acm icpc

David Van Brackle
Chief Judge, Southeast USA Region, ICPC

2015 Southeast USA Regional Contest Problems

Airports

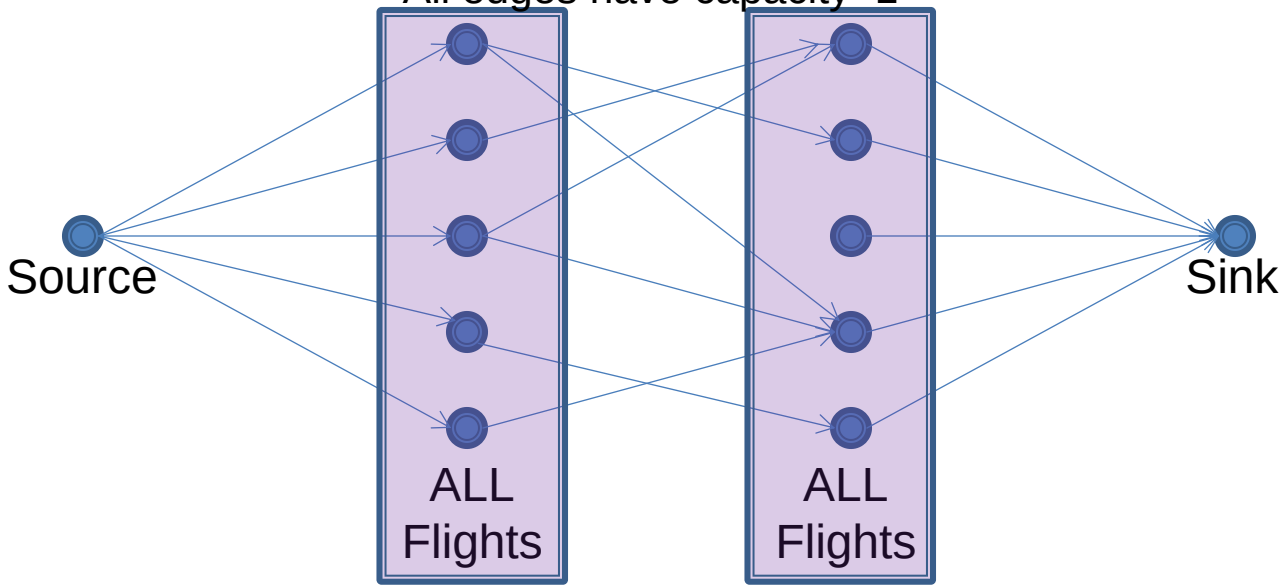
- Division 1 (2 correct/30 submissions)
- Given a list of scheduled flights, airports, turn times, etc., determine the minimum number of planes needed.

- Suppose you used a different plane for all `m` flights.
  - Then you'd need `m` planes.
- Supposed you were able to reuse a plane from flight **A** for flight **B**.
  - Then you'd need `m-1` planes
- Suppose there were `5` such instances
  - Then you'd need `m-5` planes
- So, to minimize the number of planes, you must maximize plane reuse.

Max Flow

The problem becomes an optimum pairing problem, solvable by Max Flow.

Link flights A and B iff the same plane can be used for A and B, and A≠B.
All edges have capacity=1



If the result of the Max Flow is **`x`**, then the final answer is **`m-x`**

Gotchas

acm icpc

- Note that the Triangle Inequality does NOT hold.
  - If might be quicker to fly A→C→B than A→B
  - You've got to do something like Floyd-Warshall to get all shortest paths
- However, if it's a scheduled flight, you must go directly.
  - So, you need TWO distance matrices: one for direct flights, one for shortest distances

Blur

acm icpc

- Division 2 (5/28)
- Given a 2D array of 1s and 0s, Blur it by averaging the 9 pixels around any given pixels
  - Wrap around the edges
- Do this many times
- Output: Number of unique colors needed

- Trick 1: use ‘mod’ to wrap around edges
- Trick 2: Use the sum, not the average!
  - This keeps the numbers as integers and avoids roundoff error

Checkers

- Division 1 (4/73)
- How many Black Kings can jump all of the White Kings?

- Set it up as a graph
  - Checkerboard squares are nodes
  - Jumps are undirected edges
  - Build a new graph for each Black King, using DFS or BFS, and only including possible jumps
- Is there an Euler path?
  - An Euler Path uses every edge in the graph exactly once

Euler Path

- Fleury's algorithm is very simple (but not terribly efficient):
  - From a node, choose any edge that isn't a *Bridge*
  - If all edges are *Bridge* edges, take any one
  - Traverse the chosen edge to the other node, and delete the edge
  - Continue until all edges are deleted, or until you're stuck. If all edges are deleted, then there is an Euler path
- How do you tell if an edge is a *Bridge*?
  - Do a DFS from the node, and count visited nodes
  - Delete the node (temporarily) and do another DFS
  - If the two counts are different, then the edge is a *Bridge*
- This isn't very efficient, but it's easy to code, and the numbers in this problem are small enough that it will run in time.

