Weekly Lab

Sort Algorithms

In this lab session, we will experiment with sort algorithms.

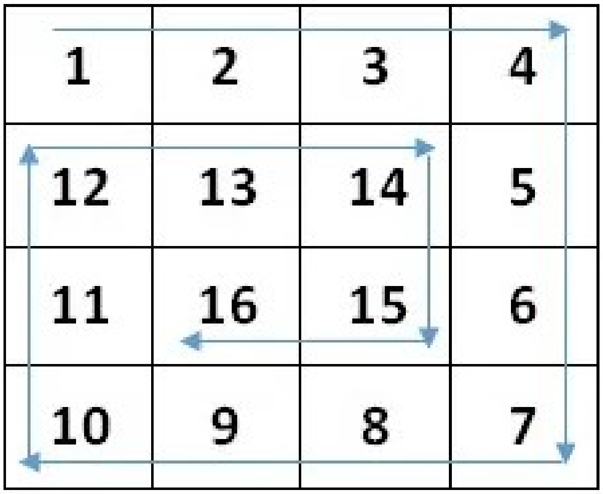
Exercise 1

Sort an integer array with even numbers in ascending order and odd numbers in descending order. Note that, positions of even and odd parts is no change. For example:

Input:
```
9
1 2 4 7 15 18 8 9 11
```
Output:
```
15 2 4 11 9 8 18 7 1
```

Exercise 2

Sort an arbitrary two-dimensional list in spiral order as following image:



1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

Figure 1: Spiral order.

Please note that the two-dimensional list does not have to be square matrix.

Exercise 3

Performs the following operations encountered when processing data for an admission exam:

1. Calculate the average scores for the candidates.
2. Assign places to the accepted candidates, based on `m` options they have, and print the list.
3. List all rejected candidates in descending order.

The exam is composed of two tests, graded with reals in the range [0, 10]. When average scores (truncated to two decimal positions right of the point) are equal, the score of the first test and then the scores of the second test is used to decide position. If equality persists, increase the number of available positions from a certain option.

Input format (`exam.txt`):

- Number of options `m` alone on one line.
- Pairs option maximum number of admitted candidates, separated by blanks, each pair on one line.
- Candidate data, one candidate on each line.

For example:

```
4
1 25
2 30
3 35
4 20
"Doe John",9.30,9.80,4,2,1,3
"Doe Jane",9.70,9.70,1,2,4,3
"Smith Alice",8.50,9.00,2,3,1,4
"Johnson Bob",9.00,8.90,3,1,2,4
"Brown Charlie",9.50,9.40,1,2,3,4
"Taylor Eve",8.90,9.10,4,3,2,1
"White Frank",9.20,9.50,2,1,3,4
"Green Hannah",8.70,8.80,1,3,2,4
...
```

Output format (console):

- Successful candidates for each option.
- Unsuccessful candidates.

For example:

```
Successful candidates for option 1
1. Doe Jane 9.70
...
Successful candidates for option 4
1. Doe John 9.55
...
Unsuccessful candidates
1. Jones Jim 4.99
...
```

Exercise 4

Read the data from the `employee.txt` file and sort it based on age in descending order. If two employees have the same age, sort them by name in alphabetical order.
The file consists of three columns:

- `id`: Unique identifier (numeric, may have leading zeros).
- `name`: Full name (first name + last name).
- `birth_year`: Encoded birth year (numeric).

Write the sorted data to another file named `sorted_employees.txt`.

Please note that:

- If the birth year is incorrectly formatted (e.g., contains more than four digits), correct it by extracting the last two digits and converting them to a four-digit year format.

  For example: `120433` should be corrected to `1933`.
- Use the `datetime` library to retrieve the current year.

Exercise 5. Sorting Algorithms Performance Comparison

Implement the following sorting algorithms for an positive integer array in increasing order:

1. Insertion Sort
2. Selection Sort
3. Interchange Sort
4. Heap Sort
5. Quick Sort
6. Merge Sort
7. Counting Sort
8. Radix Sort
9. Flash Sort

Measure the running time for each algorithm on arrays of different sizes:

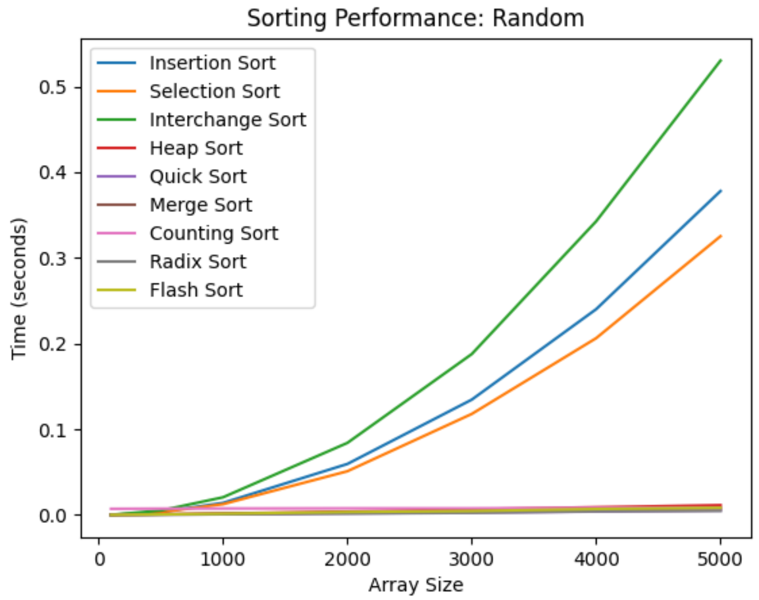
- 100
- 500
- 1000
- 2000
- 3000
- 4000
- 5000

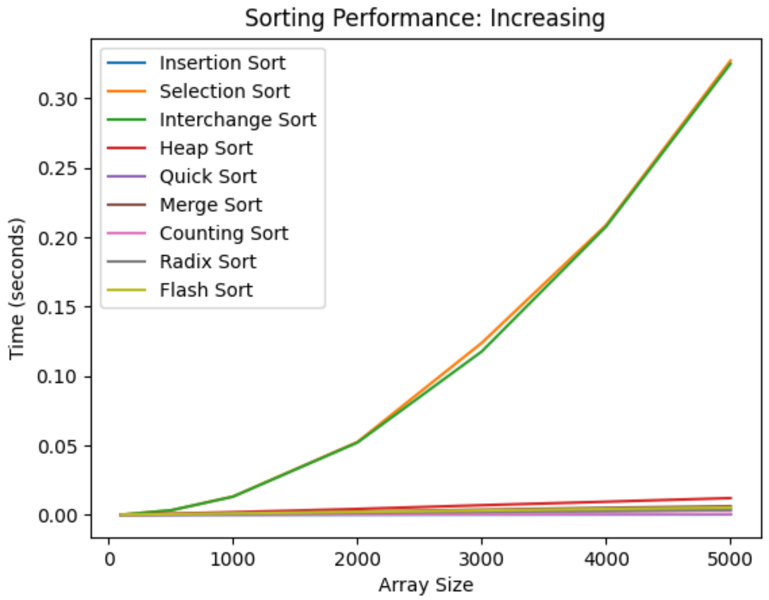
Perform experiments on the following types of arrays:

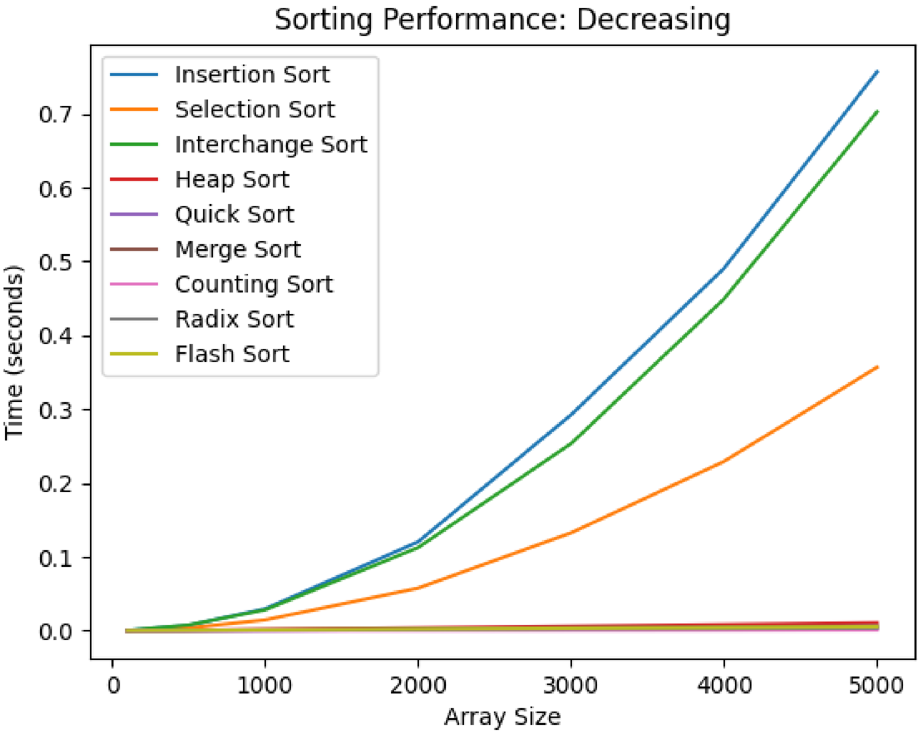
- Random array.
- Sorted in increasing order.
- Sorted in decreasing order.
- Nearly sorted (mostly sorted in increasing order with a few random elements out of place).

