Tryouts Breakdown

UF Programming Team

Feedback

- How was the problem set?
- Were the questions too hard? Too easy?
- Any issues?
- Other

Final Results

- 11 problems
- 104 accepted solutions
- 503 total submissions

n	Number of people who solve *n* problems
1	8
2	10
3	7
4	7
5	3
6	2

Final Results

Congratulations to the top 5!

1) Nick Jiang (6 solves)
2) Daniel Gollahon (6 solves)
3) Andrew Sack (5 solves)
4) Sahir Boghani (5 solves)
5) Collin Irwin (5 solves)

We are still unsure of how many we are taking to SER

➔ we will know by Thursday's practice

Download the Problem Set

goo.gl/2i5tF1

Problem Set Breakdown

In order of solves...

B. Polling

E. The *n* Days of Christmas

K. Profits

J. Palindrometer

H. Balloons

F. Vampire Numbers

Problems that didn't get solved...

A. You Win!

C. Gold Leaf

D. Stained Carpet

G. Word Ladder

I. Sunday Drive

B. Polling

Condensed problem statement

Given the votes of *n* people for an election with a candidate's name, print out the winner of the election. If there are multiple candidates with the same number of winning votes, print them out alphabetically

B. Polling

Solution

Create a *HashMap* that maps a candidate's name to the number of votes they have received. Also create a variable, *mostVotes*, that is used to keep track of the most votes you have seen a candidate have.

After reading in all of the input, iterate over each key (candidate's name) in the *HashMap*, checking to see if the value (number of votes) was equal to *mostVotes*, and adding the candidate's name to a *List* if it was. Sort the *List*, then print out the names.

Results

35 accepted solutions out of 93 submissions (37%)

E. The *n* Days of Christmas

Condensed problem statement

On the first day of Christmas, you are given a *partridge in a pear tree* (you have one gift).

On the second day of Christmas, you are given *two turtle doves and a partridge in a pear tree* (you get three gifts today, and have one from the previous day, so four total).

On the third day of Christmas, you are given *three french hens, two turtle doves, and a partridge in a pear tree* (you get six gifts today, and have four from the previous days combined, so you have 10 total)

How many gifts will you have accumulated on the *n*th day?

E. The *n* Days of Christmas

Solution

Let's write out the math...

First day: **1**

Second day: **2 + 1 (+ 1) = 4**

Third day: **3 + 2 + 1 (+ 2 + 1 + 1) = 3 + 2 + 1 (+ 4) = 10**

Fourth day: **4 + 3 + 2 + 1 (+ 3 + 2 + 1 + 2 + 1 + 1) = 4 + 3 + 2 + 1 (+ 10) = 20**

E. The *n* Days of Christmas

For any day, *D*, we are adding the previous day's sum to 1 + 2 + 3 + ... + (*D* - 1) + *D*

So, the amount of gifts you have on day *D* is

$$\sum_{1}^{1} n + \sum_{1}^{2} n + \sum_{1}^{3} n + \ldots + \sum_{1}^{D-1} n + \sum_{1}^{D} n$$

Which can be calculated using two for-loops!

... but because *D* can be up to 1,000,000, D^2 can be up to 1,000,000,000,000

THAT IS TOO BIG

E. The *n* Days of Christmas

There is a *nice* formula for the sum of numbers between 1 and *n*

$$1 + 2 + 3 + \ldots + (n - 1) + n = \frac{1}{2} * n * (n + 1)$$

We will be using this formula to solve the problem!

E. The *n* Days of Christmas

Precompute the number of gifts we will have for any day *D* so we don't have to iterate over large numbers every time.

```
long[] gifts = new long[1000001]; // use long!!!
gifts[1] = 1;
for (int i = 2; i <= 1000000; i++) {
    gifts[i] = gifts[i-1] + ((long)i + 1) * (long)i / 2;
}
```

Now, we can read in the input, and easily give the answer by calling `gifts[n]`

Results

25 accepted solutions out of 239 submissions (10%)

K. Profits

Condensed problem statement

You are given *N* integers, which represent the amount of profit made each day. Negative values on any day mean a loss of money.

