CSCI E-124-Spring, 2004 Programming Assignment 3

Russell Lowke, May 14th 2004

1. Give a dynamic programming solution to the Number Partition problem.

In essence, with the number partition problem we're trying to get the sums of the numbers as close to 0 as possible. However, for a recursive/dynamic programming solution, we can vary the target number. The residue will not be defined as the difference between the sum of the numbers and the target sum. Since the solution to the problem is some assignment of number into two sets, a recursive solution is apparent; for some array of size n, the best assignment for elements 1 - i (for some i between 1 n) is the one which allows the last n-j elements to have the smallest residue. So we assign an element, recompute the rest assuming the first one was correct, and pick the minimum

For the problem to work we want to keep shortening the length of the problem array P. First we pick an assignment for element P[i]. Then we make the target number for the rest of the array -P[i] and start it again. We do this for both assignments and pick the one closest to the target value. If we want to make this a dynamic programming algorithm, we'll have to store (for every possible point on the array), how close we can guide it toward a given target value. This could be a 2D array that's indexed into by [n][target number].

2. Explain briefly how the Karmarker-Karp algorithm can be implemented in O (n log n) steps, assuming the values in A are small enough that arithmetic operations take one step.

In max. O (n log(n)) steps one can make a priority queue from the items (inserting them individually one by one, log n steps to put them in the proper place). Once the items are in the priority queue, we're popping the top two and inserting one back. This means that, regardless of what else happens, the priority queue's size will shrink by one every time. So, at maximum, this part of the algorithm can run for O(n) steps. Retrieving and inserting items into the priority queue are all max. O(log n), so this part of the algorithm runs in O(n log n) steps. So the creation of the heap is O (n log n) and the loop is O (n log n) thus the entire thing is O(n log n).

Main Problem assignment:

Generate 50 random instances of the problem as described.

[for entire data see the log file included called *NumberPartition.log*]

Here is the residue data from 50 instances each run with 25000 iterations.

```
1)
Residue from KK heuristic is:                49581
HillClimbing residue using S is:             260866163
HillClimbing residue using P is:             28145
RandomSolution residue using S is:           50621787
RandomSolution residue using P is:           8276
SimulatedAnnealing residue using S is:       6917863
SimulatedAnnealing residue using P is:       27744

2)
Residue from KK heuristic is:                151129
HillClimbing residue using S is:             417116250
HillClimbing residue using P is:             37676
RandomSolution residue using S is:           15933338
RandomSolution residue using P is:           6548
SimulatedAnnealing residue using S is:       737892
SimulatedAnnealing residue using P is:       10145

3)
Residue from KK heuristic is:                338452
HillClimbing residue using S is:             670324241
HillClimbing residue using P is:             65628
RandomSolution residue using S is:           38737901
RandomSolution residue using P is:           1911
SimulatedAnnealing residue using S is:       1545631423353
SimulatedAnnealing residue using P is:       23170

4)
Residue from KK heuristic is:                240224
HillClimbing residue using S is:             175016115
HillClimbing residue using P is:             43572
RandomSolution residue using S is:           7377975
RandomSolution residue using P is:           9043
SimulatedAnnealing residue using S is:       57190091
```

```
SimulatedAnnealing residue using P is:      31477

5)
Residue from KK heuristic is:               447220
HillClimbing residue using S is:            442287458
HillClimbing residue using P is:            66315
RandomSolution residue using S is:          48531292
RandomSolution residue using P is:          5979
SimulatedAnnealing residue using S is:      7624046
SimulatedAnnealing residue using P is:      7679

6)
Residue from KK heuristic is:               192662
HillClimbing residue using S is:            31972752
HillClimbing residue using P is:            33773
RandomSolution residue using S is:          20791978
RandomSolution residue using P is:          8296
SimulatedAnnealing residue using S is:      34517760
SimulatedAnnealing residue using P is:      73881416

7)
Residue from KK heuristic is:               91980
HillClimbing residue using S is:            105973498
HillClimbing residue using P is:            31458
RandomSolution residue using S is:          21775606
RandomSolution residue using P is:          5940
SimulatedAnnealing residue using S is:      1993217787048
SimulatedAnnealing residue using P is:      7851

8)
Residue from KK heuristic is:               546099
HillClimbing residue using S is:            555194923
HillClimbing residue using P is:            11118
RandomSolution residue using S is:          780109
RandomSolution residue using P is:          2338
SimulatedAnnealing residue using S is:      1373281061925
SimulatedAnnealing residue using P is:      46896307

9)
Residue from KK heuristic is:               57422
HillClimbing residue using S is:            335320337
HillClimbing residue using P is:            11363
```

```
RandomSolution residue using S is:          3791165
RandomSolution residue using P is:          7364
SimulatedAnnealing residue using S is:      14146053
SimulatedAnnealing residue using P is:      27620

10)
Residue from KK heuristic is:               122860
HillClimbing residue using S is:            235612025
HillClimbing residue using P is:            46701
RandomSolution residue using S is:          6599137
RandomSolution residue using P is:          7543
SimulatedAnnealing residue using S is:      32806655
SimulatedAnnealing residue using P is:      282391

11)
Residue from KK heuristic is:               266171
HillClimbing residue using S is:            147254639
HillClimbing residue using P is:            41642
RandomSolution residue using S is:          448455
RandomSolution residue using P is:          5770
SimulatedAnnealing residue using S is:      5248813
SimulatedAnnealing residue using P is:      1531739

12)
Residue from KK heuristic is:               51913
HillClimbing residue using S is:            284180902
HillClimbing residue using P is:            50521
RandomSolution residue using S is:          67211686
RandomSolution residue using P is:          4427
SimulatedAnnealing residue using S is:      11484088
SimulatedAnnealing residue using P is:      66216

13)
Residue from KK heuristic is:               89146
HillClimbing residue using S is:            30936818
HillClimbing residue using P is:            71669
RandomSolution residue using S is:          47894066
RandomSolution residue using P is:          2094
SimulatedAnnealing residue using S is:      1011862
SimulatedAnnealing residue using P is:      76287
```