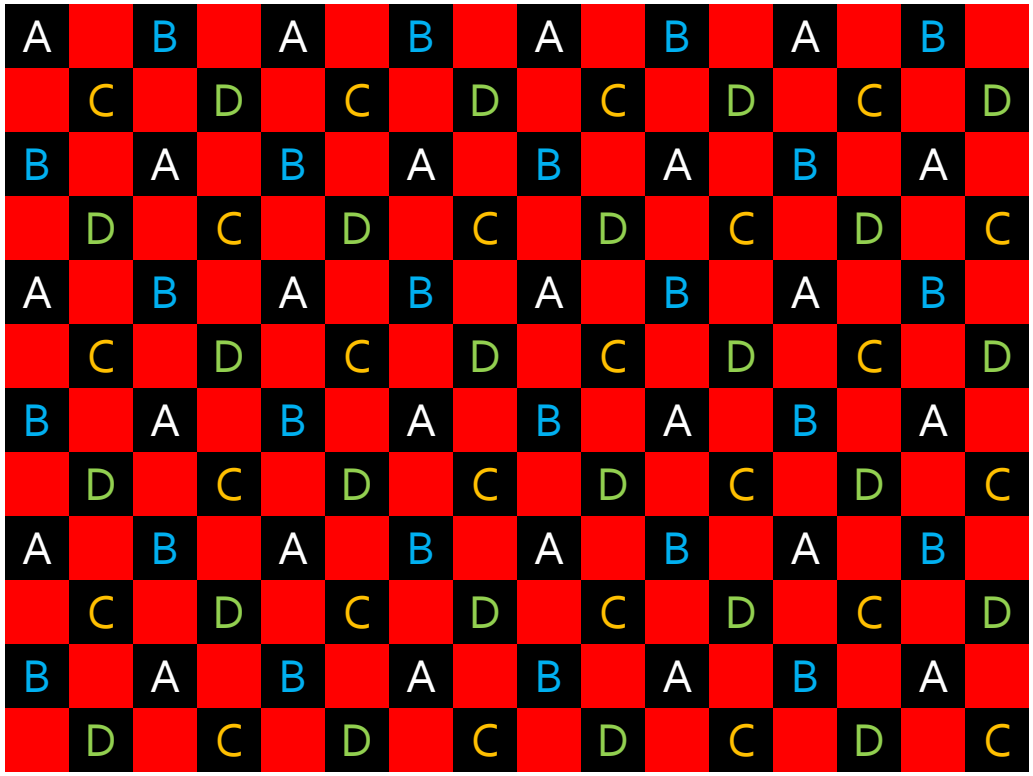
But Wait...

What if there are two different edges representing jumps over the same White King? Aren't you in danger of taking a jump over a King that is no longer there?

It's not a problem.

If jumps are your only move, then the black squares are partitioned into 4 groups:

It's impossible to jump from one group to another.



So, it's impossible for a Black King to jump a White King from two different angles.

A Classy Problem

acm icpc

- Division 2 (12/36)
- Sort people by their “class”
  - `queenelizabeth: upper upper class`
  - `mom: upper upper lower middle class`
- Parsing
- Create a “key” based on classes
  - In reverse order
  - Might not be the same length
    - Fill in missing with “Middle”
  - In the above example:
    - Let **a**=upper, **b**=middle, **c**=lower
    - **queenelizabeth** key might be **aabb**
    - **mom** key might be **bcaa**

Coverage

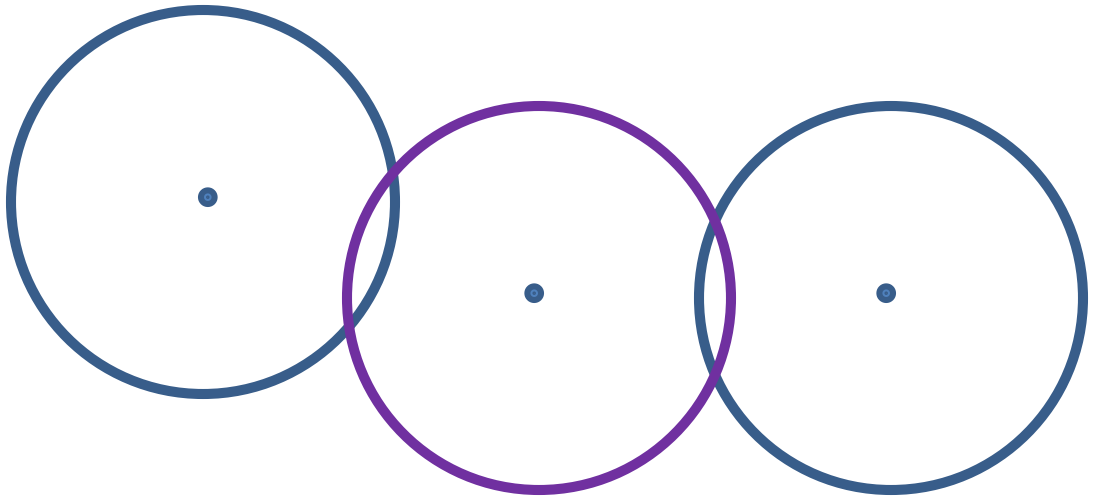
acm icpc

- Division 1 (2/9)
- Given a set of cell towers with 1km ranges, none within 1km of another, what's the largest connected group you can form by adding one tower?

- Firstly, Connected Components
  - Create a graph, with towers as nodes
  - Edge between if towers are within 2km
  - While you're at it, remember nodes that are within 4km
    - We'll use this later
    - Both of these lists will be small.