Determine the greatest amount of profit made over any range of days

Example: -3, 4, 9, -2, -5, 8

From days 2 to 6, a profit of 14 was made (4 + 9 + (-2) + (-5) + 8)

K. Profits

Solution

One way to determine the max profit, is to pick two days, *i* and *j*, and find the profit made between those two days.

```
for (int i = 1; i <= N; i++) {
    for (int j = i; j <= N; j++) {
        // iterate from i to j and determine profit
    }
}
```

However, if *N* can be up to 250,000, this is going to exceed the time limit

K. Profits

It turns out we only need to iterate over the list of profits ***just once***.

We need two variables, *maxProfit* and *currentSum*, while iterating over the profits.

maxProfit will keep track of the maximum profit we have made up to a certain day while iterating.

currentSum will represent the sum we have accumulated up to the current day. If the value of *currentSum* ever becomes negative, **we have to reset it to zero**.

Why do we reset it to zero?

K. Profits

```
int maxProfit = Integer.MIN_VALUE;
int currentSum = 0;

for (int i = 0; i < N; i++) {
    currentSum += in.nextInt();
    maxProfit = Math.max(maxProfit, currentSum);
    if (currentSum < 0) currentSum = 0;
}
```

Results

21 accepted solutions out of 68 submissions (30%)

J. Palindrometer

Condensed problem statement

Given an odometer (2 to 9 digits long), determine how many more miles you will need to drive before the odometer is a palindrome

Example

10000 → **10001** (1 mile away)

000121→ **001100** (979 miles away)

00456 → **00500** (44 miles away)

J. Palindrometer

Solution

Let us take the first half of the odometer and reflect it to form a palindrome.

10000 → 10001

000121 → 000000

00456 → 00400

2931 → 2992

If this new number is bigger than our current odometer, we have found our target palindrome, so we just need to return `target - currentOdometer`

J. Palindrometer

However, if the target is *smaller* than our current odometer, we need to create a new target odometer.

We can create this by taking the first half of the current odometer again, but this time we add 1 to it, and form a palindrome. This will guarantee us a target odometer that is greater than our current odometer.

00456 → 00400 (smaller)

Take the first half and add 1 (**004 → 005**) and form a palindrome

005 → 00500 is now bigger than **00456**

J. Palindrometer

000121 → 00000 (smaller)

000 + 1 → 001 (add 1 to the first half of the odometer)

001 → 001100 (form a new palindrome)

001100 is greater than **000121**

Results

16 accepted solutions out of 69 submissions (23%)

F. Vampire Numbers

Condensed problem statement

A *vampire number* is a number that has a pair of factors, *a* and *b*, that contain the same digits as the vampire number itself. Given a number *n*, find the smallest vampire number that is greater than or equal to *n*.

Example

126 = 6 * 21

10251 = 51 * 201

702189 = 9 * 78021

29632 = 32 * 926

F. Vampire Numbers

Solution

One way of solving this problem is to precompute all vampire numbers and hard-code them into your program, that way you avoid all computation needed to check for a vampire number while iterating over the numbers greater than *n*.

F. Vampire Numbers

Another way [*probably the desired way*] that you can solve this problem is to keep checking numbers greater than or equal to *n* to see if it is a vampire number.

```
public static String signature( String s ) {
    char[] a = s.toCharArray();
    Arrays.sort( a );
    return new String( a );
}
```

This method will take a number and turn it into a string whose digits are sorted. This will make comparing numbers to see if they contain the same digits easier.

F. Vampire Numbers

```
public boolean isVampire( int x ) {
    String vsig = signature( "" + x );
    for (int i = 2; i*i <= x; i++) {
        if (x % i == 0) {
            String isig = signature( "" + i + "" + (x / i));
            if (isig.equals(vsig)) {
                return true;
            }
        }
    }
    return false;
}
```

Results

4 accepted solutions out of 19 submissions (21%)

Other problems

- **You Win!**
  - bitmask dynamic programming
- **Gold Leaf**
  - brute force
- **Stained Carpet**
  - ternary search
- **Word Ladder**
  - breadth-first search
- **Balloons**
  - greedy
- **Sunday Drive**
  - dynamic programming

Solve!

All problems can be found on SPOJ:

www.spoj.com/problems/UFPT2015X

Replace '**X**' with the problem code you want to solve.