ECOO 2012

Programming Contest

Solutions & Notes

Local Competition (Round 1)

After March 18, 2012

Problem 1: Sluggers

In the test data sets, the team names were all fixed to be the same number of characters. This was to make it easier for the judges to spot formatting errors.

The Test Data

Each data set consists of actual data for the top 10 teams from that season.

Solution to DATA11.txt

```
2010 Regular Season
====================
NYYankees: .267 .436
Boston___: .268 .451
Tampa_Bay: .247 .403
Cincinnat: .272 .436
Texas____: .276 .419
Minnesota: .273 .422
Philadelp: .260 .413
Colorado_: .263 .425
Toronto__: .248 .454
ChicagoSo: .268 .420
====================
Big 10 Av: .264 .428
```

Solution to DATA12.txt

```
2000 Regular Season
====================
ChicagoSo: .286 .470
Colorado_: .294 .455
Cleveland: .288 .470
Oakland__: .270 .458
Houston__: .278 .477
SanFranci: .278 .472
Seattle__: .269 .442
St.Louis_: .270 .455
KansasCit: .288 .425
NYYankees: .277 .450
====================
Big 10 Av: .280 .457
```

Problem 2: Decoding DNA

This problem is based on some of the actual processes of RNA Transcription, but everything is simplified. The only tricky part here is finding the terminator sequence. It gets easier if you notice that you only have to look for pairs of sequences of length 6, since longer repeated sequences will also contain a pair of length 6 sequences.

Recommended Approach

To find the start of the transcription unit, just search for “TATAAT” using the built-in string functions in whatever language you are using, and add 10 to the index where you first find it. To find the terminator sequence, one approach is to start at the beginning of the transcription unit and look at each sequence of 6 bases in turn. For each sequence, reverse it and compute its complement, then search for this reverse complement starting after the original 6 base sequence. As soon as you find a match, you have the end of the sequence.

Possible pitfalls include:

1. Terminator-like sequences that occur before the start of the transcription unit.
2. Palindromic or partially palindromic sequences like ATCGAT – its reverse complement is also ATCGAT.
3. Zero length transcription units.

The Test Data

Each set contains at least one of each of the possible pitfalls.

Solution to DATA21.txt

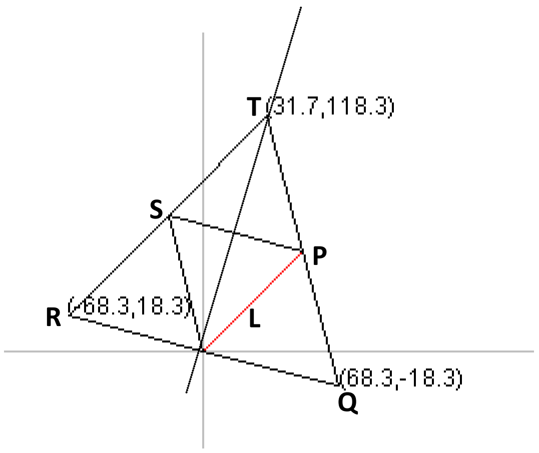
```
1: U
2: ACC
3: GA
4: UGAGU
5: GUUAGCUAG
```

Solution to DATA22.txt

```
1: UUUGG
2: CCUGUUAAGGAAUUCCUCGCUUUCGUGUAACC
3: UCUAUGAACACGUUUCUGUGC
4: AGUUACGUGUUGUAAUAGACCGAUAAGUCCCUAAAGCUAC
5: A
```

Problem 3: Triangles

Recommended approach



First, figure out the angle (theta) and length (L) of the line from the origin to the given point P (use $\tan^{-1}$ for the angle and the Pythagorean Theorem for the length). Points Q and R are also at a distance of L from the origin, but at angles *theta – 60^o* and *theta + 120^o* respectively. You can use sin and cosine to get these points. (Another way to do this is to notice that the points are reflections of one another in the line y=x, so once you have one you just flip the sign of the coordinates to get the other.)

The third point is trickier. Figure out where point S is (it's at angle *theta + 60^0* and at a distance of L from the origin). Now the final point you want is on the perpendicular bisector of PS, at a distance of *2h* from the origin, where *h* is the height of the small triangles (use the Pythagorean Theorem to get *h*). You can get the angle of the perpendicular bisector using the negative reciprocal of the slope of PS and $\tan^{-1}$. And you know this line goes through the origin. Now use sin and cos to get the coordinates of T.