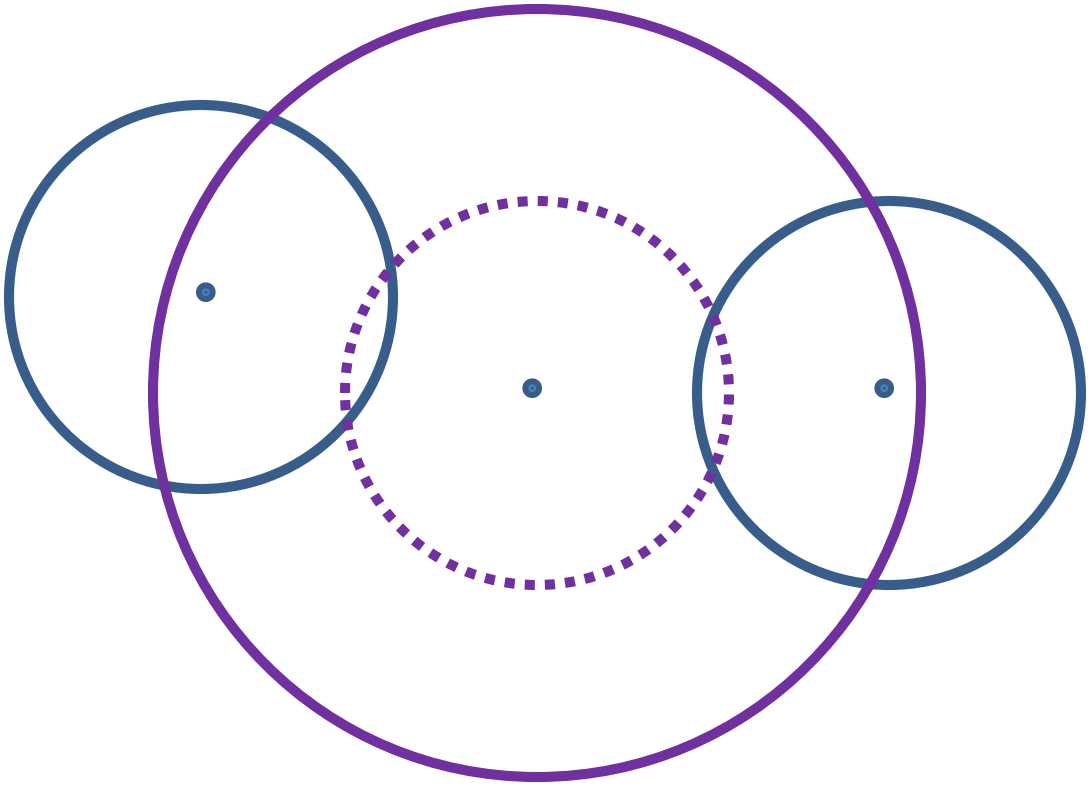
Observation #1

If a 1km circle connects two other 1km circles…

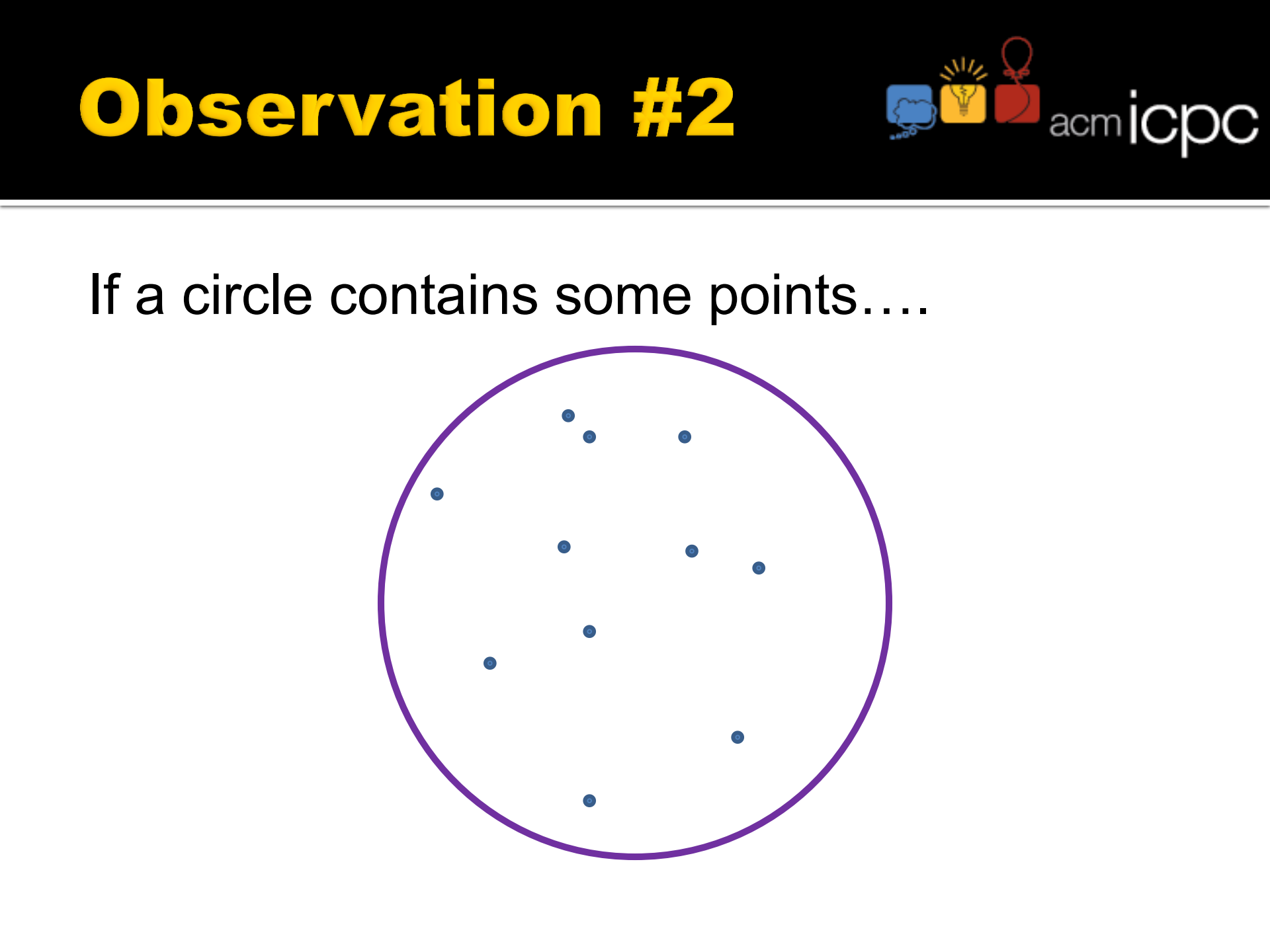


Observation #1

… Then their centers must me inside a circle with 2km radius



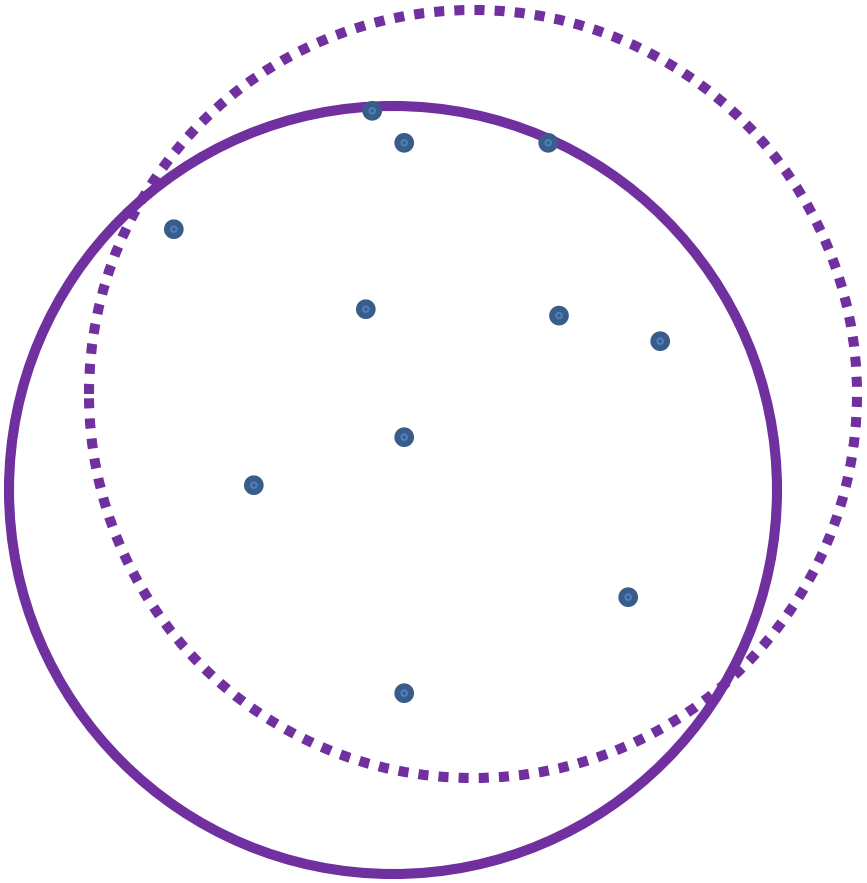
Observation #2



If a circle contains some points….

Observation #2

… Then there is another circle with the same radius, containing the same points

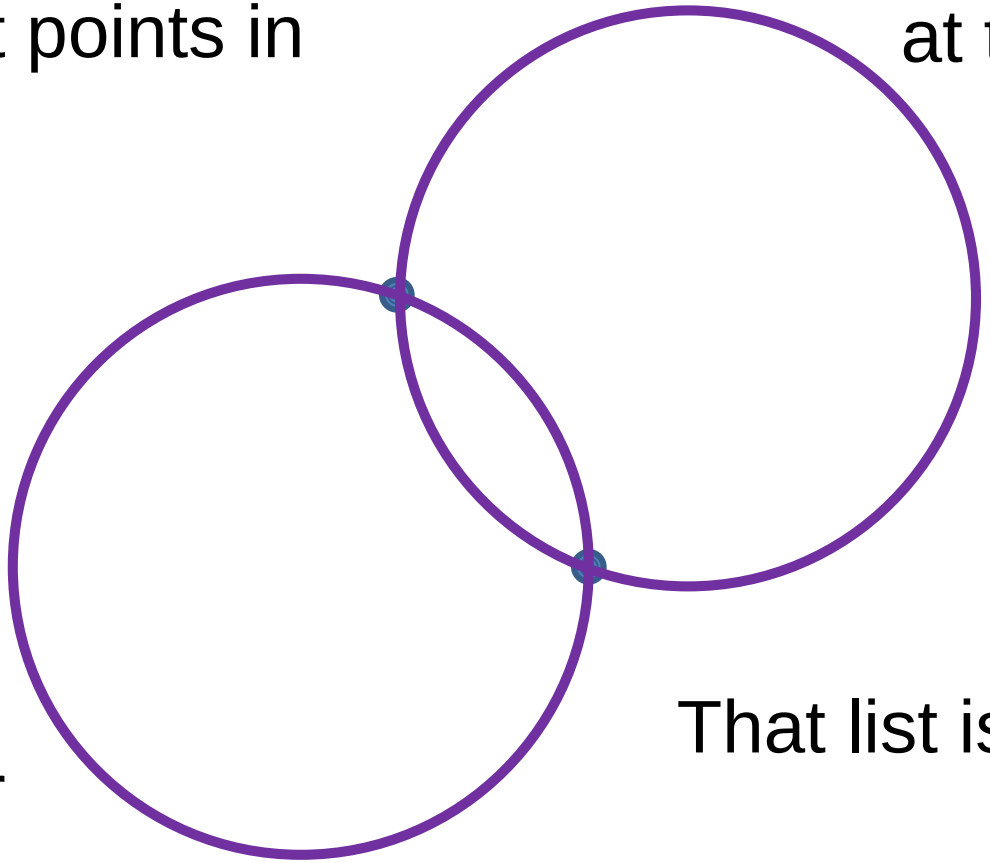


With at least 2 of the points on the edge of the circle

The Strategy

Pick pairs of points, form 2 2km circles, and in turn, count points in each.

We only need to consider points that are within 4km of each other



That's why we formed that list at the beginning

Remember, because the towers' ranges are non-intersecting,

That list is very small!

Egg Drop

- Division 2 (34/112)
- Given the results of dropping an egg off of a building at a certain floor (**`SAFE`** or **`BROKEN`**) determine:
  - The highest floor where the egg might be safe
  - The lowest floor where the egg might be broken
- We're guaranteed that it's safe on floor **`1`**, and that it will break on floor **`k`**
- So, the highest possible safe floor is the lowest recorded broken floor minus one