14)
Residue from KK heuristic is: 40266
HillClimbing residue using S is: 1195455038
HillClimbing residue using P is: 12521
RandomSolution residue using S is: 44306190
RandomSolution residue using P is: 4313
SimulatedAnnealing residue using S is: 43183620
SimulatedAnnealing residue using P is: 95100

15)
Residue from KK heuristic is: 88928
HillClimbing residue using S is: 605603924
HillClimbing residue using P is: 18999
RandomSolution residue using S is: 126800882
RandomSolution residue using P is: 2867
SimulatedAnnealing residue using S is: 42363608
SimulatedAnnealing residue using P is: 1931325

16)
Residue from KK heuristic is: 147371
HillClimbing residue using S is: 936448445
HillClimbing residue using P is: 58611
RandomSolution residue using S is: 37719913
RandomSolution residue using P is: 1765
SimulatedAnnealing residue using S is: 12064547
SimulatedAnnealing residue using P is: 503821

17)
Residue from KK heuristic is: 71857
HillClimbing residue using S is: 352392812
HillClimbing residue using P is: 49956
RandomSolution residue using S is: 26738604
RandomSolution residue using P is: 7897
SimulatedAnnealing residue using S is: 16898778
SimulatedAnnealing residue using P is: 13247

18)
Residue from KK heuristic is: 522113
HillClimbing residue using S is: 131760234
HillClimbing residue using P is: 63818
RandomSolution residue using S is: 22210354
RandomSolution residue using P is: 9817

```
SimulatedAnnealing residue using S is:       5473242
SimulatedAnnealing residue using P is:       36805

19)
Residue from KK heuristic is:                243700
HillClimbing residue using S is:             496513667
HillClimbing residue using P is:             13254
RandomSolution residue using S is:           86244667
RandomSolution residue using P is:           3273
SimulatedAnnealing residue using S is:       25133607
SimulatedAnnealing residue using P is:       33562

20)
Residue from KK heuristic is:                155908
HillClimbing residue using S is:             283969005
HillClimbing residue using P is:             14249
RandomSolution residue using S is:           64478261
RandomSolution residue using P is:           10116
SimulatedAnnealing residue using S is:       7105987
SimulatedAnnealing residue using P is:       943547

21)
Residue from KK heuristic is:                60320
HillClimbing residue using S is:             98153197
HillClimbing residue using P is:             91551
RandomSolution residue using S is:           310247677
RandomSolution residue using P is:           1461
SimulatedAnnealing residue using S is:       13392535
SimulatedAnnealing residue using P is:       7380

22)
Residue from KK heuristic is:                267378
HillClimbing residue using S is:             426021282
HillClimbing residue using P is:             9862
RandomSolution residue using S is:           79338358
RandomSolution residue using P is:           6932
SimulatedAnnealing residue using S is:       36668630
SimulatedAnnealing residue using P is:       3890

23)
Residue from KK heuristic is:                470058
HillClimbing residue using S is:             73457393
```

```
HillClimbing residue using P is:            48395
RandomSolution residue using S is:          38336959
RandomSolution residue using P is:          1880
SimulatedAnnealing residue using S is:      617397602887
SimulatedAnnealing residue using P is:      1309408

24)
Residue from KK heuristic is:               236901
HillClimbing residue using S is:            522757963
HillClimbing residue using P is:            98094
RandomSolution residue using S is:          7999419
RandomSolution residue using P is:          4195
SimulatedAnnealing residue using S is:      729645
SimulatedAnnealing residue using P is:      14628

25)
Residue from KK heuristic is:               1260186
HillClimbing residue using S is:            102646413
HillClimbing residue using P is:            21447
RandomSolution residue using S is:          20631073
RandomSolution residue using P is:          751
SimulatedAnnealing residue using S is:      899681030117
SimulatedAnnealing residue using P is:      253582

26)
Residue from KK heuristic is:               181974
HillClimbing residue using S is:            197239432
HillClimbing residue using P is:            52128
RandomSolution residue using S is:          10337642
RandomSolution residue using P is:          7526
SimulatedAnnealing residue using S is:      1672343032316
SimulatedAnnealing residue using P is:      20376810

27)
Residue from KK heuristic is:               184941
HillClimbing residue using S is:            97442591
HillClimbing residue using P is:            41802
RandomSolution residue using S is:          77117447
RandomSolution residue using P is:          14138
SimulatedAnnealing residue using S is:      6091225
SimulatedAnnealing residue using P is:      981262
```

