# Programmer’s documentation

This program allows users to visualize multiple sorting algorithms, and compare between them. For each step of chosen algorithm, it generates a svg picture showing a current progress of sorting.

The inspiration for this program was a video, showing the process of sorting when using certain sorting algorithms. My goal was to try to make an interactive version, in which the user selects required parameters and gets similar result.

## Code architecture

This program consists of these main classes:

### Class SelectParamWin

Class for operating the visual window, in which the parameters are put. After clicking “Select folder”, method ChooseFolder_click is called, creating new FolderBrowserDialog to choose a folder. The path to the folder is stored in static variable filePath, declared at the beginning of the class. Choosing a folder, along with setting the size of an array is necessary to run the algorithm.

The size is stored in another static variable called size. If these parameters are not set after clicking button “Generate”, method Generate_clicks, which first calls method CheckErrors calls method ErrorWindow which displays a window announcing what’s wrong, letting user to correct the input. It also sets a maximum size of an array to number 9998, showing an error when the size is greater than this number. There is also an option to choose multithreaded sorting, by checking isThreaded checkbox. If the parameters are correct, an iteration of calling method Algorithms begins.

First, this method generates an array of size from 0 to size-1, where size is selected by the user, and then for each selected algorithm runs sorting. After that, it returns back, and if needed, runs another iteration. After that, a label in the bottom of the window announces that all svg files were successfully generated.

### Classes of sorting algorithms

There is a class for each of these algorithms:

**Bubble Sort.** It is the simplest algorithm, it works by repeatedly swapping adjacent elements, that are in the wrong order.

**Shakersort,** also sometimes called cocktailsort, is advanced bubblesort, that returns when it reaches an end of the array.

**Selection Sort.** It sorts an array by repeatedly finding a maximal element and putting it at the end of the array.

**Insertion Sort** iterates through the array and when it encounters an element, puts it where it belongs in a sorted array.

**Quicksort** is a divide and counter algorithm, it picks a pivot and partitions an array around the pivot. Usually the implementation is recursive, however, for my purposes, I chose the iterative implementation, for easier generating steps of algorithm in a correct order. However, I still use an auxiliary method for finding the pivot.

**Bucket Sort** doesn't use any comparing, instead this algorithm sorts an array by putting numbers in buckets according to the decimal value.

**Merge Sort** sorts with the idea, that two sorted arrays can be easily merged together. In this case I chose the recursive implementation.

Each of these sorting algorithms consist of two two stages. In first stage, in single threaded sorting, a copy of an array is sorted to measure time needed for sorting the array. In case of multithreaded sorting, the method returns after this stage.

In second stage, the array is sorted again, but this time, we count comparisons, accesses, and generate svg images.

I decided to split the sorting into these two stages, so the time can be measured as accurately as possible.

## Classes of generating files

Class **SVGgenerate** contains method GenerateSVG, which takes an array and necessary parameters, and creates a svg file into a folder set by static variable filePath from SelectWinForm class. Due to svg being xml based, we can write the code to get an image. So using a streamwriter, the header of svg file is written into a file, following by creating a rectangle for each element of an array, height of the rectangle is calculated according to the size of given element. These rectangles create a graph representing an array.
Under the array, following parameters are written: Name of the algorithm, number of comparisons, accesses to elements in array and elapsed time in the current run.

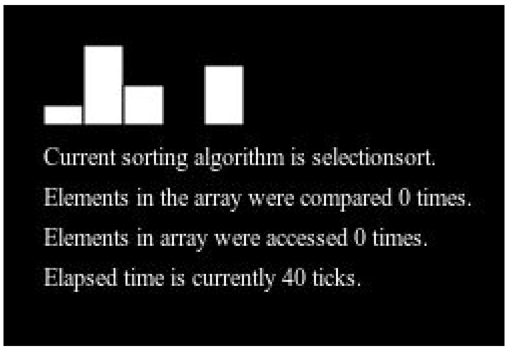
**InfoFile** is a class for generating a text file summarizing the output of current sorting: It writes out the size of an array, algorithm used and time needed to sort the array in ticks. If the sorting ran on multiple threads, number of threads is also written here, otherwise, number of comparisons and accesses is shown.

