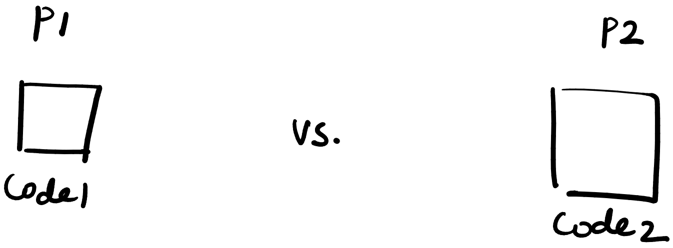
Recap: Abstract code (template) $\longrightarrow$ Variations

PS: question

input $\rightarrow$ $\boxed{\text{code}}$ $\rightarrow$ output

You don't just want to be correct, you want to be fast as well



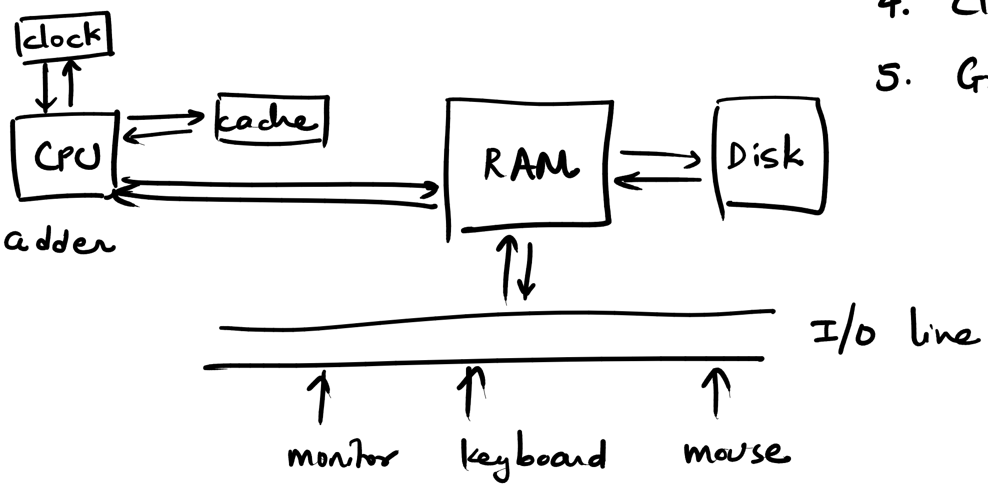
better?

1. Correctness
2. Speed

Laptop $\longrightarrow$ what specs you look for?

1. Processor / CPU chip
2. RAM
3. SSD over HDD
4. Clock
5. Graphics card

Bottlenecks.



i7 $\longrightarrow$ 2.8 GHz

1.8 GHz $\longrightarrow$ $1.8 \times 10^9 \; \frac{1}{\text{sec}}$ fundamental ops.

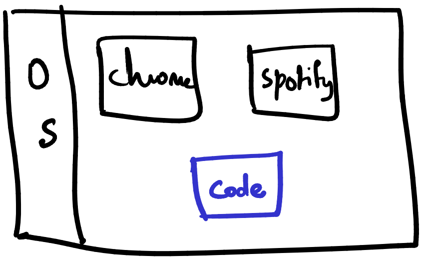
1. OS, chrome, connectivity, RAM $\rightleftharpoons$ I/O + HDD drivers Task manager

Safe assumption: My code gets 10^8 ops / second.

Bottleneck

10^8 numbers $\longrightarrow$ add them all RAM

↳ 1 second!



10^{10} numbers $\longrightarrow$ add them $\longrightarrow$ 100 s!

Time Complexity:

$\boxed{\text{code}}$ $\xrightarrow{\text{associate it with}}$ a mathematical class.

$y(n)$: ops you are going to perform to get o/p with input of size n

$y(n) = n$

$y(n) = n^2$

```
for (int i=0 ; i<n ; i++)
    for (int j=i+1; j<n ; j++)
        print (arr[i] , arr[j])
```

checks

Increments

arr[n] = { . . . }

Generate all possible pairs!

[1, 3, 4, 2, 5, 7]
↑i ↑j   n=6

(1,3) (3,4) (4,2) (2,5) (5,7)
(1,4) (3,2) (4,5) (2,7)
(1,2) (3,5) (4,7)
(1,5) (3,7)
(1,7)

$$\sum_{i=1}^{n} = \frac{n(n+1)}{2}$$

count
print $= (n-1) + (n-2) + \dots + 2 + 1$

$= 1 + 2 + \dots + (n-2) + (n-1)$

$$\#\text{print} = \frac{(n-1)n}{2}$$

$$y(n) = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

$\boxed{\text{code}} \longrightarrow y(n)$ Just for print

1) Taking input → ×
2) Checks → ×
3) Increments → ×

Exact $y(n)$ is very complex. class.

$\boxed{\text{Code1}} \longrightarrow O(n^2) \Rightarrow$ my code is quadratic !!

$\boxed{\text{Code2}} \longrightarrow O(n) \Rightarrow$ my code is linear !!

Which code is better ?

Code2 ✓

But $O(n^2) \rightarrow \frac{n^2}{2} - \frac{n}{2}$ $\quad O(n) = 10000n$

Line : Given sufficiently large input, quadratic will always be worse than linear.

$\boxed{n = 20,001}$

Real world appl^n are dealing with large n.

Rule : No code should take more than 1 s.

