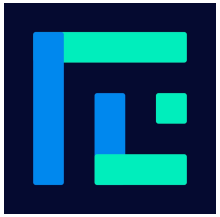


Advanced Binary Search

Binary Search Revision

Time Complexity Discussion

We are reducing our search space at every step into half of current search space

Recurrence:

$$T(n) = T(n / 2) + 1$$

$$T(1) = 1$$

$$T(8) = T(4) + 1$$

$$T(4) = T(2) + 1$$

$$T(2) = T(1) + 1$$

$$T(1) = 1$$

Time Complexity: O(logn)

$$T(8) = T(4) + 1$$

$$= T(2) + 1 + 1$$

$$= T(1) + 1 + 1 + 1$$

$$=$$

$$= 1 + 1 + 1 + 1 = 4$$

$$T(n) = T(n/2) + 1$$

$$= T(n/4) + 2$$

$$= T(n/8) + 3$$

$$= T(n/2^k) + k$$

$T(n/2^k) = 1 \rightarrow T(1) = 1$

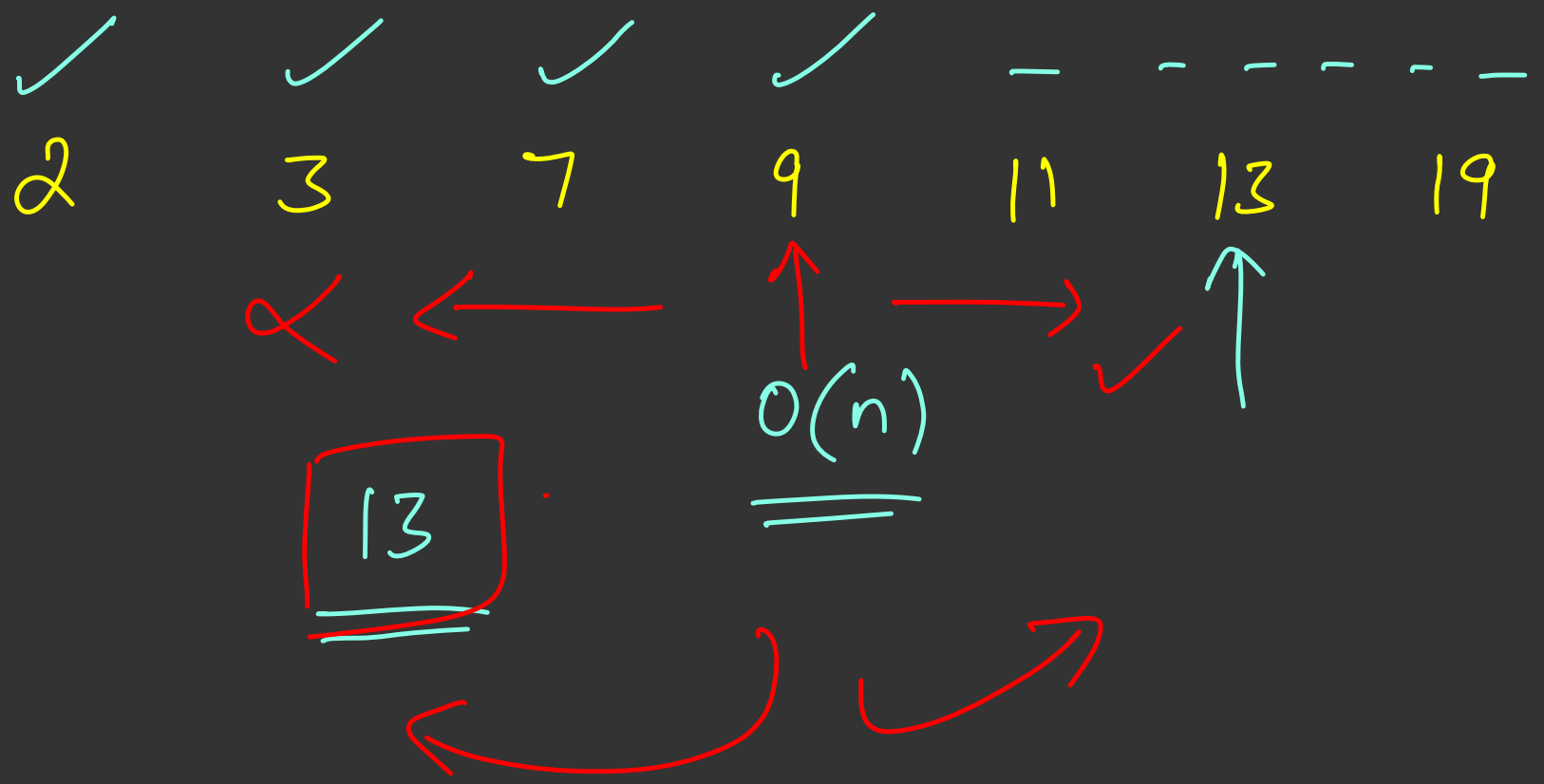
when $k = \log n$ $\rightarrow$ $n/2^k = 1$, $k = \log n$

$T(n) = T(n/2^k) + k$

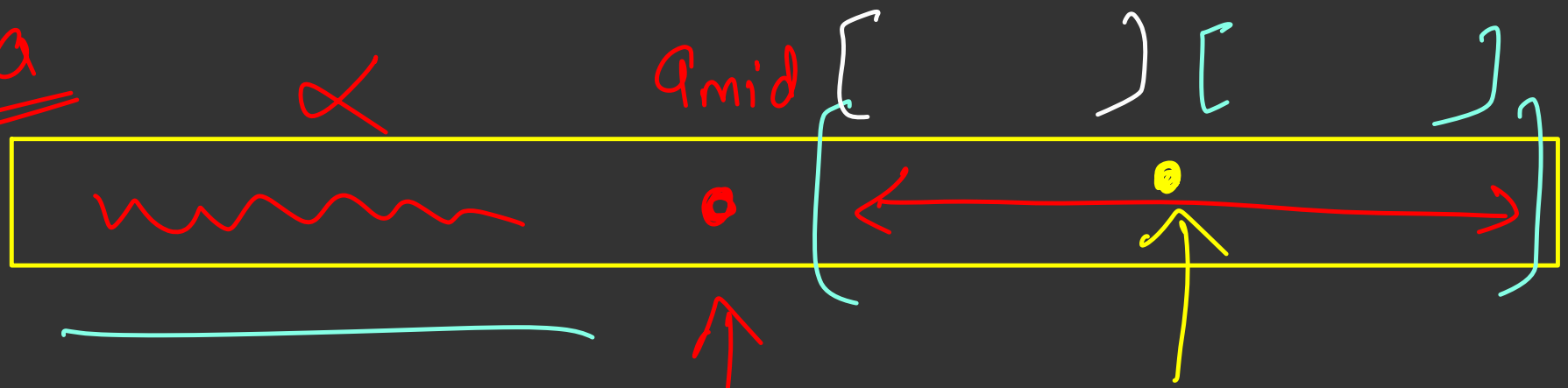
$= T(n/2^{\log n}) + \log n$

$= T(1) + \log n$

$= 1 + \log n = O(\log n)$


2
3
7
9
11
13
19
O(n)
13

Array of size (n)