# Example

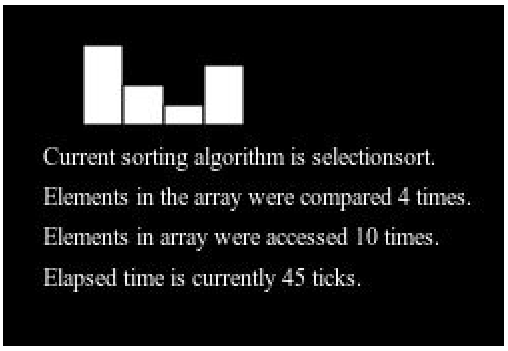
## 1. Input parameters:

Algorithm "Selection sort" is checked, size of an array is set to 5, number of iterations is 1, only single threaded sorting.

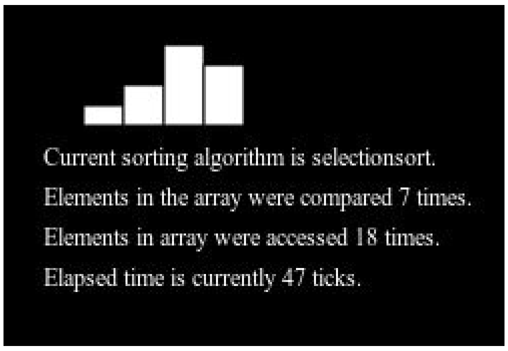
On the next page are generated images of the sorting algorithm.



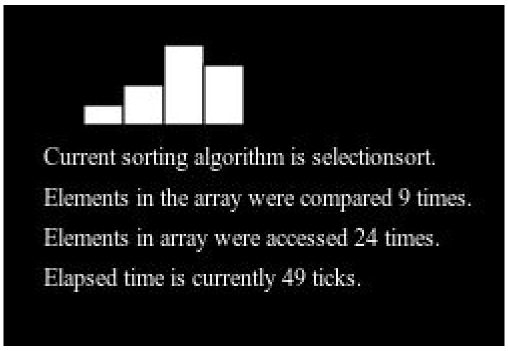
Current sorting algorithm is selectionsort.
Elements in the array were compared 0 times.
Elements in array were accessed 0 times.
Elapsed time is currently 40 ticks.



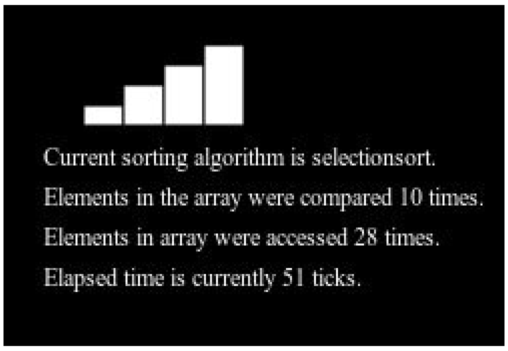
Current sorting algorithm is selectionsort.
Elements in the array were compared 4 times.
Elements in array were accessed 10 times.
Elapsed time is currently 45 ticks.



Current sorting algorithm is selectionsort.
Elements in the array were compared 7 times.
Elements in array were accessed 18 times.
Elapsed time is currently 47 ticks.



Current sorting algorithm is selectionsort.
Elements in the array were compared 9 times.
Elements in array were accessed 24 times.
Elapsed time is currently 49 ticks.



Current sorting algorithm is selectionsort.
Elements in the array were compared 10 times.
Elements in array were accessed 28 times.
Elapsed time is currently 51 ticks.

