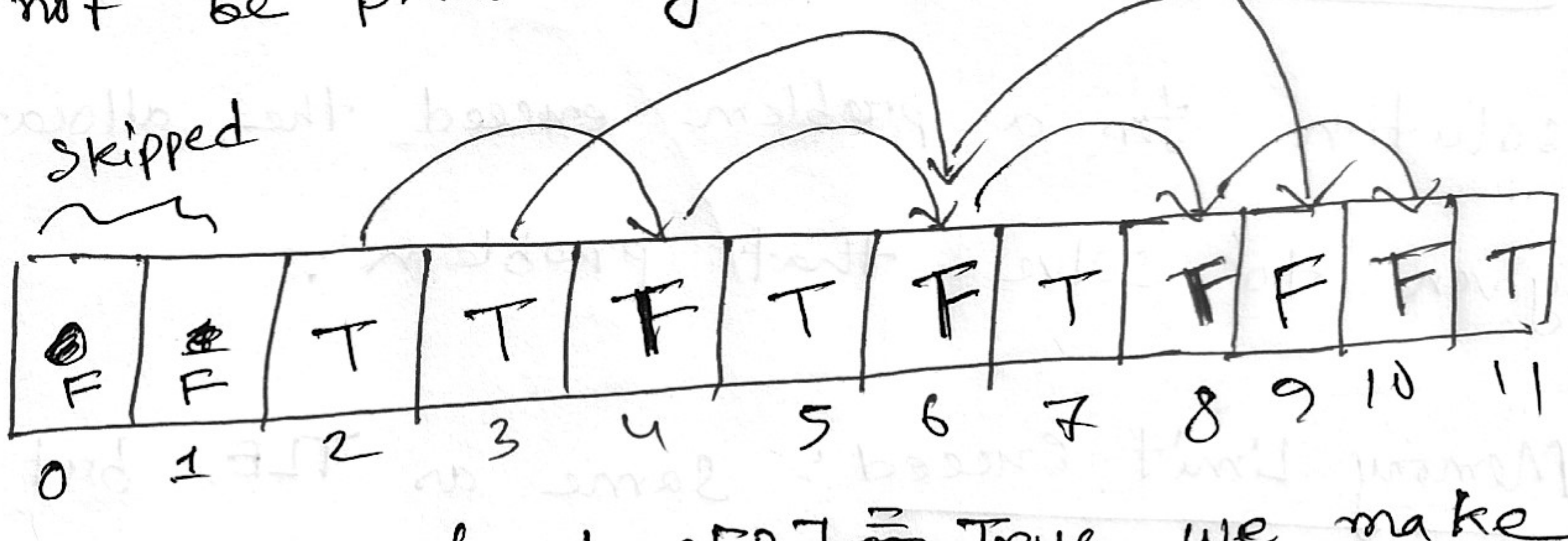
SHADAB IQBAL
ID: 19101072

Ans. to Q No-3

'Sieve of Eratosthenes' is an algorithm to find the prime numbers ~~from~~ between a certain range. The idea is simple. We take a boolean array of size n. Then we start iterating from 1. If the index is ~~false~~ True we mark that index as a prime number and we make all the multiples of that numbers "false", because those can not be prime anymore.



Here, when we found a[2] ~~==~~ True, we make the multiple indexes i.e {4, 6, 8, 10} = False, for, a[3] = True, {6, 9} = False.

This is how this algorithm works,

Ans to the Q No – 7

Online Judge: Platform where there are problemsets given and people ~~can~~ try to solve those problems and verify their answers through OJ.

Problemset: A set of problems which can be Solved ~~doing~~ writing some lines of codes.

Ranklist: During a programming competition, this ranklist keeps track of who has solved how many problems within what amount of time.

Time Limit Exceeded: This happens when a users' solution for a problem exceed the allocated time given to solve that problem.

Memory Limit Exceed: Same as TLE, but this time, it exceeds the allocated memory.

Test data: The inputs for which the user's solution will be test against.

Corner case: Some tricky test cases which has higher probability of invalidating the user's solution.

ICPC: International Collegiate programming contest. (World cup of competitive programming).

Ans to Q No - 5

Yes, theoretically time complexity of ternary search is faster than binary search. But this is not always the case in practical coding. When ternary search is applied on a huge dataset, it does a significant number of comparisons which ~~overtak~~ overthrows the time saved by reduced number of iterations. Theoretically, we ignore the constants but the constants in ternary search is relatively larger than binary search which causes poor time complexity when ternary search is applied on large dataset.

