Laboratory work #1 (Siarhei Savaniuk AI-10)

Source code (write in Java):

File Individual.java:

```
import java.util.Arrays;
import java.util.concurrent.ThreadLocalRandom;

class Individual implements Comparable<Individual> {

    public static final int GENE_LENGTH = 1000;

    private int[] genes;
    private int fitnessValue;

    public Individual(boolean initialize) {
        genes = new int[GENE_LENGTH];

        if (initialize) {
            generateIndividual();
            fitnessValue = getFitness();
        }
    }

    public Individual(int[] genes) {
        this.genes = genes;
        fitnessValue = getFitness();
    }

    public int getFitnessValue() {
        return fitnessValue;
    }

    public void setFitnessValue(int fitnessValue) {
        this.fitnessValue = fitnessValue;
    }

    public void generateIndividual() {
        for (int i = 0; i < GENE_LENGTH; ++i) {
            genes[i] = ThreadLocalRandom.current().nextInt(2);
        }
    }

    public int getFitness() {
        int fitness = 0;

        for (int gene : genes) {
            if (gene == 1) {
                ++fitness;
            }
        }

        return fitness;
    }

    public int[] getGenesBeforeCutPoint(int cutPoint) {
        int[] genes = new int[cutPoint];

        System.arraycopy(this.genes, 0, genes, 0, cutPoint);

        return genes;
    }
```

```
    public int[] getGenesAfterCutPoint(int cutPoint) {
        int[] genes = new int[GENE_LENGTH - cutPoint];

        System.arraycopy(this.genes, cutPoint, genes, 0, genes.length);

        return genes;
    }

    @Override
    public String toString() {
        return "Individual{" +
                "fitnessValue=" + fitnessValue +
                ", genes=" + Arrays.toString(genes) +
                '}' + '\n';
    }

    @Override
    public int compareTo(Individual o) {
        return (o.getFitnessValue() > fitnessValue ? 1 : (o.getFitnessValue()
== fitnessValue) ? 0 : -1);
    }
}
```

File Population.java:

```
import java.util.Arrays;

class Population {

    public static final int POPULATION_SIZE = 100;

    private Individual[] individuals;

    public Population(boolean initialize) {
        individuals = new Individual[POPULATION_SIZE];

        if (initialize) {
            for (int i = 0; i < POPULATION_SIZE; ++i) {
                individuals[i] = new Individual(true);
            }
        }
    }

    public Population(Individual[] individuals) {
        this.individuals = new Individual[POPULATION_SIZE];
        System.arraycopy(individuals, 0, this.individuals, 0,
individuals.length);
    }

    public Individual getIndividual(int index) {
        return individuals[index];
    }

    public void addIndividual(int index, Individual individual) {
        individuals[index] = individual;
    }

    public Individual[] getHalfFittestIndividuals() {
        Individual[] fittestIndividuals = new Individual[POPULATION_SIZE /
2];

        System.arraycopy(individuals, 0, fittestIndividuals, 0,
fittestIndividuals.length);

        return fittestIndividuals;
```

```
    }

    public int getMaxFitness() {
        return individuals[0].getFitnessValue();
    }

    public double getAverageFitness() {
        double sum = 0;

        for (int i = 0; i < POPULATION_SIZE; ++i) {
            sum += individuals[i].getFitnessValue();
        }

        return sum / POPULATION_SIZE;
    }

    public Individual[] getAllIndividuals() {
        return individuals;
    }

    @Override
    public String toString() {
        return "Population{\n" + Arrays.toString(individuals) + "}\n";
    }
}
```

File GeneticAlgorithm.java:

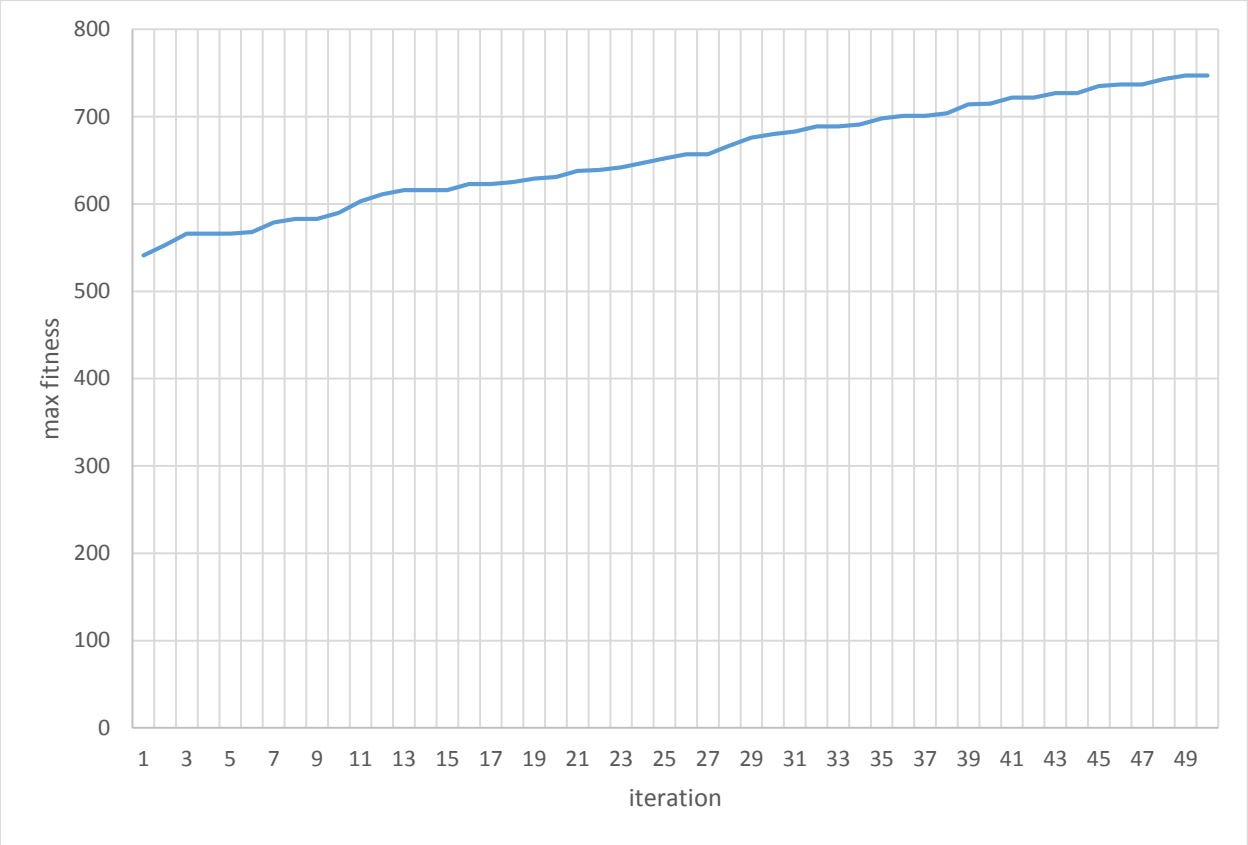
```
import java.util.Arrays;
import java.util.concurrent.ThreadLocalRandom;

public class GeneticAlgorithm {

    private Population population;

    public GeneticAlgorithm(Population population) {
        this.population = population;
    }

    public Population run() {
        Arrays.sort(population.getAllIndividuals());

        Population halfPopulation = new
Population(population.getHalfFittestIndividuals());
        Population nextGeneration = new
Population(halfPopulation.getAllIndividuals());

        for (int i = 0, j = Population.POPULATION_SIZE / 2; i <
Population.POPULATION_SIZE / 4; ++i, j += 2) {
            Individual[] parents = chooseParents(halfPopulation);

            int cutPoint = ThreadLocalRandom.current().nextInt(1000);

            Individual[] descendants = crossover(parents, cutPoint);

            nextGeneration.addIndividual(j, descendants[0]);
            nextGeneration.addIndividual(j + 1, descendants[1]);
        }

        population = nextGeneration;

        Arrays.sort(population.getAllIndividuals());

        return population;
    }
```

```
    private Individual[] chooseParents(Population fittestIndividuals) {
        return new Individual[]
{fittestIndividuals.getIndividual(ThreadLocalRandom.current().nextInt(50)),

fittestIndividuals.getIndividual(ThreadLocalRandom.current().nextInt(50))};
    }

    private Individual[] crossover(Individual[] parents, int curPoint) {
        Individual[] descendants = new Individual[2];

        int[] firstDescendantGenes =
concat(parents[0].getGenesBeforeCutPoint(curPoint),

parents[1].getGenesAfterCutPoint(curPoint));
        int[] secondIndividualGenes =
concat(parents[0].getGenesAfterCutPoint(curPoint),

parents[1].getGenesBeforeCutPoint(curPoint));

        descendants[0] = new Individual(firstDescendantGenes);
        descendants[1] = new Individual(secondIndividualGenes);

        return descendants;
    }

    private int[] concat(int[] genes1, int[] genes2) {
        int[] genes = new int[Individual.GENE_LENGTH];

        System.arraycopy(genes1, 0, genes, 0, genes1.length);
        System.arraycopy(genes2, 0, genes, genes1.length, genes2.length);

        return genes;
    }
}
```