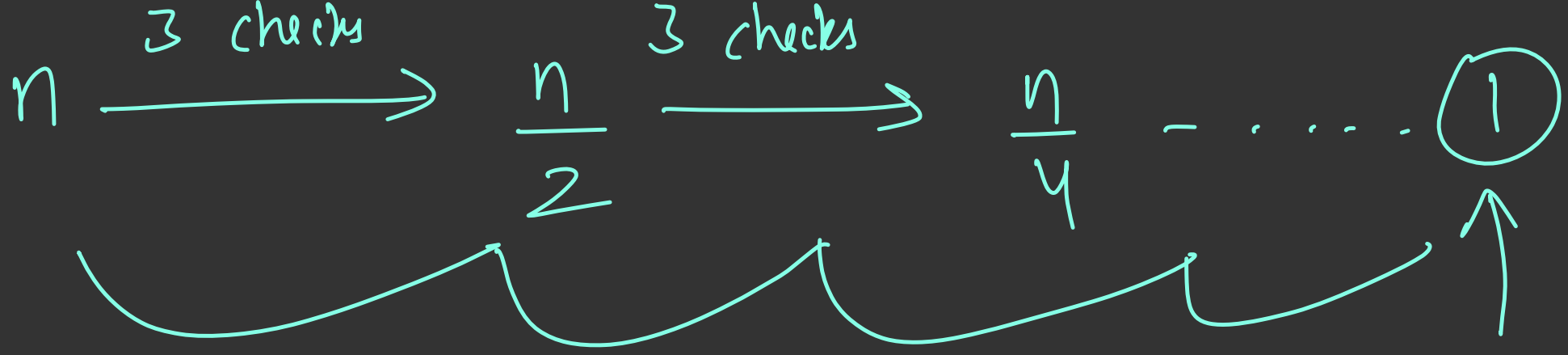


$a_{mid} < target$

$(n) \rightarrow \left(\frac{n}{2}\right) \rightarrow (n/4)$

$a_{mid} > target$

$a_{mid} == target$



$$T(n) \longrightarrow T\left(\frac{n}{2}\right) + O(1)$$

$n \longrightarrow \left(\frac{n}{2}\right) + 3$

iteration

Time to search in array of size

N = Time to search in array of size $\frac{N}{2} + 3$

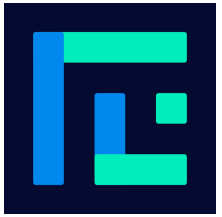
$T(n) = T\left(\frac{n}{2}\right) + 3$

$T(n) = T(n/2) + O(1)$

2 5 9 1 7 6 13

Searching for 7

Binary Search Revision

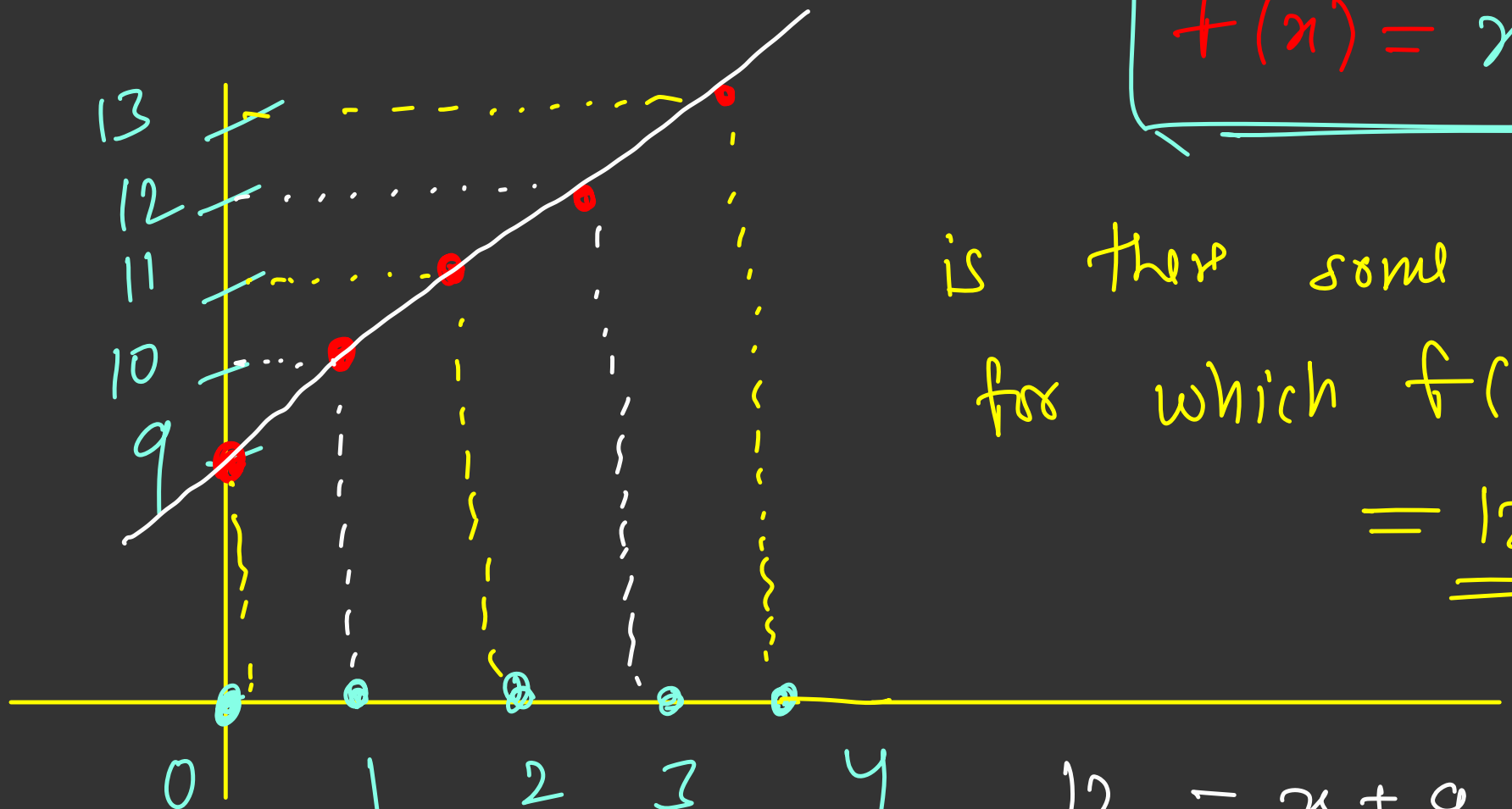


Requirement for using Binary Search

Monotonicity

- f(x) > f(y) iff x > y (increasing monotonic)
- f(x) < f(y) iff x > y (decreasing monotonic)

