1.

$$\mathrm{f(s)} = \frac{\left|\sum_1^{120} w_{s_i} x_i\right| + 5 * \left|\sum_1^{120} w_{s_i} y_i\right|}{\sum_1^{120} w_{s_i}}$$

2.

$$t_i(\mathrm{s,a,b}) = \begin{cases} s_a, & \mathrm{i = b} \\ s_b, & \mathrm{i = a} \\ s_i, & \mathrm{otherwise} \end{cases}$$

3. Move the container at position a to an empty position b.
4. b=a+60 or a=b+60 Those two container are at the same position
   a=b Swap the container with itself
   $w_{s_a} = w_{s_b}$ Swap two containers which have the same weight
5. N(s)=t(s,a,b) where $|a-b| \neq 60, a \neq b \; and \; w_{s_a} \neq w_{s_b}, a, b \in \{1,2,\dots,120\}$
6. $\Delta dx = w_{s_a} x_b + w_{s_b} x_a - w_{s_a} x_a - w_{s_b} x_b$
   $\Delta dy = w_{s_a} y_b + w_{s_b} y_a - w_{s_a} y_a - w_{s_b} y_b$
   $f\big(t(s,a,b)\big) = |dx(s) + \Delta dx| + 5 * |dy(s) + \Delta dy|$
   dx(s) is the center of gravity in the x direction for solution s
   dy(s) is the center of gravity in the y direction for solution s
7. For this problem, a 2D array can be used to flag whether two containers are baned. Thus, each time we don't need to iterate the ban list and we can directly access the 2D array. Only when one iteration finished, two container swaped and the array need to be freshed, we use the ban list to update the array with a very short time.
8. Banning random pair of containers can be a suitable approach because we want to use tabu search to leave local minimum and banning random pair can achieve this goal

Ship Loading Positions

1	7	13	19	25	31	37	43	49	55
2	8	14	20	26	32	38	44	50	56
3	9	15	21	27	33	39	45	51	57
4	10	16	22	28	34	40	46	52	58
5	11	17	23	29	35	41	47	53	59
6	12	18	24	30	36	42	48	54	60

61	67	73	79	85	91	97	103	109	115
62	68	74	80	86	92	98	104	110	116
63	69	75	81	87	93	99	105	111	117
64	70	76	82	88	94	100	106	112	118
65	71	77	83	89	95	101	107	113	119
66	72	78	84	90	96	102	108	114	120

Index of Container in each Loading Position (0=empty)

0	0	0	73	12	28	2	43	18	40
0	0	34	51	79	31	71	23	78	68
0	0	56	1	39	13	14	8	74	44
0	0	94	9	93	5	46	60	95	6
0	0	30	65	55	59	54	36	82	62
0	0	0	7	37	22	41	16	53	47

49	0	67	48	42	100	89	32	88	27
84	3	91	10	61	80	97	81	21	76
0	4	86	35	11	87	75	64	45	17
0	0	52	85	99	92	29	26	20	66
0	63	33	72	98	77	19	90	96	57
83	0	50	58	70	38	69	15	24	25

Total Weight in each Loading Position

37.50877195	0	268.1450586	787.2285767	1049.064357	1169.6485	1430.220829	1109.334164	1297.00026	904.9233463
32.97310875	207.764796	250.1432376	641.4672993	1565.148268	1584.907166	1301.055782	1528.762466	1146.56652	1225.673877
0	47.55080316	404.9923865	1136.664353	1698.88019	1840.177415	1083.312996	1489.119984	1091.653063	734.9252626
0	0	288.9542718	1268.111731	1911.287833	1483.532603	1481.613102	1264.186363	690.6172741	939.236752
0	127.9442425	768.947682	1366.687538	1458.988985	1195.091794	1284.367131	915.4051612	779.1252937	1271.84023
40.22807832	0	926.5681888	1840.278255	1337.295208	1286.234912	995.1705093	774.8262708	555.4681056	526.5731984
110.709959	383.2598417	2907.750825	7040.437752	9020.664842	8559.59239	7575.740349	7081.634409	5560.430516	5603.172666

ID Number 12345678

---- Solution Quality ----

ProbA

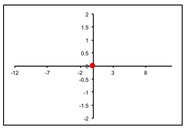
dX=	-9.7E-06
dY=	2.38E-06
Obj Fn=	2.17E-05