28)
Residue from KK heuristic is: 361901
HillClimbing residue using S is: 243232579
HillClimbing residue using P is: 38698
RandomSolution residue using S is: 1726653
RandomSolution residue using P is: 10188
SimulatedAnnealing residue using S is: 10210819
SimulatedAnnealing residue using P is: 6640

29)
Residue from KK heuristic is: 243957
HillClimbing residue using S is: 188531025
HillClimbing residue using P is: 19261
RandomSolution residue using S is: 11681555
RandomSolution residue using P is: 8241
SimulatedAnnealing residue using S is: 5816837
SimulatedAnnealing residue using P is: 7937

30)
Residue from KK heuristic is: 288768
HillClimbing residue using S is: 703344716
HillClimbing residue using P is: 8942
RandomSolution residue using S is: 15335770
RandomSolution residue using P is: 7454
SimulatedAnnealing residue using S is: 3635568
SimulatedAnnealing residue using P is: 32605807

31)
Residue from KK heuristic is: 469062
HillClimbing residue using S is: 168207033
HillClimbing residue using P is: 122834
RandomSolution residue using S is: 56977093
RandomSolution residue using P is: 1392
SimulatedAnnealing residue using S is: 1147555939995
SimulatedAnnealing residue using P is: 23259

32)
Residue from KK heuristic is: 302175
HillClimbing residue using S is: 222319732
HillClimbing residue using P is: 60253
RandomSolution residue using S is: 39459834
RandomSolution residue using P is: 9901

```
SimulatedAnnealing residue using S is:       6165080
SimulatedAnnealing residue using P is:       6271

33)
Residue from KK heuristic is:                132094
HillClimbing residue using S is:             122953189
HillClimbing residue using P is:             98113
RandomSolution residue using S is:           99936821
RandomSolution residue using P is:           2850
SimulatedAnnealing residue using S is:       21557185
SimulatedAnnealing residue using P is:       5877788

34)
Residue from KK heuristic is:                245253
HillClimbing residue using S is:             20919184
HillClimbing residue using P is:             42356
RandomSolution residue using S is:           54293484
RandomSolution residue using P is:           4870
SimulatedAnnealing residue using S is:       1506424136236
SimulatedAnnealing residue using P is:       47711

35)
Residue from KK heuristic is:                127085
HillClimbing residue using S is:             61658253
HillClimbing residue using P is:             18280
RandomSolution residue using S is:           4420607
RandomSolution residue using P is:           2666
SimulatedAnnealing residue using S is:       11905777
SimulatedAnnealing residue using P is:       232592

36)
Residue from KK heuristic is:                193833
HillClimbing residue using S is:             180840151
HillClimbing residue using P is:             52160
RandomSolution residue using S is:           28609197
RandomSolution residue using P is:           6896
SimulatedAnnealing residue using S is:       21352153
SimulatedAnnealing residue using P is:       8529149

37)
Residue from KK heuristic is:                103224
HillClimbing residue using S is:             562688245
```

```
HillClimbing residue using P is:            57942
RandomSolution residue using S is:          40180353
RandomSolution residue using P is:          5788
SimulatedAnnealing residue using S is:      31669269
SimulatedAnnealing residue using P is:      32579

38)
Residue from KK heuristic is:               32144
HillClimbing residue using S is:            108635074
HillClimbing residue using P is:            56660
RandomSolution residue using S is:          27193872
RandomSolution residue using P is:          3391
SimulatedAnnealing residue using S is:      5633000
SimulatedAnnealing residue using P is:      73867

39)
Residue from KK heuristic is:               354015
HillClimbing residue using S is:            3056277
HillClimbing residue using P is:            70031
RandomSolution residue using S is:          20900759
RandomSolution residue using P is:          13434
SimulatedAnnealing residue using S is:      61169207
SimulatedAnnealing residue using P is:      5201048

40)
Residue from KK heuristic is:               4987
HillClimbing residue using S is:            237032265
HillClimbing residue using P is:            38422
RandomSolution residue using S is:          3672845
RandomSolution residue using P is:          8918
SimulatedAnnealing residue using S is:      3096811
SimulatedAnnealing residue using P is:      4272

41)
Residue from KK heuristic is:               1028887
HillClimbing residue using S is:            266516000
HillClimbing residue using P is:            38506
RandomSolution residue using S is:          13030218
RandomSolution residue using P is:          4293
SimulatedAnnealing residue using S is:      17684396
SimulatedAnnealing residue using P is:    7325
```