$$f(x) = x + 9$$

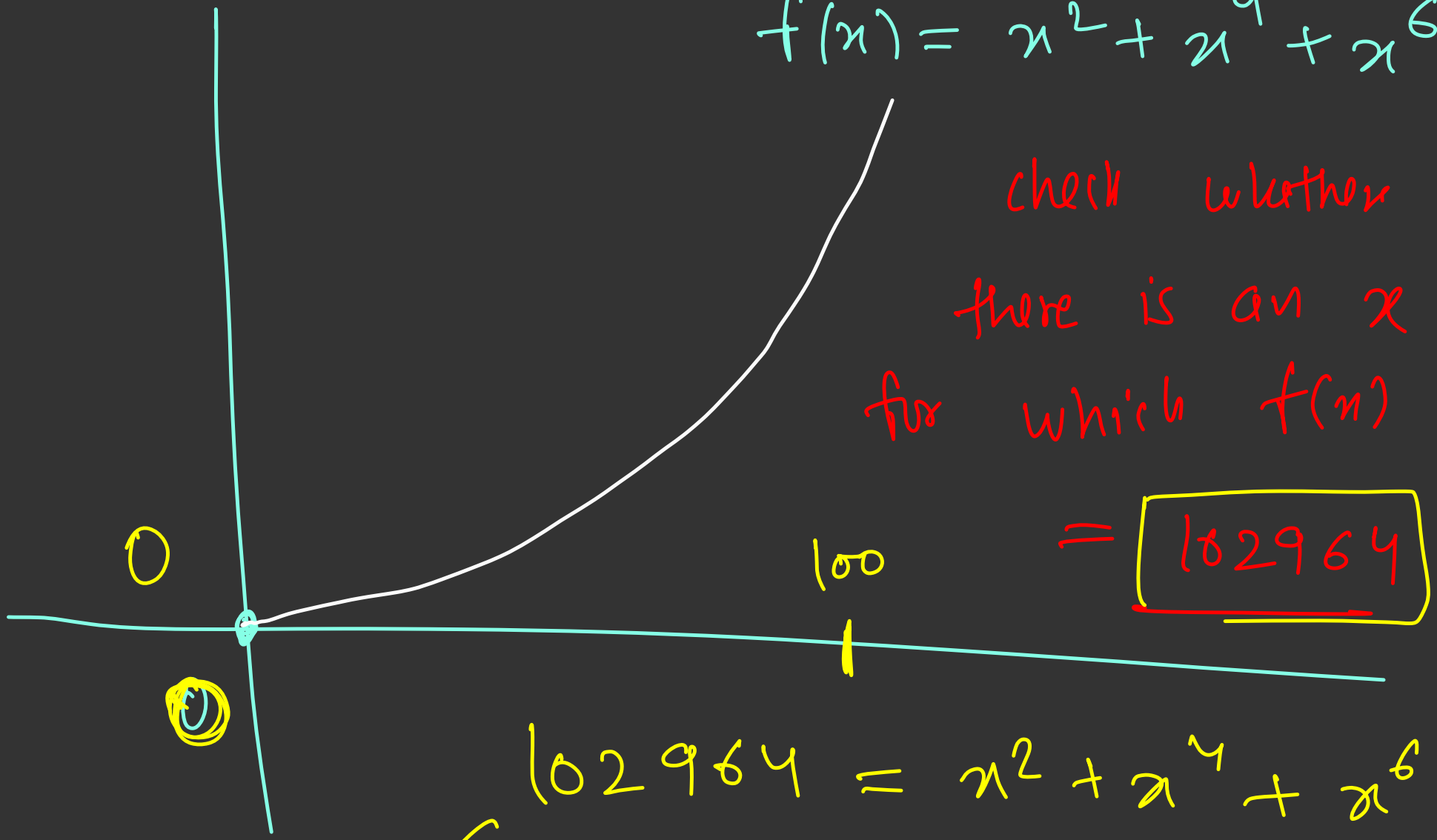


is there some x for which $f(x) = 12$

$$12 = x + 9$$

$$x = 3$$

$$f(x) = x^2 + x^4 + x^6 + 2$$

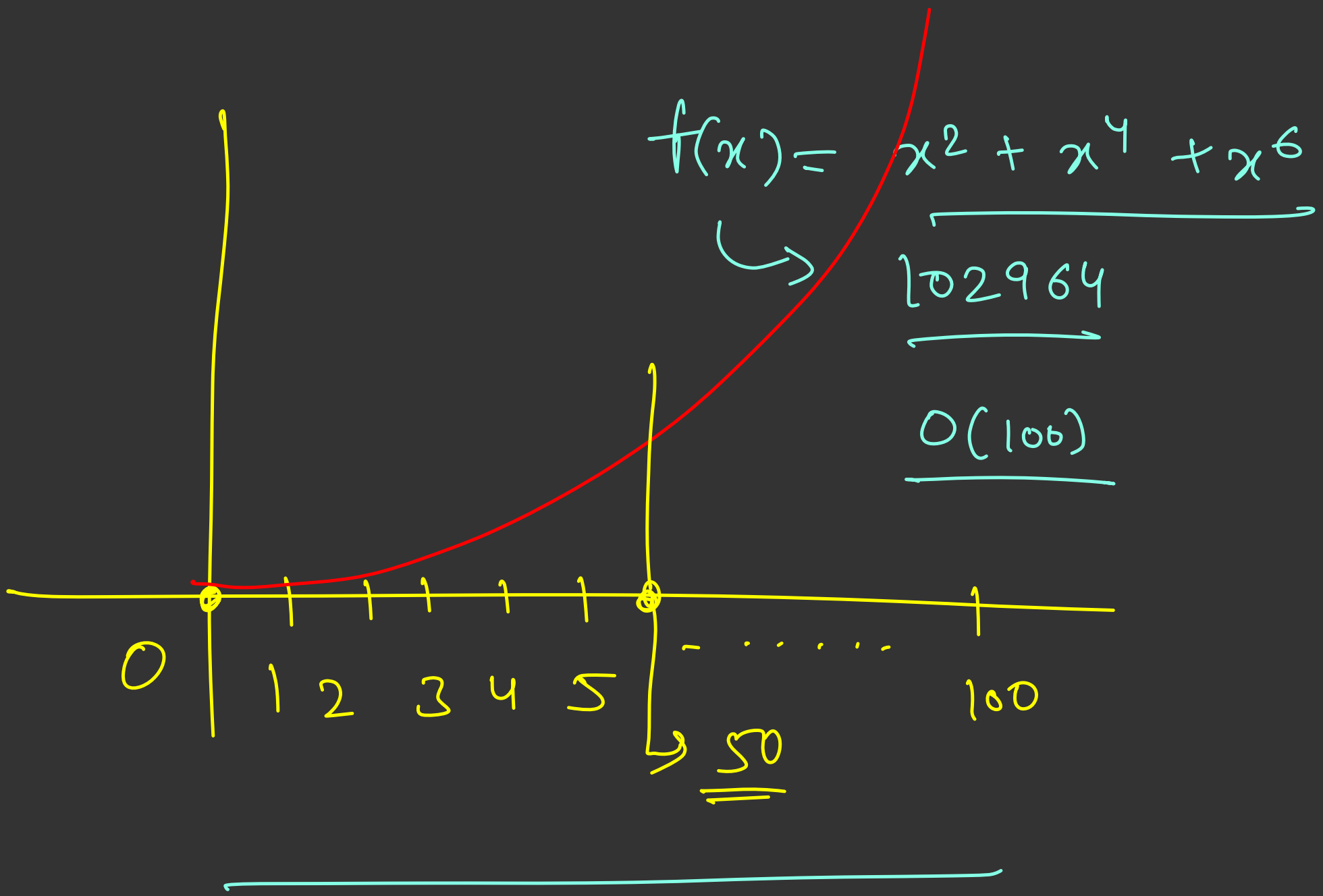


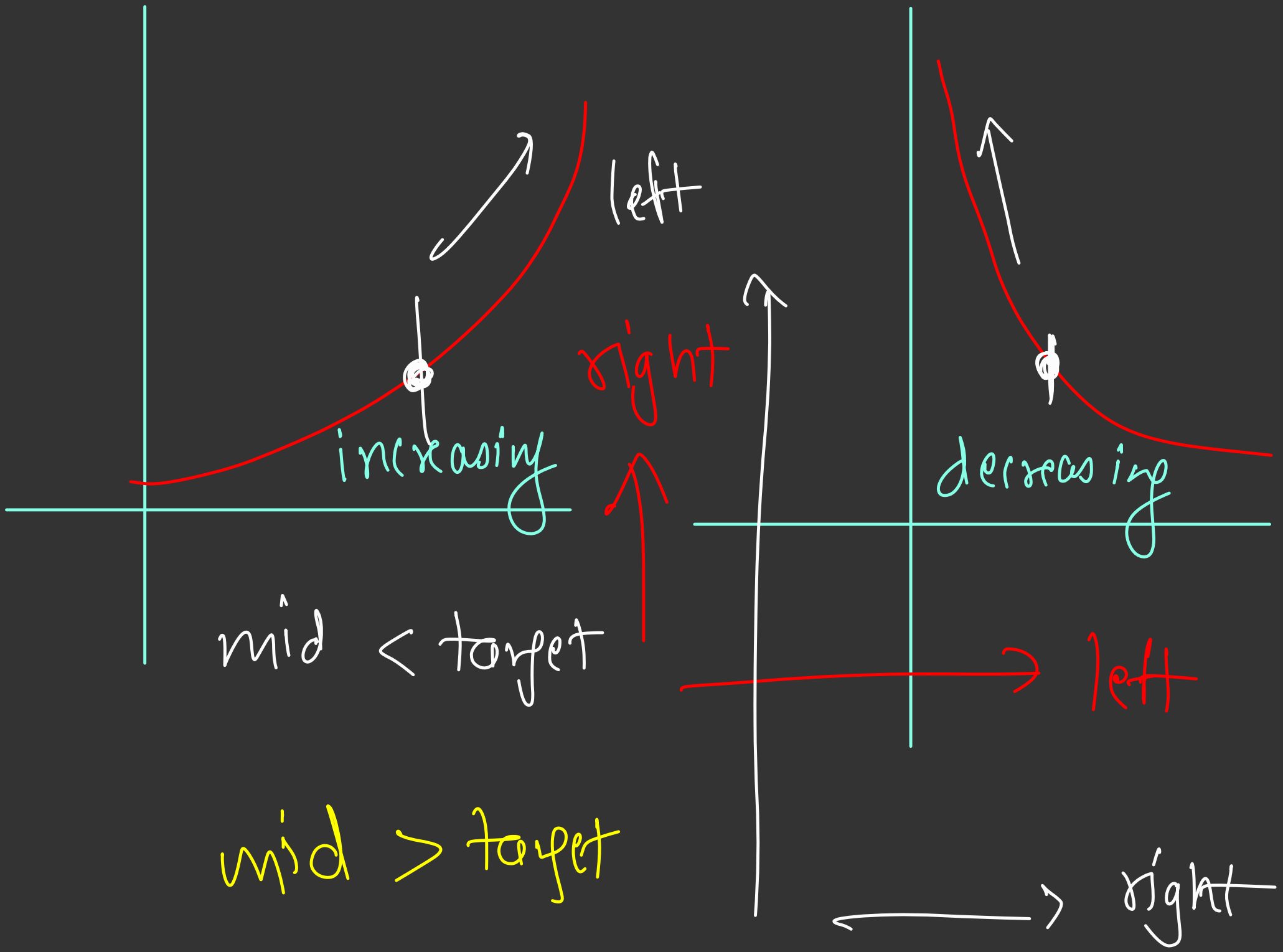
check whether there is an x for which $f(x)$ $= \boxed{102964}$