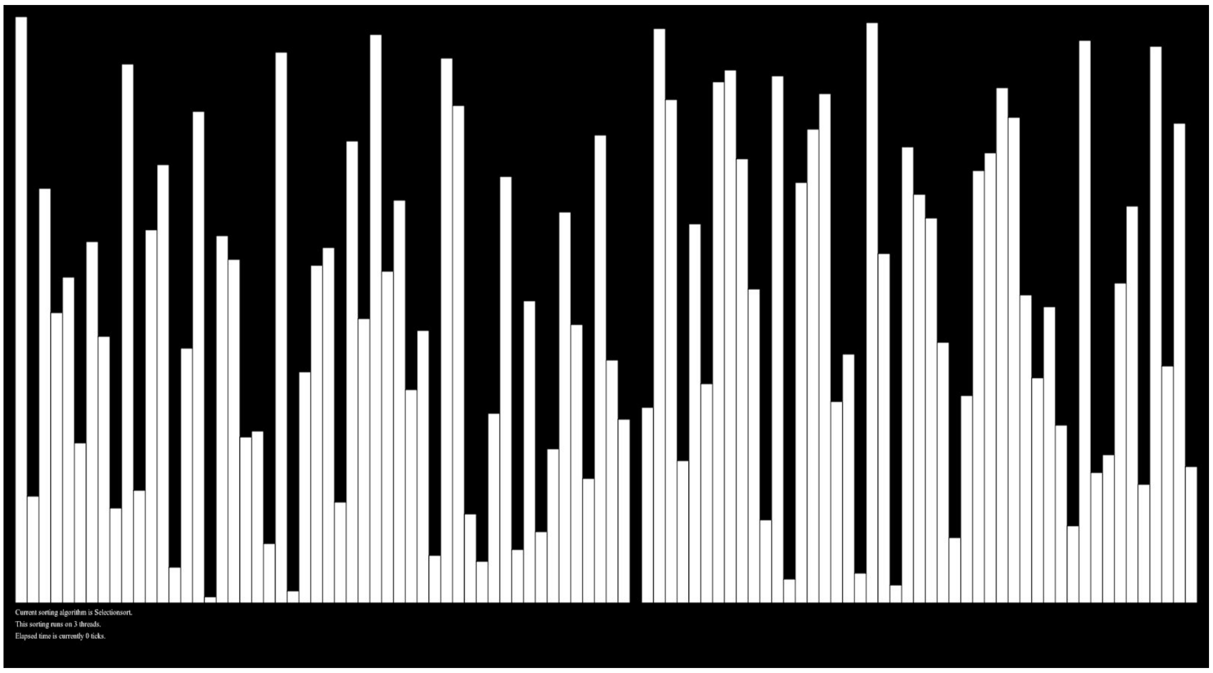
This is what is contained in affiliated text file:

An array with 5 items has been sorted.
It took 55 ticks to finish sorting with selectionsort.
There were needed 10 comparisons and 28 accesses to the elements of an array.

