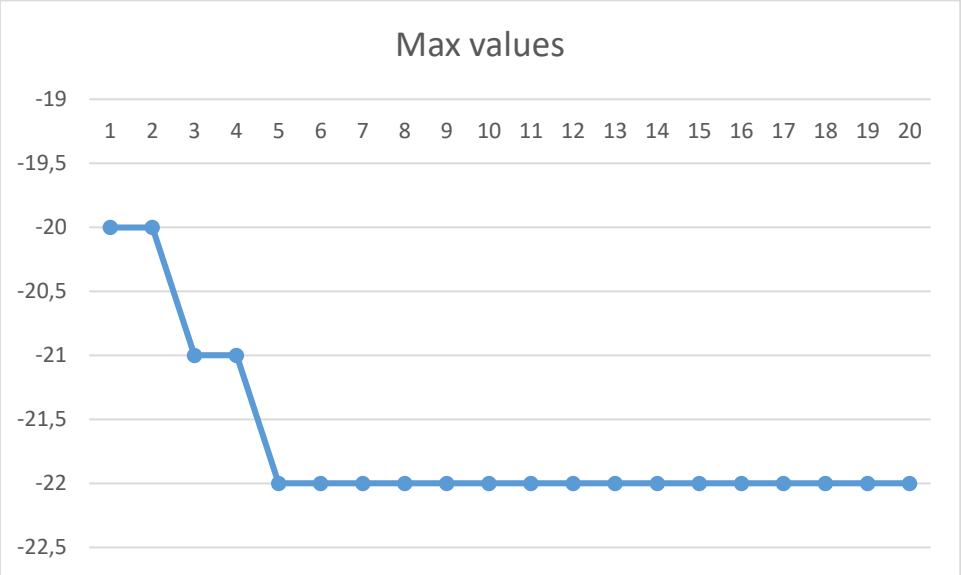
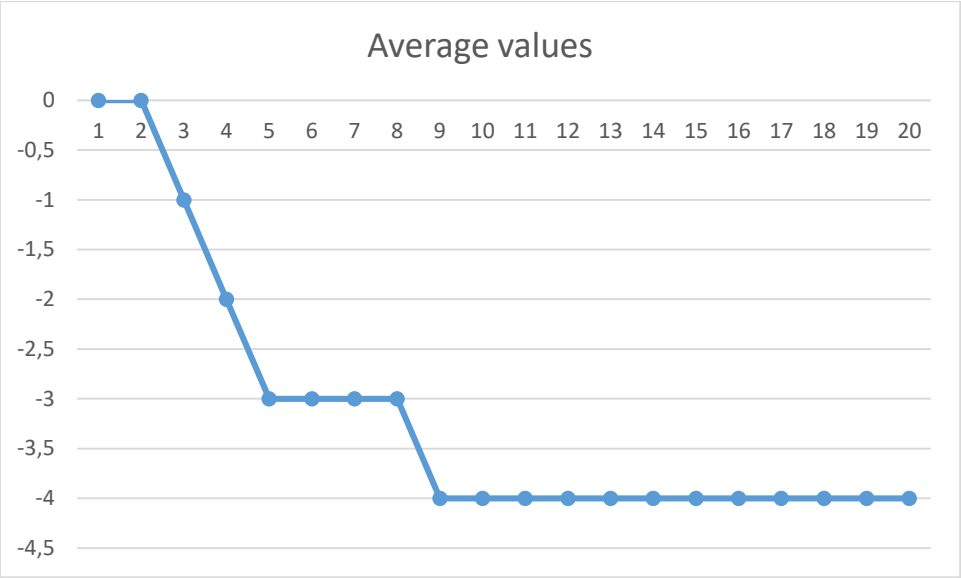
Task-1

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <math.h>
#include <time.h>

using namespace std;

typedef vector<double> Individual;
typedef vector<Individual> Population;

void makePopulation();
void getPopulation();
int ChromosomeSumm(const Individual& population);
bool ChromosomeSort(const Individual& population1, const Individual& population2);
void print();
void crossing();

Population population;

int main()
{
	srand(time(NULL));

	makePopulation();
	getPopulation();

	sort(population.begin(), population.end(), ChromosomeSort);

	ofstream file("DataForGraph.txt", ios::out);

	file << "Max Average";
	file << '\n';

	for (int i = 0; i < 20; i++)
	{
		crossing();

		sort(population.begin(), population.end(), ChromosomeSort);

		int Max = 0;
		int Mean = 0;

		for (int k = 0; k < population[0].size(); k++) {
			Max += population[0][k];
		}

		for (int j = 0; j < population.size(); j++) {
			for (int k = 0; k < population[j].size(); k++) {
				Mean += population[j][k];
			}
		}
		Mean = Mean / 20;

		file << Max << " " << Mean;
		file << '\n';
	}

	file.close();

	print();
```

```
	system("Pause");
	return 0;
}

void makePopulation() {
	ofstream file("population.txt", ios::out);
	for (int i = 0; i < 20; i++) {
		for (int j = 0; j < 20; j++) {
			file << ((rand() % 1000) - 500)/100.0;
			if (j == 20 - 1) file << '\n';
			else file << ' ';
		}
	}
	file.close();
}

void getPopulation() {
	ifstream file("population.txt");
	for (int i = 0; i < 20; i++) {
		Individual row;
		for (int j = 0; j < 20; j++) {
			double value;
			file >> value;
			row.push_back(value);
		}
		population.push_back(row);
	}
	file.close();
}

