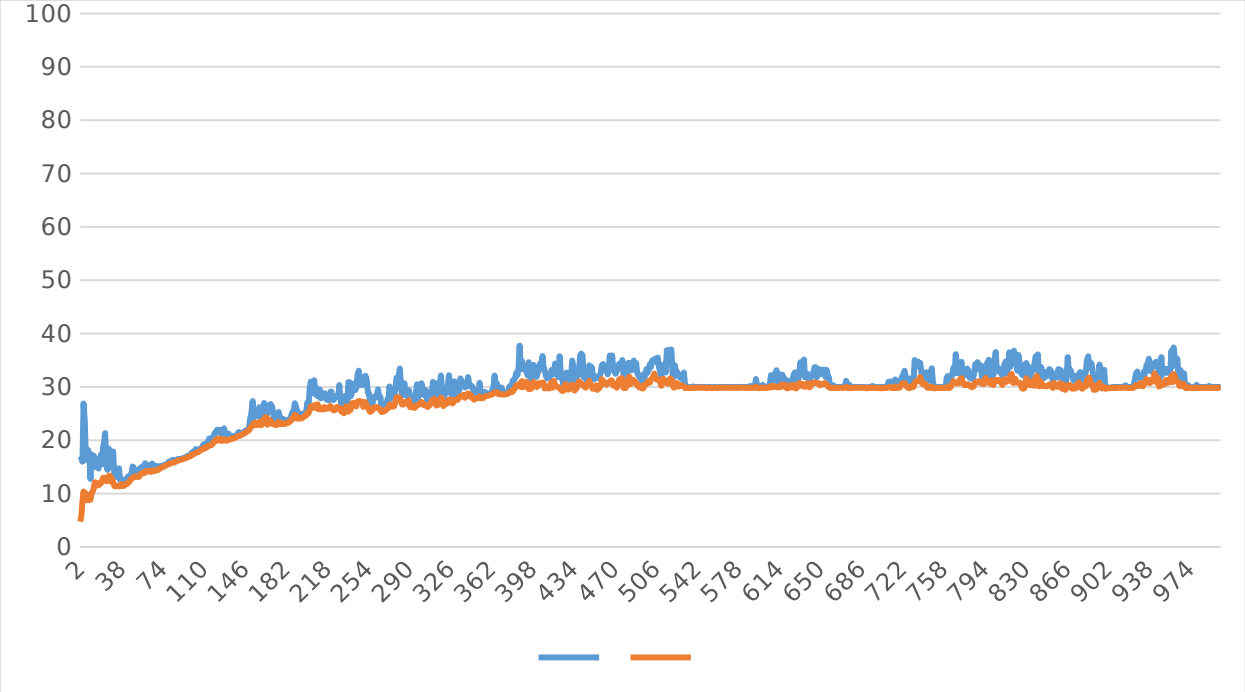
3- Roma Rudsky

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab3_siit_
{
    class Dot
    {
        public List<double> chromo;
        public double fit;
        public Dot(List<double> chromo)
        {
            this.chromo = chromo;
            fit = getFit(chromo);
        }
        private double getFit(List<double> chromo)
        {
            double fit = chromo.Count;
            for (int i=0;i<chromo.Count;i++)
            {
                double temp = chromo[i]* chromo[i] - Math.Cos(2 * Math.PI *
chromo[i]);
                fit += temp;
            }
            return fit;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab3_siit_
{
    class Population
    {
        public List<Dot> dots;
        public double theBestFit;
        public double avarageFit;
        public Population(List<Dot> dogs)
        {
            this.dots = dogs;
            theBestFit = getTheBestFit(dogs);
            avarageFit = getAvarageFit(dogs);
        }
        public double getTheBestFit(List<Dot> dogs)
        {
            double bestFit = 9999;
            for (int i = 0; i < dogs.Count; i++)
                if (bestFit > dogs[i].fit)
                    bestFit = dogs[i].fit;
            return bestFit;
        }
        public double getAvarageFit(List<Dot> dogs)