- Number of Containers -

Containers	100

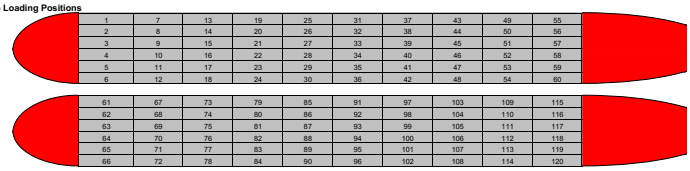
- Number of Positions -

Positions	120



-------- Solution --------		--- Container Data ----		-------------- Workings ---------------			-------- Workings ---------	
Ship Loading Positions	Index of Container at Posn	Container Index	Container Weight	Objective Calculation Workings Position	Position	Position	Duplicate Check	All Containers
		0	0				OK	OK
1	0	1	280.7546	0	-33	7.5	TRUE	TRUE
2	0	2	669.389	0	-33	4.5	TRUE	TRUE
3	0	3	207.7648	0	-33	1.5	TRUE	TRUE
4	0	4	47.5508	0	-33	-1.5	TRUE	TRUE
5	0	5	830.8559	0	-33	-4.5	TRUE	TRUE
6	0	6	275.2896	0	-33	-7.5	TRUE	TRUE
7	0	7	942.0694	0	-27	7.5	TRUE	TRUE
8	0	8	514.9914	0	-27	4.5	TRUE	TRUE
9	0	9	346.4315	0	-27	1.5	TRUE	TRUE
10	0	10	387.0416	0	-27	-1.5	TRUE	TRUE
11	0	11	888.9294	0	-27	-4.5	TRUE	TRUE
12	0	12	589.9275	0	-27	-7.5	TRUE	TRUE
13	0	13	967.7608	0	-21	7.5	TRUE	TRUE
14	34	14	585.6589	101.3493	-21	4.5	TRUE	TRUE
15	56	15	405.8013	189.8707	-21	1.5	TRUE	TRUE
16	94	16	369.025	229.7909	-21	-1.5	TRUE	TRUE
17	30	17	389.7931	206.4649	-21	-4.5	TRUE	TRUE
18	0	18	956.3363	0	-21	-7.5	TRUE	TRUE
19	73	19	745.112	202.2523	-15	7.5	TRUE	TRUE
20	51	20	391.4605	254.4257	-15	4.5	TRUE	TRUE
21	1	21	802.0752	280.7546	-15	1.5	TRUE	TRUE
22	9	22	548.2732	346.4315	-15	-1.5	TRUE	TRUE
23	65	23	615.7557	835.9396	-15	-4.5	TRUE	TRUE
24	7	24	319.5606	942.0694	-15	-7.5	TRUE	TRUE
25	12	25	262.8152	589.9275	-9	7.5	TRUE	TRUE
26	79	26	864.8866	589.4647	-9	4.5	TRUE	TRUE
27	39	27	617.4171	809.9508	-9	1.5	TRUE	TRUE
28	93	28	925.0999	988.1317	-9	-1.5	TRUE	TRUE
29	55	29	903.1375	931.2013	-9	-4.5	TRUE	TRUE
30	37	30	206.4649	733.6511	-9	-7.5	TRUE	TRUE
31	28	31	734.3443	925.0999	-3	7.5	TRUE	TRUE
32	31	32	354.7015	734.3443	-3	4.5	TRUE	TRUE
33	13	33	562.4828	967.7608	-3	1.5	TRUE	TRUE
34	5	34	101.3493	830.8559	-3	-1.5	TRUE	TRUE
35	59	35	855.9098	734.1092	-3	-4.5	TRUE	TRUE
36	22	36	398.4976	548.2732	-3	-7.5	TRUE	TRUE
37	2	37	733.6511	669.389	3	7.5	TRUE	TRUE
38	71	38	737.9617	686.8336	3	4.5	TRUE	TRUE
39	14	39	809.9508	585.6589	3	1.5	TRUE	TRUE
40	46	40	287.5062	578.4756	3	-1.5	TRUE	TRUE
41	54	41	394.7741	539.2551	3	-4.5	TRUE	TRUE
42	41	42	459.1369	394.7741	3	-7.5	TRUE	TRUE
43	43	43	754.6327	754.6327	9	7.5	TRUE	TRUE
44	23	44	345.1322	615.7557	9	4.5	TRUE	TRUE
45	8	45	419.8283	514.9914	9	1.5	TRUE	TRUE
46	60	46	578.4756	399.2997	9	-1.5	TRUE	TRUE
47	36	47	263.758	398.4976	9	-4.5	TRUE	TRUE
48	16	48	584.9763	369.025	9	-7.5	TRUE	TRUE
49	18	49	37.50877	956.3363	15	7.5	TRUE	TRUE
50	78	50	926.5682	344.4914	15	4.5	TRUE	TRUE
51	74	51	254.4257	671.8248	15	1.5	TRUE	TRUE
52	95	52	59.16333	299.1568	15	-1.5	TRUE	TRUE
53	82	53	235.9075	291.2842	15	-4.5	TRUE	TRUE
54	53	54	539.2551	235.9075	15	-7.5	TRUE	TRUE
55	40	55	931.2013	287.5062	21	7.5	TRUE	TRUE
56	68	56	189.8707	704.7202	21	4.5	TRUE	TRUE
57	44	57	520.1078	345.1322	21	1.5	TRUE	TRUE
58	6	58	898.2088	275.2896	21	-1.5	TRUE	TRUE
59	62	59	734.1092	751.7324	21	-4.5	TRUE	TRUE
60	47	60	399.2997	263.758	21	-7.5	TRUE	TRUE
61	49	61	975.6835	37.50877	-33	7.5	TRUE	TRUE
62	84	62	751.7324	32.97311	-33	4.5	TRUE	TRUE
63	0	63	127.9442	0	-33	1.5	TRUE	TRUE
64	0	64	974.1286	0	-33	-1.5	TRUE	TRUE
65	0	65	835.9396	0	-33	-4.5	TRUE	TRUE
66	83	66	663.9471	40.22808	-33	-7.5	TRUE	TRUE
67	0	67	268.1451	0	-27	7.5	TRUE	TRUE
68	3	68	704.7202	207.7648	-27	4.5	TRUE	TRUE
69	4	69	600.3964	47.5508	-27	1.5	TRUE	TRUE
70	0	70	603.6441	0	-27	-1.5	TRUE	TRUE
71	63	71	686.8336	127.9442	-27	-4.5	TRUE	TRUE
72	0	72	530.7479	0	-27	-7.5	TRUE	TRUE
73	67	73	202.2523	268.1451	-21	7.5	TRUE	TRUE
74	91	74	671.8248	148.794	-21	4.5	TRUE	TRUE
75	86	75	497.6541	215.1217	-21	1.5	TRUE	TRUE
76	52	76	520.9537	59.16333	-21	-1.5	TRUE	TRUE
77	33	77	460.9826	562.4828	-21	-4.5	TRUE	TRUE
78	50	78	344.4914	926.5682	-21	-7.5	TRUE	TRUE
79	48	79	589.4647	584.9763	-15	7.5	TRUE	TRUE
80	10	80	850.5629	387.0416	-15	4.5	TRUE	TRUE
81	35	81	913.0067	855.9098	-15	1.5	TRUE	TRUE
82	85	82	291.2842	921.6802	-15	-1.5	TRUE	TRUE
83	72	83	40.22808	530.7479	-15	-4.5	TRUE	TRUE
84	58	84	32.97311	898.2088	-15	-7.5	TRUE	TRUE
85	42	85	921.6802	459.1369	-9	7.5	TRUE	TRUE
86	61	86	215.1217	975.6835	-9	4.5	TRUE	TRUE
87	11	87	872.4166	888.9294	-9	1.5	TRUE	TRUE
88	99	88	340.664	923.1561	-9	-1.5	TRUE	TRUE
89	98	89	760.8318	527.7877	-9	-4.5	TRUE	TRUE
90	70	90	516.9076	603.6441	-9	-7.5	TRUE	TRUE
91	100	91	148.794	244.5486	-3	7.5	TRUE	TRUE
92	80	92	652.6767	850.5629	-3	4.5	TRUE	TRUE
93	87	93	988.1317	872.4166	-3	1.5	TRUE	TRUE
94	92	94	229.7909	652.6767	-3	-1.5	TRUE	TRUE
95	77	95	299.1568	460.9826	-3	-4.5	TRUE	TRUE
96	38	96	487.8411	737.9617	-3	-7.5	TRUE	TRUE
97	89	97	614.2222	760.8318	3	7.5	TRUE	TRUE
98	97	98	527.7877	614.2222	3	4.5	TRUE	TRUE
99	75	99	923.1561	497.6541	3	1.5	TRUE	TRUE
100	29	100	244.5486	903.1375	3	-1.5	TRUE	TRUE
101	19	101		745.112	3	-4.5	TRUE	
102	69	102		600.3964	3	-7.5	TRUE	
103	32	103		354.7015	9	7.5	TRUE	
104	81	104		913.0067	9	4.5	TRUE	
105	64	105		974.1286	9	1.5	TRUE	
106	26	106		864.8866	9	-1.5	TRUE	
107	90	107		516.9076	9	-4.5	TRUE	
108	15	108		405.8013	9	-7.5	TRUE	
109	88	109		340.664	15	7.5	TRUE	
110	21	110		802.0752	15	4.5	TRUE	
111	45	111		419.8283	15	1.5	TRUE	
112	20	112		391.4605	15	-1.5	TRUE	
113	96	113		487.8411	15	-4.5	TRUE	
114	24	114		319.5606	15	-7.5	TRUE	
115	27	115		617.4171	21	7.5	TRUE	
116	76	116		520.9537	21	4.5	TRUE	
117	17	117		389.7931	21	1.5	TRUE	
118	66	118		663.9471	21	-1.5	TRUE	
119	57	119		520.1078	21	-4.5	TRUE	
120	25	120		262.8152	21	-7.5	TRUE	