Pitfalls

You have to remember that your computer probably thinks in radians, not degrees. So you have to do the proper conversions. Every time you compute a new angle for a point, you may have to correct it to make sure you are within 0 to 360^0, and you may also have to correct it to make sure it is in the same quadrant as the x and y you used to compute it. (e.g. if x=-10 and y=-10,$\tan^{-1}$(y/x) = 45^o, but this puts you in Quadrant I. You need to add 180^o in this case to get the real angle.) You also have to watch out for vertical and horizontal lines and you may need to treat them as special cases.

Solution to DATA31.txt

```
(-54.6,14.6) (54.6,-14.6) (25.4,94.6)
(54.6,-14.6) (-54.6,14.6) (-25.4,-94.6)
(-153.2,-134.6) (153.2,134.6) (-233.2,265.4)
(-173.2,-100.0) (173.2,100.0) (-173.2,300.0)
(100.0,-173.2) (-100.0,173.2) (-300.0,-173.2)
```

Solution to DATA32.txt

```
(-99.5,0.3) (99.5,-0.3) (0.5,172.3)
(49.5,86.3) (-49.5,-86.3) (149.5,-85.7)
(17.3,10.0) (-17.3,-10.0) (17.3,-30.0)
(-11.5,19.9) (11.5,-19.9) (34.5,19.9)
(-51.0,-97.6) (51.0,97.6) (-169.0,88.4)
```

Problem 4: Boggled

The basic approach here is to write a backtracking search that takes the board, the start location, and a word as inputs and returns true if the word is there, false otherwise. Then write a loop that goes through the entire board calling the backtracker at each spot until the word is found or no spots are left. When exploring possible paths through the board, you need to mark squares that you have already visited to avoid overlapping paths. A 2D array of Booleans or ints could serve this purpose.

Potential Pitfalls…

There are traps in this question. The sample data contains only "realistic" situations. There are some insidious bugs that are easy to introduce such that most cases will still work, but certain special cases will not (in some cases testing with other words that are on the boards in the given data but not present in the lists may reveal the bugs).

First pitfall: You could create a solution without a "visited" array, but you might find words that aren't there. For example, in the board shown in the problem, the word SALTALT would be wrongly identified as being in the board.

Second pitfall: Even with a "visited" array, you could run into trouble in cases where there are two paths to go down, but only one leads to finding the word (this happens a lot with repeated letters). In a recursive backtracker this means that if the current part of the search finds nothing, you have to mark the current location as not visited before returning. If not you could end up not finding something that is in fact there. In two examples below, depending on your search order (e.g. up down left right vs. left right down up) you will get two of the four words wrong if you have made this mistake. Again, cases like this are not present in the sample data.

```
Board:    AADA        Board:      ADAA
          DDPC                    ADAA
          AACA                    DPCC
          AACA                    ACAA

Words:    PDDD        Words:      PDDD
          PCCC                    PCCC
```

The Test Data

If you don't use a visited list, only 1 of the 5 cases in each data set will be correct in each data set. If you don't remember to unmark visited nodes (as described above) only 3 of the 5 cases in each data set will be correct.

Solution to DATA41.txt

```
Your score: 17 (16 good, 5 not found, 1 too short, 1 repeated)
Your score: 33 (20 good, 4 not found, 2 too short, 2 repeated)
Your score: 23 (21 good, 5 not found, 2 too short, 4 repeated)
Your score: 9 (4 good, 1 not found, 0 too short, 0 repeated)
Your score: 24 (4 good, 1 not found, 0 too short, 0 repeated)
```

Solution to DATA42.txt

```
Your score: 22 (21 good, 7 not found, 2 too short, 3 repeated)
Your score: 25 (14 good, 4 not found, 0 too short, 0 repeated)
Your score: 42 (29 good, 3 not found, 0 too short, 2 repeated)
Your score: 15 (4 good, 1 not found, 0 too short, 0 repeated)
Your score: 24 (4 good, 1 not found, 0 too short, 0 repeated)
```

ECOO 2012
Programming Contest
Solutions & Notes

Regional Competition (Round 2)

April 28, 2012

Problem 1: Prime Time

Recommended Approach

Every number is a product of two numbers: The message code number in the range 0 to 3030, and the prime number key in the range 100000 to 500000. The easiest thing to do is simply search one of the message numbers sequentially for the first factor you find starting at 100000.

