Задача:

Решить с помощью генетического алгоритма задачу: распределить числа от 1 до 10 в таком порядке, чтобы большие элементы были как можно ближе к центру, а малые – как можно ближе к краям. Примеры хорошего распределения: а) 1 3 5 7 9 10 8 6 4 2, б) 1 4 5 8 9 10 7 6 3 2.

Генетический алгоритм

Переменные:

- Размер популяции: N.
- Вероятность мутации: P-mutation .
- Число поколений: G.

Шаги решения:

1. **Инициализация популяции**: создать N случайных перестановок чисел 1...10.
2. **Оценка приспособленности**: вычислить fitness для каждой перестановки.
3. **Селекция**: отобрать лучших индивидов.
4. **Кросовер**: создать новых индивидов путём обмена частями между родителями.
5. **Мутация**: случайно поменять местами два числа в перестановке с вероятностью P-mutation .
6. **Обновление популяции**: заменить старую популяцию новой.
7. Повторять шаги G-раз.

Алгоритм вычисления fitness:

Для оценки качества распределения нужно учесть два условия:

1. Чем ближе большие числа к центру, тем лучше.
2. Чем ближе малые числа к краям, тем лучше.

Формализуем это:

- Центральность числа x вычисляется как расстояние до центра: $dist(x)=|index(x)-5.5|$.

- Функция штрафа: $fitness_x = x \cdot dist(x)$, где dist(x) — расстояние числа x от центра.
- Сумируем штрафы всех чисел в последовательности $fitness = fitness_{x1} + fitness_{x2} + fitness_{xn}$ ….

При таком подходе алгоритм будет стремится минимизировать число fitness соответственно найдет идеальную последовательность чисел

Код

```
using System;

using System.Collections.Generic;

using System.Linq;


class GeneticAlgorithm

{

  // Параметры

  const int PopulationSize = 100;

  const int Generations = 500;

  const double MutationProbability = 0.1;

  static Random random = new Random();


  static void Main(string[] args)

  {

    // Инициализация популяции

    List<int[]> population = InitializePopulation(PopulationSize, 10);


    // Эволюция

    for (int generation = 0; generation < Generations; generation++)

    {
```

```
// Вычисление fitness для каждого индивида
Dictionary<int[], double> scores = population.ToDictionary(
   individual => individual,
   individual => Fitness(individual)
);

// Селекция
List<int[]> selected = TournamentSelection(population, scores, PopulationSize / 2);

// Кроссовер
List<int[]> children = new List<int[]>();
for (int i = 0; i < selected.Count; i += 2)
{
   if (i + 1 < selected.Count)
   {
      var (child1, child2) = Crossover(selected[i], selected[i + 1]);
      children.Add(child1);
      children.Add(child2);
   }
}

// Мутация
foreach (var child in children)
{
   Mutate(child);
}
```

```
        // Обновление популяции
        population = children;
    }

    // Поиск лучшего решения
    int[] best = population.OrderBy(Fitness).First();
    Console.WriteLine("Лучшее распределение: " + string.Join(" ", best));
    Console.WriteLine("Значение fitness: " + Fitness(best));
}

// Функция fitness
static double Fitness(int[] sequence)
{
    double center = sequence.Length / 2.0;
    return sequence.Select((value, index) => value * Math.Abs(index + 1 - center)).Sum();
}

// Инициализация популяции
static List<int[]> InitializePopulation(int populationSize, int sequenceLength)
{
    var population = new List<int[]>();
    for (int i = 0; i < populationSize; i++)
    {
        int[] individual = Enumerable.Range(1, sequenceLength).OrderBy(_ =>
random.Next()).ToArray();
```

```
        population.Add(individual);
    }
    return population;
}

// Турнирная селекция
static List<int[]> TournamentSelection(List<int[]> population, Dictionary<int[], double>
scores, int selectedSize)
{
    var selected = new List<int[]>();
    for (int i = 0; i < selectedSize; i++)
    {
        int[] best = population[random.Next(population.Count)];
        for (int j = 0; j < 2; j++) // Турнир из 3
        {
            int[] contender = population[random.Next(population.Count)];
            if (scores[contender] < scores[best])
            {
                best = contender;
            }
        }
        selected.Add(best);
    }
    return selected;
}
```

```
    // Кроссовер
    static (int[], int[]) Crossover(int[] parent1, int[] parent2)
    {
        int point = random.Next(1, parent1.Length - 1);
        int[] child1 = parent1.Take(point).Concat(parent2.Where(x
=> !parent1.Take(point).Contains(x))).ToArray();
        int[] child2 = parent2.Take(point).Concat(parent1.Where(x
=> !parent2.Take(point).Contains(x))).ToArray();
        return (child1, child2);
    }

    // Мутация
    static void Mutate(int[] sequence)
    {
        if (random.NextDouble() < MutationProbability)
        {
            int i = random.Next(sequence.Length);
            int j = random.Next(sequence.Length);
            (sequence[i], sequence[j]) = (sequence[j], sequence[i]);
        }
    }
}
```