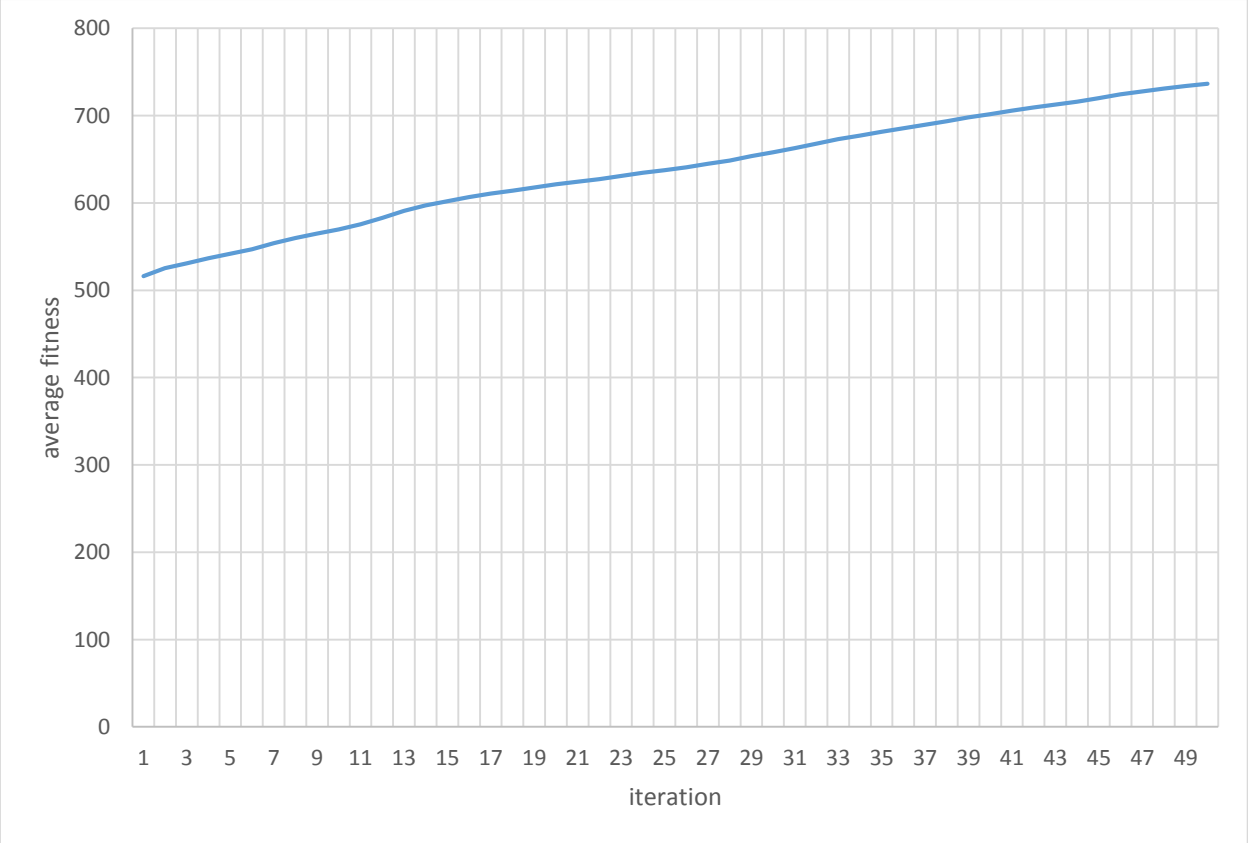
File Main.java:

```
import java.io.*;

public class Main {

    public static void main(String[] args) throws IOException {
        Population population = new Population(true);
        GeneticAlgorithm geneticAlgorithm = new GeneticAlgorithm(population);

        for (int i = 0; i < 100; ++i) {
            Population newPopulation = geneticAlgorithm.run();
            System.out.println("Max = " + newPopulation.getMaxFitness() + ",
Average = " + newPopulation.getAverageFitness());
        }
    }
}
```