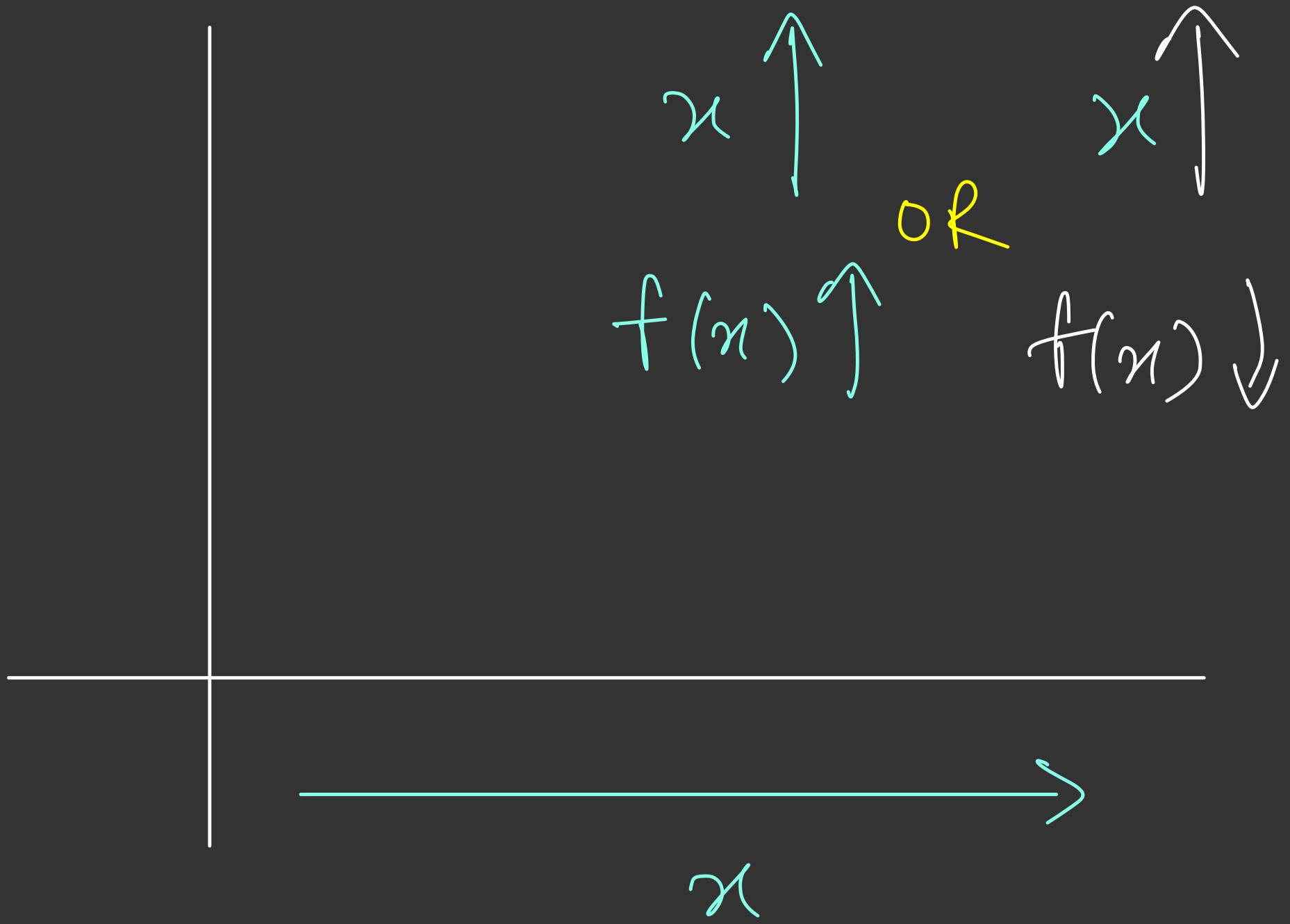
$102964 = x^2 + x^4 + x^6$

$x = 100$

100^6


f(x) = x^2 + x^4 + x^6
102964
O(100)
0
1 2 3 4 5
50
100


left
increasing
right
decreasing
mid < target
left
mid > target
right


x ↑
x ↑
OR
f(x) ↑
f(x) ↓
x

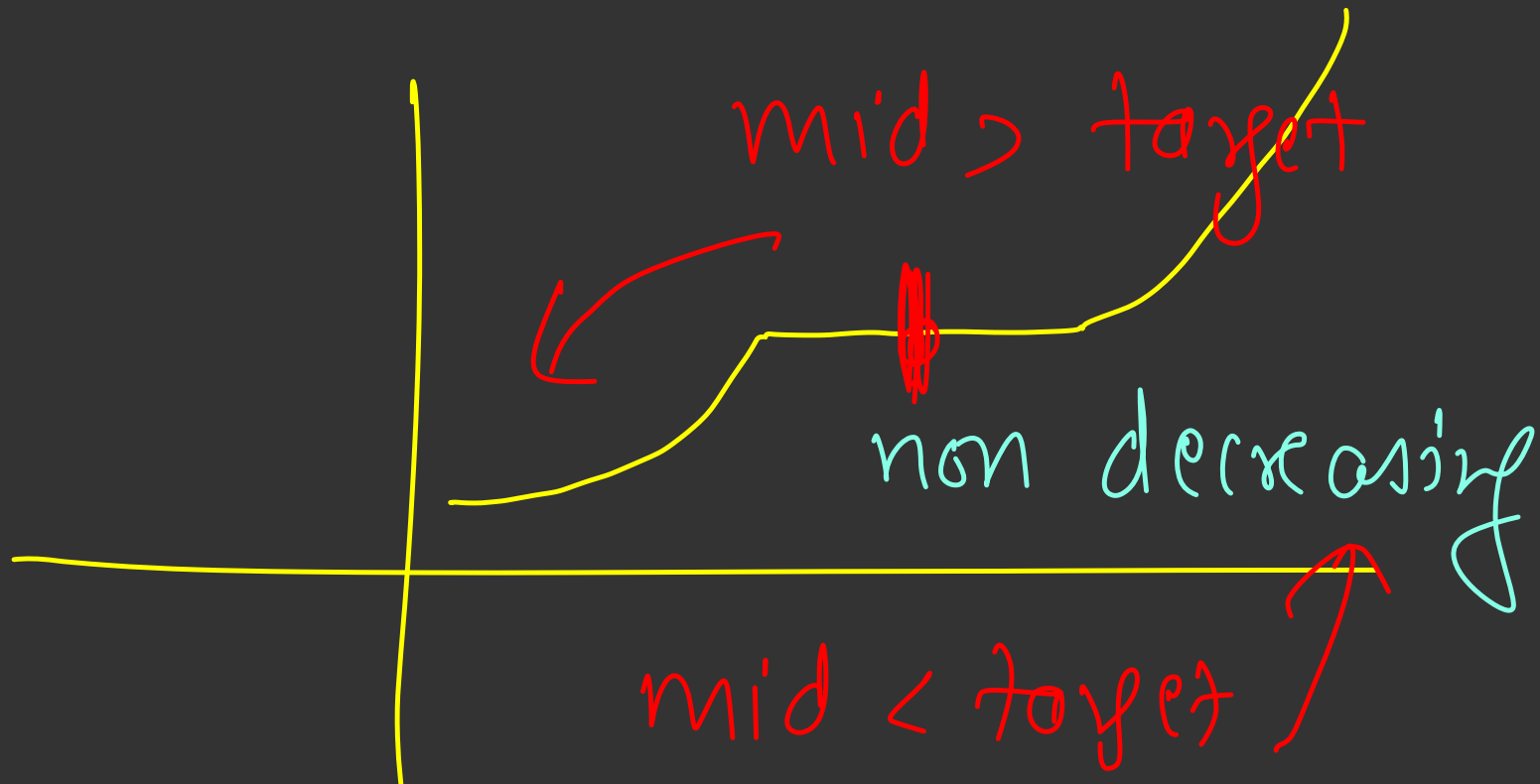
increasing $x_1 > x_2$

$f(x_1) > f(x_2)$

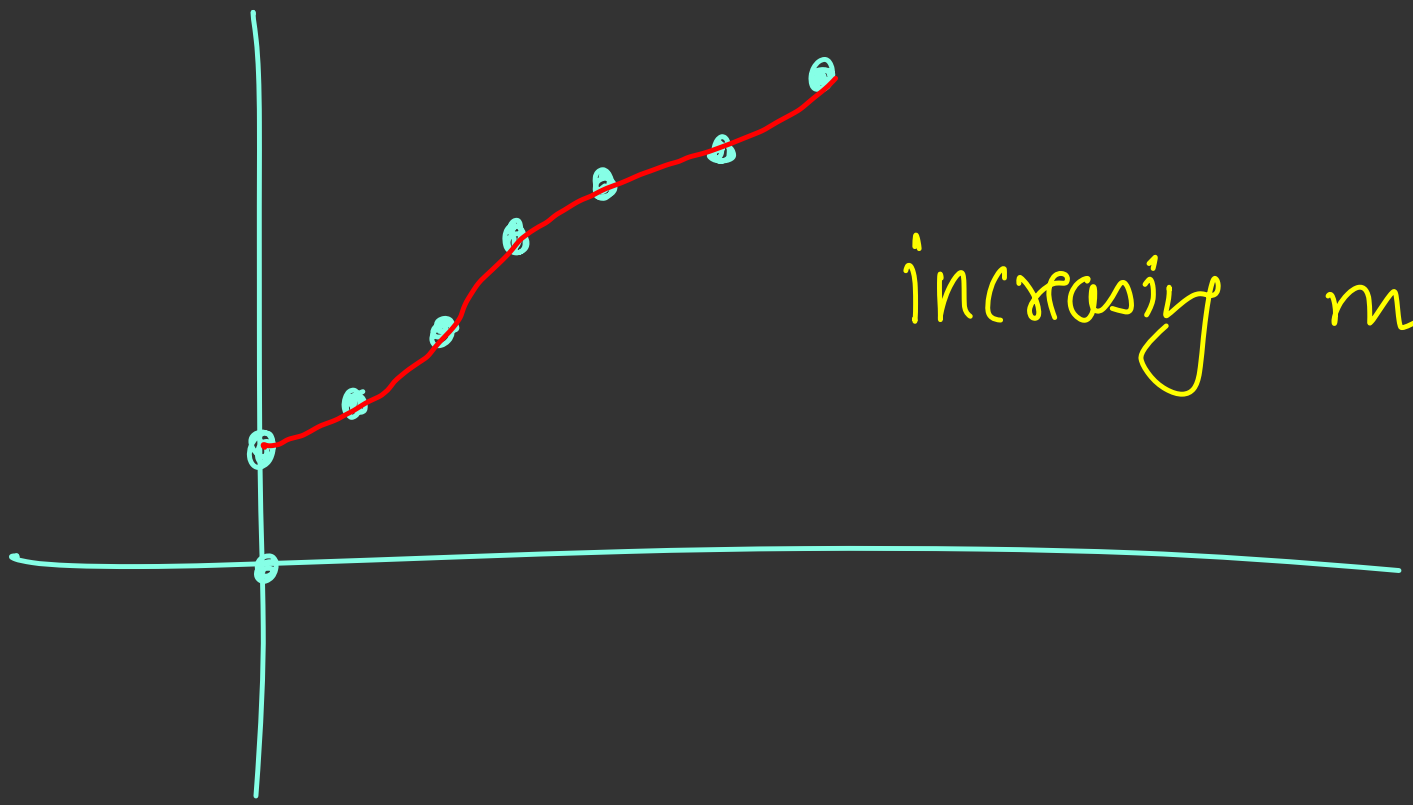
Non decreasing $x_1 > x_2$

$f(x_1) \geq f(x_2)$


increasing

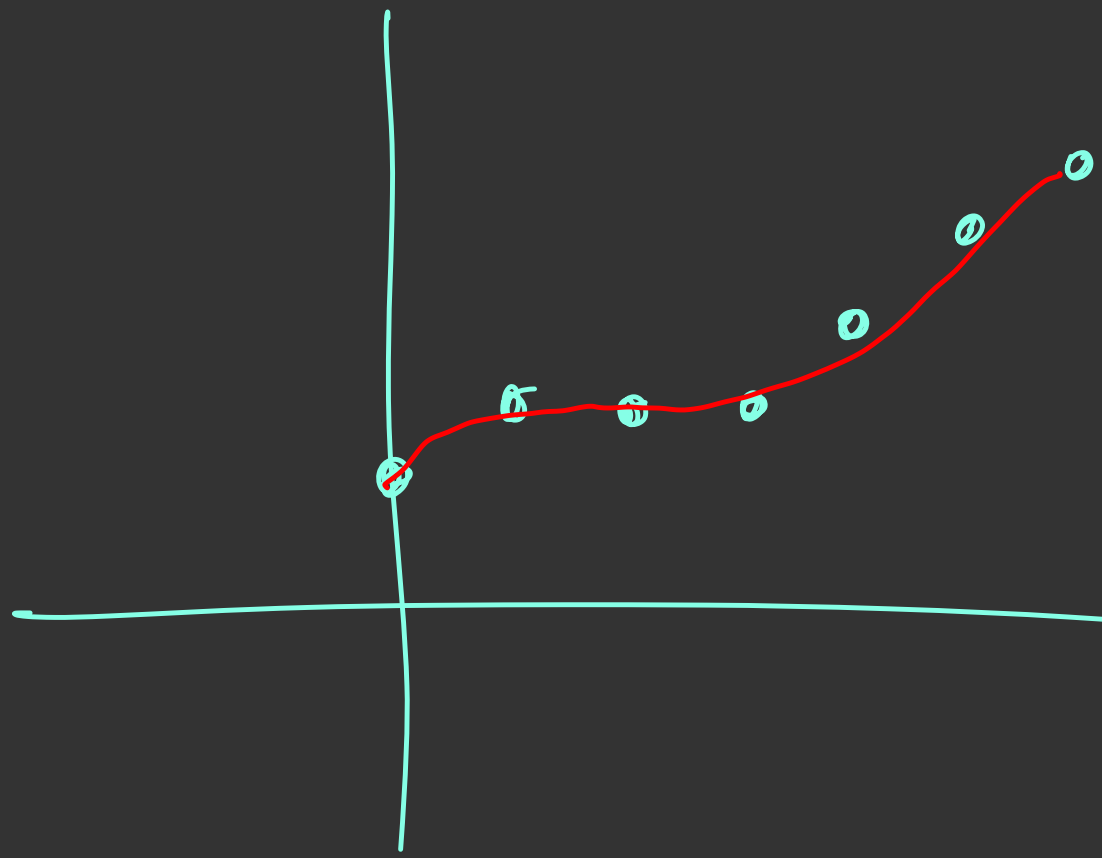

mid > target
non decreasing
mid < target