  - No broken floors? Then it's **`k-1`**
- Likewise, the lowest possible broken floor is the highest safe floor plus one
  - No safe floors? Then it's **`2`**

Excellence

acm icpc

- Division 2 (30/75)
- Given a list of an even number of scores, find the smallest number **x** where the list can be organized into pairs, and every pair's sum is ≥**x**

- Sort the numbers. Pair the smallest with the largest, next-smallest with next-largest, and so on.
- Take the smallest of all of those pair sums

A quick proof

- Let x be the best. The second best is **`x-a`**, **`a>0`**.
- Let y be the worst. The second worst is **`y+b`**, **`b>0`**.
- Here's the order:

```
x
x-a
y+b
y
```

- Our answer is **`min( x+y, (x-a)+(y+b) )`**.
- Suppose we paired them differently: **`x`** with **`y+b`**, **`y`** with **`x-a`**
- Then our answer would be **`min( x+y+b, x+y-a )`** = **`x+y-a`**
  - **`a>0`**, **`b>0`**, so **`b>-a`**
- But since **`a>0`** and **`b>0`**, **`x+y-a`** has to be worse than **`x+y`**, and it also has to be worse than **`x+y-a+b`**.
- Any ordering you can come up with can be obtained from the claimed optimal solution by a series of such swaps, each one making the answer worse.
- So, the claimed optimal solution is, indeed, optimal.

