

Task 1

1. Create 100 random binary-chromosomes each with 1000 genes.
2. Fitness is the number of "1" in one chromosome - the more the better.
3. Select 2 chromosomes at random from the better half of the population.
4. Create a child chromosome by a onr-point-crossover.
5. Give the child a mutation with a probability of $1/1000 = 0.001$.
6. Repeat from 2. to 5. 100 times and create the next generation.
7. Repeat 6. until the fitness value does not change any more.
8. Show the result:
 - (1) Display the best chromosome in the 1st, an intermediate & final generation.
 - (2) Display the best and average fitness vs. generation.

Source code (Java)

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;

public class Main
{
    static class Chromosome
    {
        int[] genes;
        Chromosome()
        {
            genes = new int[1000];
            for(int i = 0; i < genes.length; i++)
                genes[i] = random.nextInt(2);
        }
        int getFitness()
        {
            int cnt = 0;
            for(int i : genes)
                if(i == 1)
                    cnt++;
            return cnt;
        }
    }

    static class Generation
    {
        Chromosome[] chromosomes;
        boolean isSorted = false;
        Generation()
        {
            chromosomes = new Chromosome[100];
            for(int i = 0; i < chromosomes.length; i++)
                chromosomes[i] = new Chromosome();
        }
    }
}
```

```

int getAverageFitness()
{
    int cnt = 0;
    for(Chromosome i : chromosomes)
        cnt += i.getFitness();
    return cnt / chromosomes.length;
}

void sort()
{
    Arrays.sort(chromosomes, (o1, o2) ->
    {
        int o1fit = o1.getFitness();
        int o2fit = o2.getFitness();
        if(o1fit < o2fit)
            return 1;
        else if(o1fit > o2fit)
            return -1;
        else
            return 0;
    });
    isSorted = true;
}

Chromosome[] get2RandomChromosomes()
{
    if(!isSorted)
        sort();
    Chromosome[] random2 = new Chromosome[2];

    random2[0] = chromosomes[random.nextInt(50)];
    random2[1] = chromosomes[random.nextInt(50)];

    return random2;
}

Chromosome getBestChromosome()
{
    if(!isSorted)
        sort();
    return chromosomes[0];
}
}

static Random random;
static ArrayList<Generation> generations;
static ArrayList<Integer> fitnessHistory;

static Chromosome crossover(Chromosome first, Chromosome second)
{
    int crossoverPoint = random.nextInt(first.genes.length);

    Chromosome child = new Chromosome();

    System.arraycopy(first.genes, 0, child.genes, 0, crossoverPoint);
    System.arraycopy(second.genes, crossoverPoint, child.genes, crossoverPoint,
        second.genes.length - crossoverPoint);

    if(random.nextInt(1000) == 871)
    {
        int pos = random.nextInt(child.genes.length);
        if(child.genes[pos] == 0)
            child.genes[pos] = 1;
        else
            child.genes[pos] = 0;
    }

    return child;
}

```

```

static boolean isLastGenerationUnchanged()
{
    int tmp = fitnessHistory.get(fitnessHistory.size() - 1);
    if(tmp==1000)
        return true;
    if(fitnessHistory.size() <= 100)
        return false;
    for(int i = 0; i < 100; i++)
        if(fitnessHistory.get(fitnessHistory.size()-i-1) != tmp)
            return false;
    return true;
}

public static void main(String[] args) throws InterruptedException
{
    random = new Random(System.currentTimeMillis());
    fitnessHistory = new ArrayList<>();
    generations = new ArrayList<>();
    generations.add(new Generation());
    generations.get(0).sort();
    fitnessHistory.add(generations.get(0).getAverageFitness());
    System.out.println("Generation: 0; Fitness: " +
        generations.get(0).getAverageFitness() +
        "; Best chromosome: " + generations.get(0).getBestChromosome().getFitness());

    for(int i = 1; ; i++)
    {
        Generation newGeneration = new Generation();
        for(int j = 0; j < newGeneration.chromosomes.length; j++)
        {
            Chromosome[] random2 = generations.get(generations.size()-
1).get2RandomChromosomes();
            newGeneration.chromosomes[j] = crossover(random2[0], random2[1]);
        }
        newGeneration.sort();
        generations.add(newGeneration);
        generations.remove(0);
        fitnessHistory.add(newGeneration.getAverageFitness());
        System.out.println(i + " " + newGeneration.getAverageFitness() + " " +
newGeneration.getBestChromosome().getFitness());

        if(isLastGenerationUnchanged())
            break;
    }

    Thread.sleep(1000);
}
}

```

Result

- 1) The best chromosome in the 1st generation - 542;
The best chromosome in the 36000th generation - 982;
The best chromosome in the last (73386th) generation - 1000;

2)