Ans to Q No-2

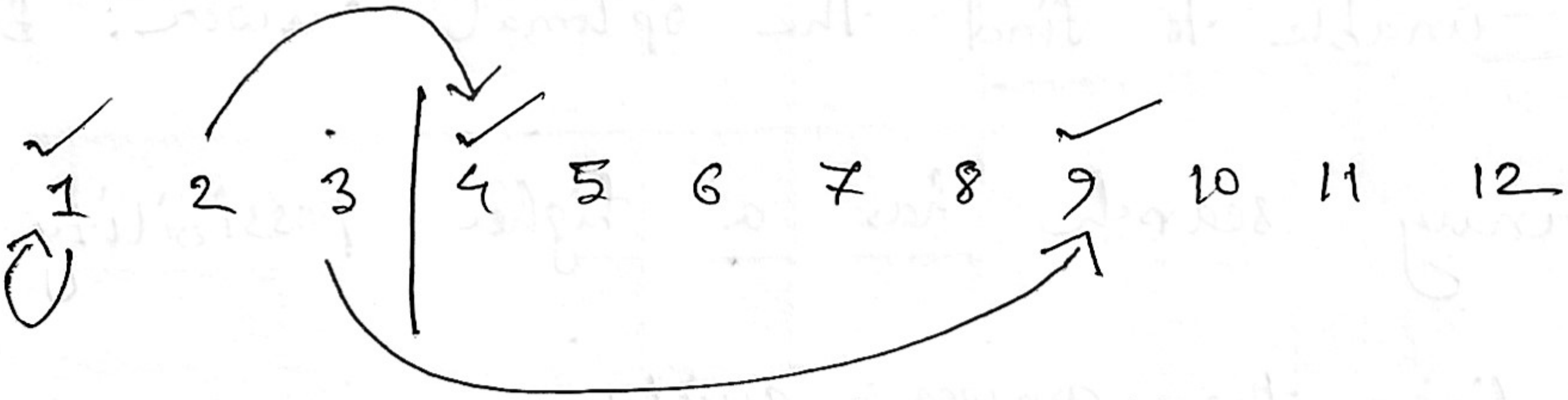
& ~~count~~

The answer will be Floor (sqrt (n)).

Reason: Only those numbers ~~are~~ have odd number of divisors, which are perfect squares. And, between (1 – n) range, how many perfect squares will we get ?

We will get ⌊sqrt(n)⌋ perfect squares.

As, the question told us to only find the count of beautiful numbers, time complexity of my presented solution is 1.



This is a simulation of how it works,

Ans to Q No - 6

```
string str;   count = 0;

cin >> str;

for (int i= str.size()-1;  i>=0;  --i) {
    if (str[i] == '1') {
        break;
    }
    ++ count;
}

answer = count.
```

This is a linear approach. It works because to solve this problem, we basically have to count the number of consecutive zeros at the right of given integer.

Example:

$$101\overset{1}{0} \% 2^{1} = 0$$

$$100\,1\overbrace{0000}^{4} \% 2^{4} = 0$$

$$101\overset{0}{1} \% 2^{0} = 0$$

So, we can find the answer just by counting number of zeros.

Ans. to Q No-1

Time complexity of the given GCD algo is $O(\log(\min(a,b)))$

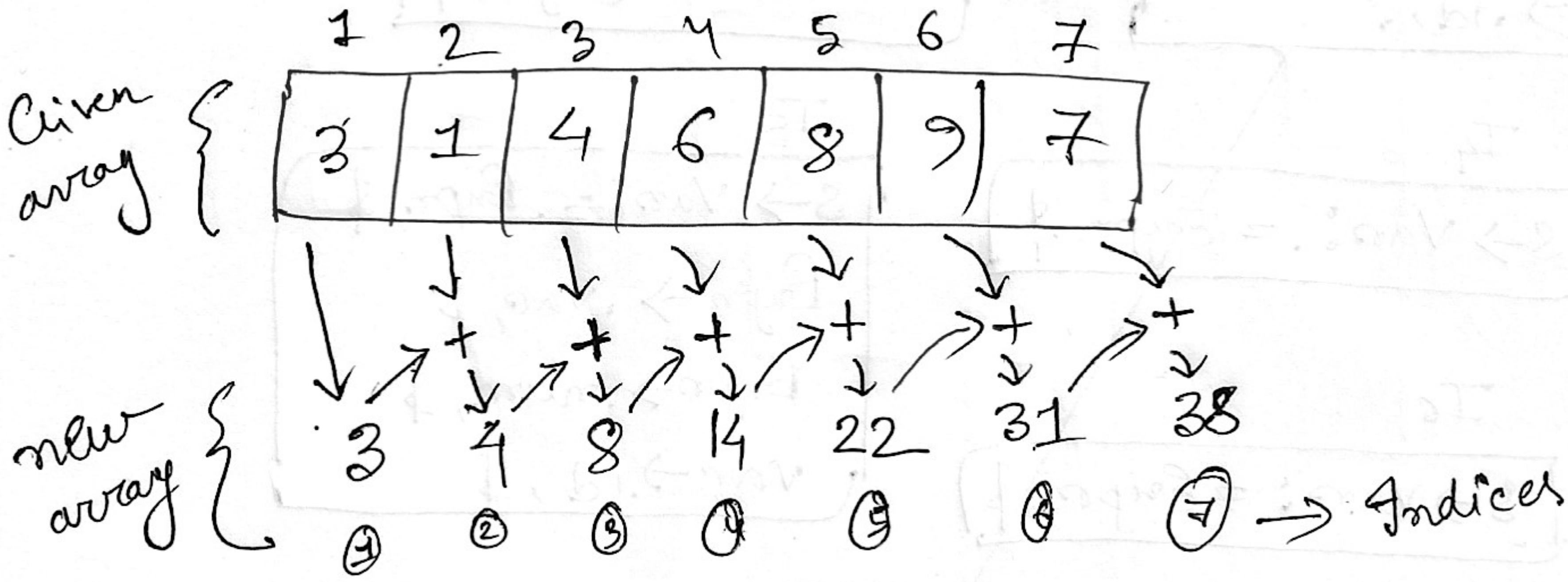
Here, as we are doing modulus in every iteration, the number of iterations is at most linear in the number of digits in the minimum number

So, we can infer that

no. of iterations before a number turns to 0 = at most logarithmic in the smaller input numbers.

Ans to Q No - 4

We make a new ~~array~~ array with the sums.



If query = 1,3 | ans = new_array [3]

" query = 2,6 | ans = new array [6] − new_array [1]

" query = 4,7 | ans = new_array [7] − new_array [3]

This solution has a time complexity of o(N).