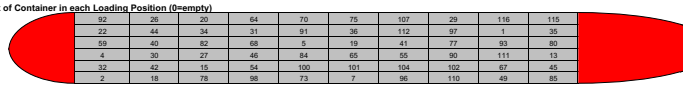
Ship Loading Positions



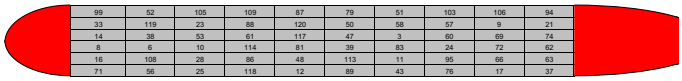
1	7	13	19	25	31	37	43	49	55
2	8	14	20	26	32	38	44	50	56
3	9	15	21	27	33	39	45	51	57
4	10	16	22	28	34	40	46	52	58
5	11	17	23	29	35	41	47	53	59
6	12	18	24	30	36	42	48	54	60

61	67	73	79	85	91	97	103	109	115
62	68	74	80	86	92	98	104	110	116
63	69	75	81	87	93	99	105	111	117
64	70	76	82	88	94	100	106	112	118
65	71	77	83	89	95	101	107	113	119
66	72	78	84	90	96	102	108	114	120

Index of Container in each Loading Position (0=empty)

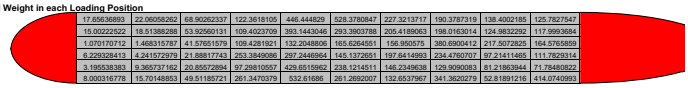


92	26	20	64	70	75	107	29	116	115
22	44	34	31	91	36	112	97	1	35
59	40	82	68	5	19	41	77	93	80
4	30	27	46	84	65	55	90	111	13
32	42	15	54	100	101	104	102	67	45
2	18	78	98	73	7	96	110	49	85



99	52	105	109	87	79	51	103	106	94
33	119	23	88	120	50	58	57	9	21
14	38	53	61	117	47	3	60	69	74
8	6	10	114	81	39	83	24	72	62
16	108	28	86	48	113	11	95	66	63
71	56	25	118	12	89	43	76	17	37

Total Weight in each Loading Position



17.65636893	22.06058262	68.90262337	122.3618105	446.444829	528.3780847	227.3213717	190.3787319	138.4002185	125.7827547
15.00222522	18.51388288	53.92560131	109.4023709	393.1443046	293.3903788	205.4189063	198.0163014	124.9832292	117.9993684
1.070170712	1.468315787	41.57651579	109.4281921	132.2048806	165.6264551	156.950575	380.6900412	217.5072825	164.5765859
6.229328413	4.241572979	21.88817743	253.3849086	297.2446964	145.1372651	197.6414993	234.4760707	97.21411465	111.7829314
3.195538383	9.365737162	20.85572894	97.29810557	429.6515962	238.1214511	146.2349638	129.9090083	81.21863944	71.78480822
6.000316778	15.70148853	49.51187721	261.3470379	532.61686	261.2892007	132.6537967	341.3620279	52.81891216	414.0740993

ID Number 12345678

-------- Solution --------

Ship Loading Positions	Index of Container at Posn
1	92
2	22
3	59
4	4
5	32
6	2
7	26
8	44
9	40
10	30
11	42
12	18
13	20
14	34
15	82
16	27
17	15
18	78
19	64
20	31
21	68
22	46
23	54
24	98
25	70
26	91
27	5
28	84
29	100
30	73
31	75
32	36
33	19
34	65
35	101
36	7
37	107
38	112
39	41
40	55
41	104
42	96
43	29
44	97
45	77
46	90
47	102
48	110
49	116
50	1
51	93
52	111
53	67
54	49
55	115
56	35
57	80
58	13
59	45
60	85
61	99
62	33
63	14
64	8
65	16
66	71
67	52
68	119
69	38
70	6
71	108
72	56
73	105
74	23
75	53
76	10
77	28
78	25
79	109
80	88
81	61
82	114
83	86
84	118
85	87
86	120
87	117
88	81
89	48
90	12
91	79
92	50
93	47
94	39
95	113
96	89
97	51
98	58
99	3
100	83
101	11
102	43
103	103
104	57
105	60
106	24
107	95
108	76
109	106
110	9
111	69
112	72
113	66
114	17
115	94
116	21
117	74
118	62
119	63
120	37