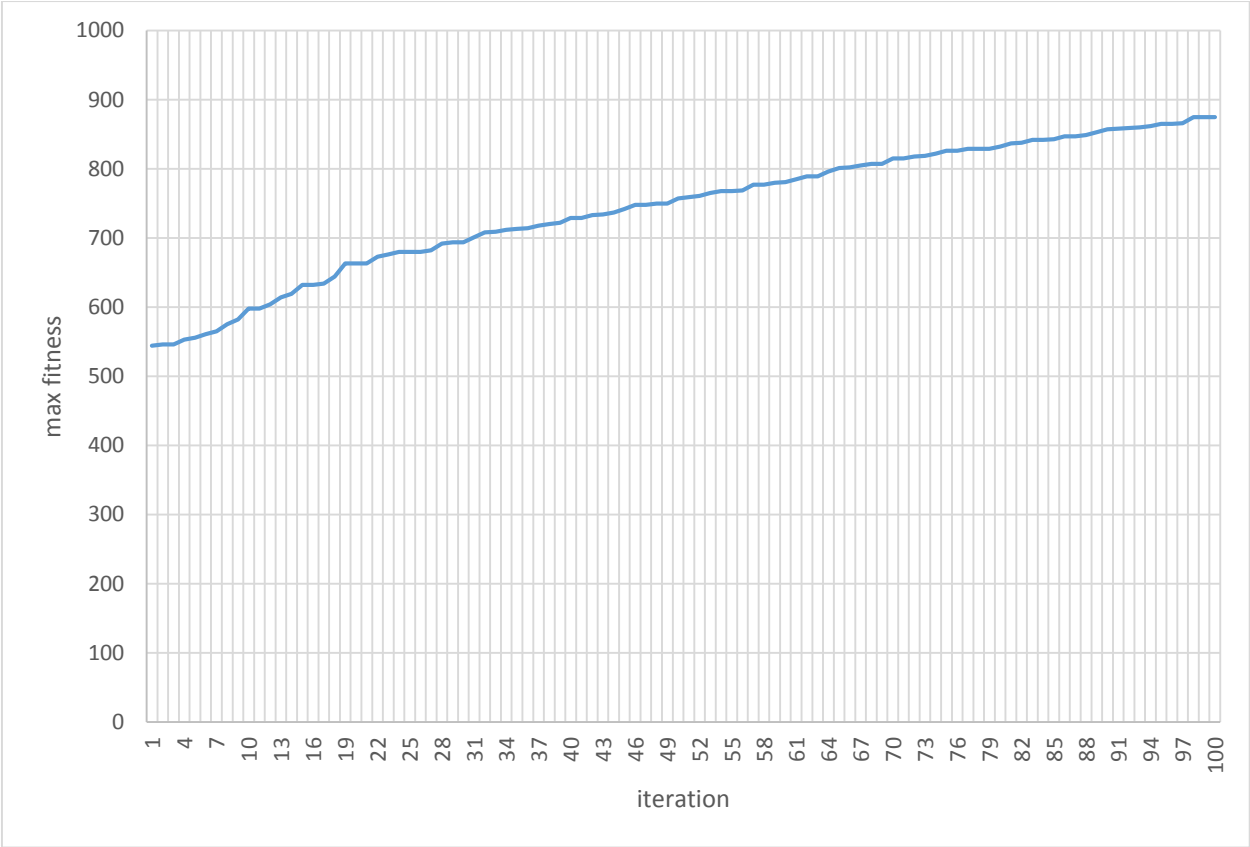
Testing:



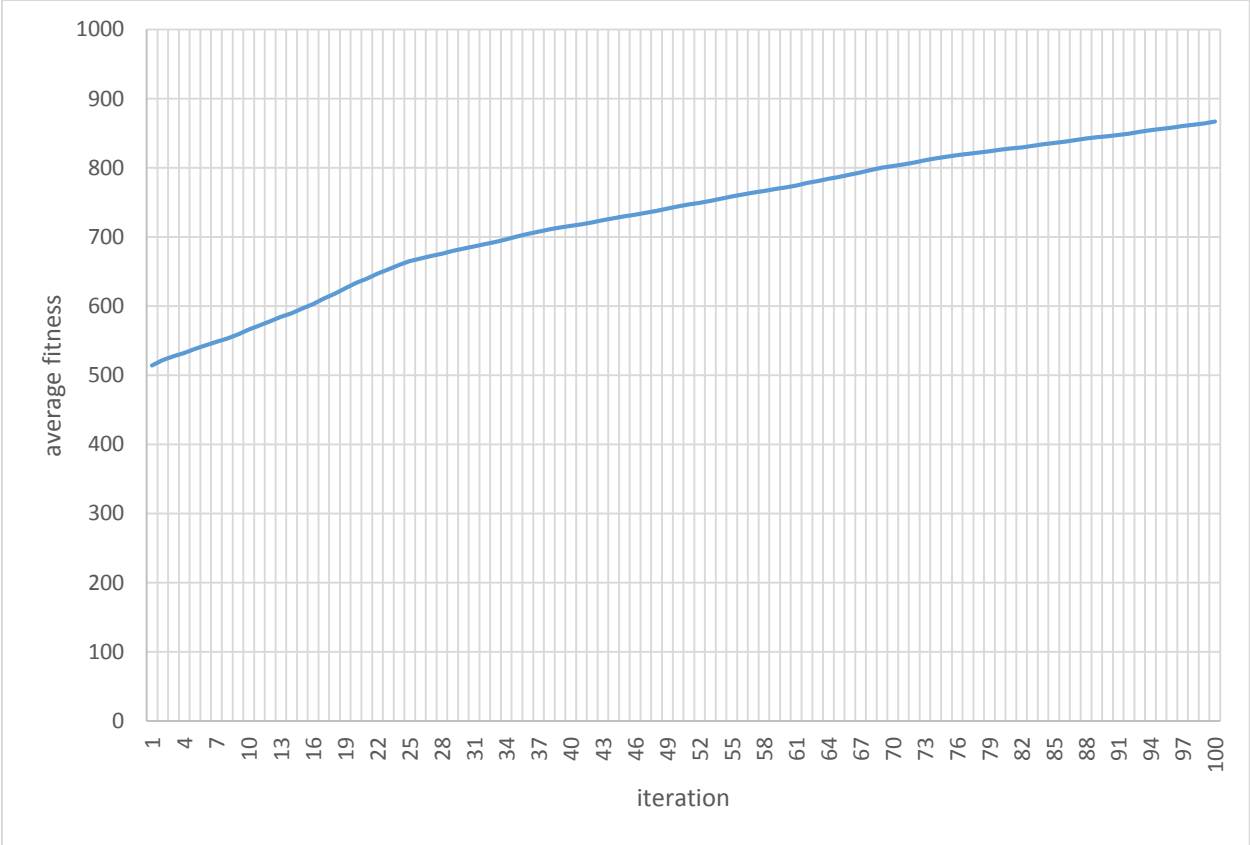
Graphics - Dependence of the max value of fitness function from iteration (**50** iterations)



Graphics - Dependence of the average value of fitness function from iteration **(50** iterations)



Graphics - Dependence of the max value of fitness function from iteration (**100** iterations)



Graphics - Dependence of the average value of fitness function from iteration (**100** iterations)