42)
Residue from KK heuristic is: 204889
HillClimbing residue using S is: 283561607
HillClimbing residue using P is: 14806
RandomSolution residue using S is: 18531987
RandomSolution residue using P is: 5966
SimulatedAnnealing residue using S is: 2409
SimulatedAnnealing residue using P is: 11165

42)
Residue from KK heuristic is: 204889
HillClimbing residue using S is: 283561607
HillClimbing residue using P is: 14806
RandomSolution residue using S is: 18531987
RandomSolution residue using P is: 5966
SimulatedAnnealing residue using S is: 2409
SimulatedAnnealing residue using P is: 11165

43)
Residue from KK heuristic is: 153274
HillClimbing residue using S is: 254275104
HillClimbing residue using P is: 11940
RandomSolution residue using S is: 26721764
RandomSolution residue using P is: 14100
SimulatedAnnealing residue using S is: 14131804
SimulatedAnnealing residue using P is: 12585

44)
Residue from KK heuristic is: 410935
HillClimbing residue using S is: 380969392
HillClimbing residue using P is: 69096
RandomSolution residue using S is: 159450584
RandomSolution residue using P is: 2969
SimulatedAnnealing residue using S is: 10171806
SimulatedAnnealing residue using P is: 6115

45)
Residue from KK heuristic is: 64124
HillClimbing residue using S is: 23046365
HillClimbing residue using P is: 4924
RandomSolution residue using S is: 2017685

RandomSolution residue using P is: 9512
SimulatedAnnealing residue using S is: 1681605013119
SimulatedAnnealing residue using P is: 1300013

46)
Residue from KK heuristic is: 113875
HillClimbing residue using S is: 28279841
HillClimbing residue using P is: 56373
RandomSolution residue using S is: 25970167
RandomSolution residue using P is: 2637
SimulatedAnnealing residue using S is: 77035131725
SimulatedAnnealing residue using P is: 31798162

47)
Residue from KK heuristic is: 103949
HillClimbing residue using S is: 608835347
HillClimbing residue using P is: 34110
RandomSolution residue using S is: 37189157
RandomSolution residue using P is: 8294
SimulatedAnnealing residue using S is: 1054931010293
SimulatedAnnealing residue using P is: 36915611

48)
Residue from KK heuristic is: 83868
HillClimbing residue using S is: 141800635
HillClimbing residue using P is: 25343
RandomSolution residue using S is: 20559543
RandomSolution residue using P is: 15674
SimulatedAnnealing residue using S is: 3109773
SimulatedAnnealing residue using P is: 4003528

49)
Residue from KK heuristic is: 268670
HillClimbing residue using S is: 245547715
HillClimbing residue using P is: 12975
RandomSolution residue using S is: 1363053
RandomSolution residue using P is: 2690
SimulatedAnnealing residue using S is: 546699
SimulatedAnnealing residue using P is: 1347251

```
50)
Residue from KK heuristic is:                590228
HillClimbing residue using S is:             338893877
HillClimbing residue using P is:             124303
RandomSolution residue using S is:           19094291
RandomSolution residue using P is:           2637
SimulatedAnnealing residue using S is:       8956391
SimulatedAnnealing residue using P is:       2372658
```

Compare the results and discuss.

The KK algorithm always returned a reasonable answer within five or six digits, not always great but reasonable, much better than the set of type S. Quite surprisingly the closest answer was consistently the random solution using P [instance #25 was the best], revealing how crucial it is to find a good starting point, in this case found purely randomly over many iterations.

When using set P type heuristics, we get much better answers. Because testing the residue involved us feeding this back into the KK algorithm, our answers were always very low. Using P with the random solutions always led to a MUCH closer result than the other representation or even the KK algorithm running by itself, the random solution being the most effective, having a good chance of accidentally starting near the correct answer.

Simulated Annealing in type P gave varied results, yet when effective its result was usually better then its Hill Climbing type P counterpart. There is a risk with Hill Climbing and Simulated Annealing that when the solution space is so large and when starting from a random result, it could easily end up so far away from a good answer that it simply can't make it to a reasonable solution.