---- Solution Quality ----

ProbB

dX=	-1.3E-06
dY=	5.08E-08
Obj Fn=	1.58E-06

- Number of Containers -

Containers	120

- Number of Positions -

Positions	120

--- Container Data ---

Container Index	Container Weight
0	0
1	87.51651
2	0.87015
3	61.58868
4	0.745666
5	80.59673
6	2.884038
7	81.78999
8	5.483662
9	37.46672
10	12.91584
11	67.36751
12	248.008
13	79.73182
14	0.717683
15	10.34096
16	2.424872
17	27.63356
18	5.244997
19	99.16572
20	7.49868
21	32.51991
22	0.105823
23	50.00625
24	182.9773
25	38.50207
26	2.625794
27	8.972342
28	10.51477
29	99.98858
30	1.357535
31	18.70266
32	0.770666
33	14.8964
34	3.919351
35	85.47946
36	203.2026
37	95.60781
38	0.508907
39	70.31611
40	0.959409
41	95.3619
42	2.358016
43	53.30152
44	5.398177
45	32.04686
46	19.17551
47	66.46074
48	358.62
49	25.18535
50	90.18779
51	80.63375
52	19.43479
53	33.10859
54	14.33645
55	55.72874
56	10.45649
57	82.91769
58	71.53949
59	0.352488
60	95.06674
61	87.56534
62	32.05111
63	39.73795
64	15.70216
65	74.82116
66	36.18721
67	45.03143
68	21.86285
69	46.50483
70	96.55847
71	7.130167
72	37.90606
73	284.6089
74	79.98338
75	391.0193
76	48.20882
77	285.6233
78	11.00978
79	137.3588
80	84.59321
81	209.7193
82	8.467922
83	141.9128
84	87.52544
85	318.4663
86	82.96165
87	349.8864
88	90.69971
89	179.4792
90	51.49878
91	299.6533
92	1.662801
93	171.0025
94	39.84121
95	51.53105
96	79.35228
97	115.0986
98	22.68183
99	15.99357
100	71.03162
101	163.9521
102	78.37796
103	90.39015
104	78.86745
105	61.40394
106	43.96271
107	146.6876
108	7.007721
109	106.6597
110	293.1532
111	59.30805
112	133.8794
113	74.16934
114	234.2094
115	85.94154
116	94.43751
117	51.60815
118	238.6652
119	13.11571
120	93.49102

---------- Workings ----------

Objective Calculation Workings		
Position	Position	Position
1.662801	-33	7.5
0.105823	-33	4.5
0.352488	-33	1.5
0.745666	-33	-1.5
0.770666	-33	-4.5
0.87015	-33	-7.5
2.625794	-27	7.5
5.398177	-27	4.5
0.959409	-27	1.5
1.357535	-27	-1.5
2.358016	-27	-4.5
5.244997	-27	-7.5
7.49868	-21	7.5
3.919351	-21	4.5
8.467922	-21	1.5
8.972342	-21	-1.5
10.34096	-21	-4.5
11.00978	-21	-7.5
15.70216	-15	7.5
18.70266	-15	4.5
21.86285	-15	1.5
19.17551	-15	-1.5
14.33645	-15	-4.5
22.68183	-15	-7.5
96.55847	-9	7.5
299.6533	-9	4.5
80.59673	-9	1.5
87.52544	-9	-1.5
71.03162	-9	-4.5
284.6089	-9	-7.5
391.0193	-3	7.5
203.2026	-3	4.5
99.16572	-3	1.5
74.82116	-3	-1.5
163.9521	-3	-4.5
81.78999	-3	-7.5
146.6876	3	7.5
133.8794	3	4.5
95.3619	3	1.5
55.72874	3	-1.5
78.86745	3	-4.5
79.35228	3	-7.5
99.98858	9	7.5
115.0986	9	4.5
285.6233	9	1.5
51.49878	9	-1.5
78.37796	9	-4.5
293.1532	9	-7.5
94.43751	15	7.5
87.51651	15	4.5
171.0025	15	1.5
59.30805	15	-1.5
45.03143	15	-4.5
25.18535	15	-7.5
85.94154	21	7.5
85.47946	21	4.5
84.59321	21	1.5
79.73182	21	-1.5
32.04686	21	-4.5
318.4663	21	-7.5
15.99357	-33	7.5
14.8964	-33	4.5
0.717683	-33	1.5
5.483662	-33	-1.5
2.424872	-33	-4.5
7.130167	-33	-7.5
19.43479	-27	7.5
13.11571	-27	4.5
0.508907	-27	1.5
2.884038	-27	-1.5
7.007721	-27	-4.5
10.45649	-27	-7.5
40.40394	-21	7.5
50.00625	-21	4.5
33.10859	-21	1.5
12.91584	-21	-1.5
10.51477	-21	-4.5
38.50207	-21	-7.5
106.6597	-15	7.5
90.69971	-15	4.5
87.56534	-15	1.5
234.2094	-15	-1.5
82.96165	-15	-4.5
238.6652	-15	-7.5
349.8864	-9	7.5
93.49102	-9	4.5
51.60815	-9	1.5
209.7193	-9	-1.5
358.62	-9	-4.5
248.008	-9	-7.5
137.3588	-3	7.5
90.18779	-3	4.5
66.46074	-3	1.5
70.31611	-3	-1.5
74.16934	-3	-4.5
179.4792	-3	-7.5
80.63375	3	7.5
71.53949	3	4.5
61.58868	3	1.5
141.9128	3	-1.5
67.36751	3	-4.5
53.30152	3	-7.5
90.39015	9	7.5
82.91769	9	4.5
95.06674	9	1.5
182.9773	9	-1.5
51.53105	9	-4.5
48.20882	9	-7.5
43.96271	15	7.5
37.46672	15	4.5
46.50483	15	1.5
37.90606	15	-1.5
36.18721	15	-4.5
27.63356	15	-7.5
39.84121	21	7.5
32.51991	21	4.5
79.98338	21	1.5
32.05111	21	-1.5
39.73795	21	-4.5
95.60781	21	-7.5

---------- Workings ----------

Duplicate Check	All Containers
OK	OK
TRUE	TRUE

Ship Loading Positions

1	7	13	19	25	31	37	43	49	55
2	8	14	20	26	32	38	44	50	56
3	9	15	21	27	33	39	45	51	57
4	10	16	22	28	34	40	46	52	58
5	11	17	23	29	35	41	47	53	59
6	12	18	24	30	36	42	48	54	60

61	67	73	79	85	91	97	103	109	115
62	68	74	80	86	92	98	104	110	116
63	69	75	81	87	93	99	105	111	117
64	70	76	82	88	94	100	106	112	118
65	71	77	83	89	95	101	107	113	119
66	72	78	84	90	96	102	108	114	120

Index of Container in each Loading Position (0=empty)

0	0	0	0	18	19	14	0	0	0
0	0	0	0	5	1	0	0	0	0
0	0	0	0	3	0	0	0	0	0
0	0	0	0	0	0	0	0	7	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	9	0	20	0	0	0