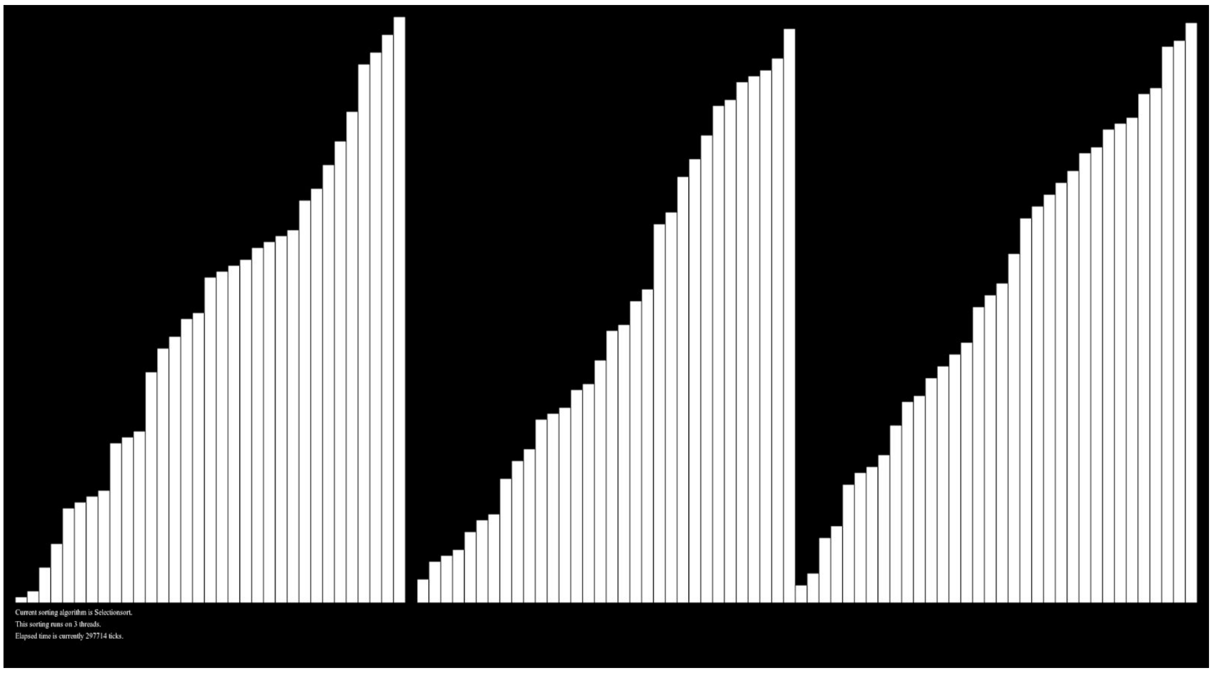
## 2. Input parameters:

Algorithm “Selection sort” is checked, size of an array is set to 100, number of iterations is 1, number of threads is 3.

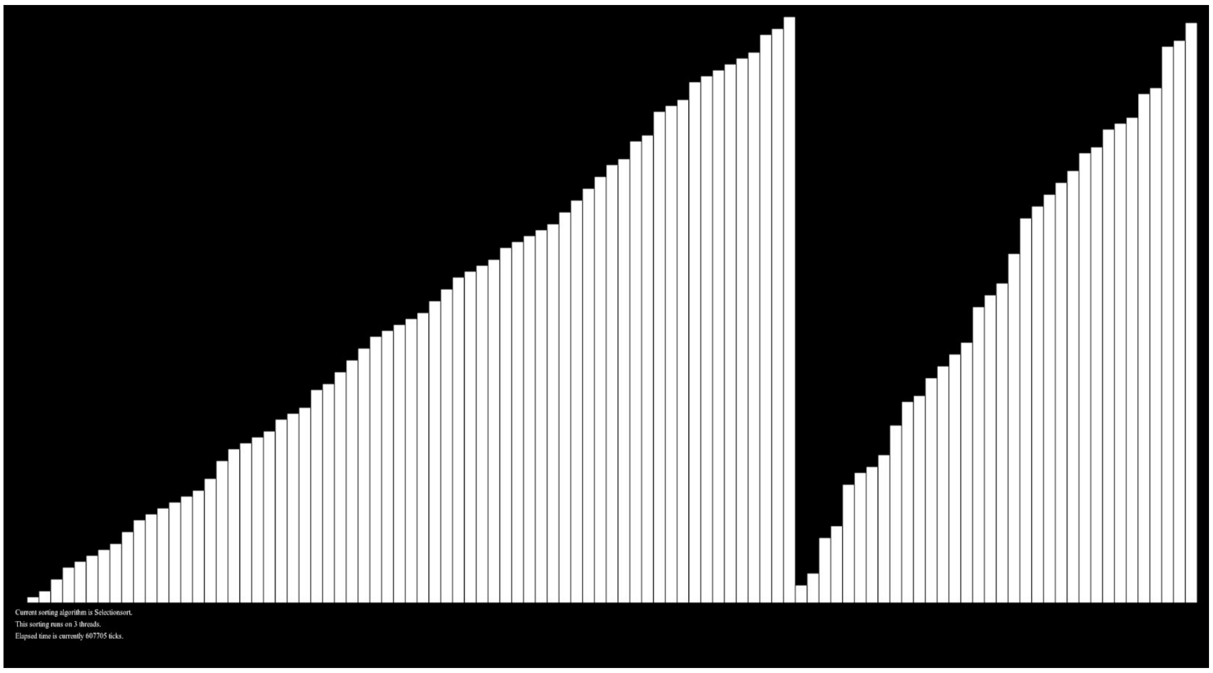
Below are generated images of the sorting algorithm.



