SOLUTIONS FOR 25TH OF JULY

ABSURD TRIANGLE 3

```
puts(".");
for(int i = 1; i < n; i++) {
  putchar('|');
  for(int j = 0; j < i-1; j++) {
    putchar((i < n-1) ? ' ' : '_');
  }
  putchar('\\');
  puts("");
}
```

BINARY SYSTEM TO OCTAL SYSTEM

If you used **long long** or **int** type variables to read the input, then **you get wrong** regardless of your algorithm. The statement says $n < 2^{1000}$, so **inputs with 1000 digits can be given!** Please, before submitting your solution to the server, try to check whether your solution works with the **upper bound of the input constraint!**

So first, read n as a string (char[] array or an STL string) Then, add some leading zeroes to the front of the given number, to make the length a multiple of 3. So for example, convert "1011001011" into "001011001011". After that, split this number into groups of 3 digits: "001/011/001/011". Now, convert each group into a octal system: "1313". We can use this algorithm because $8 = 2^3$, 3 digits in the binary system represents a digit in the octal system.

COUNTING DIVISORS

If you try to iterate all integers from 1 to n, and try to count numbers that are divisors of n, you'll probably get time limit exceeded. $n \leq 10^9$, so this $O(n)$ algorithm won't work. (refer to the 1e8 rule)

So, we can use a way that we proposed on the $O(\sqrt{n})$ primality test. Suppose we can decompose $n = a \cdot b$ such that $a \leq b$. Then, $a \leq \sqrt{n}$ must holds, because if $a > \sqrt{n}$, $a \cdot b > \sqrt{n} \cdot \sqrt{n} = n$. Therefore we can iterate all a from 1 to $\lfloor\sqrt{n}\rfloor$, and count the divisors like:

- If a is not a divisor of n, ignore.
- Otherwise, compute $b = \frac{n}{a}$. If $a = b$, add 1 to the answer. If $a \neq b$, add 2 to the answer. (since we found two distinct divisors a and b)

This algorithm has time complexity $O(\sqrt{n})$ and can be accepted comfortably.

Distributing Cookies

```
int ans = 0;
for (int i = 1; i <= n; i++) {
  int p;
  scanf("%d", &p);
  ans += (p / k);
}
printf("%d\n", ans);
```

Note that integer division automatically floors the result.

Elephants Wearing Hats

We will not reveal the solution, as still nobody solved this problem. Anyway we will upload the solution before the solution starts (or right after the solution session ends)

Instead we give some hints: This problem doesn't require sorting. This problem can be solved by case analysis, but the cases are not that much.

Fancy Dancing

The problem statement says that if k increases, the corresponding T decreases. Therefore we can use a binary search to find the minimum k such that $T(k) \leq l$. Now, we just need to focus on how to calculate $T(k)$ for fixed k.

Actually, it is easy to do. We just need to simulate the process given in the problem statement:

- First, add first k horses. Make an array $P[1..k]$ that stores the time each horse stops dancing.
- For each remaining horse $\#i$:
  - First, find the minimum element among $P[1..k]$ in $O(k)$ time. Suppose the element is $P[j]$.
  - Add d_i to $P[j]$, which means there is a new horse that starts dancing at time $P[j]$, and dances for durating d_i (so the new horse ends dancing at time $P[j] + d_i$).
- After considering all horses, we can calculate $T(k)$ by maximum of $P[1..k]$.

This straightforward algorithm is enough, since it takes $O\big(k(n-k)\big)$ time and $k(n-k) \leq \frac{n^2}{4} \leq 25{,}000{,}000$. Although we repeat this algorithm $O(\log n)$ times (due to binary search), the value of $k(n-k)$ becomes smaller, so this solution was enough to pass. (this is not actually a proof, but this is what we got)

However, our intended algorithm was not like this. We will add that solution if we have time..

Gigantic Addition

Do as the problem statement says. The main problem is just on implementation. Some tips on implementation:

- Reverse the input strings. Then you can add numbers from the *front* of the strings (without aligning the numbers to the right), so it will be convenient.

- Initially fill the array with spaces, and print the char[] array.