0	0	0	12	11	6	2	0	4	0
0	0	0	0	10	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	8	0
0	0	0	0	0	0	0	15	16	0
0	0	0	0	0	17	13	0	0	0

Total Weight in each Loading Position

0	0	0	130.781	453.1908	429.282	189.628	0	15.958	0
0	0	0	0	110.557	1.234	0	0	0	0
0	0	0	0	8.258	0	0	0	0	0
0	0	0	0	0	0	0	0	98.856	0
0	0	0	0	0	0	0	218.649	256.452	0
0	0	0	0	70.973	298.215	612.357	0	0	0

ID Number 12345678

---- Solution Quality ----

ProbC

dX=	0.002788
dY=	0.003081
Obj Fn=	0.018195

- Number of Containers -

Containers	20

- Number of Positions -

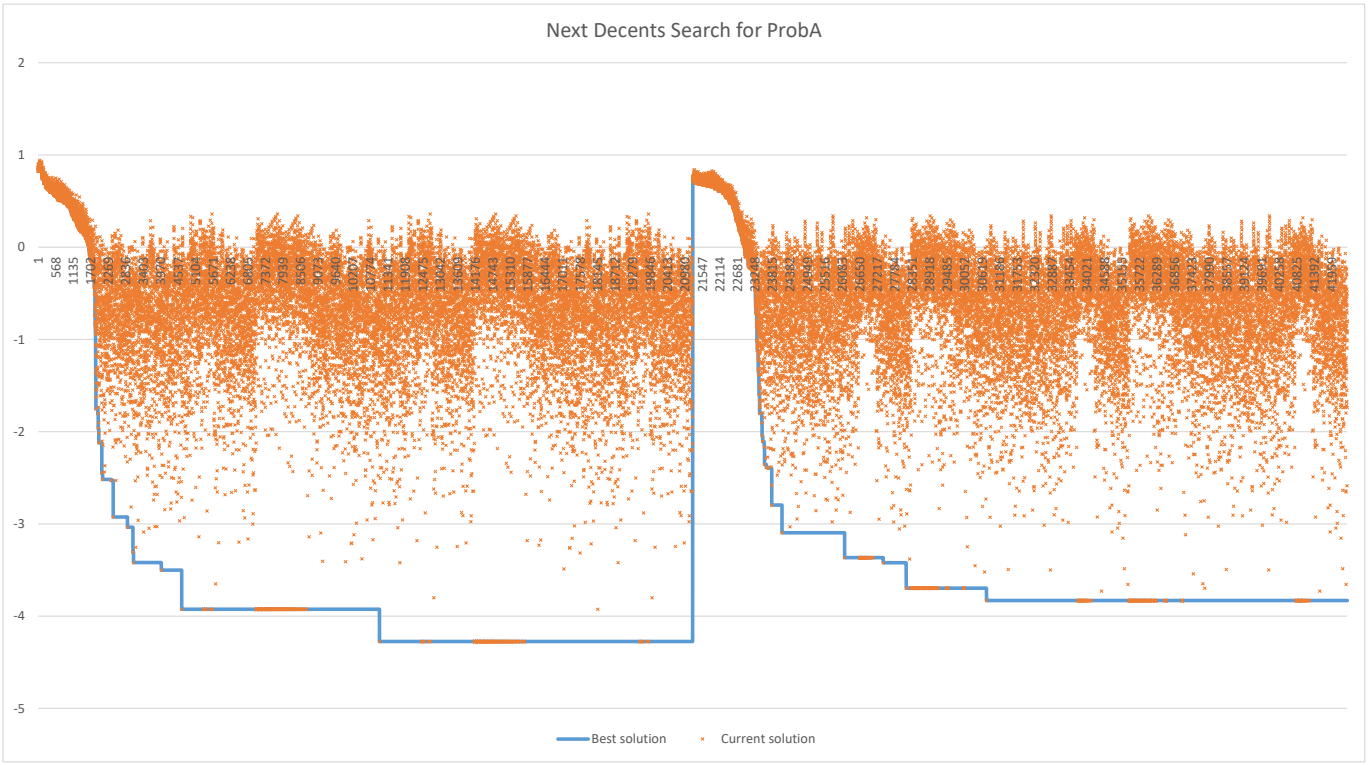
Positions	120

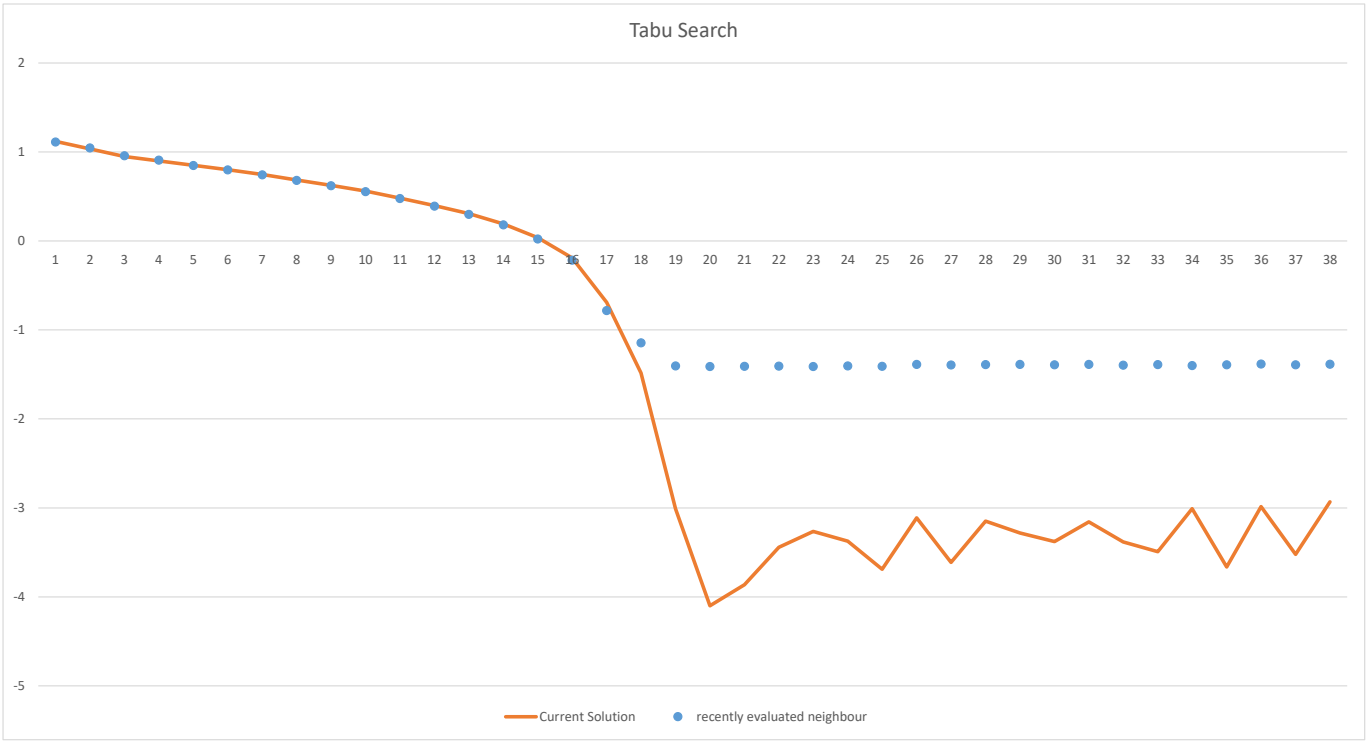
--- Container Data ----

Container Index	Container Weight
0	0
1	1.234
2	4.259
3	8.258
4	15.958
5	23.299
6	32.159
7	43.258
8	55.598
9	70.973
10	87.258
11	107.2458
12	130.781
13	156.589
14	185.369
15	218.649
16	256.452
17	298.215
18	345.945
19	397.123
20	455.768
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	
41	
42	
43	
44	
45	
46	
47	
48	
49	
50	
51	
52	
53	
54	
55	
56	
57	
58	
59	
60	
61	
62	
63	
64	
65	
66	
67	
68	
69	
70	
71	
72	
73	
74	
75	
76	
77	
78	
79	
80	
81	
82	
83	
84	
85	
86	
87	
88	
89	
90	
91	
92	
93	
94	
95	
96	
97	
98	
99	
100	

-------- Solution -------- / --------------- Workings --------------- / -------- Workings ---------

Ship Loading Positions	Index of Container at Posn	Objective Calculation Workings: Position	Position	Position	Duplicate Check OK	All Containers OK
1	0	0	-33	7.5	TRUE	TRUE
2	0	0	-33	4.5	TRUE	TRUE
3	0	0	-33	1.5	TRUE	TRUE
4	0	0	-33	-1.5	TRUE	TRUE
5	0	0	-33	-4.5	TRUE	TRUE
6	0	0	-33	-7.5	TRUE	TRUE
7	0	0	-27	7.5	TRUE	TRUE
8	0	0	-27	4.5	TRUE	TRUE
9	0	0	-27	1.5	TRUE	TRUE
10	0	0	-27	-1.5	TRUE	TRUE
11	0	0	-27	-4.5	TRUE	TRUE
12	0	0	-27	-7.5	TRUE	TRUE
13	0	0	-21	7.5	TRUE	TRUE
14	0	0	-21	4.5	TRUE	TRUE
15	0	0	-21	1.5	TRUE	TRUE
16	0	0	-21	-1.5	TRUE	TRUE
17	0	0	-21	-4.5	TRUE	TRUE
18	0	0	-21	-7.5	TRUE	TRUE
19	0	0	-15	7.5	TRUE	TRUE
20	0	0	-15	4.5	TRUE	TRUE
21	0	0	-15	1.5	TRUE	
22	0	0	-15	-1.5	TRUE	
23	0	0	-15	-4.5	TRUE	
24	0	0	-15	-7.5	TRUE	
25	18	345.945	-9	7.5	TRUE	
26	5	23.299	-9	4.5	TRUE	