With set P, having to pass answers through the KK algorithm to test them (although it slowed down the code) allowed a "bad" answer to be pulled close to the optimum and hence almost always a vast improvement over set S was found. Also, the random groupings of some numbers together allowed the KK algorithm to get over one of its biggest pitfalls; sometimes two large numbers need to be together for the optimal solution. Simulated annealing was a personal disappointment, but I think this is because, again, the length of the running time wasn't enough to see it pull close to a good solution. It was, in essence, taking steps that were just too small and going in the wrong direction a little too often.

Discuss briefly how you could use the solution from the Karmarkar-Karp algorithm as a starting point for randomized algorithms, and suggest what effect that might have.

You can run the KK algorithm and take the set of answers (of the form { +1, -1, ...} and feed that into a hill climbing algorithm and get a better result than randomly choosing a starting point. If you randomly choose a starting point for the hill climbing algorithm, you can possibly end up very far from the solution (and can't get there in the allotted time) but this guarantees that you'll be very close to the right answer so the randomized algorithms will produce results at least as good as the KK algorithm, which itself is a pretty good answer.

CODE

Please note, I'm using the C++ Number Theory Library, "ntl-5.3.1.1" as obtained from **http://www.shoup.net/ntl/index.html** to deal with large integers [`typedef ZZ bigint;`] and large real numbers [`typedef RR bigdouble;`]

```
//
//
//  Heuristics.cpp
//
//       AUTHOR: Russell Lowke
//
//               Date: 5/15/2004
//
//  Implementing some randomized and local search heuristics for solving NP-complete
problems.
//


#include <iostream>
using std::cin;
using std::cout;
using std::endl;

#include <NTL/ZZ.h>
#include <NTL/RR.h>
using namespace NTL;
typedef RR bigdouble;
typedef ZZ bigint;

#include <queue>
#include <vector>
typedef std::priority_queue< bigint > PriorityQueue;
```

```
const int MAX_ITER = 250000;
const float MATH_E = 2.71828182846;

//
//  random generator
//
void randomizer()
{
    time_t      timeval;

    time(&timeval);
    srand(timeval);
    SetSeed(to_ZZ(rand()));
}

double randomReal() {
    return rand() / (double) RAND_MAX;
}

int randomInt(int min, int max) {
    int range = (max + 1) - min;
    return (int) (randomReal() * range + min);
}


/* produces an array of 100 randomly generated integers in the range [1, 10^12]*/
void generateRandomInstance(unsigned long randomInstance[100]) {

    long bound = (long) pow(10, 12);

    bigint bi = to_ZZ("120347129034871029834710284018934701297834019873 4");
    cout << "The bigint in generateRandomInstance, bigbound, is: " << bi << endl;
    for (int i= 0; i < 100; ++i) {

        unsigned long current = (rand() % bound) + 1;
        /* for creating 64 random bits
         current << 32;
         current |= ((rand() % bound) + 1);
        */
         randomInstance[i] = current;
    }
}

/* produces an array of 100 randomly generated integers in the range [1, 10^12]*/
void generateRandomInstance(bigint randomInstance[100]) {

    //long bound = (long) pow(10, 12);
    bigint bound = power_ZZ(10, 12);

    for (int i= 0; i < 100; ++i) {
         bigint current = RandomBnd(bound) + 1;
         randomInstance[i] = current;
    }

    cout << "Instance is:\n{ ";
    for (int i = 0; i < 100; ++i){
```

```
        cout << randomInstance[i] << " ";
    }
    cout << "}" << endl;

}


void generateRandomInts(int randomInstance[100], int from, int to) {

       for (int i = 0; i < 100; ++i) {
              randomInstance[i] = randomInt(from, to);
       }
}


void copyArray(int arraySource[100], int arrayDest[100]) {
    for (int i = 0; i < 100; ++i) {
        arrayDest[i] = arraySource[i];
    }
}


void copyBigIntArray(bigint arraySource[100], bigint arrayDest[100]) {
    for (int i = 0; i < 100; ++i) {
        arrayDest[i] = arraySource[i];
    }
}


bigint* doGroupings(bigint instance[100], int groupings[100]) {

       bigint* result = new bigint[100];
       copyBigIntArray(instance, result);

       for (int i = 0; i < 100; ++i) {
              int firstCaseOf = -1;

              for (int j = 0; j < 100; ++j) {

                     if (groupings[j] == i) {
                            if (firstCaseOf == -1) {
                                   firstCaseOf = j;
                            } else {
                                   result[firstCaseOf] += result[j];
                                   result[j] = 0;
                            }
                     }
              }