        {
            double avarageFit = 0;
            for (int i = 0; i < dogs.Count; i++)
                avarageFit += dogs[i].fit;
            avarageFit /= dogs.Count;
            return avarageFit;
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
```

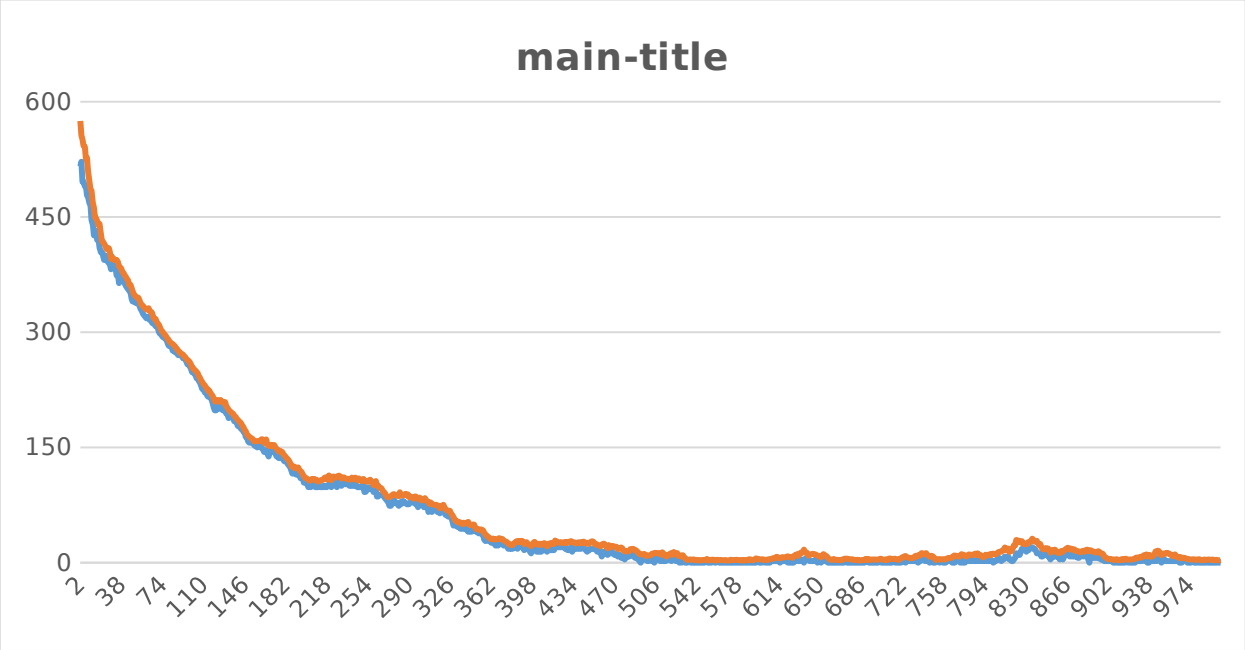
```
using System.Threading.Tasks;

namespace lab3_siit_
{
    class GA
    {
        Random random;
        List<Population> histor;
        public GA()
        {
            random = new Random();
            List<Dot> dots = new List<Dot>();
            for (int i = 0; i < 20; i++)
            {
                List<double> chromo = new List<double>();
                for (int j = 0; j < 20; j++)
                {
                    chromo.Add(random.NextDouble() % 2 -1);
                }
                dots.Add(new Dot(chromo));
            }
            //Sorting by fit
            for (int i = 0; i < dots.Count; i++)
            {
                for (int j = dots.Count - 1; j > i; j--)
                {
                    if (dots[j].fit > dots[j - 1].fit)
                    {
                        Dot tempDot = dots[j];
                        dots[j] = dots[j - 1];
                        dots[j - 1] = tempDot;
                    }
                }
            }
