotlib` library for all your visualizations.
   - For grade G we expect:
     (a) A concrete formulation of the 3-sum problem.
     (b) A Python implementation of the *Brute Force* solution to the problem.
     (c) A description of your experimental setup.
     (d) That you find a range of suitable input sizes with execution times in range 0.01 to (say) 5 seconds.
     (e) A visualization showing 3 separate runs in a single plot to see if the result fluctuates from one run to another.
     (f) Implement support for making each data point (time measurement) the average of 5 executions. Make a `matplotlib` size vs time plot.
     (g) Decide time complexity of your implementation using the log-log and linear regression approach. A print the resulting coefficient $k$.
     (h) Plot log-log and straight line fit in a single plot.
   - **For grade VG we expect in addition:**
     (a) That you also implement the two faster approaches: *Two-Pointers* and *Caching* that was discussed in Lecture 2.
     (b) That you make experiments to evaluate their performance (compare them).
     (c) That you make experiments to determine their time complexities.
     (d) One is faster than the other. In your opinion, why?
   - Visit tutoring sessions to discuss with your TA/Teacher how you best can organize your code in a project like this. For example,
     – It might be a good idea to keep your algorithms in a separate folder.
     – The linear regression approach to fit a straight line to a data set will be used in other exercises. Hence, maybe store it in a separate folder?
     – etc.

     But to start with, read our example report *The 3-sum Problem* to get an idea of what we are looking for.



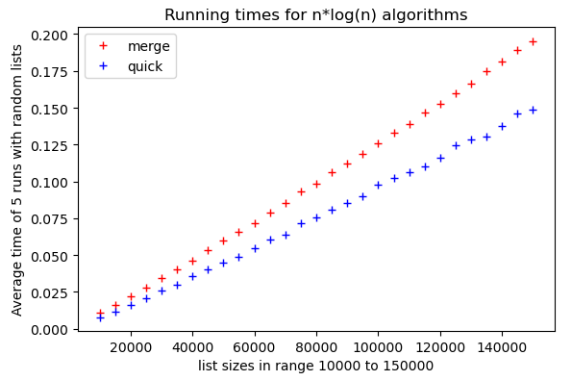


## Lecture 3: Sorting Algorithms

There is no way around it, any basic course about algorithms should include many sorting algorithms! **Note: You can easily find implementations of these algorithms on the Internet, using ChatGPT, in the texbook by Liang, or in our lecture slides. Don't just copy them! Try instead to implement them yourself just from understanding the basic idea. It is more fun and you will appreciate it when we have our written exam!**

Also, for all sorting algorithms, they should return a sorted list (smallest element first) and they must not change the input list.

1. **Implement and evaluate $O(n^2)$ algorithms**
   (a) Implement the three $O(n^2)$ algorithms Selection Sort, Bubble Sort, and Insertion Sort. Store the algorithms in a separate folder named `sort_algorithms.py`.
   (b) Evaluate the performance of the three algorithms and compare them with each other.
   (c) Verify the time complexity of your algorithms using the log-log and linear regression approach.
   (d) We want the program to display two plots as shown above. The first one is answering (b), the second one is answering (c).



2. **Implement and evaluate $O(n \cdot \log(n))$ algorithms**
   (a) Implement the two $O(n \cdot \log(n))$ algorithms Merge Sort and Quick Sort. Store the algorithms in the same folder `sort_algorithms.py` used in Exercise 1
   (b) Evaluate the performance of the two algorithms and compare them with each other by producing a plot as the one shown above.
   (c) **VG part of Exercise 2.**
   - As discussed during the lecture, a major problem with using the first element as pivot for Quick Sort is when handling sorted, or almost sort lists.
   - Implement a fix to this problem where we as pivot take the median element of the first, last and mid element in the list.
   - Can you come up with an experiment that shows that this fix works? Compare it with ordinary Quick Sort. Try also to sort a reversely sorted list. What happens?