       }

       return result;
}


/* produces an array of 100 randomly generated integers which are either -1 or +1 */
void generateRandomSolution(int randomSolution[100]) {
    for (int i = 0; i < 100; ++i) {
```

```
        int num = rand() % 2;
        if (num == 0) {
                num = -1;
            }
         randomSolution[i] = num;
    }
    return;
}


/* must pass in a copy of original data.  It will be modified into a neighbor */
void generateRandomNeighbor(int newNeighbor[100]) {
    int i = random() % 100 + 1;
    int j;
    do {
        j = random() % 100 + 1;
    } while (i == j);

    //now that i and j are different indices, flip i
    newNeighbor [i] *= -1;

    //and flip j with 50% probability
    if ( randomReal() < 0.50) {
        newNeighbor[j] *= -1;
    }
}

void generateRandomNeighborFromP(int newNeighbor[100]) {
    int i = random() % 100 + 1;
    int j;
    for (int k = 0; k < 100; ++k) {
        j = random() % 100 + 1;

            if (newNeighbor[i] != newNeighbor[j]){
                newNeighbor[i] = newNeighbor[j];
                break;
            }
    }
}


long calculateResidue(unsigned long instance[100], int assignments[100]) {
    long residue = 0;
    for (int i = 0; i < 100; ++i) {
        residue += instance[i] * assignments[i];
    }
    return abs(residue);
}


bigint calculateResidue(bigint instance[100], int assignments[100]) {
    bigint residue;
    for (int i = 0; i < 100; ++i) {
        residue += instance[i] * assignments[i];
    }
    return abs(residue);
}
```

```
/* implementing the KK heuristic */
bigint KKsolution(bigint instance[100]){

	// make a priority queue of size 100;
	// feed all of the numbers from instance into it
	PriorityQueue queue;
	for(int i = 0; i < 100; ++i) {
		if ( instance[i] != 0 ){
			queue.push(instance[i]);
		}
	}

	bigint a, b, diff;

	while (queue.size() > 2) {
		a = queue.top();
		queue.pop();

		b = queue.top();
		queue.pop();
		if (b == 0) {
			queue.push(a);
			break;
		}

		diff = abs(a - b);
		queue.push(diff);
	}

	return (queue.top());
}


bigint calculateResidueWithP(bigint instance[100], int assignments[100]) {
    bigint residue;
       bigint* temp = doGroupings(instance, assignments);
    residue = KKsolution(temp);
       delete[] temp;
    return abs(residue);
}


int* randomSolution(unsigned long* instance) {

    // Start with a random solution S
    int* s1 = new int[100];
    int* s2;
    unsigned long r1, r2;
     generateRandomSolution(s1);
     r1 = calculateResidue(instance, s1);

    // for iter = 1 to max_iter
    //        S` = a random solution
    for (int i = 0; i < MAX_ITER; ++i) {
        s2 = new int[100];
```

```
        //
         generateRandomSolution(s2);
         r2 = calculateResidue(instance, s2);

        // if residue(S`) < residue(S) then S = S`
        if (r2 < r1) {
            delete[] s1;
            s1 = s2;
            r1 = r2;
        } else {
            delete[] s2;
        }
    }

    // resturn S
    return s1;
}

int* randomSolutionWithP(bigint* instance) {

    // Start with a random solution S
    int* s1 = new int[100];
       generateRandomInts(s1, 1, 100);
    int* s2;
    bigint r1, r2;
     r1 = calculateResidueWithP(instance, s1);

    // for iter = 1 to max_iter
    //         S` = a random solution
    for (int i = 0; i < MAX_ITER; ++i) {
        s2 = new int[100];

         generateRandomSolution(s2);
               generateRandomInts(s2, 1, 100);
         r2 = calculateResidueWithP(instance, s2);
        // if residue(S`) < residue(S) then S = S`
        if (r2 < r1) {
            delete[] s1;
            s1 = s2;
            r1 = r2;

        } else {
            delete[] s2;
        }
    }

    // resturn S
    return s1;
}

int* randomSolution(bigint* instance) {

    // Start with a random solution S
    int* s1 = new int[100];
    int* s2;
    bigint r1, r2;
     generateRandomSolution(s1);
```

```
     r1 = calculateResidue(instance, s1);

    // for iter = 1 to max_iter
    //        S` = a random solution
    for (int i = 0; i < MAX_ITER; ++i) {
        s2 = new int[100];

         generateRandomSolution(s2);
         r2 = calculateResidue(instance, s2);

        // if residue(S`) < residue(S) then S = S`
        if (r2 < r1) {
            delete[] s1;
            s1 = s2;
            r1 = r2;
        } else {
            delete[] s2;
        }
    }

    // resturn S
    return s1;
}

int* hillClimbing(bigint* instance) {

    // Start with a random solution S
    int* s1 = new int[100];
    int* s2;
    bigint r1, r2;
     generateRandomSolution(s1);
     r1 = calculateResidue(instance, s1);

    // for iter = 1 to max_iter
    for (int i = 0; i < MAX_ITER; ++i) {