            Population startPopulation = new Population(dots);
            histor = new List<Population>();
            histor.Add(startPopulation);
            int k = 0;
            while (!isReady(histor))
            {
                Thread.Sleep(10);
                histor.Add(getNextPupulation(histor[k]));
                k++;
            }
        }
        private Population getNextPupulation(Population parent)
        {
            List<Dot> childrenPopulationDots = new List<Dot>();

            for (int i = 0; i < 10; i++)
            {
                List<Dot> childrenDots = getChildren(
                    parent.dots[random.Next() % 10 + 10],
                    parent.dots[random.Next() % 10 + 10]
                    );
                childrenPopulationDots.AddRange(childrenDots);
            }
            //Sorting by fit
            for (int i = 0; i < childrenPopulationDots.Count; i++)
            {
                for (int j = childrenPopulationDots.Count - 1; j > i; j--)
                {
                    if (childrenPopulationDots[j].fit > childrenPopulationDots[j -
1].fit)
                    {
                        Dot tempDot = childrenPopulationDots[j];
                        childrenPopulationDots[j] = childrenPopulationDots[j - 1];
                        childrenPopulationDots[j - 1] = tempDot;
                    }
                }
            }
            Population childrenPopulation = new Population(childrenPopulationDots);
            return childrenPopulation;
        }

        Boolean isReady(List<Population> histor)
        {
            if (histor.Count < 100)
                return false;
```

```
            else
            {
                for (int i = histor.Count - 100; i < histor.Count; i++)
                {
                    if (histor[histor.Count - 100].avarageFit != histor[i].avarageFit)
                        return false;
                }
                return true;
            }
        }
        private List<Dot> getChildren(Dot father, Dot mother)
        {
            List<Dot> childrenDots = new List<Dot>();
            int pointCross = random.Next()%20;
            List<double> firstChromo = new List<double>();
            List<double> secondChromo = new List<double>();
            for (int j = 0; j < 20; j++)
            {
                if(j<pointCross)
                {
                    firstChromo.Add(father.chromo[j]);
                    secondChromo.Add(mother.chromo[j]);
                }
                else
                {
                    firstChromo.Add(mother.chromo[j]);
                    secondChromo.Add(father.chromo[j]);
                }