Gears

acm icpc

- Division 1 (23/28)
- Given a set of gears, determine the effect on a target gear of turning a source gear
  - The source gear cannot move
  - The source gear is not connected to the target gear
  - The source gear turns the target gear by some ratio

Solution

- Build a graph with gears as nodes, edges between if they're connected (i.e. they "touch")
  - $(x1-x2)^2 + (y1-y2)^2 = (radius1 + radius2)^2$
  - Don't take the square root – keep it in integers!
- Use a graph search to analyze the graph
  - Is the source gear connected to the target gear?
  - Keep track of Parity (1 or 0).
    - This corresponds to Clockwise vs CounterClockwise
    - If two gears are connected, then they must have opposite parity
  - If a gear has neighbors with opposite parities, then the source gear cannot turn.
- If they're connected. The ratio answer is just the ratio of their radii
  - Intervening gears do not matter!

Grid

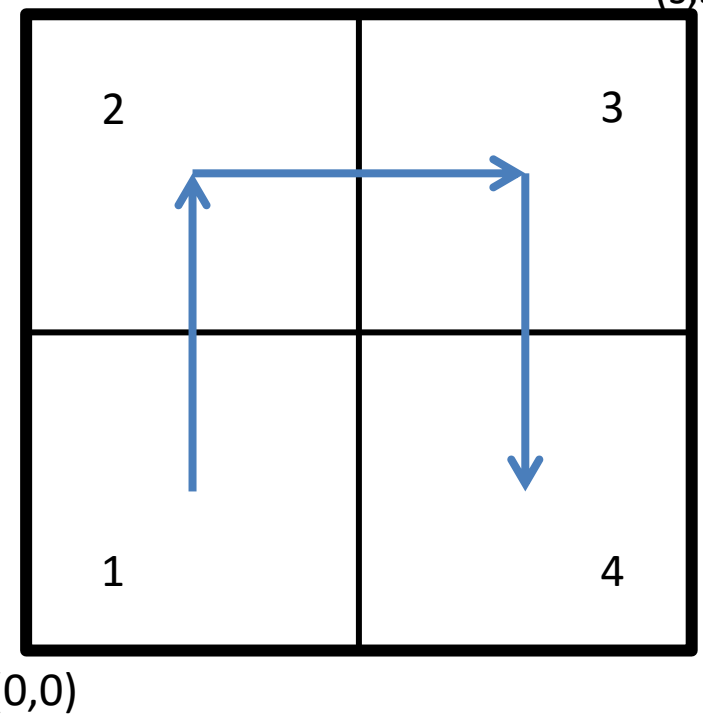
acm icpc

- Both Divisions (33/74,2/15)
- Given a grid of digits 1-9, each digit represents the number of squares you can move, up, down, left or right
- Find the length of the shortest path from the top-left to the bottom-right
  - No wrapping around the edges
- Just Breadth-First Search

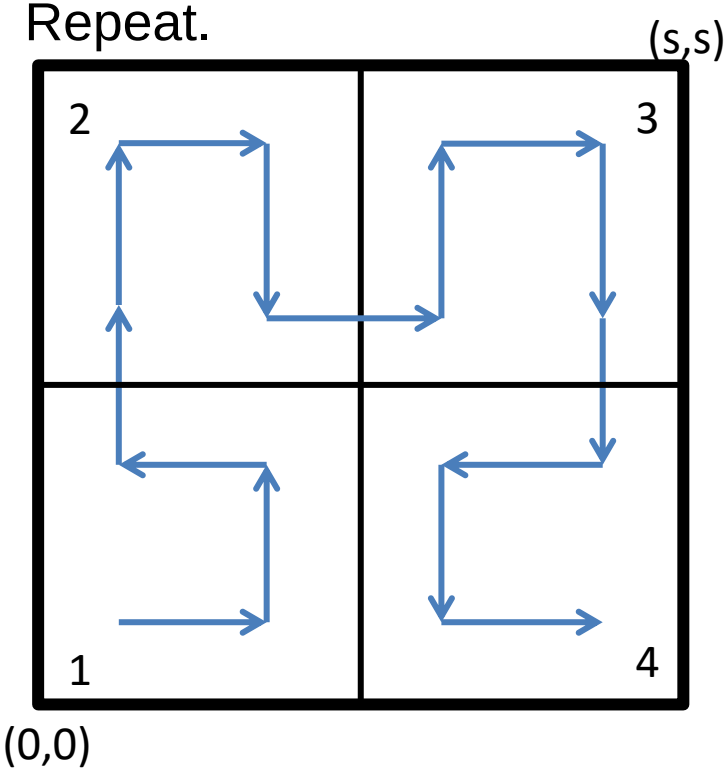
Hilbert Sort

- Both Divisions (7/19,0/0)
- Sort points by where they fall on a Hilbert Curve

- Start with a point in the center of a square.
- Duplicate that square in each of 4 quadrants (with some flipping), and connect them.



