Venkat and Random Numbers

The problem reduces to finding sum of n/i over all i from 1 to n.
Observation :- Number of distinct values of n/i are at most 2*sqrt(n).
Proof :- let i<=sqrt(n) then number of values that can be got are sqrt(n) and for i > sqrt(n) the value of n/i will be at most sqrt(n) leading to at most 2*sqrt(n) values in total.
It can be implemented by iterating over all the distinct values of n/i as it is easy to see that values of i that have same value of n/i form a continuous range.
Complexity :- O(sqrt(n))

Code

L and Convex Polygons

Observation :- There is an optimal solution with only triangles.
The solution is dynamic programming over subsets. For any subset, pick a point, iterate over pairs of all other points, answer for this subset is minimum of current value, area of the triangle + answer for smaller subset obtained by removing these three points.
Time complexity is 2^15. 15^2. If we did not make that triangle observation and try all subsets, the complexity becomes 3^15, which will not run in time.
Govind Lahoti's and Ram Prakash's teams are the only two to solve this question.
Kudos to them :)

Code

Shindou Hikaru and his three friends

Observation :- Any number that satisfies the properties mentioned would be divisible by 11.
It is very easy to see from the divisibility rule of 11. So the only prime number is 11 that satisfies all the properties
Many people got WA as they did not realize that 11 is a prime number.
Complexity :- O(1)
Fun fact - We too did not realise that 11 is a prime number until Shantanu pointed out during testing :)

Code

Emperor Lelouch vi Britannia

Observation :- The time complexity for recursive solution is a1 + a1.a2 + .. + a1.a2..an.
Many got Time Limit Exceeded because they submitted this recursive solution.
There are lot of 1's. Just remove them. Now, there are at most logN numbers. The previous solution will work. Another way is to reverse sort the input and use the previous recursive solution.

Bonus :- Can you come up with a non-recursive solution?
Fun fact :- We thought of making stack limit small thus allowing only non-recursive solutions, but did not do that thinking that it might too difficult.
Code

Saitama and Pairs of Strings

Observation :- If string s is a prefix of string t, it's useless to keep s in the subset. It's better to keep t instead of s.
There are several solutions for this .

1. For every string, check if it's prefix of any other string. Take all those strings which are not prefix of other strings. Make sure you don't remove a string simply because it occurs twice.
2. Greedily start with a list of all strings. Iterate over all pairs and find s and t such that s is prefix of t, remove s. Stop when you cannot find such pairs.
3. Sort the strings by length. Start from largest one, keep picking strings as long as it is not prefix of any string you have.

Complexity :- O(n^3) there are better complexities as stated above

Code

Sandeep and Lost Matrix

Observation :- On multiplying the matrices we get
x1 + x2 * 1 + x3 * 1^2 + … xn * 1^(n - 1) = y[1]
x1 + x2 * 2 + x3 * 2^2 + … xn * 2^(n - 1) = y[2]
.
.
.
x1 + x2 * n + x3 * n^2 + … xn * n^(n - 1) = y[n]
So if we see x1, x2, …., xn are the coefficients of a (n-1)-degree polynomial (say f) for which f(1) = y[1], f(2) = y[2], ….., f(n) = y[n].
So we have n points through which the polynomial passes. So we can find x1, x2, …, xn using any standard methods like lagrange interpolation or newton method in O(n*n).
I will explain the further solution using lagrange interpolation.
We can precompute all the denominators of lagrange interpolation in O(n*n).
Then for every point we can compute the value of (p[i] - 1) * (p[i] - 2) * ….. (p[i] - n).
Then we can calculate the value of each polynomial in lagrange (li(x)) in O(1) time by precomputing all the inverses of numbers from 1 to 10^6. Thus making an overall O(n) time for each point p[i]. So the overall complexity is O(n*n).

Code
Code

We did not expect any team to solve F, but we expected more teams to solve B.