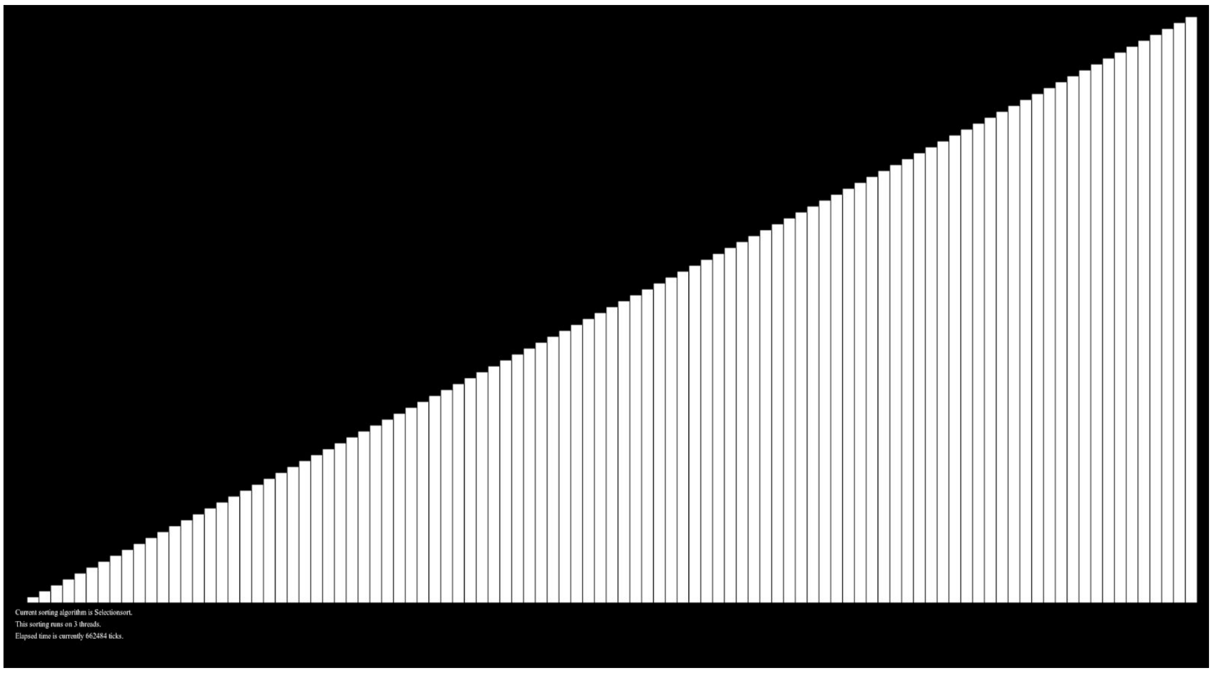
```
Current sorting algorithm is Selectionsort.
This sorting runs on 3 threads.
Elapsed time is currently 0 ticks.
```



```
Current sorting algorithm is Selectionsort.
This sorting runs on 3 threads.
Elapsed time is currently 297714 ticks.
```



```
Current sorting algorithm is Selectionsort.
This sorting runs on 3 threads.
Elapsed time is currently 607705 ticks.
```



```
Current sorting algorithm is Selectionsort.
This sorting runs on 3 threads.
Elapsed time is currently 662484 ticks.
```

This is what is contained in affiliated text file:

An array with 100 items has been sorted.
It took 730672 ticks to finish sorting with Selectionsort.
There were 3 threads sorting the array.