Condition. We have 100 chromosomes which include 1000 gens. There are gens of two kinds: 0 or 1, where 1 - is better than 0. Genes are (roulette-wheel selection) determined. Then we choose the best chromosomes use sorting them by number of 1. 50 best chromosomes we cross our between each other until there appear a chromosome with all 1.

Cod Programs:

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <time.h>
#include <vector>
#include <algorithm>
#include <math.h>
using namespace std;


typedef vector<int> Chromosome;
typedef vector<Chromosome> Population;

Population generation[100][1000];
int good[50][1000];
int childes[50][1000];
int childCount = 0;
int checkCount = 0;
int prev = 0;

int _tmain(int argc, _TCHAR* argv[])
{
	srand(time(NULL));
	createGenerate();
	while (true) {
		int max = rowCount(generation[0]);
		if (prev==max)
			checkCount++;
		else
			checkCount = 0;
		prev = max;
		cout <<max + " ";
		cout << average();
		if (checkCount > 25) {
			return;
		}
		setGood();

		childCount = 0;
		for (int i = 0; i < 25; i++) {
			createChildes(good[rand() % 50], good[rand() % 50]);
		}
		createNewGeneration();
	}

	return 0;
```

```
}


bool operator==(const Chromosome& chr1, const Chromosome& chr2) {
	for (auto i = 0; i < chr1.size(); i++) {
		if (chr1[i] != chr2[i]) return false;
	}
	return true;
}

	 void createGenerate()
	 {
		int tmp = rand() % 1000;
		double val1 = (double)tmp / 1000;
		double val2 = (double)tmp / 1000;
		while (val1 == val2)
		{
			tmp = rand() % 1000;
			val2 = (double)tmp / 1000;
		}

		double summProb = 0.0;
		int parent1, parent2;
		for (int i = 0; i < Prob.size(); i++)
		{
			summProb += Prob[i];
			if (summProb >= val1)
			{
				parent1 = i;
				break;
			} }
		summProb = 0.0;
		for (int i = 0; i < Prob.size(); i++)
		{
			summProb += Prob[i];
			if (summProb >= val2)
			{
				parent2 = i;
				break;
			}}

		Chromosome child1 = population[parent1];
		Chromosome child2 = population[parent2];
		int cut = rand() % 1000;
		for (int i = cut; i < 1000; i++)
		{
			child1[i] = population[parent2][i];
			child2[i] = population[parent1][i];
		}
		result.push_back(child1);
		result.push_back(child2);

	 }

		int rowCount(int generation[]) {
			int count = 0;
			for (int i = 0; i<1000; i++) {
				count += generation[i];
			}
			return count;
		}
```

```
        void setGood() {
               for (int i = 0; i < 50; i++) {
                      for (int j = 0; j < 50; j++) {
                             good[i][j] = generation[i][j];
                      } }
        }

    void createChildes(int parent1[], int parent2[]) {

           int del = rand() % 1000;
          int child1  [1000];
          int child2 [1000];
          for (int i = 0; i<1000; i++) {
                 if (i <= del) childes[childCount][i] = parent1[i];
                 else childes[childCount][i] = parent2[i];
          } childCount++;
          for (int i = 0; i<1000; i++) {
                 if (i <= del) childes[childCount][i] = parent2[i];
                 else childes[childCount][i] = parent1[i];
          }childCount++;

    }

          void createNewGeneration() {
                 for (int i = 0; i<100; i++) {
                        for (int j = 0; j < 1000; j++) {
                               if (i < 50) generation[i][j] = good[i][j];
                               else generation[i][j] = childes[i - 50][j];
                        }
                 }
          }