            }

            childrenDots.Add(new Dot(firstChromo));
            childrenDots.Add(new Dot(secondChromo));

            //Mutation
            for (int j = 0; j < childrenDots.Count; j++)
            {
                int prob = random.Next(0, 20);
                if (prob == 7)
                {
                    int number = random.Next(20);
                    childrenDots[j].chromo[number] = random.NextDouble() % 2 - 1;
                }
            }
            return childrenDots;
        }
    }
}
```

Share fitness vs generation



fitness vs generation



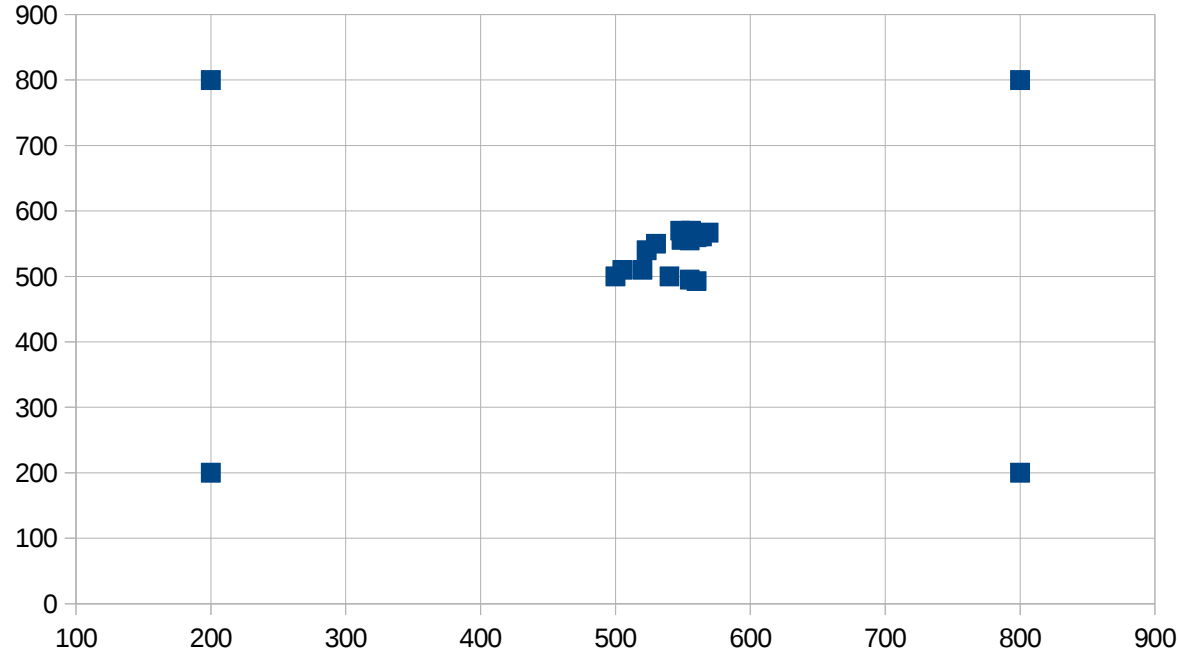
gen	points	fitness	share fitness
1	[479, 539]	540	15.6
2	[511, 433]	522	14.21052632
3	[469, 523]	546	6.6
4	[530, 464]	534	6
5	[490, 462]	552	5.777777778
6	[539, 487]	548	5
7	[492, 464]	556	4.561403509
8	[526, 474]	548	4.237288136
9	[523, 473]	550	4.210526316
10	[515, 467]	552	4.090909091
11	[533, 503]	564	4.047619048
12	[514, 520]	566	3.928571429
13	[524, 480]	556	3.913043478
14	[533, 497]	564	3.695652174
15	[522, 512]	566	3.636363636
16	[478, 502]	576	3.396226415
17	[505, 469]	564	3.265306122
18	[492, 476]	568	3.25
19	[504, 522]	574	1.86440678
20	[509, 487]	578	1.86440678

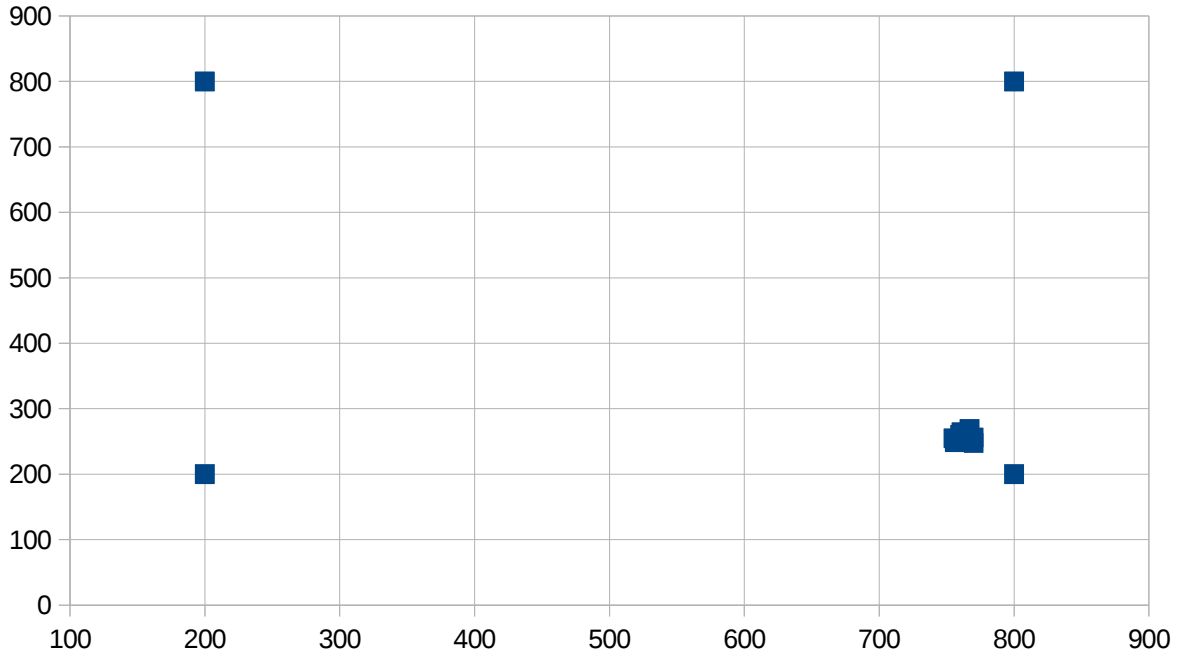
mid generation:

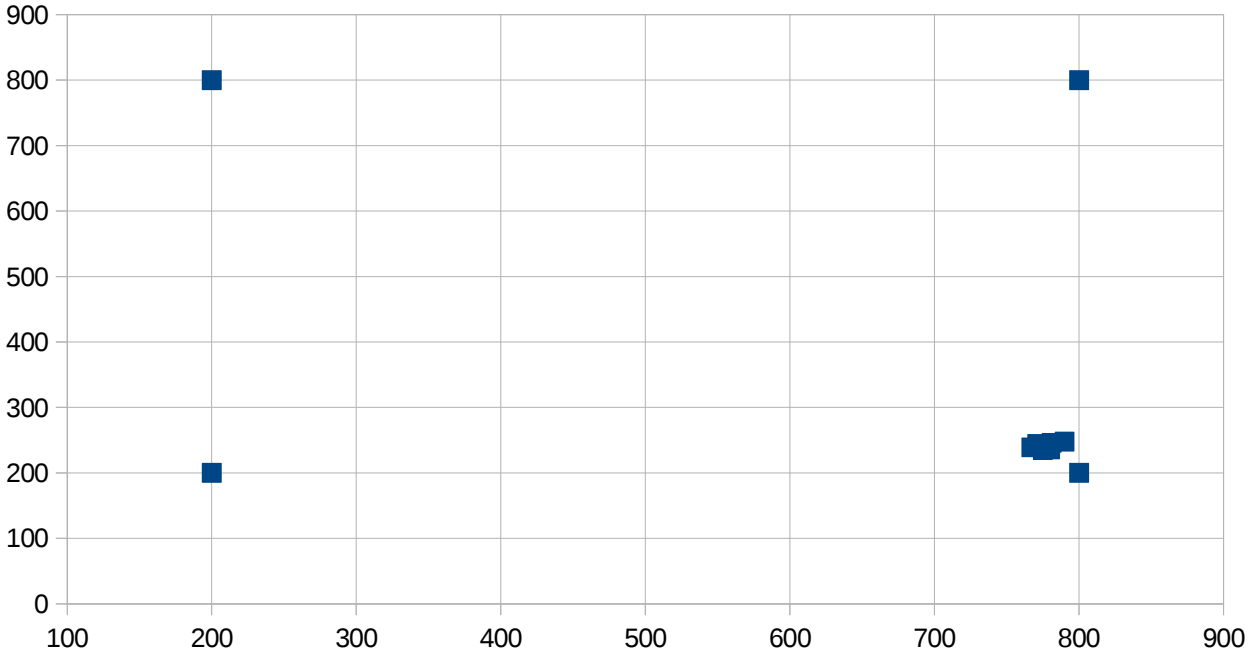
gen	points	fitness	share fitness