It may also be possible in some cases to find the GCF (greatest common factor) of two of the message numbers using the Euclidean algorithm or some other method. But this method may not find the right key. For example if the two message numbers you select are both even, the GCF will actually be double the key.

Solution to DATA11.txt

```
REMEMBER WHEN YOU WERE YOUNG? YOU SHONE LIKE THE SUN...

NOW THERES A LOOK IN YOUR EYE, LIKE BLACK HOLES IN THE SKY.

SHINE ON YOU CRAZY DIAMOND!

YOU WERE CAUGHT IN THE CROSSFIRE OF CHILDHOOD AND STARDOM, BLOWN ON
THE STEEL BREEZE.

COME ON YOU RAVER, YOU SEER OF VISIONS. COME ON YOU PAINTER, YOU
PIPER, YOU PRISM AND SHINE!
```

Solution to DATA12.txt

```
ZIGGY PLAYED GUITAR, JAMMING GOOD WITH WIERD AND GILLY AND THE SPIDERS
FROM MARS.

HE PLAYED IT LEFT HAND, BUT MADE IT TOO FAR...

BECAME THE SPECIAL MAN! THEN WE WERE ZIGGYS BAND.

ZIGGY PLAYED FOR TIME, JIVING US THAT WE WERE VOODOO. THE KIDS WERE
JUST CRASS!

WHEN THE KIDS HAD KILLED THE MAN I HAD TO BREAK UP THE BAND...
```

Problem 2: Password Strength

Notes

There are no hidden traps here, but very few test cases are given so you have to interpret the criteria and test your solution carefully before submitting. The hardest part is probably finding the longest sequences. This can be done by looping through the string, keeping track of the last character seen, and keeping count of the current and longest sequence seen so far.

Solution to DATA21.txt

```
Very Weak (score = 4)
Very Weak (score = 3)
Good (score = 44)
Good (score = 57)
Weak (score = 33)
Very Strong (score = 93)
Strong (score = 78)
Very Strong (score = 100)
Very Weak (score = 15)
Strong (score = 80)
```

Solution to DATA22.txt

```
Very Weak (score = 14)
Good (score = 41)
Very Weak (score = 5)
Very Strong (score = 92)
Very Strong (score = 100)
Very Weak (score = 3)
Very Weak (score = 4)
Strong (score = 75)
Weak (score = 28)
Very Weak (score = 20)
```

Problem 3: Airport Radar

Recommended Approach

Compute the end point of the flight using trigonometry functions ($x = r \cdot cos(\theta),\ y = r \cdot sin(\theta)$). Then compute equation of the line of flight path ($y = mx$, where the slope is the tangent of the angle of flight). Then, for each tower:

1. Use length of line segment formula to check if the origin or end point of the flight is within radar range. If so, count the tower.
2. If not, compute the equation of the line perpendicular to the flight path passing through the radar tower point, then compute the intersection point. If it is on the line segment (between the start and end point), compute the distance to see if it's in radar range. If it is, count the tower.

This procedure may have special cases for horizontal and vertical lines, depending on the programming environment.

Simulation Approach

Compute the end point of the flight as above, then get the slope of the plane and divide rise and run by a big number to get small x and y increments for simulation. Simulate the flight of the plane along its path, checking each tower that has not seen the plane yet at each step to see if it can see it at this point (using length of a line segment formula). Count all the towers that see the plane at some point during the simulation.

Solution to DATA31.txt

```
The jet will appear on 15 radar screens.
The jet will appear on 9 radar screens.
The jet will appear on 2 radar screens.
The jet will appear on 5 radar screens.
The jet will appear on 1 radar screens.
```

Solution to DATA32.txt

```
The jet will appear on 1 radar screens.
The jet will appear on 8 radar screens.
The jet will appear on 8 radar screens.
The jet will appear on 4 radar screens.
The jet will appear on 26 radar screens.
```

Problem 4: Lo Shudoku

Exhaustive Search

An exhaustive search of all possible moves would be difficult to write and will not finish in time, so another strategy is needed. Perhaps a breadth first search would work in some cases, but still in the worst case would still have a bad running time and memory requirements. As a ball park estimate, in the worst case you have to consider 8 swaps for each square, giving you 8^9 combinations (more than 130 million) to try.

