List of Exercises:

**Week 1**

Ex 1:

* Write a method that combines two strings, by taking one character from the first string, then one from the second string and so on. Once one string has no characters left it should carry on with the other string. For example, for strings "day" and "time" should give the result "dtaiyme".

Ex 2:

* Check if a 3 digit number is an Armstrong number. An Armstrong number of three digits is an integer such that the sum of the cubes of its digits is equal to the number itself. For example, 371 is an Armstrong number since 3\*\*3 + 7\*\*3 + 1\*\*3 = 371. Input: a number, such as 371 Output: “Yes” if the number is an Armstrong number, “No” if the number is not an Armstrong

**Week 2**

Ex 1:

Write a program to solve the problem of the eight queens: i.e. find their placement on the chess board so that they have do not have the possibility to attack each other.

**Week 3**

Ex 1:

* Write a program that reads n words from the standard input, separated by spaces and prints them mirrored (the mirroring function should be implemented recursively). What is the time complexity of the algorithm? Use the BigO notation to express it.

Ex 2:

* Write a recursive version of linear search on an array of integers. What is the time complexity of the algorithm? Use the BigO notation to express it.

***Week 4***

Ex 1:

* Manually arrange the sequence [2 7 9 4 1 5 3 6 0 8] in ascending order using insertion sort, bubble sort and selection sort, showing at each step the new configuration of the sequence. How many comparisons and how many element moves were used by each method? Which is the best performing method for sorting this array of integers? Which would be the worst arrangement of this sequence?

Ex 2:

* Use binary search in implementing a guessing game. One thinks of a number between 1 and 20000, the program attempts to guess the number and feedback is given whether my number is higher or lower. The program then makes a new guess and so on until it guesses the right number

***Week 5***

Ex 1:

* Based on the Python code or the C++ code provided in class as a starting point, implement a function that deletes duplicate value nodes from the list.

***Week 6***

Ex 1:

* Implement a function that deletes a node in a binary search tree in a language of your choice.

***Week 7***

Ex 1:

* Implement an unweighted, undirected graph structure in the programming language of your choice. Implement the BFS or DFS traversal for this graph

Ex 2:

* For the same graph, implement a function named havePath(v, w), where v ∈ V and w ∈ V, to check if there is a path between the two given nodes.

***Week 8***

Ex 1:

* Adapt the previous graph structure to support weighted connections and implement Dijkstra’s algorithm.