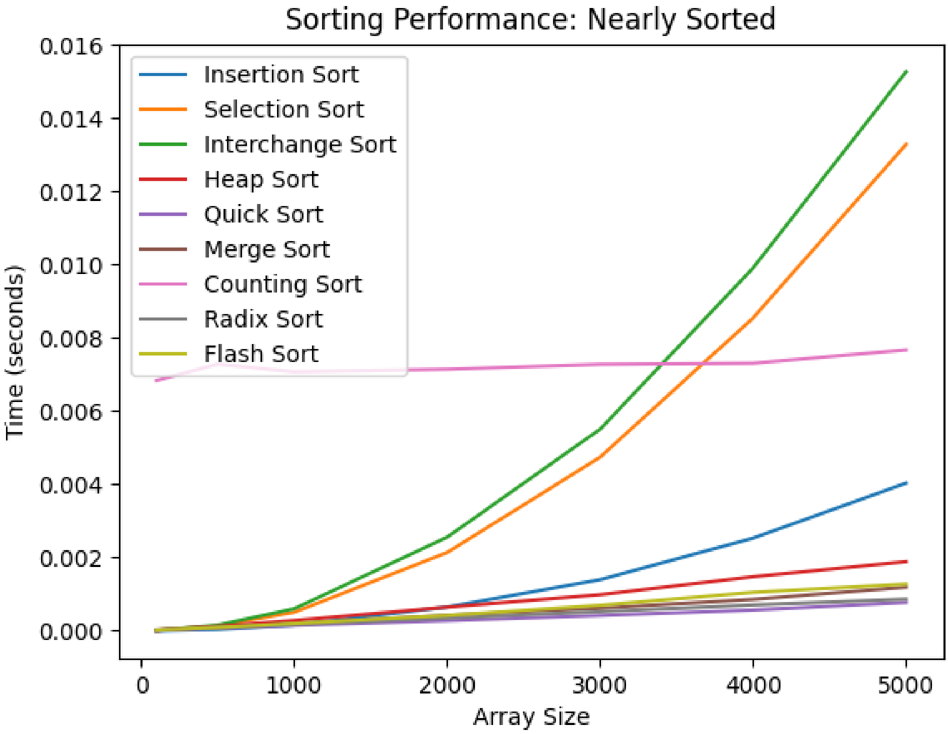
Ensure that all algorithms operate on the same initial list to maintain fairness in the comparison.

After obtaining the execution times, plot graphs using `matplotlib` to visualize the performance of each algorithm across different input sizes and array types. Label the axes appropriately and include a legend for clarity.






Sorting Performance: Decreasing
Insertion Sort
Selection Sort
Interchange Sort
Heap Sort
Quick Sort
Merge Sort
Counting Sort
Radix Sort
Flash Sort
Time (seconds)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
1000
2000
3000
4000
5000
Array Size


Sorting Performance: Nearly Sorted
Insertion Sort
Selection Sort
Interchange Sort
Heap Sort
Quick Sort
Merge Sort
Counting Sort
Radix Sort
Flash Sort
Time (seconds)
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
0.016
0
1000
2000
3000
4000
5000
Array Size

Regulations

Please follow these guidelines:

- You may use any Python IDE.
- After completing assignment, check your submission before and after uploading to Moodle.
- Do not use the following modules: `numpy`, `pandas`, `collections`, `heapq`, and `deque`.
- You may use `list`, `tuple`, and `set` but no external libraries.

Your submission must be contributed in a compressed file, named in the format `StudentID.zip`, with the following structure:

```
StudentID
├── Exercise_1.py
├── Exercise_2.py
├── Exercise_3.py
├── Exercise_4.py
└── Exercise_5.py
```

The end.