Observations

If you can find a fast enough way to transform a 3x3 square into one of the magic squares, you can just run it 8 times – once for each of the eight possible magic squares – and look for the smallest required number of moves.

There is an upper bound of 9 moves for transforming any 3x3 square that has no blanks into any given magic square. This is because every swap can be used to put at least one number in its correct spot. In fact, there is no faster approach when there are no blanks than to simply cycle through all nine spots and swap the correct piece into place if necessary.

It actually does not matter to the number of moves required whether you fill in the blanks at the beginning or at the end.

Final Strategy

Try each of the 8 magic squares. For each one, you transform the given 3x3 square by first swapping into place as many pieces as you can, and then by filling in the blanks at the end with their correct numbers. Whichever of the 8 squares requires the least moves gives you the answer.

Solution to DATA41.txt

```
446            999            888            485            000
554            999            888            769            080
553            999            888            564            000
---            ---            ---            ---            ---
```

Solution to DATA42.txt

```
188            565            000            988            544
797            336            078            988            564
882            554            778            878            444
---            ---            ---            ---            ---
```

ECOO 2012
Programming Contest
Solutions and Notes

Final Competition (Round 3)

May 12, 2012

Problem 1: The Word Garden

Recommended Approach

Trying to generate and draw the trees on the screen all at once will be quite difficult. A better idea is to create a two-dimensional array of characters (24 rows, 40 columns), "draw" the trees into this array, then draw the array to the screen. Once the array is created, you can fill it with spaces and draw the ground in the final row immediately. Then you need an algorithm (preferably in a method, function, or procedure) for drawing a tree with its trunk at a given location. Test that algorithm to make sure you can draw a single word tree anywhere you want. It's a good idea to put it in a method, procedure, or function.

The next part is figuring out where to draw each tree. The first one is not hard. If the word is 5 characters long, its trunk should be in the 5th column of the array. But figuring out where to draw each of the next trees is harder. You could try to do some mathematical or logical reasoning to figure out where each one should go, but once you have the drawing algorithm done, you can copy it and modify it so that instead of actually drawing the tree, it checks all the array locations the tree would need for a "collision" with other trees. This second algorithm can be used to test possible locations for the next tree. If you know where the trunk of the last tree was, you can start at that location plus 1 and then keep moving to the right until your tree checking algorithm says it's ok. Then draw the tree there.

The Test Cases

The text for both data sets is from Martin Luther King Jr.'s famous "I have a dream" speech.

Solution to DATA11.txt

```
          d
         drd
   h    drerd    t
  hah  dreaerd  tht
 havahdreamaerdthaht  o    d     t
havevah   d   thatahtono  dad   tht
   h      r      t  onenodayadotheht
   a      e      h    o    d ono t
I  va     a      a    n    a  o  h
I  ea     m      t    e    y  n  e
========================================
                G
               GeG
              GeoeG            f
             GeoroeG          fof
       h    GeorgroeG        forof
      hih  GeorgigroeG      formrof
     hilihGeorgiaigroeG s  formemrof
    hilllih     G      sosformeremrof
  rhillsllih    e  t  sonos    f
 rer   h        o thtsonsnos   o
reder  i o      rtheht  s o    r
  r    lofo     g  t    oofo   m
  e    l o      i  h    n o    e
  d    s f      a  e    s f    r
========================================

     s                   f
    sls                 fof
   slals               forof      s
  slavals             formrof    sls
 slavevals        s  formemrof  slals
slavesevals      sosformeremrofslavals
     s  a    t  sonos    f    slavevals
     l ana  thtsonsnos   o        s
     aandnatheht  s o    r        l
     v  a    t    oofo   m        a
     e  n    h    n o    e        v
     s  d    e    s f    r        e
========================================

     o
    owo
   ownwo
  ownenwo
 ownerenwo  w      a         d
ownersrenwowiw    aba       dod
     o    wiliw  ablba  s  dowod
     w   willliwablelbasisdownwod
     n      w b    a tsitis  d
     e      ibeb   btot s    o
     r      l b    l t  i    w
     s      l e    e o  t    n
========================================
                      b
                     brb
                    brorb
                   brotorb
                  brothtorb
                 brothehtorb
       t        brotherehtorb
      tot      brotherhrehtorb
     togot    brotherhohrehtorb
    togegot  brotherhooohrehtorb
   togetegotbrotherhoodoohrehtorb
  togethtegot         b
 togethehtegot   t    r
togetherehtegot tat   o
       t       tabat  t
       o      tablbat h
       g    ttablelbate
       e   tht   t    r
       t atheht  a o  h
       hata t    bofo o
       e a  h    l o  o
       r t  e    e f  d
========================================
```