27	3	8.258	-9	1.5	TRUE	
28	0	0	-9	-1.5	TRUE	
29	0	0	-9	-4.5	TRUE	
30	9	70.973	-9	-7.5	TRUE	
31	19	397.123	-3	7.5	TRUE	
32	1	1.234	-3	4.5	TRUE	
33	0	0	-3	1.5	TRUE	
34	0	0	-3	-1.5	TRUE	
35	0	0	-3	-4.5	TRUE	
36	0	0	-3	-7.5	TRUE	
37	14	185.369	3	7.5	TRUE	
38	0	0	3	4.5	TRUE	
39	0	0	3	1.5	TRUE	
40	0	0	3	-1.5	TRUE	
41	0	0	3	-4.5	TRUE	
42	20	455.768	3	-7.5	TRUE	
43	0	0	9	7.5	TRUE	
44	0	0	9	4.5	TRUE	
45	0	0	9	1.5	TRUE	
46	0	0	9	-1.5	TRUE	
47	0	0	9	-4.5	TRUE	
48	0	0	9	-7.5	TRUE	
49	0	0	15	7.5	TRUE	
50	0	0	15	4.5	TRUE	
51	0	0	15	1.5	TRUE	
52	7	43.258	15	-1.5	TRUE	
53	0	0	15	-4.5	TRUE	
54	0	0	15	-7.5	TRUE	
55	0	0	21	7.5	TRUE	
56	0	0	21	4.5	TRUE	
57	0	0	21	1.5	TRUE	
58	0	0	21	-1.5	TRUE	
59	0	0	21	-4.5	TRUE	
60	0	0	21	-7.5	TRUE	
61	0	0	-33	7.5	TRUE	
62	0	0	-33	4.5	TRUE	
63	0	0	-33	1.5	TRUE	
64	0	0	-33	-1.5	TRUE	
65	0	0	-33	-4.5	TRUE	
66	0	0	-33	-7.5	TRUE	
67	0	0	-27	7.5	TRUE	
68	0	0	-27	4.5	TRUE	
69	0	0	-27	1.5	TRUE	
70	0	0	-27	-1.5	TRUE	
71	0	0	-27	-4.5	TRUE	
72	0	0	-27	-7.5	TRUE	
73	0	0	-21	7.5	TRUE	
74	0	0	-21	4.5	TRUE	
75	0	0	-21	1.5	TRUE	
76	0	0	-21	-1.5	TRUE	
77	0	0	-21	-4.5	TRUE	
78	0	0	-21	-7.5	TRUE	
79	12	130.781	-15	7.5	TRUE	
80	0	0	-15	4.5	TRUE	
81	0	0	-15	1.5	TRUE	
82	0	0	-15	-1.5	TRUE	
83	0	0	-15	-4.5	TRUE	
84	0	0	-15	-7.5	TRUE	
85	11	107.2458	-9	7.5	TRUE	
86	10	87.258	-9	4.5	TRUE	
87	0	0	-9	1.5	TRUE	
88	0	0	-9	-1.5	TRUE	
89	0	0	-9	-4.5	TRUE	
90	0	0	-9	-7.5	TRUE	
91	6	32.159	-3	7.5	TRUE	
92	0	0	-3	4.5	TRUE	
93	0	0	-3	1.5	TRUE	
94	0	0	-3	-1.5	TRUE	
95	0	0	-3	-4.5	TRUE	
96	17	298.215	-3	-7.5	TRUE	
97	2	4.259	3	7.5	TRUE	
98	0	0	3	4.5	TRUE	
99	0	0	3	1.5	TRUE	
100	0	0	3	-1.5	TRUE	
101	0	0	3	-4.5	TRUE	
102	13	156.589	3	-7.5	TRUE	
103	0	0	9	7.5	TRUE	
104	0	0	9	4.5	TRUE	
105	0	0	9	1.5	TRUE	
106	0	0	9	-1.5	TRUE	
107	15	218.649	9	-4.5	TRUE	
108	0	0	9	-7.5	TRUE	
109	4	15.958	15	7.5	TRUE	
110	0	0	15	4.5	TRUE	
111	0	0	15	1.5	TRUE	
112	8	55.598	15	-1.5	TRUE	
113	16	256.452	15	-4.5	TRUE	
114	0	0	15	-7.5	TRUE	
115	0	0	21	7.5	TRUE	
116	0	0	21	4.5	TRUE	
117	0	0	21	1.5	TRUE	
118	0	0	21	-1.5	TRUE	
119	0	0	21	-4.5	TRUE	
120	0	0	21	-7.5	TRUE	


