Project 3: Sorting algorithms

dr Szymon Murawski

In this project we will measure the performance of sorting algorithms. For starters please refer to lecture notes for quick review on the topic. We introduced three main classes of sorting algorithms: simple, advanced and counting, let's then implement one algorithm of each class and check how it performs in reality.

Task 1: *Array generators*

As you might remember from the lecture, performance of sorting algorithms might depend on data distribution. Because of that we would like to create a set of functions, each returning differently distributed array.

1. Remember `ArrayGenerator` class from first project? Copy it to this project.
2. That class should already have methods for creating randomly distributed array and ordered array
3. Expand it by adding methods for more data distributions: reverse ordered, constant, A and V-shaped. Refer to lecture notes (Lecture 03, slide 9) for reference

Task 2: *Simple sorting algorithm*

1. Choose one of the simple sorting algorithms (selection, insertion, bubble or cocktail sort) and implement it
2. Run the analysis, by using your implementation of sorting algorithm to sort arrays of different sizes and different distributions and measuring time it takes to perform this task.
3. Gather your data in a table like the one below. Use some Excel-like software to produce this table:

size	randomDistribution	ascendingDistribution	vShapedDistribution	...
1000	0.032ms	0.05ms	0.028ms	...
10000	0.3ms	0.5ms	0.2ms	...
...	...	...	...	...

4. For sorting to take any reasonable time use array sizes starting from 10^6 - for smaller it might end instantly
5. Make a graph out of this data

Task 3: *Advanced sorting algorithm*

1. Basically repeat the steps from the previous task, but this time choose one of the advanced algorithms (mergesort, quicksort, heapsort)

Task 4: *Counting sort*

1. Almost the same as above, choose one of counting algorithms (counting sort, radix sort) and run analysis
2. This time however not only the size of the array matters, but also how big the numbers to sort are.
3. Because of that it is better to run analysis using arraySize/digits (A/D) ratio. Take one size of array big enough (maybe 10^6) and run analysis for $A/D = 0.01, 0.1, 1, 10, 100, 1000$. For example $A/D = 10$ would mean that array is of size 10^6 and we are sorting digits that are at max 10^5 long
4. Present data gathered in a table like in previous tasks and make a graph out of this data.