**Midterm Simulation**

**Algorithms and Data Structures 2023** Due Date: 16th April 2023

**Information**

- This is a simulation midterm exam. The actual exam will consist of different questions with similar topics presented in class.
- Try to solve these exercises amounting to 120 points in maximum of 2 hours. During the actual exam you will most likely have only 2 hours to solve the given problems.
- Try to solve the problems without discussing the possible solutions with other students and using additional resources. During the actual exam you will not be able to communicate with anyone and use the internet (or any other resource) for help.
- You are allowed to submit either a solution written as a Python file or submit a handwritten pseudocode solving the given problems. During the exam you will not have access to a computer, so it might be useful practicing writing code as a pseudocode representation on paper. Submit *only one* PDF or Python file containing the solutions to the problems you solved.
- This simulation is made for your benefit. Follow the instructions to recreate as realistic simulation as possible so you are better prepared for the exam and know what to expect.
- In case of questions, please contact the assistants by email or write on the iCorsi forum.
- Please carefully read the instructions on the next page and proceed with solving the given exercises.

**Instructions**

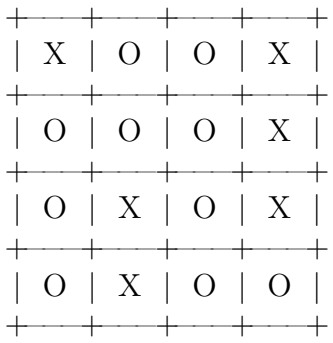
- Write in a readable and understanding handwriting. Anything that we cannot understand will not be taken into consideration while grading.
- Submit only what is required.
- You may only use the following, limited subset of the Python 3 language and libraries:
  - You may only use the built-in numeric types (e.g., `int`) and sequence types (e.g., arrays).
  - With arrays or other sequence types, you may only use the following operations:
    - direct access to an element by index, as in return `A[7]` or `[i+1] = A[i]`
    - append an element, as in `A.append(10)`
    - delete the last element, as in `A.pop()` or `del A[len(A)-1]`
    - read the length, as in `n = len(A)`
  - You may use the `range` function, typically in a for-loop, as in `for i in range(10)`
  - You may not use any library or external function other than the ones listed above.
  - You may use the `A.sort()` function to sort an array. This sorts the array in place in `nlog(n)` complexity.

---

10 ▶ **Exercise 1.1** Write a function called `peak_element(A)` that given an array `A` of type `int`, returns the index `i` of a single peak element of that array. A peak element is an element that is strictly greater than its left and right neighbor. If an array contains multiple peaks, return the index of any of those peaks. For example, the function when given the input `A = [1,2,1,3,5,6,4]` should return either 1 or 5.

20 ▶ **Exercise 1.2** Analyze the complexity of your algorithm. Write an algorithm that is functionally equivalent to the original algorithm but with a time complexity of `O(logn)`. If your algorithm already satisfies this constraint, just answer the first part of this question.

20 ▶ **Exercise 2.1** Given a matrix board `B` of size `(m rows x n columns)`, where each cell can either be a battleship (or a part of one) marked `"X"` or an empty cell marked `"O"`, write a function `count_battleships(B)` that returns the number of battleships placed on the matrix board `B`. Battleships can be placed horizontally or vertically on the matrix board and their shape can be of size `(1 x k)` or `(k x 1)`, where `k` can be of any size within the range of the board. Every battleship is separated from each other by at least one empty cell.




```
count_battleships([['X','O','O','X'], ['O','O','O','X'], ['O','X','O','X'],
['O','X','O','O']])
>>> 3
```

10 ▶ **Exercise 2.2** Analyze the time complexity of your solution in terms of big-O notation. Express your complexity in terms of the board size (m and n).

▶ **Exercise 3** Consider the following algorithm `Algo-X(A, B)` operating on two arrays of numbers `A` and `B` of total length `A.length + B.length = n`:

```
Algo–X(A,B):
    C = [False] * A.length
    for j = 1 to B.length
        i = 1
        while i <= A.legnth and (C[i] == True or A[i] != B[j])
            i = i + 1
        if i <= A.length
            C[i] = True
        else return False
    for i = 1 to A.length
        if C[i] == False
            return False
    return True
```

10 ▶ **Exercise 3.1** Explain what `Algo-X` does. Do not simply paraphrase the code. Instead, explain the high-level semantics, independent of the code. Also, analyze the best and worst-case complexity of `Algo-X`.

20 ▶ **Exercise 3.2** Write an algorithm called `Better-Algo-X` that does exactly the same thing as `Algo-X`, but with a strictly better worst-case time complexity and equal or better best-case complexity. Analyze the complexity `Better-Algo-X`. Notice that if `Algo-X` modifies the content of the input arrays `A` and `B`, then `Better-Algo-X` must do the same. Otherwise, if `Algo-X` doesn't modify `A` and `B`, then `Better-Algo-X` must not modify `A` and `B`.

30 ▶ **Exercise 4.1** Given an array of unique points in the Cartesian coordinate system (2D), where `points[i] = [xi, yi]`, write a function `find_min_rectangle(points)` that returns the minimum area of a rectangle formed from these points. The sides of the rectangles must be parallel to the `X` and `Y` axis correspondingly. In case no rectangle is formed, return 0.

```
find_min_rectangle([[1,1], [1,3], [3,1], [3,3], [2,2]])
>>> 4
find_min_rectangle([[1,1], [1,3], [3,1], [3,3], [4,1], [4,3]])
>>> 2
find_min_rectangle([[1,1], [1,3], [3,1]])
>>> 0
```