INSCRIPTION 2014 EDITORIALS

Problem 1:
Lucy's dish

Problem Link:
https://www.hackerrank.com/contests/inscription2014/challenges/lucys-dish

Problem Setter:
Anirudh Raja

Solution:
This was the simplest problem of the set. Given an array find the element that repeats the maximum number of times. As all elements of the array are positive and less than or equal to 1000 we can use an array of size 1001 as a hash table and count occurrences of each element. In the hash table, hash[i] represents the number of occurrences of N. The only thing is that we need to ensure only one number occurs maximum number of times otherwise we need to print -1.

x-x-x-x-x-x-x-x-x-x-x—x-x-x-x-x-x-x—x-x-x-x

Problem 2:
The Dragon Warrior

Problem Link:
https://www.hackerrank.com/contests/inscription2014/challenges/the-dragon-warrior

Problem Setter:
Medhini

Solution:
The problem can be solved using a recursive logic. First of all, since before the first step Po can move to any of the possible locations, we should find the top score you can earn if we move to location (x,y) before the first move. We should try each location (x,y) as a candidate.
After the first move, Po can earn some calories depending on the location of first item. Then he can choose to move to a new point (x′,y′) which shares column or row (x,y). After this move, we are in a similar sub-problem to the initial one, you are located at position (x′,y′) and want to know the maximum calories you can make with the last Kâˆ’1 events. (Where K is the number of events).
The recurrence relation is: f(x,y,i) finds the maximum calories gained if Po is located at position (x,y) before event i.
If i=K, all items have been found and Po cannot earn any more calories. The result is 0. This is the base case. Else Po can earn some calories m equal to the Manhattan distance between (x,y) and the event's location. Finally he can move to a new point (x′,y′) and then the remaining events will happen and he might move again later. We can try to iterate through all reachable points (x′,y′) that share the column or row with (x,y) and pick the maximum f(x′,y′,i+1). The complexity of this approach is large. Imagine T was the largest dimension. There are O(T2) values for (x,y) , which means there are O(KT2) possible states for the function. In each state, we need an O(T) loop to pick the new row or the new column. In total we have an O(KT3) complexity. Note that T can be very large: T′=1000000.
In order to make the solution more efficient, we just need to notice that it is never necessary to move to a point (x,y) such that x or y don't match at least one of the items' coordinates. This means that there are only O(k) options we need to try for x and y in each step and there are O(k2) different
values. This changes the complexity to O(k4).

Finally, After computing the maximum calroies Po can earn compare it with the given Q value. If your answer is gretaer than or equal to Q then the final answer is 1, else 0.

x-x-x-x-x-x-x-x-x-x-x–x-x-x-x-x-x-x–x-x-x-x

Problem 3:
Equal Lightsabers

Problem Link:
https://www.hackerrank.com/contests/inscription2014/challenges/equal-lightsabers

Problem Setter:
Vadiraja K

Solution:
There are 3 thing needed to solve this problem.
Firstly we should know that sum of first n fibonacci numbers is F(n+2)-1. For eg n=3 1+1+2 = 4 = F(5)-1 = 5-1 = 4.
So to get 2 equal numbers F(n+2) must be odd.
Secondly we observe that fibonacci numbers go as odd,odd,even,odd,odd,even... as first 2 are odd (1 and 1) and then odd+odd is even so 3rd is even then even+odd is odd so next 2 are odd and this continues.
Lastly from Zeckendorf's theorem we know that all numbers can be represented by unique fibonacci numbers.
So our question reduces to count of Fibonacci numbers from a+2 to b+2 that are odd.

x-x-x-x-x-x-x-x-x-x-x–x-x-x-x-x-x-x–x-x-x-x

Problem 4:
Lucy's Big Family

Problem Link:
https://www.hackerrank.com/contests/inscription2014/challenges/lucybigfamily

Problem Setter:
Raghuram

Solution:
Utilises persistent Heavy Light Decomposition

x-x-x-x-x-x-x-x-x-x-x–x-x-x-x-x-x-x–x-x-x-x

Problem 5:
Save Middle Earth

Problem Link:
https://www.hackerrank.com/contests/inscription2014/challenges/save-middle-earth

Problem Setter:
Anirudh GP

Solution
Simple Ad-hoc.