2 3 5 9 11 13 17

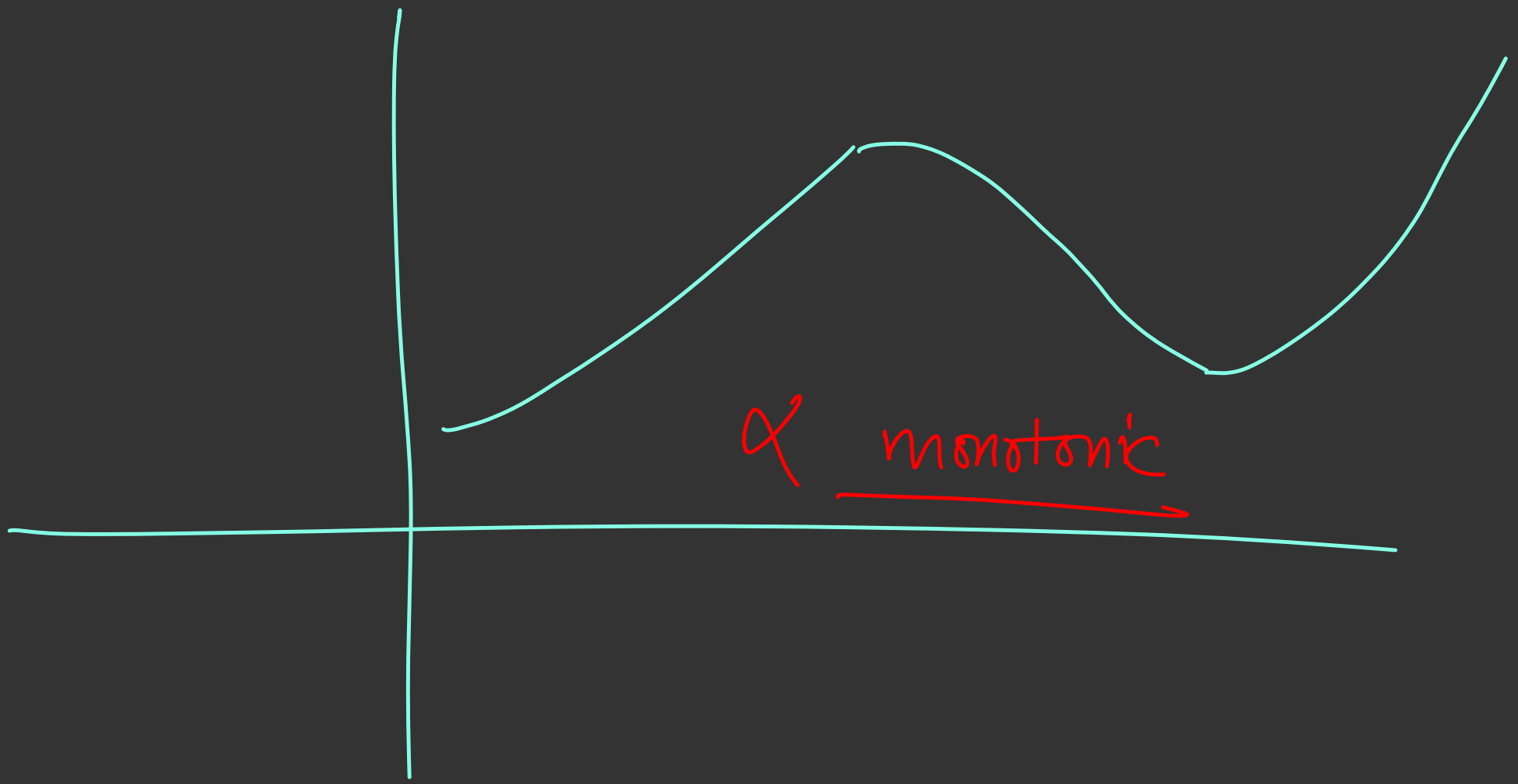


increasing monotonic

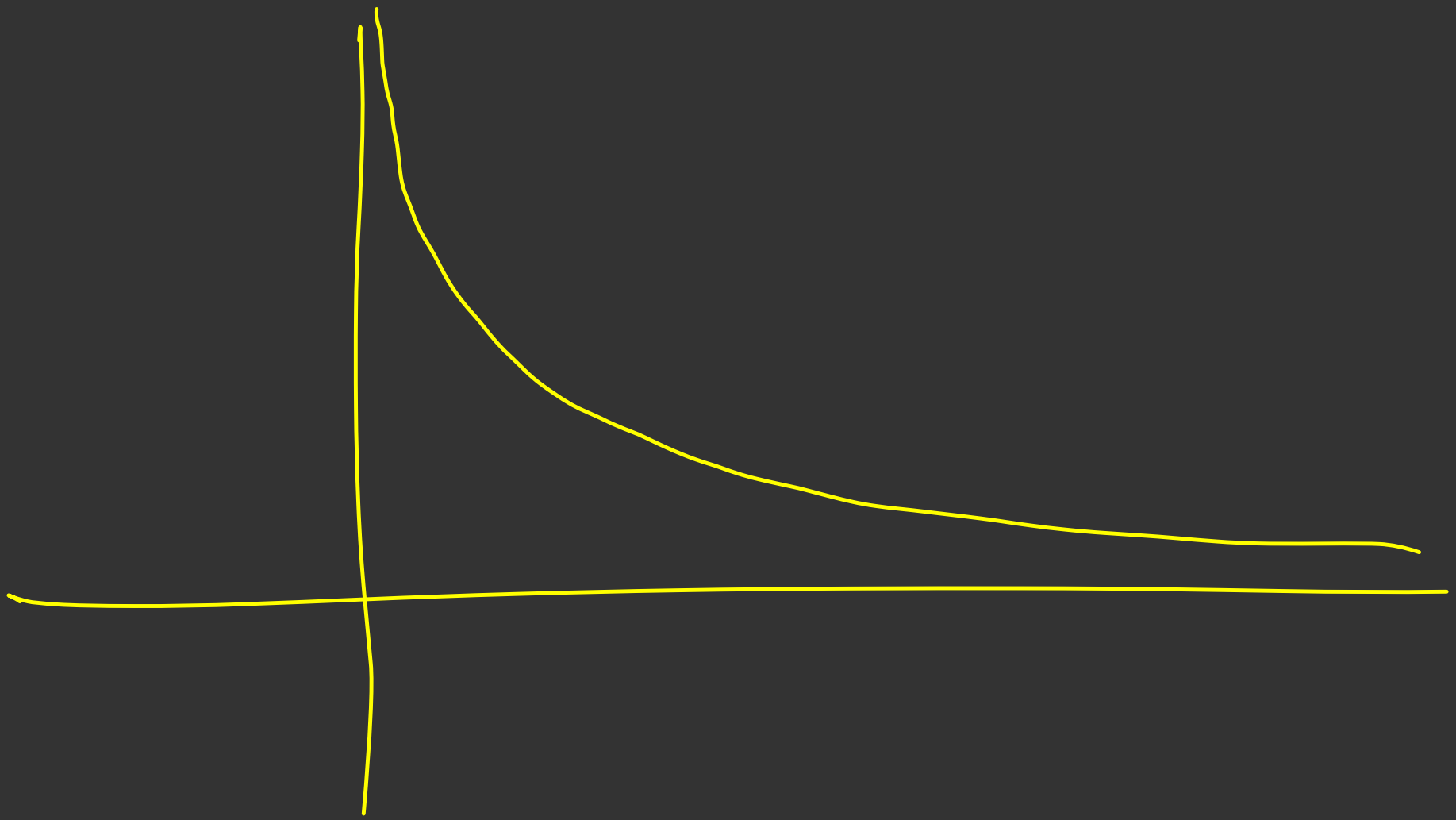
2 3 3 3 4 5 6



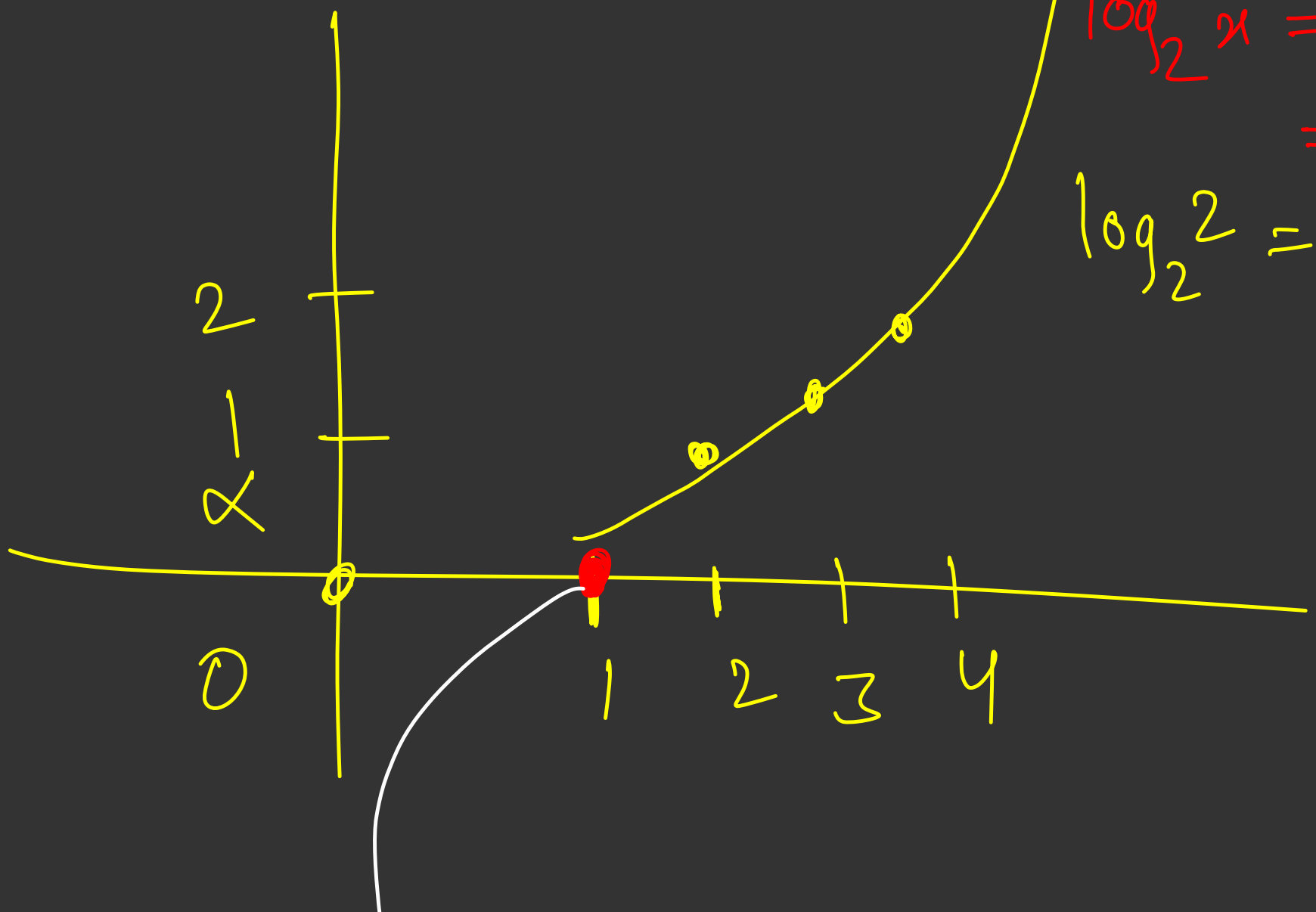
non decreasing


& monotonic

$$f(x) = \frac{1}{x}$$



$$f(x) = \log_2 x$$



$x = 1$

$\log_2 x = \log_2 1$

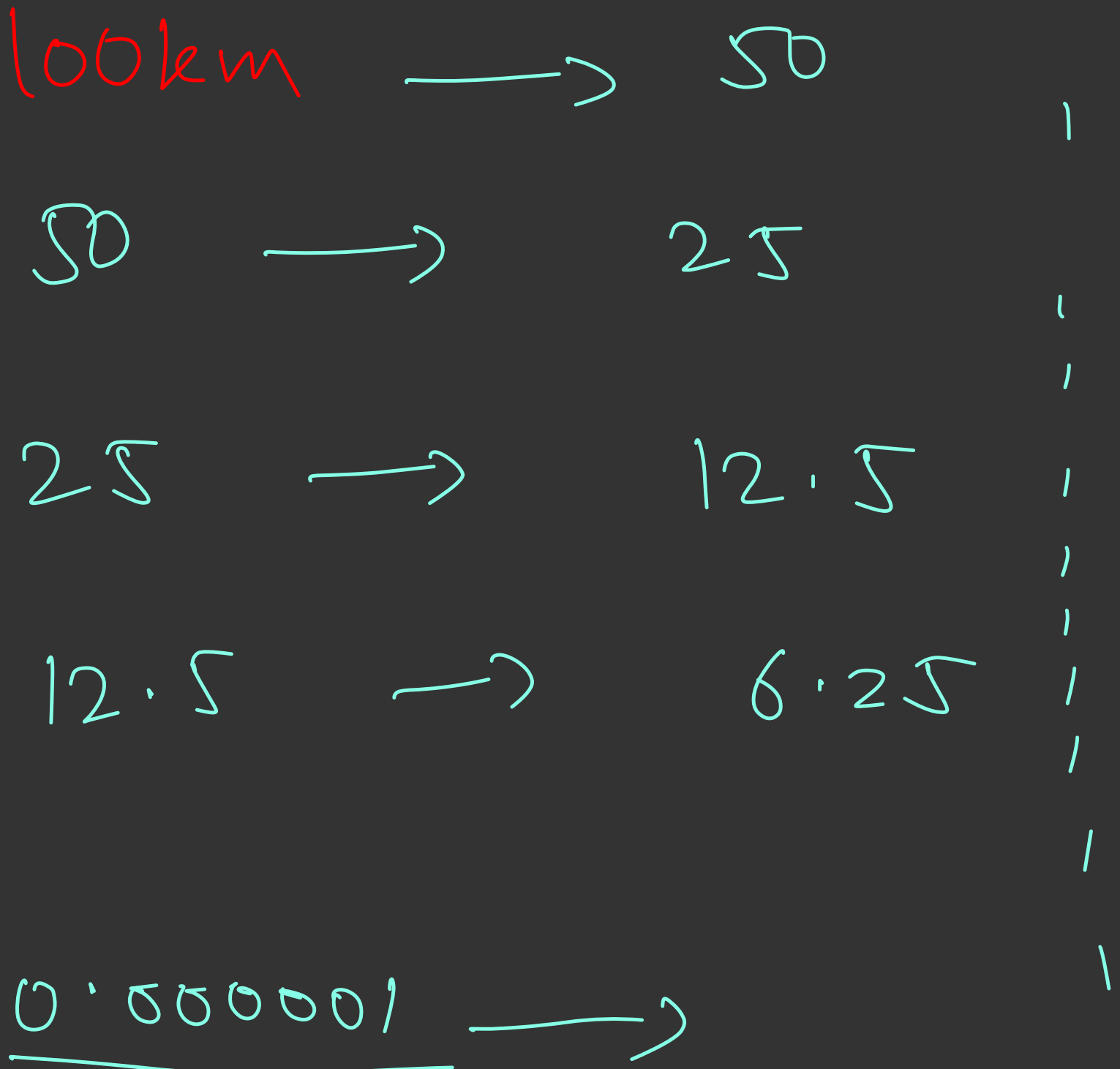
$= 0$

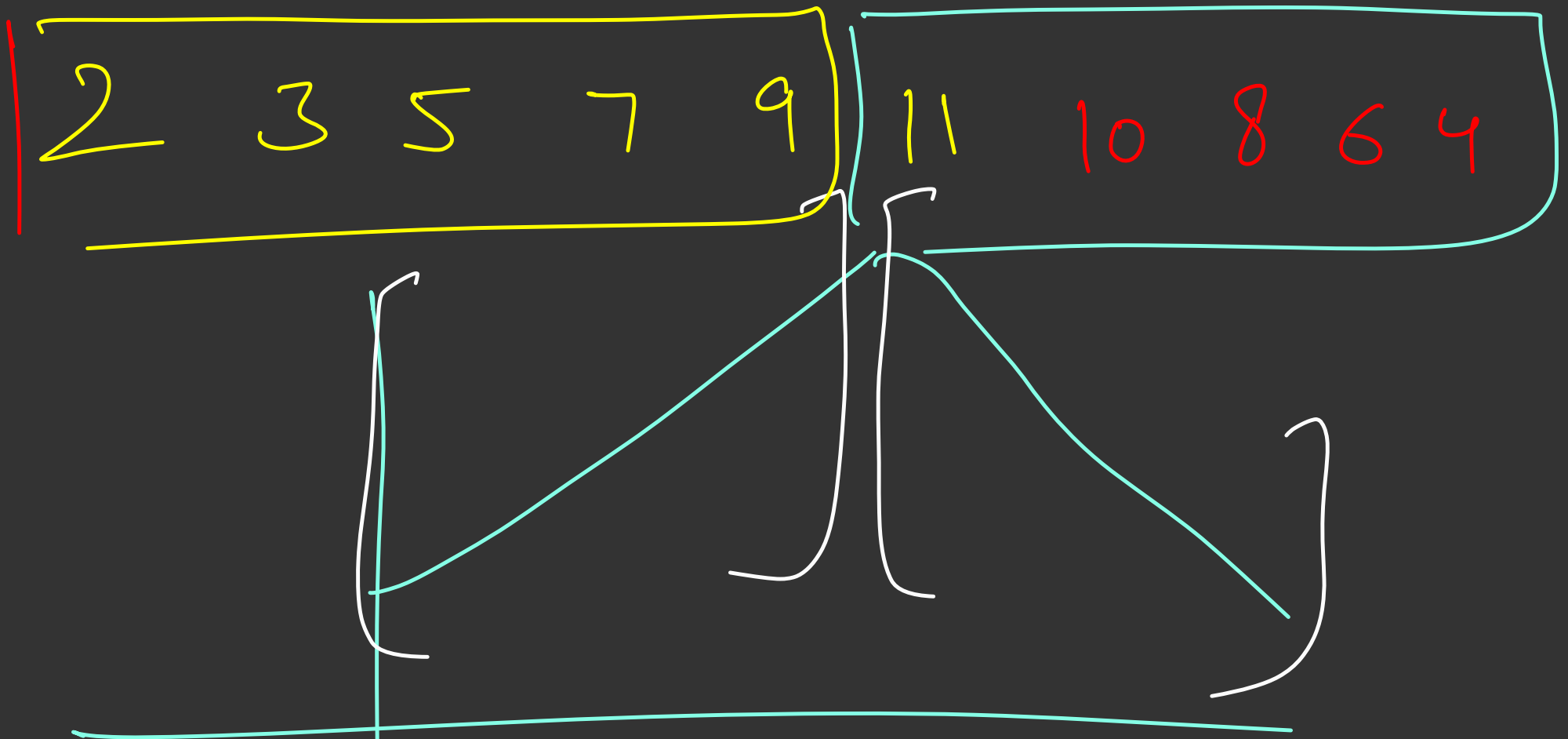
$\log_2 2 = 1$

You want to cover 100km of distance

In every second you cover half of the remaining distance

How much time to cover all 100 km


100km