           int average() {
                 int s = 0;
                 for (int i = 0; i<100; i++) {
                        s += rowCount(generation[i]);}
                 int average = s / 100;
                 return average;
          }
```

After compliting program and analyze result, we can see, that alghoritm create chromosomes more 60% good gens. Average values almost equals with max value, and have less steps by iterations.

Desplay the best-fitness generation and average-fitness generation.

x - generation
y – fitness

As we have the best parents (with each iteratoin the better) we get better children and from after 200 iterations the maximum fitness is not increased.

iteration	Average	Best
1	497	533
2	499	541
3	500	532
4	501	530
5	504	535
6	505	540
7	503	533
8	504	529
9	505	541
10	506	551
11	508	546
12	510	550
13	511	550
14	513	544
15	515	552
16	515	551
17	515	545
18	515	548
19	518	549
20	519	550
21	519	561
22	518	544
23	520	547
24	519	544
25	520	545
26	522	548
27	523	551
28	523	550
29	523	552
30	525	552
31	526	557
32	526	552
33	529	559
34	531	559
35	533	564
36	535	566
37	536	567
38	540	574
39	541	574
40	544	577
41	547	577
42	548	577
43	548	572
44	549	577
45	547	579
46	549	579
47	549	579
48	550	575
49	549	575
50	550	573
51	549	573
52	551	576
53	552	578
54	552	576

55	554	576
56	555	576
57	554	587
58	555	586
59	556	581
60	555	578
61	555	581
62	557	579
63	557	579
64	558	577
65	559	576
66	560	576
67	563	582
68	565	583
69	565	583
70	566	583
71	565	581
72	565	582
73	565	583
74	566	583
75	566	581
76	566	582
77	567	583
78	567	586
79	569	591
80	570	590
81	572	595
82	571	596
83	572	596
84	572	600
85	572	600
86	574	600
87	575	600
88	576	594
89	578	595
90	578	598
91	578	598
92	579	599
93	580	603
94	580	606
95	580	600
96	581	603
97	580	597
98	581	596
99	582	600
100	582	601
101	582	601
102	582	601
103	583	601
104	583	598
105	582	598
106	582	595
107	583	597
108	583	596
109	583	596
110	583	598
111	584	602
112	583	602
113	584	597
114	585	597
115	585	600
116	585	600
117	585	597
118	586	600
119	586	600
120	586	601
121	586	598
122	586	597
123	586	598
124	587	599
125	587	600
126	588	600
127	587	600
128	587	600
129	588	603
130	589	602
131	590	603
132	591	603
133	591	605
134	590	607
135	591	604
136	591	605
137	590	606
138	590	609
139	590	609
140	590	609
141	590	609
142	591	607
143	590	607
144	591	608
145	592	608
146	592	608
147	592	608
148	594	608
149	593	608
150	594	609
151	595	610
152	596	609
153	596	609
154	597	610
155	597	610
156	598	610
157	598	610
158	599	610
159	599	612
160	600	612
161	600	612
162	601	610
163	601	611
164	602	611
165	602	611
166	602	611
167	603	611
168	603	612
169	603	612
170	603	612
171	603	612
172	604	612
173	605	613
174	606	613
175	606	614
176	606	613
177	607	613
178	606	614
179	606	614
180	606	614
181	606	614
182	607	614
183	607	614

184	607	615
185	607	616
186	607	616
187	607	616
188	608	616
189	608	616
190	608	616
191	608	616
192	608	615
193	608	616
194	609	616
195	609	616
196	609	616

197	610	616
198	610	616
199	610	616
200	610	617
201	610	617
202	610	616
203	610	616
204	610	616
205	610	617
206	611	617
207	611	617
208	611	616
209	611	616

210	611	617
211	611	617
212	611	617
213	611	617
214	611	617
215	611	617
216	611	617
217	611	617
218	611	617
219	611	617

Tables result on (0,75,150 and last iteration): fitness generation and value for each fitness value.

0			75			150			219	
465	0,0093		567	0,0097		605	0,0099		606	0
469	0,0094		569	0,0098		605	0,0099		607	0
470	0,0094		571	0,0098		605	0,0099		607	0
470	0,0094		572	0,0098		606	0,0099		607	0
471	0,0094		573	0,0098		606	0,0099		607	0
472	0,0095		574	0,0098		606	0,0099		608	0
473	0,0095		574	0,0098		606	0,0099		608	0