x-x-x-x-x-x-x-x-x-x-x—x-x-x-x-x-x-x—x-x-x-x

Problem 6:
Tyrion's Strategy

Problem Link:
https://www.hackerrank.com/contests/inscription2014/challenges/tyrions-strategy

Problem Setter:
Ajith PS

Solution:
Easy Convex Hull

x-x-x-x-x-x-x-x-x-x-x—x-x-x-x-x-x-x—x-x-x-x

Problem 7:
NTB and Kunjoorings

Problem Link:
https://www.hackerrank.com/contests/inscription2014/challenges/ntb-and-kunjoorings

Problem Setter:
Suraj Rajan

Solution:
The problem basically boils down to finding total number of kunjoorings and subtracting it with the number of kunjoorings matching the given order. Total number of kunjoorings can be found out by (n x (n+1))/2 where n is the length of the Asteroid Straight Line Phenomenon. Then number of matching kunjoorings can be found out by using some suffix structure such as an automaton or an array. For the automaton, concatenate the main order and pass the query through it. For the array, use binary search. This number should be subtracted from the total number. The results should then be summed up to give the final answer.

x-x-x-x-x-x-x-x-x-x-x—x-x-x-x-x-x-x—x-x-x-x

Problem 8:
Hunt the Challenge!

Problem Link:
https://www.hackerrank.com/contests/inscription2014/challenges/hunt-the-challenge

Problem Setter:
VK Aadheshwar

Solution:
Pick's Theorem
Area = # Inner Integer Points + # Border Integer Points / 2 - 1

To calculate the Area, we can use cross product.
So the remain part is to get the number of integer points on its border. That equals to count how many integer points on the segment (0,0) - (dx, dy). This can be solved by Greatest Common Divisor (gcd).
Integer Points on Segment = gcd(dx, dy) + 1
which including the two ends.
Combining these together and applying the Pick's Theorem, we finally get the total integer points inside or on the border n.
If total challenges t is more than n then n on campus challenges, t-n off campus challenges, else only on campus challenges. Now solve the on campus and off campus parts of the problem separately.
For off campus challenges (if any) just add them all and that's the min time required (t2).
In the on campus challenges, because we can skip only strictly lesser values , greedy approach will not work out.
Example: If on campus times are: 6 5 4 3 we can go greedy to pick 6 and 4 (i.e sort and pick alternate times).
But if: 6 5 4 4 then we must pick 6 and 5. So dp is the way to approach…
First, process the items into <#value, #number of items with that value> tuples. Then, sort the tuples in descending order of #value.
Iterate over those tuples and let # freed be an empty multi set()
For each tuple <#val, #num>, we can calculate #max_skiptime, the maximum number of items (not value) that we can get for free up to this point easily.
So, we want to populate #freed so that it contains exactly #max_skiptime.

x-x-x-x-x-x-x-x-x-x-x—x-x-x-x-x-x-x—x-x-x-x

Problem 9:
Lucy and Sushi

Problem Link:
https://www.hackerrank.com/contests/inscription2014/challenges/lucy-and-sushi

Problem Setter:
Ashish Kedia

Solution:
Basically one has to evaluate a polynomial with matrix argument over finite field.
Now consider, the case when X is just a variable (not a matrix), then F can be evaluated using Horner's rule. Horner's rule require L (length of polynomial) multiplication, which is an optimal solution.
However, when X is a matrix, multiplication is a costly affair. As such we need to minimize the number of matrix multiplication. We can achieve this by using Paterson Stockmeyer Method which requires only L^(0.5) multiplications. The trick is to store all powers ox X from 0 to sqrt(L).
A nice explanation of Horner's Method can be found here :
http://en.wikipedia.org/wiki/Horner%27s_method

The original paper for the Paterson-Stockmeyer can be found here:
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.218.4987

The algorithm can be further optimized by using strassen's algorithm but this optimization will yield results only if the size of matrix is much larger. The solution will pass with O(n^3) matrix multiplication

x-x-x-x-x-x-x-x-x-x-x—x-x-x-x-x-x-x—x-x-x-x

Problem 10:
Fury's Network

Problem Link:
https://www.hackerrank.com/contests/inscription2014/challenges/furys-network