Thumb rule:

$f(n) < 10^8$

$\downarrow$

$n_{max} \uparrow\uparrow$

Input size (n) n_{max}	$\longleftarrow$	class of f^n / T.C.
10		$n!$
25		2^n
500-1000		n^3
10^4		n^2
10^6		$n\log_2 n$
10^8		n
10^{12}		$\sqrt{n}$ / $\log_2(n)$

RHS is first.

Read: If my code is of class $O(n^2)$, then the max input size I can handle is 10^4.

Trick: ① With time, you will know enough codes with their corresponding TCs.

Sorting $\rightarrow O(n\log n)$

Searching $\rightarrow O(\log n)$

Rotate $\rightarrow O(n)$

② In problems, under constraints part, max value of n is always given.

Eg. Constraints : $1 \leq n \leq 10^5$ $\longrightarrow$ n^2 is not going to work. $n \log n$ maybe

$\downarrow$

Sorting ??

Journey:

1. Correct sol^n

Problem $\rightarrow$ $\boxed{\text{Code}}$ $\xrightarrow[\text{Optimize}]{}$ $\boxed{\text{code}}$ TC$\downarrow$ faster!!!

Q. am I doing something redundant ?

Eliminate all redundant work!!

Amazing Problem : Bulb Toggle Problem

PS : n bulbs. Each bulb ⇒ on / off
1 0 ← representation

2nd bulb is OFF

bulbs = [1 0 0 1 0 1 1 1 0 .... 1 0 1 1]

1st bulb is ON

my last bulb is ON.

index : 0 1 2 3 4 5 6 7 8 ..... n-1

q queries / tasks : (l_i, r_i) ⇒ position l to position r ⇒ Toggle!

Eg. 3 tasks : (0, 5) (3, 8) (4, 9) | max interval (0, n-1)

n bulbs. bulbs = [n — elements —— , 0,1]

q queries : l_1 r_1 ⟹ For each range, from l_i to r_i go and toggle the bulbs.
l_2 r_2
l_3 r_3
⋮
l_q r_q

l_i and r_i are inclusive.

o/p : After finishing all q queries/tasks, the number of bulbs which are ON!

Code 1: Agenda : Correct code $\longrightarrow$ Just do as they say!

bulbs = [ ] , $\overset{100}{n}$, q , queries = $[(l_1, r_1), (l_2, r_2) \ldots (l_q, r_q)]$

(start, end) $\Rightarrow$ (17, 53)

```
for ( int i = 0 ;   i < q ;   i++)
    l, r = queries [i]
    for (int j = l ;   j <= r ;  j++)
        bulbs [j] = 1 - bulbs [j]
```

T.C

q . Effort to toggle

$O(qn)$

sum (bulbs) $\Rightarrow$ $O(n)$ Total = $O(qn + n)$

q, n are inputs. $f(q, n) \Rightarrow$ $\boxed{TC \sim O(qn)}$

Code 2: Optimize this! $\Rightarrow$ Redundant work. (X)

What it is ? $\Rightarrow$ Many bulbs you toggled that produced no effect.

Eg. 10 bulbs.

3 (l_1, r_1)
$(l_2\ r_2)$
$(l_3\ r_3)$

$\left.\begin{array}{lll} 0-4 & \Rightarrow & 5 \text{ bulbs} \\ 5-9 & \Rightarrow & 5 \text{ bulbs} \\ 0-9 & \Rightarrow & 10 \text{ bulbs} \end{array}\right\}$ Work : 20 bulbs

End effect is nothing!

All bulbs are in same state as initially.

$\left.\begin{array}{l} 0-4 \\ 0-9 \end{array}\right\} \Rightarrow$ Means toggle only 5 later bulbs.

Track of all l_i r_i $\Rightarrow$ At last I will toggle bulbs. 10

bulbs :

1	0	1	1	0	0	0	1	0	0
0	1	2	3	4	5	6	7	8	9

5 tasks $q = 5$

11 $\leftarrow n+1$

diff :

+1	+1	+1	+1	0	0	0	-1	-2	0	-1
0	1	2	3	4	5	6	7	8	9	10

l

(2,6), (3,7), (7,9) (1,6) (0,7)

1 1 x 1 1

diff[l] = +1

diff[r+1] = −1

prefix sum / cumulative freq.

4

always be zero

prefix:

1	2	3	4	4	4	4	3	1	1	$\boxed{0}$
0	1	2	3	4	$\boxed{5}$	6	7	8	9	10

prefix[i] = prefix[i-1] + diff[i]

5

prefix[i] tells me how many times i^{th} bulb is toggled !!

if prefix[i] is odd :

only then toggle the bulb !

Optimized :

bulbs = [], n, q, queues = [(l_1, r_1), (l_2, r_2) ... (l_q, r_q)]

diff [n+1] = {0}
prefix [n+1] = {0}

```
for (int i = 0 ; i < q ; i++)
    l, r = queries[i]
    diff[l] = diff[l] + 1
    diff[r+1] = diff[r+1] - 1
```

$O(q)$

T.C.

```
prefix[0] = diff[0]
for (int i = 1 ; i <= n ; i++)
    prefix[i] = prefix[i-1] + diff[i]
```

$O(n)$

```
for (int i = 0 ; i < n ; i++)
    if prefix[i] % 2 != 0 :
        bulbs[i] = 1 - bulbs[i]
```

$O(n)$

sum(bulbs) $O(n)$

Total : $O(n + n + n + q)$

$\sim O(3n + q)$

Amazing

slower → faster

$O(nq) \longrightarrow O(n+q)$

TC $\sim$ $O(n+q)$.