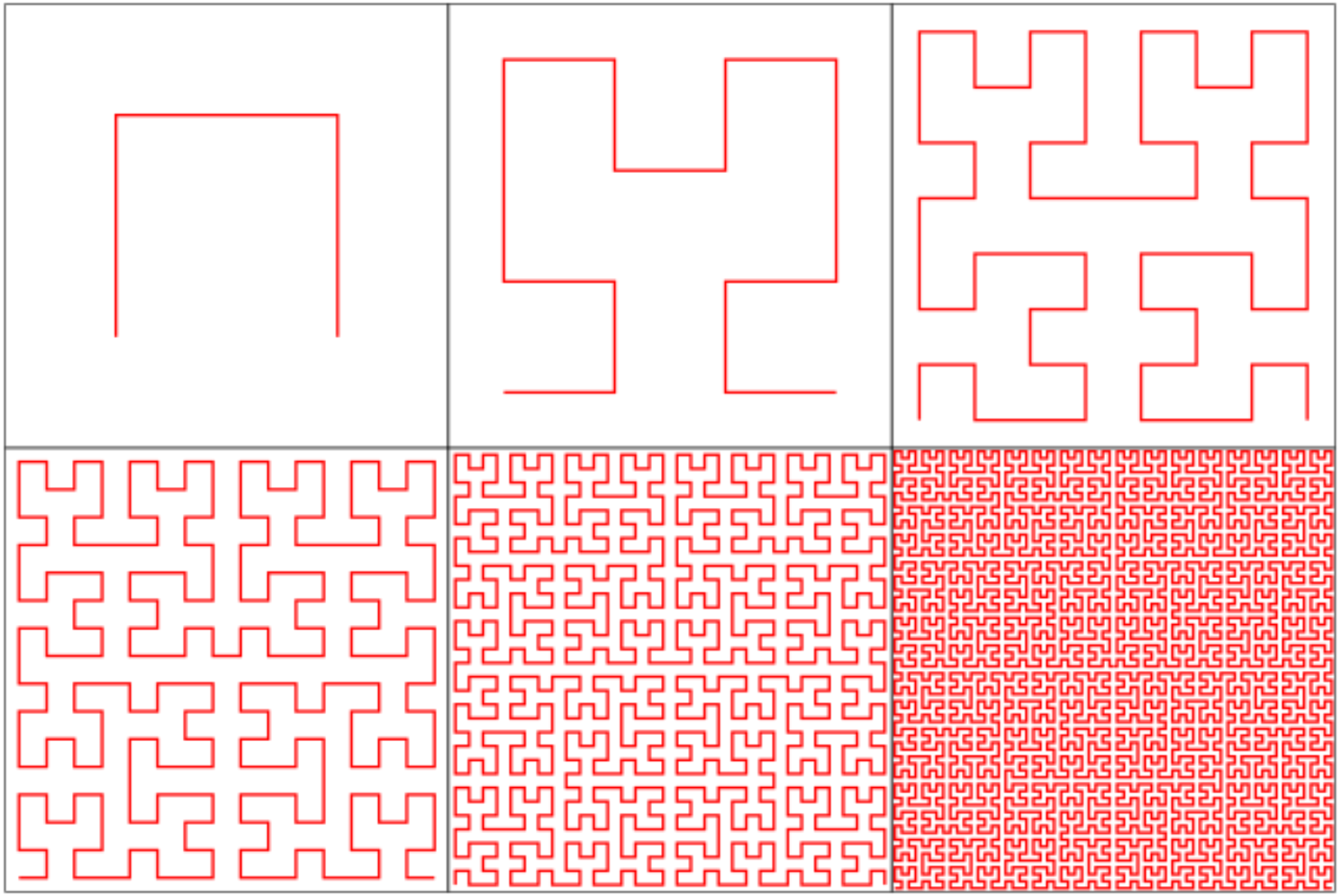
Repeat.



The Hilbert Curve

acm icpc

Repeat ad infinitum, and the curve fills the whole space.



The Trick

acm icpc

- The Hilbert Curve is complicated. But, notice that every point in Q1 comes before any point in Q2, Q2 before Q3, Q3 before Q4
  - So order the points by quadrant
- What if 2 points are in the same quadrant?
  - It's a fractal. Just repeat the process in the quadrant… and so on, and so on.
- Build a 'key' to sort on, based on quadrant, quadrant in quadrant, and so on.
- How deep should you go?
  - Well, they're between 1 and 10^9, so they're all less than 2^{30}.
  - Splitting in half 30 times should be sufficient whatever the data set.

The Devil is in the Details...

acm icpc

The translation, flipping and scaling can be tricky.

```
public String makekey( double x, double y, double s, int level )
{
    String key = "";
    if( level>0 )
    {
        --level;
        double s2 = s/2.0;
        if( x<=s2 && y<=s2 ) key = "a" + makekey( y, x, s2, level );
        else if( x<=s2 && y>s2 ) key = "b" + makekey( x, y-s2, s2, level );
        else if( x>s2 && y>s2 ) key = "c" + makekey( x-s2, y-s2, s2, level );
        else if( x>s2 && y<=s2 ) key = "d" + makekey( s2-y, s-x, s2, level );
    }
    return key;
}
```

The Magical 3

- Both divisions (21/152,2/101)
- Given a number **x**, find the smallest base in which the representation of **x** ends with a **3**

- If **x** ends with a **3** in base **b**, then **x-3** ends in **0** in base **b**, and is thus divisible by **b**.

- Then, we must find the smallest factor >3 of **x-3** (bases 2 and 3 don't have the digit 3)

- The trick: We can't just iterate from 4 to **x**-3. If **x**-3 is large and prime, then it will take too long.

The Trick

- Iterate up to $\sqrt{x-3}$
  - `for( int i=4; i*i<=x-3; i++ )`
- If you don't find anything, then **x**-3 must be a large prime, or a large prime times 2, or a large prime times 3.
  - So, the answer is the large prime
- This should be fast enough, but to make it even faster, the answer has to be 4, or 6, or 9, or a prime number. So, limit your search to 4, 6, 9, or primes.