Solution to DATA12.txt

```
                               l
                              lil
          d                  litil
         drd                litttil
   h    drerd    t      f  littlttil
  hah  dreaerd  tht    foflittlelttil
 havahdreamaerdthaht  fouof    l
havevah   d   thatahtfouruof   i
   h      r      t m    f      t
   a      e      hmym   o      t
I  va     a      a m    u      l
I  ea     m      t y    r      e
========================================

       c
      chc
     chihc
    chilihc
   childlihc                     n
  childrdlihc                   nan
 childrerdlihc                 natan
childrenerdlihc               natitan
       c   w              l  natioitan
       h  wiw            lilnationoitan
       i wiliw  o    d  livil    n
       lwillliwono  dadlivevil   a
       d   w  onenodayad  l i    t
       r   i    o    d    iini   i
       e   l    n    a    v ia   o
       n   l    e    y    e na   n
========================================
                            j
                           juj
    w                     juduj
   whw                   judgduj
  whehw    t      w     judgegduj
 wherehw  tht    wiw   judgedegduj
whererehwtheht  wiliw  n    j    t
    w   theyehtwillliwnon   u   tht
    h      t      w  notonb d btheht
    e      h      i    n bebgbyb t
    r      e      l    o  b e b  h
    e      y      l    t  e d y  e
========================================

     c
    coc
   coloc      t
  cololoc    tht
 colouoloc  theht    s
colouruoloctheieht  sks
     c    theiriehtskiks  b     t
     o        t   skiniksbub   tht
     l o      h      s  butubbtheht
     oofo     e      k    b byb t
     u o      i      i    u  b  h
     r f      r      n    t  y  e
========================================

                        c
                       chc
                      chahc
                     charahc
      c             chararahc
     coc           characarahc
    conoc         charactcarahc
   contnoc       charactetcarahc
  contetnoc    tcharacteretcarahc
 contenetnoc  tht       c
contentnetnoctheht      h
      c     theieht     a
      o    theirieht    r
      n        t        a
      t o      h        c
      eofo     e        t
      n o      i        e
      t f      r        r
========================================
```

Problem 2: Jewelry Tips

Recommended Approach

The best way to solve this is just a top to bottom, left to right sweep of the entire board. For each jewel, simulate swaps in all 4 directions (in order of priority – LT, UP, RT, DN) and check for lines, then undo the swap and continue. Keep track of the first “good” and “excellent” tips found, but quit immediately if you find a “normal” tip and return that. If you get to the end of the board without any normal tips, return the good tip you found, or the excellent tip if there was no good tip, or “game over” if no tips of any kind were found.

I found it helpful to create a special counting method (a.k.a. procedure or function) that takes the board and a location as parameters, then counts the jewels of the same colour vertically and horizontally in both directions. If there is only one line >= 3, it returns the length of it. If there are two, it adds their lengths and returns that. If there are none, it returns 0. So 0 means no line, 3 or 4 means a single line, and 5 or more means either more than one line or a line of five.

Some Further Notes

When you swap two jewels, you have to check that the other jewel in the swap doesn’t end up creating lines as well. If it does, you’ve got an excellent tip, so you shouldn’t return it as a normal tip.

There are not many situations in which you would return a good tip, since when you one jewel to create a line of 4, there is always another jewel you could move to make a line of 3 instead. It is only when this move creates multiple lines (making it excellent rather than normal) that you can return a good tip.

The Test Cases

The test cases were generated automatically by randomly generating thousands of boards. The boards were thrown away if they already contained a line of 3 or more. If not, they were checked using the jewelry tip routine written above to see what kind of tips they contained and kept if they met the criteria I was looking for.

I went with 10 cases because there are so many different kinds of errors you can make in programming this. I thought that the more test cases there were, the more errors I would shake out of the code being tested. But because rule 5 was added at the last minute to cover an ambiguity in the rules, none of the test cases required the use of that rule.