        //           S` = a random neighbor of S
        s2 = new int[100];
         copyArray(s1, s2);
         generateRandomNeighbor(s2);
         r2 = calculateResidue(instance, s2);

        // if residue(S`) < residue(S) then S = S`
        if (r2 < r1) {
                    //cout<<"disposing of s1, assigning s2 to s1"<<endl;
            delete[] s1;
            s1 = s2;
            r1 = r2;
        } else {
                    //cout<<"disposing of s2"<<endl;
            delete[] s2;
        }
    }

    // resturn S
    return s1;
}
```

```
int* hillClimbingWithP(bigint* instance) {

    // Start with a random solution S
    int* s1 = new int[100];
    int* s2;
    bigint r1, r2;
       generateRandomInts(s1, 1, 100);
     r1 = calculateResidueWithP(instance, s1);

    // for iter = 1 to max_iter
    for (int i = 0; i < MAX_ITER; ++i) {

        //            S` = a random neighbor of S
        s2 = new int[100];
         copyArray(s1, s2);
          generateRandomNeighborFromP(s2);
          r2 = calculateResidueWithP(instance, s2);

        // if residue(S`) < residue(S) then S = S`
        if (r2 < r1) {

                    if (s1 != s2){
                           delete[] s1;
                    }
            s1 = s2;
            r1 = r2;
        } else {

            delete[] s2;
        }
    }

    // resturn S
    return s1;
}

float temperature(int iter){
    // the real temperature calculation
    return (pow(10, 10) * pow(0.9, (iter/300.0)));
}


int* simulatedAnnealing(bigint* instance) {

    // Start with a random solution S
    int* s1 = new int[100];
    int* s2;
    bigint r1, r2, r3;
     generateRandomSolution(s1);
     r1 = calculateResidue(instance, s1);
       float temper;
    // S`` = S
    int* s3 = new int[100];
     copyArray(s1, s3);
    r3 = r1;
```

```
    // for iter = 1 to max_iter
    for (int i = 0; i < MAX_ITER; ++i) {

        //           S` = a random neighbor of S
        s2 = new int[100];
        copyArray(s1, s2);
        generateRandomNeighbor(s2);
        r2 = calculateResidue(instance, s2);

        // if residue(S`) < residue(S) then S = S`
        if (r2 < r1) {
                    if (s1 != s3) {
                           delete[] s1;
                    }
            s1 = s2;
                    r1 = r2;
        } else {

            // S = S` with probability exp(-(res(S`) - res(S`)) / T(iter))
                    temper = (temperature(i) > 50 ? temperature(i) : 50);

            if (randomReal() < power(to_RR(MATH_E), ( - (to_long(r2 - r1)) / temper ))) {
                           //                    power (float   ,   zz / float
                           if (s1 != s3) {
                                  delete[] s1;
                           }
                s1 = s2;
                           r1 = r2;
            } else {
                delete[] s2;
            }
        }

        // if residue(S) < residue(S``) then S = S`
        if (r1 < r3) {
            delete[] s3;
            s3 = s1;
            r3 = r1;
        }
    }

    if (s1 != s3) {
        delete[] s1;
    }

    // resturn S``
    return s3;
}

int* simulatedAnnealingWithP(bigint* instance) {

    // Start with a random solution S
    int* s1 = new int[100];
    int* s2;
       float temper;
    bigint r1, r2, r3;
       generateRandomInts(s1, 1, 100);
```

```
    r1 = calculateResidueWithP(instance, s1);

    // S`` = S
    int* s3 = new int[100];
    copyArray(s1, s3);
    r3 = r1;

    // for iter = 1 to max_iter
    for (int i = 0; i < MAX_ITER; ++i) {

        //           S` = a random neighbor of S
        s2 = new int[100];
        copyArray(s1, s2);
        generateRandomNeighborFromP(s2);
        r2 = calculateResidueWithP(instance, s2);

        // if residue(S`) < residue(S) then S = S`
        if (r2 < r1) {
                    if (s1 != s3) {
                        delete[] s1;
                    }
            s1 = s2;
                    r1 = r2;
        } else {

            // S = S` with probability exp(-(res(S`) - res(S`)) / T(iter))
                    temper = (temperature(i) > 50 ? temperature(i) : 50);

            if (randomReal() < power(to_RR(MATH_E), ( - (to_long(r2 - r1)) / temper ))) {
                        //                    power (float   ,   zz / float
                        if (s1 != s3) {
                            delete[] s1;
                        }
                s1 = s2;
                        r1 = r2;
            } else {
                delete[] s2;
            }
        }

        // if residue(S) < residue(S``) then S = S`
        if (r1 < r3) {
            delete[] s3;
            s3 = s1;
            r3 = r1;
        }
    }

    if (s1 != s3) {
        delete[] s1;
    }