Next Decents Search for ProbA
2
1
0
-1
-2
-3
-4
-5
Best solution
Current solution


Tabu Search
2
1
0
-1
-2
-3
-4
-5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
Current Solution
recently evaluated neighbour

```
//Author: Baiwei Chen
//ID: 896127611
//UPI:bche722

package entry;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {

	private static String fileName = "ProbA";

	private static double[] weight;
	private static double[][] position;
	private static int[] solution;

	private static List<Double> bestMeasures;
	private static List<Double> tempMeasures;

	public static void main(String args[]) {
		weightReader("src/data/" + fileName + ".txt");
		positionReader("src/data/Positions.txt");
		bestMeasures = new ArrayList<Double>();
		tempMeasures = new ArrayList<Double>();

		nextDescentSearch(solution);
		nextDescentSearch(solution);
		write();

		nextDescentSearch200("ProbA");
		nextDescentSearch200("ProbB");
		nextDescentSearch200("ProbC");

		tabuSearch();
	}

	public static void weightReader(String path) {
		File file = new File(path);
		try {
```

```
            Scanner scanner = new Scanner(file);
            int lineNumber = Integer.parseInt(scanner.nextLine());
            weight = new double[lineNumber + 1];
            solution = new int[120];
            weight[0] = 0;
            for (int i = 1; i < lineNumber + 1; i++) {
                weight[i] = Double.parseDouble(scanner.nextLine().trim());
                solution[i - 1] = i;
            }
            scanner.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }

    public static void positionReader(String path) {
        File file = new File(path);
        try {
            Scanner scanner = new Scanner(file);
            int lineNumber = Integer.parseInt(scanner.nextLine().trim());
            position = new double[lineNumber][2];
            for (int i = 0; i < lineNumber; i++) {
                String[] temp = scanner.nextLine().split("¥¥s+");
                position[i][0] = Double.parseDouble(temp[1]);
                position[i][1] = Double.parseDouble(temp[2]);
            }
            scanner.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }

    public static void tabuSearch() {
        weightReader("src/data/ProbA.txt");
        bestMeasures = new ArrayList<Double>();
        tempMeasures = new ArrayList<Double>();
        boolean[][] flag = new boolean[120][120];
        int banListSize = Math.min(20, weight.length / 3);
        int[][] banList = new int[banListSize][2];
        int count = 0;
        int worsen = 50000;

        double currentX = 0;
        double currentY = 0;
```

```
            double sumWeight = 0;
            for (int i = 0; i < 120; i++) {
                currentX += weight[solution[i]] * position[i][0];
                currentY += weight[solution[i]] * position[i][1];
                sumWeight += weight[solution[i]];
            }
            int[] current = solution.clone();
            while (count < 100000 && count < worsen * 2) {
                int[] currentPosition = new int[2];
                double iterationBestX = Double.POSITIVE_INFINITY;
                double iterationBestY = Double.POSITIVE_INFINITY;
                double tempMeasure = 0;
                // Find the best solution in each iteration with the tabu list
                for (int a = 0; a < 120; a++) {
                    for (int b = a + 1; b < 120; b++) {
                        if (flag[a][b] || (weight[current[a]] == 0 && weight[current[b]] == 0) || a
== b - 60) {
                            continue;
                        } else {
                            double tempX = currentX + weight[current[a]] * position[b][0]
                                    + weight[current[b]] * position[a][0] - weight[current[a]]
* position[a][0]
                                    - weight[current[b]] * position[b][0];
                            double tempY = currentY + weight[current[a]] * position[b][1]
                                    + weight[current[b]] * position[a][1] - weight[current[a]]
* position[a][1]
                                    - weight[current[b]] * position[b][1];
                            tempMeasure = Math.abs(tempX) + 5 * Math.abs(tempY);
                            double iterationBestMeasure = Math.abs(iterationBestX) + 5 *
Math.abs(iterationBestY);
                            if (tempMeasure < iterationBestMeasure) {
                                iterationBestX = tempX;
                                iterationBestY = tempY;
                                currentPosition[0] = a;
                                currentPosition[1] = b;
                            }
                        }
                    }
                }
                // check whether the solution become worse and change the stop condition
                double currentMeasure = Math.abs(currentX) + 5 * Math.abs(currentY);
                double iterationBestMeasure = Math.abs(iterationBestX) + 5 *
Math.abs(iterationBestY);
                tempMeasures.add(Math.log10(currentMeasure/sumWeight));
```

```
                bestMeasures.add(Math.log10(tempMeasure/sumWeight));
                if (currentMeasure < iterationBestMeasure && worsen == 50000) {
                    worsen = count;
                }
                // change the current solution to the best solution in this iteration
                int temp = current[currentPosition[0]];
                current[currentPosition[0]] = current[currentPosition[1]];
                current[currentPosition[1]] = temp;
                currentX = iterationBestX;
                currentY = iterationBestY;
                // Add the swap to the ban list and using the ban list to update the matrix
                flag[banList[count % banListSize][0]][banList[count % banListSize][1]] = false;
                flag[currentPosition[0]][currentPosition[1]] = true;
                banList[count % banListSize][0] = currentPosition[0];
                banList[count % banListSize][1] = currentPosition[1];
                // Increase the iteration count
                count++;
            }

            try {
                PrintWriter pw = new PrintWriter("src/output/TabuSearch.txt");
                for (int i = 0; i < tempMeasures.size(); i++) {
                    pw.write(bestMeasures.get(i) + "    ");
                    pw.write(tempMeasures.get(i) + "¥n");
                }
                pw.flush();
                pw.close();
            } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

    public static void nextDescentSearch(int[] s) {