Solution to DATA21.txt

```
Norm: Y.UP@7,3
Good: W.UP@6,2
Excl: O.RT@4,3
Game Over
Norm: W.RT@3,4
Norm: Y.UP@7,0
Good: W.RT@6,2
Excl: B.DN@6,2
Norm: O.LT@6,3
Excl: O.RT@6,2
```

Solution to DATA22.txt

```
Excl: P.DN@4,5
Excl: G.DN@2,1
Excl: P.RT@7,3
Excl: P.RT@4,0
Good: W.DN@3,6
Good: G.RT@6,4
Norm: R.LT@0,6
Norm: Y.DN@2,2
Norm: G.DN@2,4
Norm: G.RT@3,0
```

Problem 3: Steam Arithmetic

The name “Steam” is a spoof of “Scheme”, an actual dialect of Lisp that is currently in use and is taught as a first programming language at Waterloo and some other universities. You might think that programming this in a Lisp dialect would give an advantage, but because of the requirement that the program read the data from a file and interpret it, it is not necessarily so easy.

Recommended approach

Recursion is the best approach here. Write a method (or procedure or function) that takes a 3-element list, separates the operator and the operands and applies the operator to the operands. If an operand is a list, it should be recursively evaluated before being applied.

The restrictions on spacing and the restriction of operators and operands to a single character make it easy to process the list by pulling apart the string one character at a time. There is no need for any further tokenizing , except in the case of lists as operands.

To isolate list operands, you could use a bracket counting method. That is, start to the right of the first open bracket with the counter set to 1. Step through the string one character at a time. Increment the counter at an open bracket, and decrement it at a closed bracket. When the counter hits 0 you’ve found the end of the list.

The Test Cases

In the test cases, I tried to hit a couple of simple cases, then progressively harder ones. I tried for a few cases with very deep nesting, some with tail recursion, and some with head recursion (http://en.wikipedia.org/wiki/Tail_call)

Solution to DATA31.txt

```
1
0
-35
2
75
77
130
-19683
0
61
```

Solution to DATA32.txt

```
0
10
157
8
-20
160
6
-3456
1152
-9
```

Problem 4: Splitsville

This problem is inspired by the field of Machine Learning, specifically Concept Learning Systems. In a concept learning system, you have a list of training objects (e.g. satellite pictures of possible oil spills, data from medical scanners, etc.) with category labels (e.g. is/is not an oil spill, is/is not a tumor, etc.). Each object is described using a list of properties (usually numbers) and the task is to try to form a theory that separates the objects into their different categories. This theory can be viewed as a partitioning of a multi-dimensional space.

In this problem the training objects are houses, the labels are A and B, and the list of properties for each object are just the x and y location of the house. The "theory" is the set of fences that separate the A's from the B's. In machine learning, the theory you form should eventually do well at categorizing previously unseen objects, but that would only work in this case if A and B families tend to live close to other families of the same type.

There are a number of ways to partition spaces like this, including "nearest neighbour" classification and neural network classifiers. The algorithm used in this question is a simplification of machine learning algorithms that produce decision trees or decision rules as their output.

Recommended approach

This problem lends itself well to a recursive solution in which you partition a region of space into two parts, then recursively subdivide the two new regions created. The base case for the recursion is a "pure" region (i.e. a region with only A or B houses in it). Within each region the simplest strategy is to try all possible partitions and count the number of houses out of place in each one, where the number of houses out of place is just the sum of the minima of the counts of A and B houses on each side. So if the left side has 1 A and 3 B's while the right side has 10 A's and 2 B's, the total number of out of place houses is 3.

There are at least two ways to represent a region in this space. The first is to think of it as a 2-dimensional space bounded by integer coordinates above and below and on the left and right. To process it, you try all partitions half way between each pair of integer coordinates. (To keep life simpler and avoid floating point numbers, you could also double the original coordinates and then only consider splits on odd numbered coordinates.) This approach is not very efficient but it's fast enough for problems of this size. One potential pitfall is that you have to rule out splits that would not separate at least one house from the others.

The second, more efficient, method is to think of a region as just a list of houses. If there are 12 houses in the region, you only need to try at most 11 different cuts on each dimension, even if the houses are separated by thousands of units. To do this, you would have to sort the houses first, then find a split between each pair. Because of the sorting step, and the problems of creating and managing lists, this solution is more difficult to implement.

The Test Cases

The last test case in each set requires a few seconds of processing in Java when using the less efficient representation described above.

Solution to DATA41.txt

```
3
1
60
0
132
```

Solution to DATA42.txt

```
25
0
1
65
156
```