int ChromosomeSumm(const Individual& population) {
	int summ = 0;
	for (int i = 0; i < population.size(); i++) {
		summ += population[i] * sin(abs(population[i]));
	}
	return summ;
}

bool ChromosomeSort(const Individual& population1, const Individual& population2) {
	return ChromosomeSumm(population1) > ChromosomeSumm(population2);
}

void print(){
	ofstream file("sort.txt", ios::out);
	for (int i = 0; i < population.size(); i++) {
		int tmp = 0;
		for (int j = 0; j < population[i].size(); j++) {
			file << population[i][j];
			if (j == 20 - 1) file << '\n';
			else file << ' ';
		}
	}
	file.close();
}

void crossing() {
	int CrossingPoint;
	int FirstParent, SecondParent;
	Population resultPopulation;

	for (int i = 0; i < 10; i++){
		CrossingPoint = rand() % 20;
		FirstParent = rand() % 10;
		SecondParent = rand() % 10;
		if (FirstParent == SecondParent){
```

```
                while (FirstParent != SecondParent)
                {
                        SecondParent = rand() % 10;
                }
        }

        Individual Child1, Child2;

        for (int j = 0; j < population[i].size(); j++)
        {
                if (j < CrossingPoint)
                {
                        Child1.push_back(population[FirstParent][j]);
                        Child2.push_back(population[SecondParent][j]);
                }
                else
                {
                        Child1.push_back(population[SecondParent][j]);
                        Child2.push_back(population[FirstParent][j]);
                }
        }
        resultPopulation.push_back(Child1);
        resultPopulation.push_back(Child2);
    }
    population = resultPopulation;
}
```





Task – 2

```
#include "stdafx.h"
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <math.h>
#include <time.h>

using namespace std;

typedef vector<double> Individual;
typedef vector<Individual> Population;

void makePopulation();
void getPopulation();
int ChromosomeSumm(const Individual& population);
bool ChromosomeSort(const Individual& population1, const Individual& population2);
void print();
void crossing();
double x, y;

Population population;

int main()
{
	srand(time(NULL));

	makePopulation();
	getPopulation();

	sort(population.begin(), population.end(), ChromosomeSort);

	ofstream file("DataForGraph.txt", ios::out);

	file << "X Y";
	file << '\n';

	for (int i = 0; i < 10; i++)
	{

		file << "Iteration N " << i + 1;
		file << '\n';
```

```
			crossing();

			sort(population.begin(), population.end(), ChromosomeSort);

			for (int j = 0; j < population.size(); j++) {
				double summ = 0;
				summ = population[i][1] * 1 + population[i][2] * 2 + population[i][3]
* 4 + population[i][4] * 8 + population[i][5] * 16 + population[i][6] * 32 +
population[i][7] * 64 + population[i][8] * 128 + population[i][9] * 256;
				if (population[i][0] == 1){
					summ = summ * 1;
				}
				else summ = summ * -1;
				summ = summ;
				x = summ / 1023 * 5;
				summ = summ * sin(abs(summ));
				y = summ / 1023 * 5;
				file << x << " " << y;
				file << '\n';
			}
		}

		file.close();

		print();
		system("Pause");
		return 0;
}

void makePopulation() {
		ofstream file("population.txt", ios::out);
		for (int i = 0; i < 20; i++) {
			for (int j = 0; j < 10; j++) {
				file << rand() % 2;
				if (j == 10 - 1) file << '\n';
				else file << ' ';
			}
		}
		file.close();
}

void getPopulation() {
		ifstream file("population.txt");
		for (int i = 0; i < 20; i++) {
			Individual row;
			for (int j = 0; j < 10; j++) {
				double value;
				file >> value;
				row.push_back(value);
			}
			population.push_back(row);
		}
		file.close();
}

int ChromosomeSumm(const Individual& population) {
		double summ = 0;
		summ = population[1] * 1 + population[2] * 2 + population[3] * 4 + population[4] *
8 + population[5] * 16 + population[6] * 32 + population[7] * 64 + population[8] * 128 +
population[9] * 256;
		if (population[0] == 1){
			summ = summ * 1;
		}
		else summ = summ * -1;
		summ = summ / 1023 * 5;
```

```
	summ = summ * sin(abs(summ));
	return summ;
}

bool ChromosomeSort(const Individual& population1, const Individual& population2) {
	return ChromosomeSumm(population1) > ChromosomeSumm(population2);
}

void print(){
	ofstream file("sort.txt", ios::out);
	for (int i = 0; i < population.size(); i++) {
		int tmp = 0;
		for (int j = 0; j < population[i].size(); j++) {
			file << population[i][j];
			if (j == 10 - 1) file << '\n';
			else file << ' ';
		}
	}
	file.close();
}

void crossing() {
	int CrossingPoint;
	int FirstParent, SecondParent;
	Population resultPopulation;

	for (int i = 0; i < 10; i++){
		CrossingPoint = rand() % 10;
		FirstParent = rand() % 10;
		SecondParent = rand() % 10;
		if (FirstParent == SecondParent){
			while (FirstParent != SecondParent)
			{
				SecondParent = rand() % 10;
			}
		}

		Individual Child1, Child2;

		for (int j = 0; j < population[i].size(); j++)
		{
			if (j < CrossingPoint)
			{
				Child1.push_back(population[FirstParent][j]);
				Child2.push_back(population[SecondParent][j]);
			}
			else
			{
				Child1.push_back(population[SecondParent][j]);
				Child2.push_back(population[FirstParent][j]);
			}
		}
		resultPopulation.push_back(Child1);
		resultPopulation.push_back(Child2);
	}
	population = resultPopulation;
}
```