Persistence

acm icpc

- Division 2 (53/59)
- Multiply a number's digits, keep going until the result is <10. How many steps does it take?

```
while( number >= 10 )
{
    int newnumber = 1;
    while( number>0 )
    {
        newnumber *= number%10;
        number /= 10;
    }
    number = newnumber;
    ++count;
}
ps.println( count );
```

Racing Gems

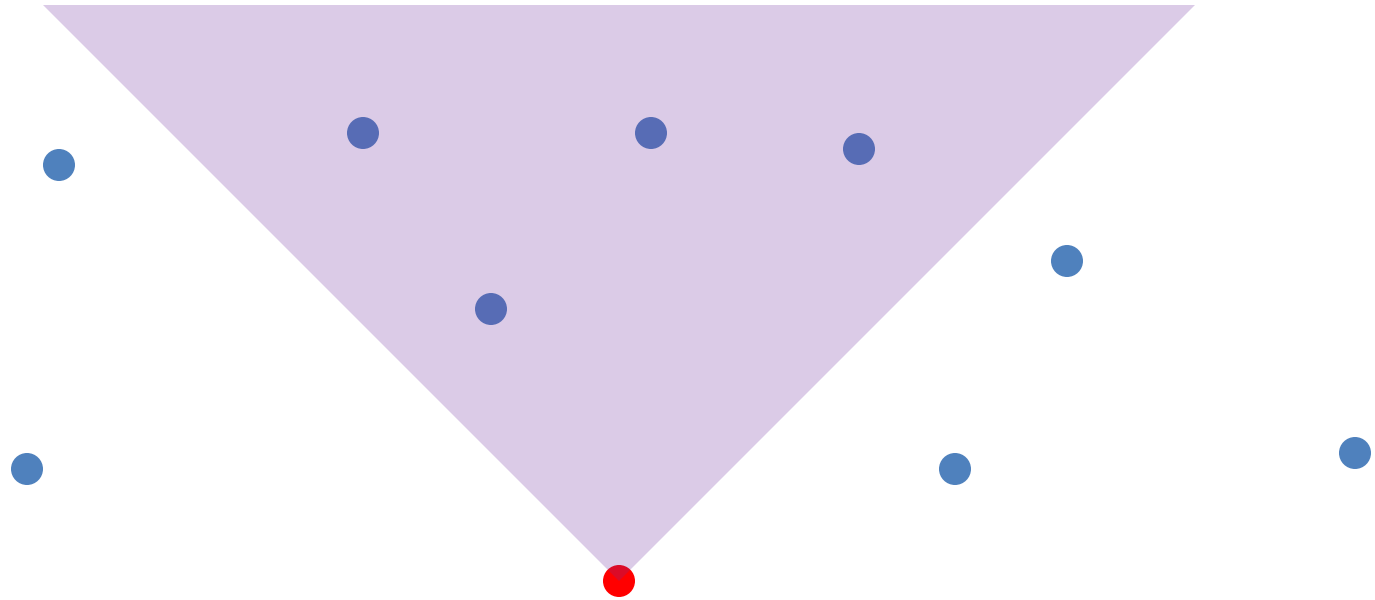
acm icpc

- Division 1 (4/16)
- You move up a grid at a fixed pace, with a limit to your horizontal speed. There are gems at points of the grid. How many gems can you collect?

- A bit of geometry here – but not as much as you might think.

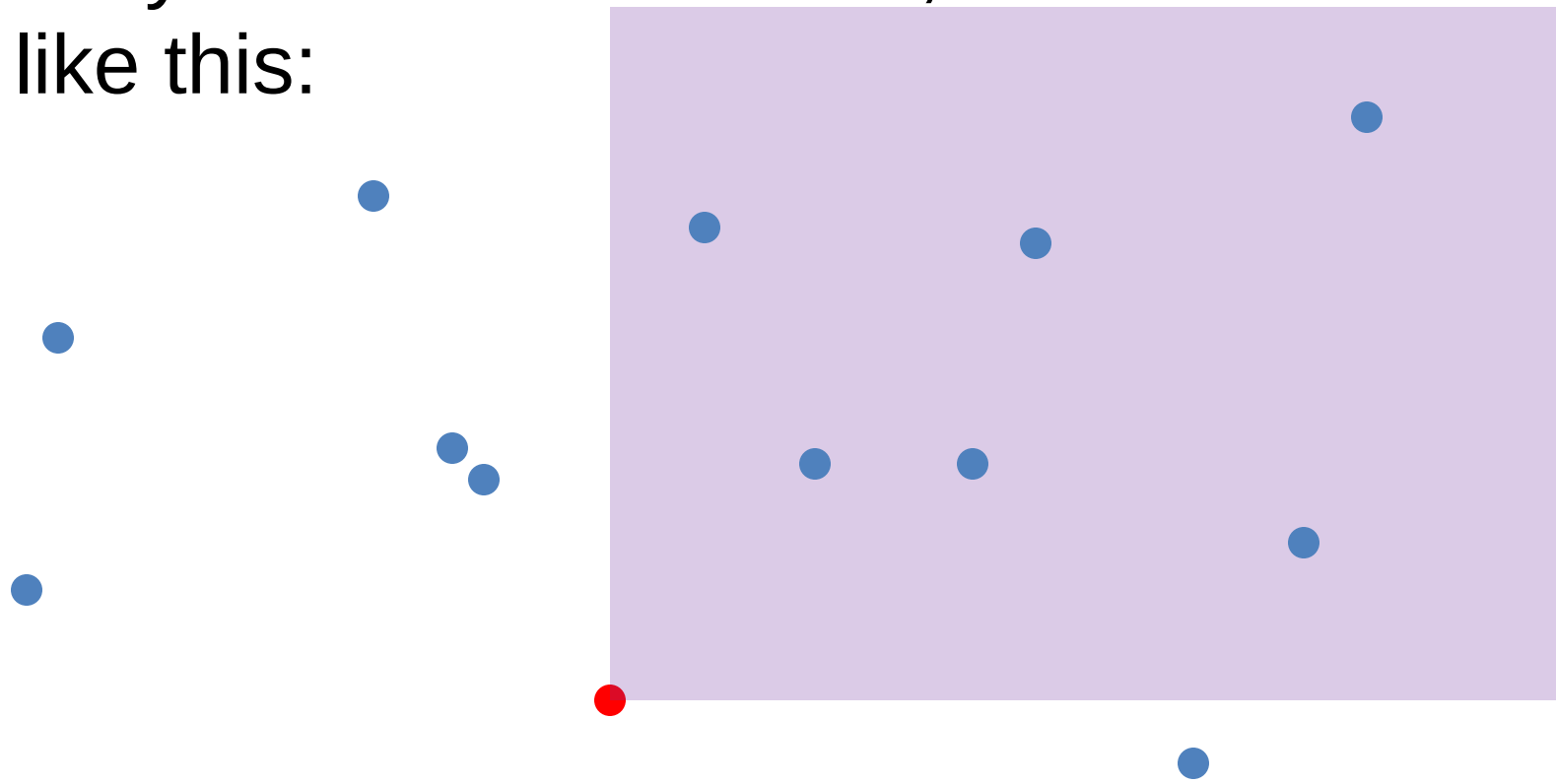
Geometry

For the sake of explanation, let **`r=1`**. Then, from any gem, the gems that you could possibly get next are in this realm:



Rotation

Transform by: `(y-x, y+x)`. This is essentially a 45º rotation. Now, that realm looks like this:



From here....

- Just sort by the **`x`** coordinate, and find the longest nondecreasing subsequence of **`y`** coordinates.
- There's that pesky rate **`r`**, but it's easily handled in the transform:
  - **`(y-r*x, y+r*x)`**
- The sort is O(n log n). What about the longest nondecreasing subsequence? Can we do that in O(n log n)?

Longest NonDec Subsequence

acm icpc

Let **a[i]** be the smallest element that can end a LNDS of length **i.**

For example, in this list:

8 9 12 10 11 1 2 3 7 5 6

a[4]=5

The **a[i]**'s have to be in sorted order. If **a[i]=x**, then there is a subsequence of size **i** that ends with **x**, which means that there's a subsequence of size **i-1** of all numbers <**x** that **x** is at the end of.

So, let **MAX** be the longest LNDS found so far. Go through the list in order. For each new value **x**, use a Binary Search to find where **x** should go in the list.

If **a[i]≤x<a[i+1]**, then **a[i+1]=x**
If **a[MAX]≤x**, then **a[++MAX]=x**

At the end, **MAX** is the length of the LNDS. You go through a list of size n, doing an O(log n) Binary Search, so the whole algorithm is O(n log n)

Simplicity

acm icpc

- Both Divisions (42/73,39/140)
- Given a string of lower-case letters, what is the smallest number of individual letters that must be removed so that the string has no more than 2 kinds of letters?
  - Example: **`ababcabxbaabbacaaa`**
  - Remove 3 (the 2 **`c`**'s and the **`x`**) and the string has all **`a`**'s and **`b`**'s

- Count the number of occurrences of each letter
- The answer is the sum of all but the two largest

Code


```
char letters[] = sc.next().trim().toCharArray();
int counts[] = new int[26];
Arrays.fill( counts, 0 );

for( char letter : letters ) ++counts[(int)(letter-'a')];

Arrays.sort( counts );

int sum = 0;
for( int i=0; i<24; i++ ) sum += counts[i];

ps.println( sum );
```

Triangles

- Division 2 (35/59)
- Given the 3 side lengths of 2 triangles, can the 2 triangles be put together to form a rectangle?

- 2 things to check:
  - Are they the same 3 side lengths (maybe in a different order)?
  - Do they form a right triangle? (use the Pythagorean theorem: $a^2 + b^2 = c^2$, where c is the longest side)

```
Arrays.sort( t1 );
Arrays.sort( t2 );
boolean ok = (t1[0]*t1[0] + t1[1]*t1[1] == t1[2]*t1[2]);
ok &= (t1[0]==t2[0] && t1[1]==t2[1] && t1[2]==t2[2]);
ps.println( ok ? "1" : "0" );
```

Weighlifting

acm icpc

- Division 1 (1/8)
- Given your energy, and the energy expended in successful and unsuccessful lifts, how close can you come to your (unknown) strength?

- Must chop the search space into as many equally-sized segments as possible

Simple Example

- If $e_{success} == e_{failure}$, then it's like binary search
  - With 1 lift, you can split the space into 2
  - With 2 lifts, if the 1st lift fails, then you can ignore the upper partition. If the 1st lift succeeds, you can ignore the lower partition. So, with 2 lifts you can split the space into 4 partitions.
  - With 3 lifts, you can partition the space into 8

The Real Deal

- If $e_{success}$ `!=` $e_{failure}$, it's a bit trickier. But, we can use a simple Dynamic Programming approach.

- Let `splits[i]` be the number of splits you can get with `i` energy remaining.

  `splits[i] = 1 + splits[i-`$e_{success}$`] + splits[i-`$e_{failure}$`]`

- Then, the number of partitions is just

  `p = splits[e]+1`

Gotchas

acm icpc

- Using recursion will run too deep. You need to iterate through the list.
- The number of splits can overflow pretty quickly. You need to find a largest feasible value.

```
public static final long INFINITY = 10000000000L;
for( int i=1; i<=e; i++ )
{
    splits[i] = Math.min(
        (i>=es?splits[i-es]:0) + (i>=ef?splits[i-ef]:0) + 1,
        INFINITY );
}
int partitions = splits[e]+1
```

One More Gotcha

acm icpc

- You have two strategies:
  - You can use all of your lifts to partition the search space [0,225]

  OR
  - You can save 1 lift to do 25, and use the remaining to partition the search space [25,225].

- Which is better?
  - Should you lose a lift to guarantee a score of at least 25, or should you use all of your lifts and risk a score of 0?
  - If you're trying to maximize your score, then saving a lift for 25 is probably better. But, that's not what the problem says. The problem is to minimize your error window.
  - You don't know a priori which one is best, so you have to account for them both.

```
Math.min( 200.0/(p-1), 225.0/p )
```

Xedni Drawkcab

- Division 2 (51/81)
- Given a list of words, reverse them, and print the reversals in alphabetical order

```
int n = sc.nextInt();
String words[] = new String[n];
for( int i=0; i<n; i++ )
{
    char word[] = sc.next().toCharArray();
    words[i] = "";
    for( char ch : word ) words[i] = "" + ch + words[i];
}
Arrays.sort( words );
for( String word : words ) ps.println( word );
```