1	[734, 212]	78	27.47368421
2	[730, 216]	86	27.05263158
3	[742, 222]	80	26.53061224
4	[739, 225]	86	26.2244898
5	[738, 224]	86	26.2244898
6	[740, 218]	78	26.1
7	[737, 227]	90	26.02040816
8	[735, 227]	92	25.91836735
9	[736, 218]	76	25.9
10	[737, 219]	82	25.9
11	[738, 220]	82	25.9
12	[736, 218]	82	25.9
13	[738, 220]	82	25.9
14	[738, 220]	82	25.9
15	[737, 223]	86	25.7
16	[735, 221]	86	25.7
17	[732, 220]	88	25.6
18	[732, 222]	90	25.5
19	[732, 222]	90	25.5
20	[734, 224]	90	25.5

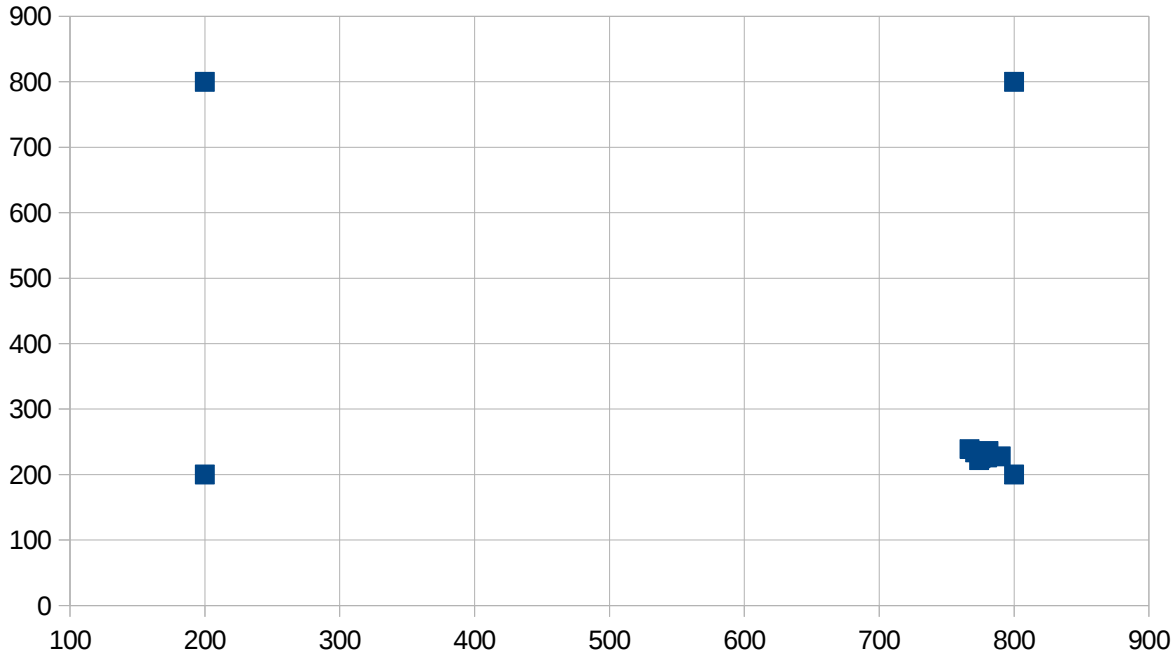
last generation:

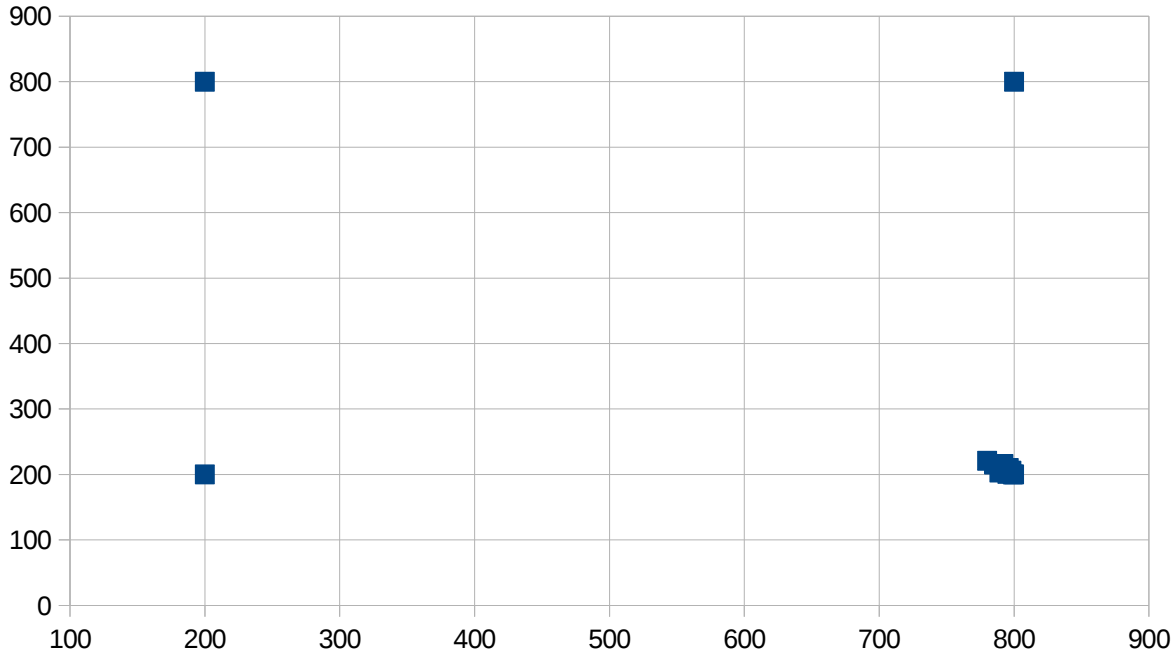
gen	points	fitness	share fitness
1	[800, 200]	0	30
2	[800, 200]	0	30
3	[800, 200]	0	30
4	[798, 200]	2	29.9
5	[798, 200]	2	29.9
6	[801, 201]	2	29.9
7	[800, 202]	2	29.9
8	[799, 199]	2	29.9
9	[802, 200]	2	29.9
10	[797, 201]	4	29.8
11	[800, 196]	4	29.8
12	[801, 197]	4	29.8
13	[798, 198]	4	29.8
14	[803, 199]	4	29.8
15	[800, 196]	4	29.8
16	[799, 203]	4	29.8
17	[797, 199]	4	29.8
18	[796, 198]	6	29.7
19	[796, 196]	8	29.6
20	[797, 207]	6	29.5




900
800
700
600
500
400
300
200
100
0
100
200
300
400
500
600
700
800
900


900
800
700
600
500
400
300
200
100
0
100
200
300
400
500
600
700
800
900


900
800
700
600
500
400
300
200
100
0
100
200
300
400
500
600
700
800
900


900
800
700
600
500
400
300
200
100
0
100
200
300
400
500
600
700
800
900