50
50
25
25
12.5
12.5
6.25
0.000001



$\log n + \log n$

$2 \log n \rightarrow O(\log n)$

Binary Search Revision

Find Number of elements in the range of l to r in a sorted array

1	2	3	5	6	7	8	10	11	16	20

Ex: Number of Elements in the range 4 to 12 are 6

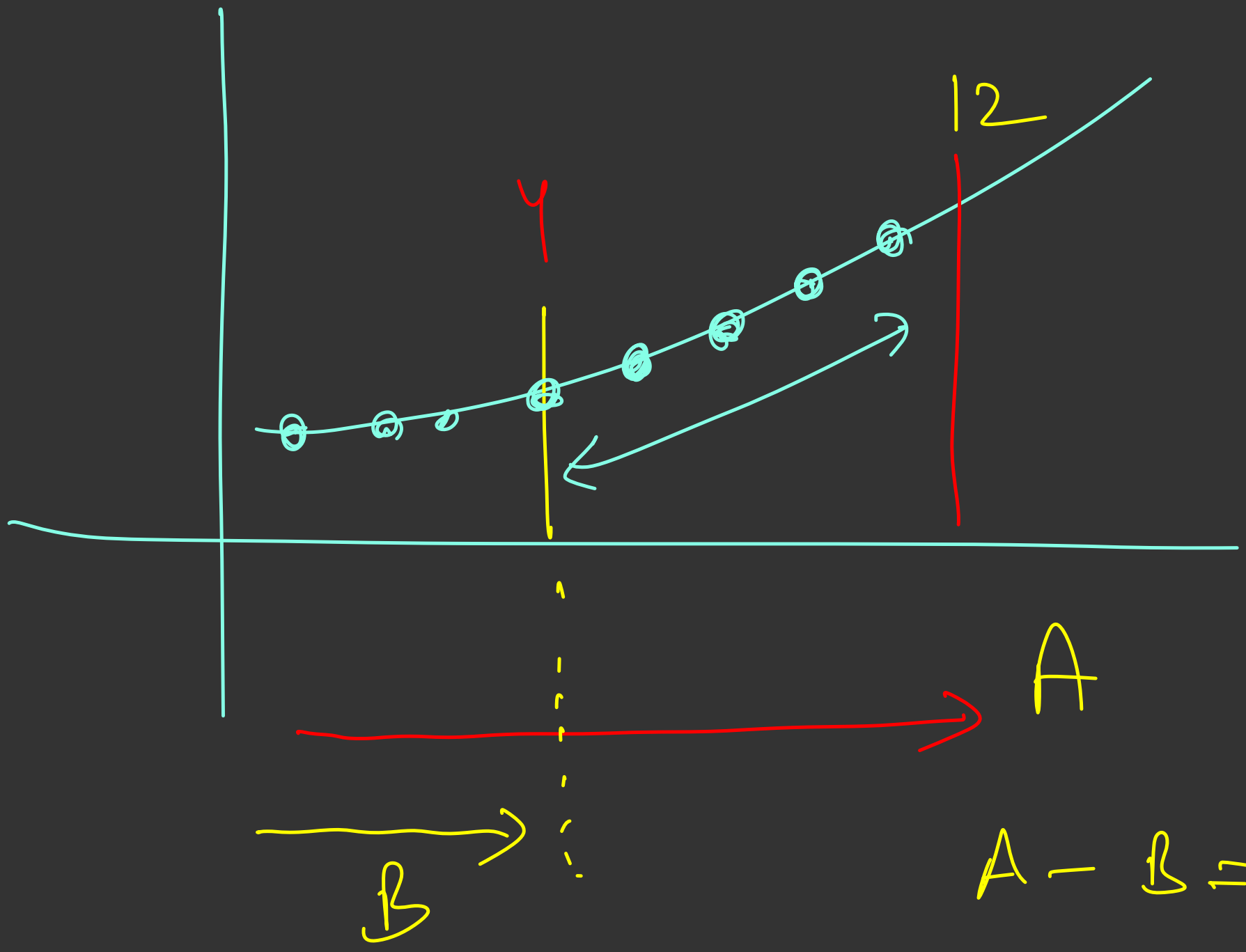
$4 \leq arr[i] \leq 12$

how many such i exist

A = no. of elements ≤ 12

B = no. of elements < 4

$A - B$ = answer


12
4
A
B
A − B = answer

find no. of elements in sorted array $\leq X$

= index of the highest number $\leq X$