473	0,0095		574	0,0098		606	0,0099		608	0
476	0,0095		575	0,0099		606	0,0099		608	0
477	0,0096		575	0,0099		606	0,0099		608	0
480	0,0096		575	0,0099		606	0,0099		608	0
480	0,0096		576	0,0099		607	0,0099		609	0
480	0,0096		576	0,0099		607	0,0099		609	0
481	0,0096		576	0,0099		607	0,0099		609	0
482	0,0097		577	0,0099		607	0,0099		609	0
483	0,0097		577	0,0099		608	0,01		609	0
483	0,0097		577	0,0099		608	0,01		609	0
484	0,0097		577	0,0099		608	0,01		609	0
485	0,0097		577	0,0099		608	0,01		609	0
486	0,0097		577	0,0099		608	0,01		609	0
486	0,0097		578	0,0099		609	0,01		609	0
487	0,0098		578	0,0099		609	0,01		609	0
487	0,0098		578	0,0099		609	0,01		609	0
488	0,0098		578	0,0099		609	0,01		609	0
488	0,0098		578	0,0099		609	0,01		609	0
488	0,0098		578	0,0099		609	0,01		609	0
488	0,0098		578	0,0099		609	0,01		610	0

489	0,0098		578	0,0099		609	0,01		610	0
490	0,0098		578	0,0099		609	0,01		610	0
491	0,0098		578	0,0099		609	0,01		610	0
491	0,0098		579	0,0099		609	0,01		610	0
491	0,0098		579	0,0099		609	0,01		610	0
491	0,0098		579	0,0099		609	0,01		610	0
491	0,0098		579	0,0099		609	0,01		610	0
492	0,0099		580	0,01		609	0,01		610	0
492	0,0099		580	0,01		609	0,01		610	0
493	0,0099		580	0,01		610	0,01		610	0
493	0,0099		581	0,01		610	0,01		610	0
494	0,0099		581	0,01		610	0,01		610	0
495	0,0099		581	0,01		610	0,01		610	0
496	0,0099		581	0,01		610	0,01		610	0
496	0,0099		581	0,01		610	0,01		610	0
496	0,0099		582	0,01		610	0,01		610	0
496	0,0099		582	0,01		610	0,01		610	0
497	0,01		582	0,01		610	0,01		610	0
497	0,01		582	0,01		610	0,01		610	0
497	0,01		583	0,01		610	0,01		610	0
498	0,01		583	0,01		610	0,01		611	0
498	0,01		583	0,01		610	0,01		611	0
500	0,01		583	0,01		610	0,01		611	0
500	0,01		583	0,01		611	0,01		611	0
501	0,01		583	0,01		611	0,01		611	0
501	0,01		583	0,01		611	0,01		611	0
501	0,01		583	0,01		611	0,01		611	0
501	0,01		583	0,01		611	0,01		611	0
502	0,0101		584	0,01		611	0,01		611	0
502	0,0101		584	0,01		611	0,01		611	0
502	0,0101		584	0,01		611	0,01		611	0
502	0,0101		584	0,01		611	0,01		611	0
502	0,0101		584	0,01		611	0,01		611	0
503	0,0101		585	0,01		611	0,01		612	0
503	0,0101		585	0,01		612	0,01		612	0
504	0,0101		585	0,01		612	0,01		612	0
504	0,0101		585	0,01		612	0,01		612	0
504	0,0101		585	0,01		612	0,01		612	0
505	0,0101		585	0,01		612	0,01		612	0
505	0,0101		586	0,0101		612	0,01		612	0
506	0,0101		586	0,0101		612	0,01		613	0
508	0,0102		586	0,0101		612	0,01		613	0
508	0,0102		587	0,0101		612	0,01		613	0

508	0,0102		587	0,0101		613	0,01		613	0
509	0,0102		587	0,0101		613	0,01		613	0
509	0,0102		587	0,0101		613	0,01		613	0
509	0,0102		588	0,0101		613	0,01		613	0
510	0,0102		588	0,0101		613	0,01		613	0
510	0,0102		588	0,0101		613	0,01		613	0
511	0,0102		589	0,0101		613	0,01		613	0
513	0,0103		589	0,0101		613	0,01		613	0
513	0,0103		589	0,0101		613	0,01		613	0
514	0,0103		589	0,0101		614	0,0101		614	0
515	0,0103		589	0,0101		614	0,0101		614	0
516	0,0103		589	0,0101		614	0,0101		614	0
516	0,0103		589	0,0101		614	0,0101		614	0
516	0,0103		589	0,0101		614	0,0101		614	0
517	0,0104		589	0,0101		614	0,0101		614	0
517	0,0104		589	0,0101		614	0,0101		614	0
517	0,0104		589	0,0101		614	0,0101		614	0
517	0,0104		589	0,0101		614	0,0101		614	0
519	0,0104		590	0,0101		614	0,0101		614	0
519	0,0104		590	0,0101		614	0,0101		614	0
519	0,0104		590	0,0101		615	0,0101		615	0
520	0,0104		591	0,0101		615	0,0101		615	0
522	0,0105		591	0,0101		615	0,0101		615	0
523	0,0105		592	0,0102		615	0,0101		616	0
526	0,0105		593	0,0102		615	0,0101		616	0
529	0,0106		593	0,0102		616	0,0101		616	0
530	0,0106		594	0,0102		616	0,0101		617	0
531	0,0106		595	0,0102		616	0,0101		617	0
536	0,0107		596	0,0102		616	0,0101		617	0
541	0,0108		601	0,0103		616	0,0101		618	0