        shuffle(s);
        double bestX = 0;
        double bestY = 0;
        double sumWeight = 0;
        for (int i = 0; i < 120; i++) {
            bestX += weight[s[i]] * position[i][0];
            bestY += weight[s[i]] * position[i][1];
            sumWeight += weight[s[i]];
        }
        int[] bestSolution = s.clone();
```

```
            double bestMeasure = Math.abs(bestX) + 5 * Math.abs(bestY);
            boolean flag = true;
            while (flag) {
                flag = false;
                for (int a = 0; a < 120; a++) {
                    for (int b = a + 1; b < 120; b++) {
                        if (a == b - 60) {
                            continue;
                        } else {
                            double tempX = bestX + weight[bestSolution[a]] * position[b][0]
                                    + weight[bestSolution[b]] * position[a][0] -
weight[bestSolution[a]] * position[a][0]
                                    - weight[bestSolution[b]] * position[b][0];
                            double tempY = bestY + weight[bestSolution[a]] * position[b][1]
                                    + weight[bestSolution[b]] * position[a][1] -
weight[bestSolution[a]] * position[a][1]
                                    - weight[bestSolution[b]] * position[b][1];
                            double tempMeasure = Math.abs(tempX) + 5 * Math.abs(tempY);
                            if (tempMeasure < bestMeasure) {
                                int temp = bestSolution[a];
                                bestSolution[a] = bestSolution[b];
                                bestSolution[b] = temp;
                                bestX = tempX;
                                bestY = tempY;
                                bestMeasure = tempMeasure;
                                flag = true;
                            }
                            bestMeasures.add(Math.log10(bestMeasure/sumWeight));
                            tempMeasures.add(Math.log10(tempMeasure/sumWeight));
                        }
                    }
                }
            }
        }

        public static void nextDescentSearch200(String name) {
            weightReader("src/data/" + name + ".txt");
            double bestX = 0;
            double bestY = 0;
            double sumWeight = 0;
            for (int i = 0; i < 120; i++) {
                bestX += weight[solution[i]] * position[i][0];
                bestY += weight[solution[i]] * position[i][1];
                sumWeight += weight[solution[i]];
```

```
            }
            int[] bestSolution = solution.clone();
            double bestMeasure = Math.abs(bestX) + 5 * Math.abs(bestY);
            for (int i = 0; i < 200; i++) {
                shuffle(solution);
                double iterationBestX = 0;
                double iterationBestY = 0;
                for (int j = 0; j < 120; j++) {
                    iterationBestX += weight[solution[j]] * position[j][0];
                    iterationBestY += weight[solution[j]] * position[j][1];
                }
                int[] iterationBestSolution = solution.clone();
                double iterationBestMeasure = Math.abs(iterationBestX) + 5 *
Math.abs(iterationBestY);
                boolean flag = true;
                while (flag) {
                    flag = false;
                    for (int a = 0; a < 120; a++) {
                        for (int b = a + 1; b < 120; b++) {
                            if (a == b - 60) {
                                continue;
                            } else {
                                double tempX = iterationBestX +
weight[iterationBestSolution[a]] * position[b][0]
                                        + weight[iterationBestSolution[b]] * position[a][0]
                                        - weight[iterationBestSolution[a]] * position[a][0]
                                        - weight[iterationBestSolution[b]] * position[b][0];
                                double tempY = iterationBestY +
weight[iterationBestSolution[a]] * position[b][1]
                                        + weight[iterationBestSolution[b]] * position[a][1]
                                        - weight[iterationBestSolution[a]] * position[a][1]
                                        - weight[iterationBestSolution[b]] * position[b][1];
                                double tempMeasure = Math.abs(tempX) + 5 *
Math.abs(tempY);
                                if (tempMeasure < iterationBestMeasure) {
                                    int temp = iterationBestSolution[a];
                                    iterationBestSolution[a] = iterationBestSolution[b];
                                    iterationBestSolution[b] = temp;
                                    iterationBestX = tempX;
                                    iterationBestY = tempY;
                                    iterationBestMeasure = tempMeasure;
                                    flag = true;
                                }
                            }
```

```
                        }
                    }
                }
                if (iterationBestMeasure < bestMeasure) {
                    bestSolution = iterationBestSolution;
                    bestMeasure = iterationBestMeasure;
                }
            }
            try {
                PrintWriter pw = new PrintWriter("src/output/" + name + "output.txt");
                pw.write(bestMeasure/sumWeight + "¥n");
                for (int i = 0; i < bestSolution.length; i++) {
                    pw.write(bestSolution[i] + "¥n");
                }
                pw.flush();
                pw.close();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            }
        }

    public static void write() {
        try {
            PrintWriter pw = new PrintWriter("src/output/output.txt");
            for (int i = 0; i < bestMeasures.size(); i++) {
                pw.write(bestMeasures.get(i) + "   ");
                pw.write(tempMeasures.get(i) + "¥n");
            }
            pw.flush();
            pw.close();
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public static void shuffle(int[] s) {
        for (int i = 0; i < 119; i++) {
            int j = i + 1 + (int) (Math.random() * (118 - i));
            int temp = s[i];
            s[i] = s[j];
            s[j] = temp;
        }
    }
```

```
}
```

I swear upon an appropriate entity of my choice that the output was produced by running the code handed in without undue assistance from others.