≤ 12

Considering 1 based indexing

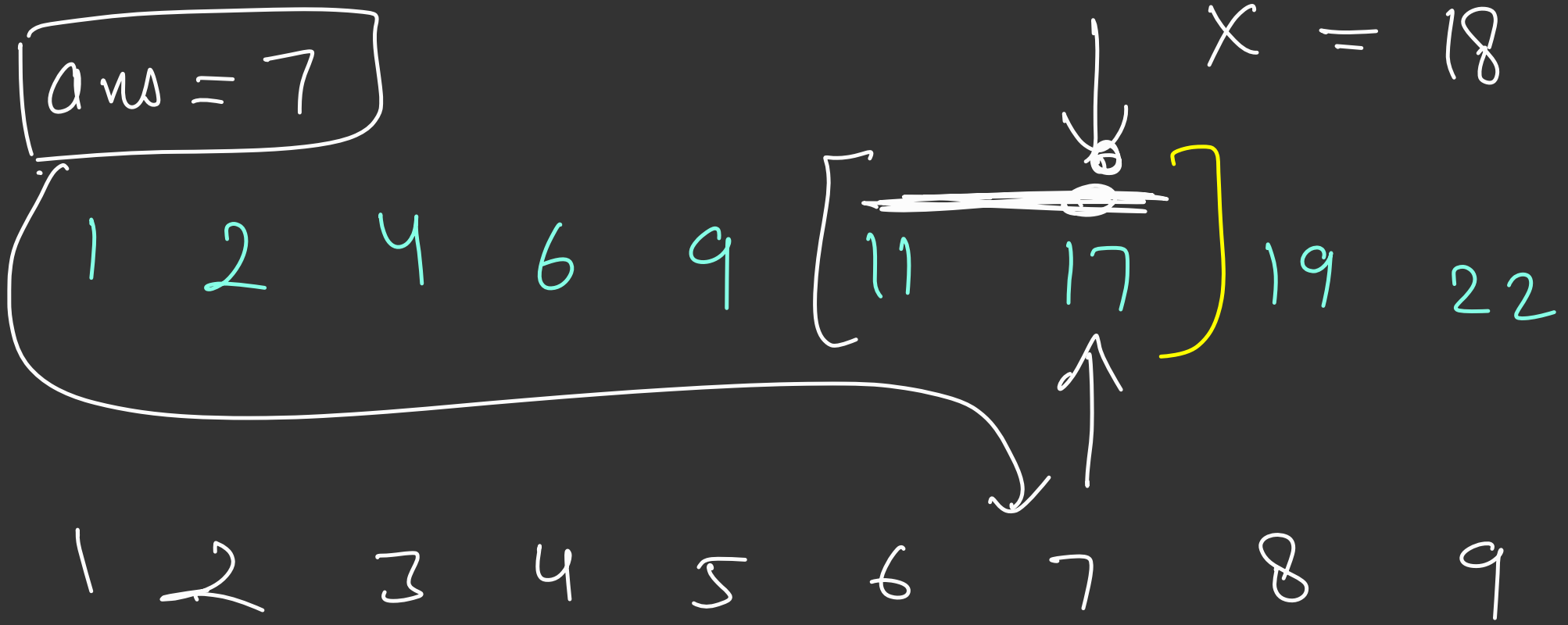
no. of elements $\leq X$

$=$ index of highest element

$\leq X$

Given a number x, find out the index of the highest element which is $\leq x$

ans = 7

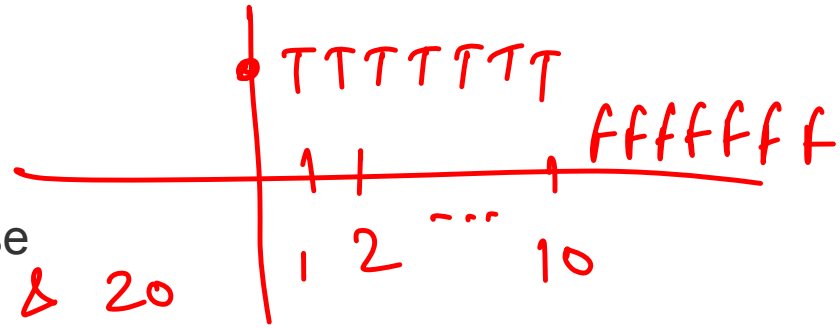


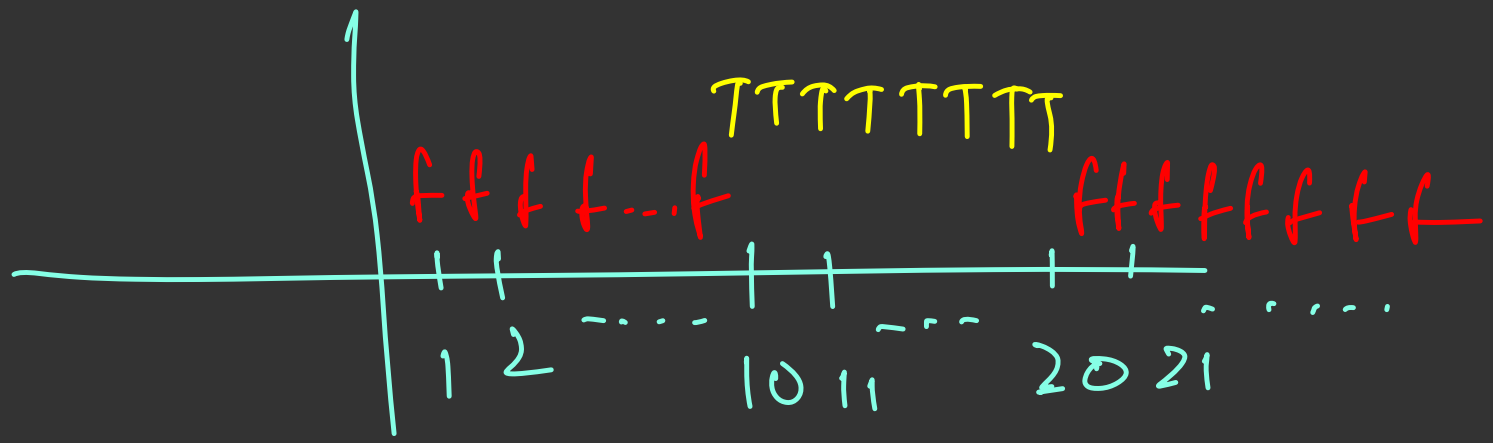
Predicate Functions

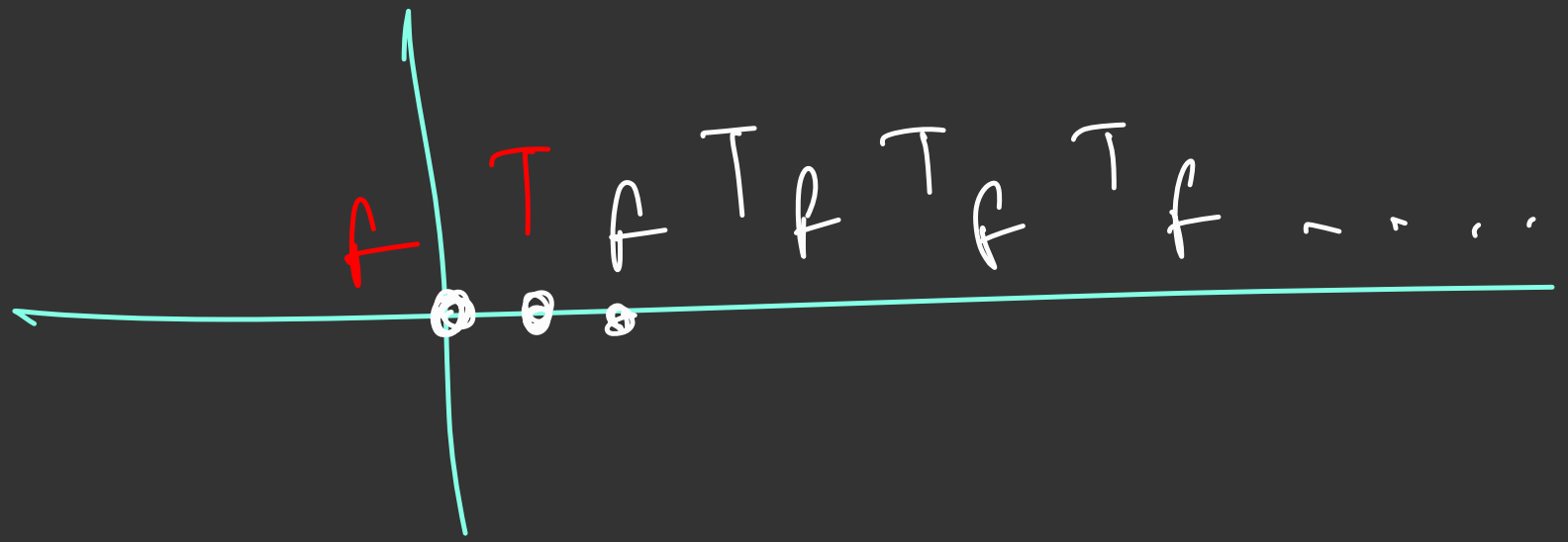
Functions that return a single TRUE or False for every input. You use predicate functions to check if your input meets some condition or not.