    // resturn S``
    return s3;
}
```

```
//
//
//  numberPartition
//
//       AUTHOR: Russell Lowke
//
//               Date: 5/15/2004
//


#include <iostream>
using std::cout;
using std::endl;

#include <fstream>
using std::fstream;
using std::ios;

#include "heuristic.h"

void printAnswer(int* answer) {
    cout << "{";
    for (int i = 0; i < 100; ++i) {
        cout << answer[i] << " ";
    }
    cout << "}" << endl;
}

void printAnswerToFile(int* answer, fstream& log) {
    log << "{";
    for (int i = 0; i < 100; ++i) {
        log << answer[i] << " ";
    }
    log << "}" << endl;
}

void printInstanceToFile(bigint* instance, fstream& log) {
    log << "{";
    for (int i = 0; i < 100; ++i) {
        log << instance[i] << " ";
    }
    log << "}" << endl;
}


int main (int argc, char * const argv[]) {
       fstream logFile("NumberPartition.log", ios::out);
       randomizer();
       for(int i = 0; i < 50; ++i){

              cout << i+1 << ")" << endl;
              logFile << i+1 << ")" << endl;

              bigint instance[100];
              generateRandomInstance(instance);
              cout << "Random instance generated\n";
              logFile << "instance: ";
```

```
            printInstanceToFile( instance, logFile );

            int* answer;
            int* answerFromP;

            cout << "Answer from KK heuristic is\n";
            cout << KKsolution(instance) << endl;

            logFile << "Answer from KK heuristic is\n";
            logFile << KKsolution(instance) << endl;

            cout << "About to start computing hill climbing answers\n";

            cout << "hillClimbing answer using S is:\n";
            answer = hillClimbing(instance);
            printAnswer(answer);
            logFile << "hillClimbing answer using S is:\n";
            printAnswerToFile(answer, logFile);

            cout << "Residue is: " << calculateResidue(instance, answer) << endl;
            logFile << "Residue is: " << calculateResidue(instance, answer) << endl;
            delete[] answer;

            cout << "hillClimbing answer using P is:\n";
            answerFromP = hillClimbingWithP(instance);
            printAnswer(answerFromP);
            cout << "Residue is: " << calculateResidueWithP(instance, answerFromP) << endl;
            logFile << "hillClimbing answer using P is:\n";
            printAnswerToFile(answerFromP, logFile);
            logFile << "Residue is: " << calculateResidueWithP(instance, answerFromP) <<
endl;
            delete[] answerFromP;


            cout << "About to start computing random answers\n";

            cout << "randomSolution answer using S is:\n";
            answer = randomSolution(instance);
            printAnswer(answer);
            logFile << "randomSolution answer using S is:\n";
            printAnswerToFile(answer, logFile);
            cout << "Residue is: " << calculateResidue(instance, answer) << endl;
            logFile << "Residue is: " << calculateResidue(instance, answer) << endl;
            delete[] answer;

            cout << "randomSolution answer using P is:\n";
            answerFromP = randomSolutionWithP(instance);
            printAnswer(answerFromP);
            logFile << "randomSolution answer using P is:\n";
            printAnswerToFile(answerFromP, logFile);

            cout << "Residue is: " << calculateResidueWithP(instance, answerFromP) << endl;
            logFile << "Residue is: " << calculateResidueWithP(instance, answerFromP) <<
endl;
            delete[] answerFromP;

            cout << "About to start computing simulatedAnnealing answers\n";
```

```
                cout << "simulatedAnnealing answer using S is:\n";
                answer = simulatedAnnealing(instance);
                printAnswer(answer);
                cout << "Residue is: " << calculateResidue(instance, answer) << endl;

                logFile << "simulatedAnnealing answer using S is:\n";
                printAnswerToFile(answer, logFile);
                logFile << "Residue is: " << calculateResidue(instance, answer) << endl;
                delete[] answer;

                answerFromP = simulatedAnnealingWithP(instance);
                cout << "simulatedAnnealing answer using P is:\n";
                printAnswer(answerFromP);
                cout << "Residue is: " << calculateResidueWithP(instance, answerFromP) << endl;

                logFile << "simulatedAnnealing answer using P is:\n";
                printAnswerToFile(answerFromP, logFile);
                logFile << "Residue is: " << calculateResidueWithP(instance, answerFromP) <<
endl;
                delete[] answerFromP;

                cout << endl;
                logFile << endl;
        }

    return 0;
}
```