Problem Setter:
Adarsh Mohata

Solution:
The problem requires knowledge of heavy light decomposition and segment tree with lazy propagation.
Let us first introduce Heavy-light decomposition briefly. For each vertex x which has at least a son, find the maximum size of the subtree of some son y of x, in case of a tie choosing any one vertex available is fine, we call y is the heavy son of x and we call p is the light son of x if p is not equal to y. That is, there is only one heavy son of x and the others are light sons. The edge between a vertex and its heavy son is called heavy edge, The edge between a vertex and its light son is called light edge.Consider the sequence of vertex {a[0],a[1],a[2],...,a[m-1]}, for every i>0, a[i-1] is the father of a[i]. We call the sequence of vertex a heavy chain iff for every i>0, a[i] is the heavy son of a[i-1] and any edge connected to some vertex in the sequence is light edge. For more detailed explanation on heavy light decomposition, visit http://blog.anudeep2011.com/heavy-light-decomposition/
Now we can have a segment tree for storing the cost values. When we have a segment tree, it's time to make it persistent and to build the heavy-light decomposition of the tree. If we store all the chains of the HLD in the single segment tree, there is no more effort required to maintain the versions of the persistence correctly. As the segment tree has been constructed, we can get any information of every interval [L,R] online in O(log n) time.
Each node of the segment tree will contain :
sum[L,R] : The sum of costs of all links between vertices L and R.
sqsum[L,R] : The sum of squares of costs of all links between vertices L and R.
cubsum[L,R] : The sum of cubes of costs of all links between vertices L and R.
value : The value to be currently added to this node (required for lazy propagation)
Now, when there is an update operation, we have to update the corresponding nodes and push up the updates.
If a value x is to be added to a node [L,R] then,
cubsum[L,R] = cubsum[L,R] + (R-L) * x * x * x + 3 * sqsum[L,R] * x + 3 * sum[L,R] * x * x
(Hint: Expansion of (a+b)^3)
sqsum[L,R] = sqsum[L,R] + (R-L) * x * x + 2 * x * sum[L,R]
(Hint: Expansion of (a+b)^2}
sum[L,R] = sum[L,R] + (R-L) * x
Note: The order of update should be the same as above since the previous sqsum and sum values will effect the cubsum.
Whenever there is a query operation, we can sum up all the cubsum of the corresponding nodes in the segment tree after pushing down the pending updates and return the result.

x-x-x-x-x-x-x-x-x-x-x—x-x-x-x-x-x-x—x-x-x-x

Problem 11:
Lucy's Hunger

Problem Link:
https://www.hackerrank.com/contests/inscription2014/challenges/lucys-hunger

Problem Setter:
Tushar Makkar

Solution:
The problem can be solved using Mobius Inversion . For more details check http://en.wikipedia.org/wiki/M%C3%B6bius_inversion_formula and the pdf attached with the email.
Basically problem is to find sum of lcm(a,b) over all pairs such that there is no integer n>1 such that n^2 divides both a and b .We want to compute sigma i=1 to a (sigma j=1 to b (m*n*F(gcd(m,n)))) .
Using F(n) = 1/t if t is square free else 0 .
Now applying mobius inversion we can convert it into a form where only primes multiplicative terms remains . These terms can be found out by using wheel sieve and segmented sieve .

x-x-x-x-x-x-x-x-x-x-x–x-x-x-x-x-x-x–x-x-x-x

Problem 12:
The War Begins

Problem Link:
https://www.hackerrank.com/contests/inscription2014/challenges/the-war-begins

Problem Setter:
Anirudh Raja

Solution:
The weights of the maximum and minimum spanning trees gives the answer for the problem.

x-x-x-x-x-x-x-x-x-x-x–x-x-x-x-x-x-x–x-x-x-x

Problem 13:
Good Subarray

Problem Link:
https://www.hackerrank.com/contests/inscription2014/challenges/sub-array-gcd

Problem Setter:
Chaithanya B.S.

Solution:
Lets denote d[1..n] as an array such that d[i] holds the size of the largest good subarray with A[i] as its last element.
The result d[i]<=d[i-1]+1 can be deduced using proof by contradiction.
also this good subarray ending at A[i] cannot extend in left beyond index j (0 <=j < i) if gcd(A[i],A[j]) =1 . Let the maximum possible j be x.
Using the above two results d[i] can be calculated as d[i]=min(d[i-1]+1 , i-x)
The answer is the maximum value in the array d[1..n].

x-x-x-x-x-x-x-x-x-x-x–x-x-x-x-x-x-x–x-x-x-x