Examples:

- F(x) = True if x > 10 otherwise False
- F(x) = True if x is a number otherwise False
- F(x) = True if x^2 is an odd number otherwise False

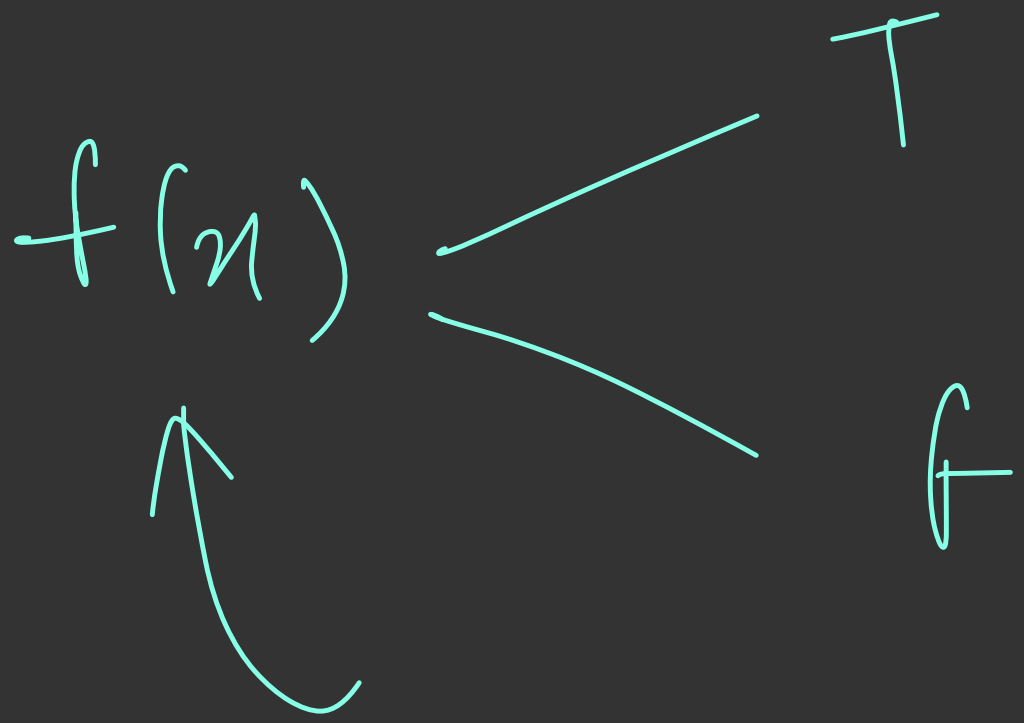






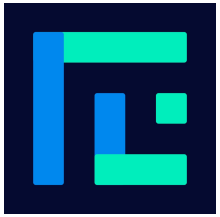
Square of an odd number = odd

Square of even no. = even



$f(n) = n + 2$

Binary Searching on Answer



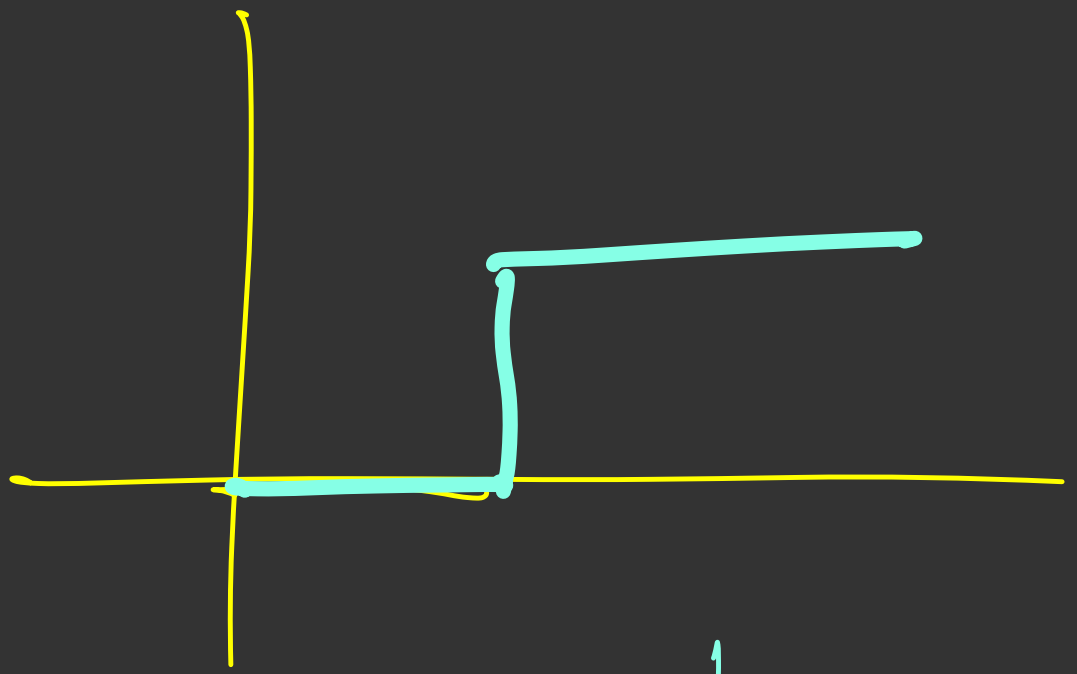
- Consider a predicate P defined over some ordered set S (the search space).
- The search space consists of candidate answers to the problem.
- We use the predicate to verify if a candidate answer is legal or not.

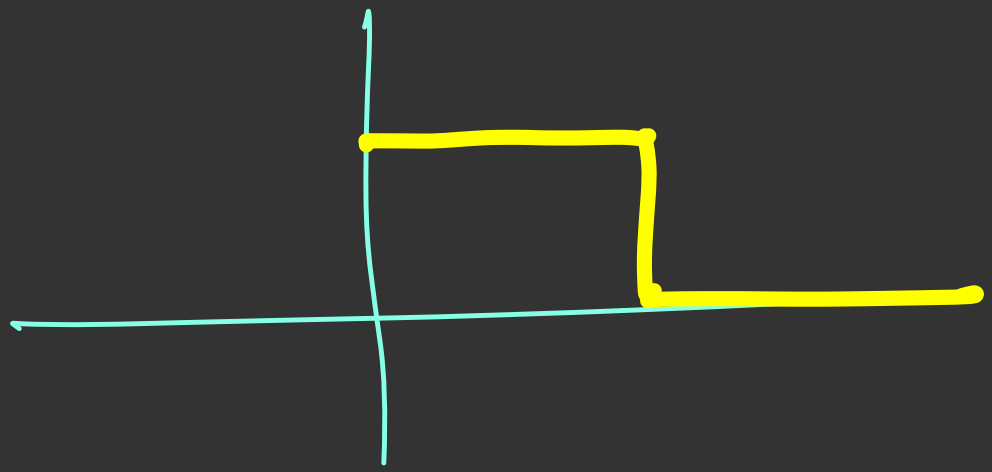
Example: We have the set of numbers {1, 2, 3, 4, 5, 6}.

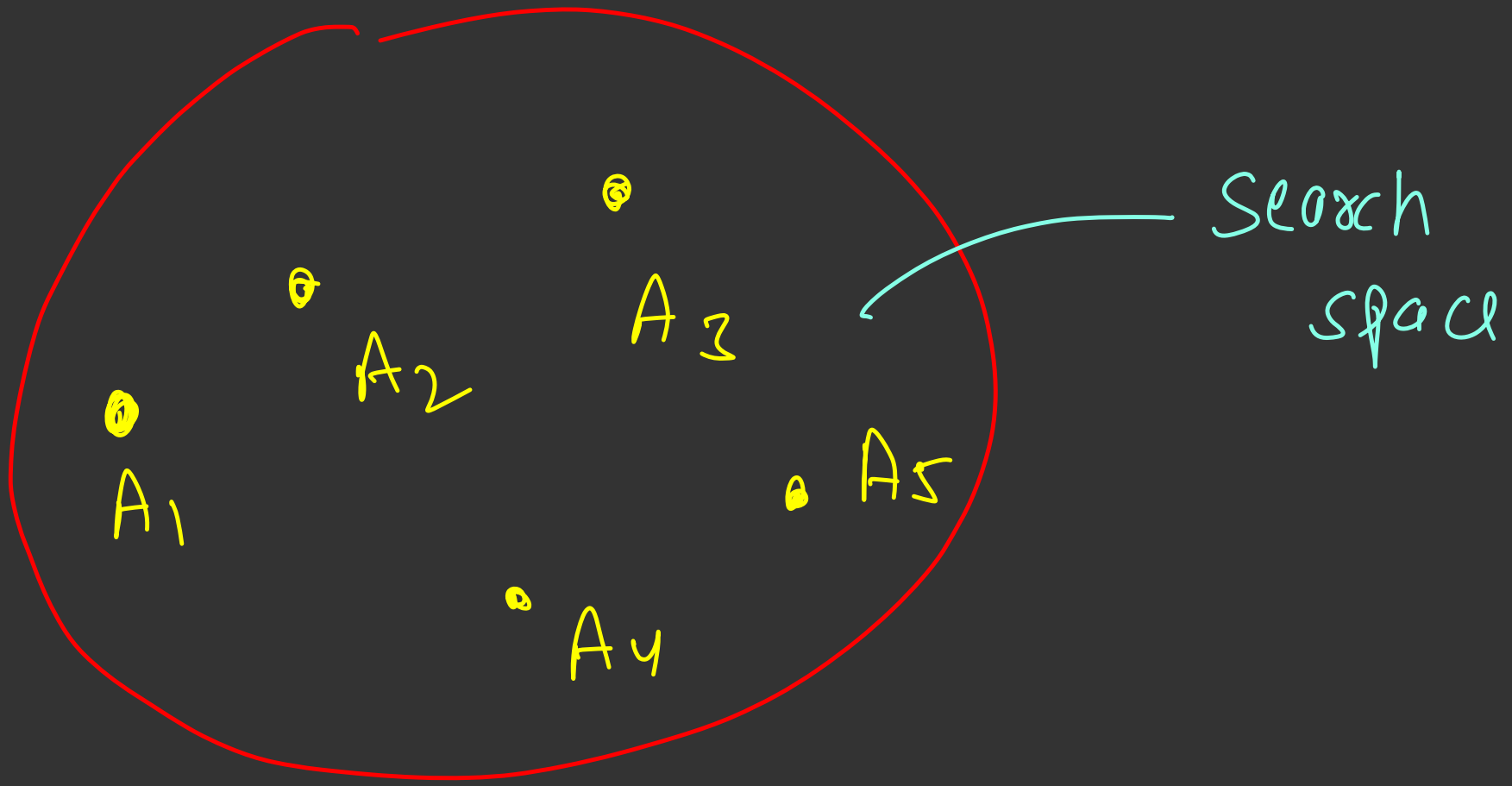
Our predicate function could be following:

- Return TRUE if the number is less than 3 and FALSE otherwise

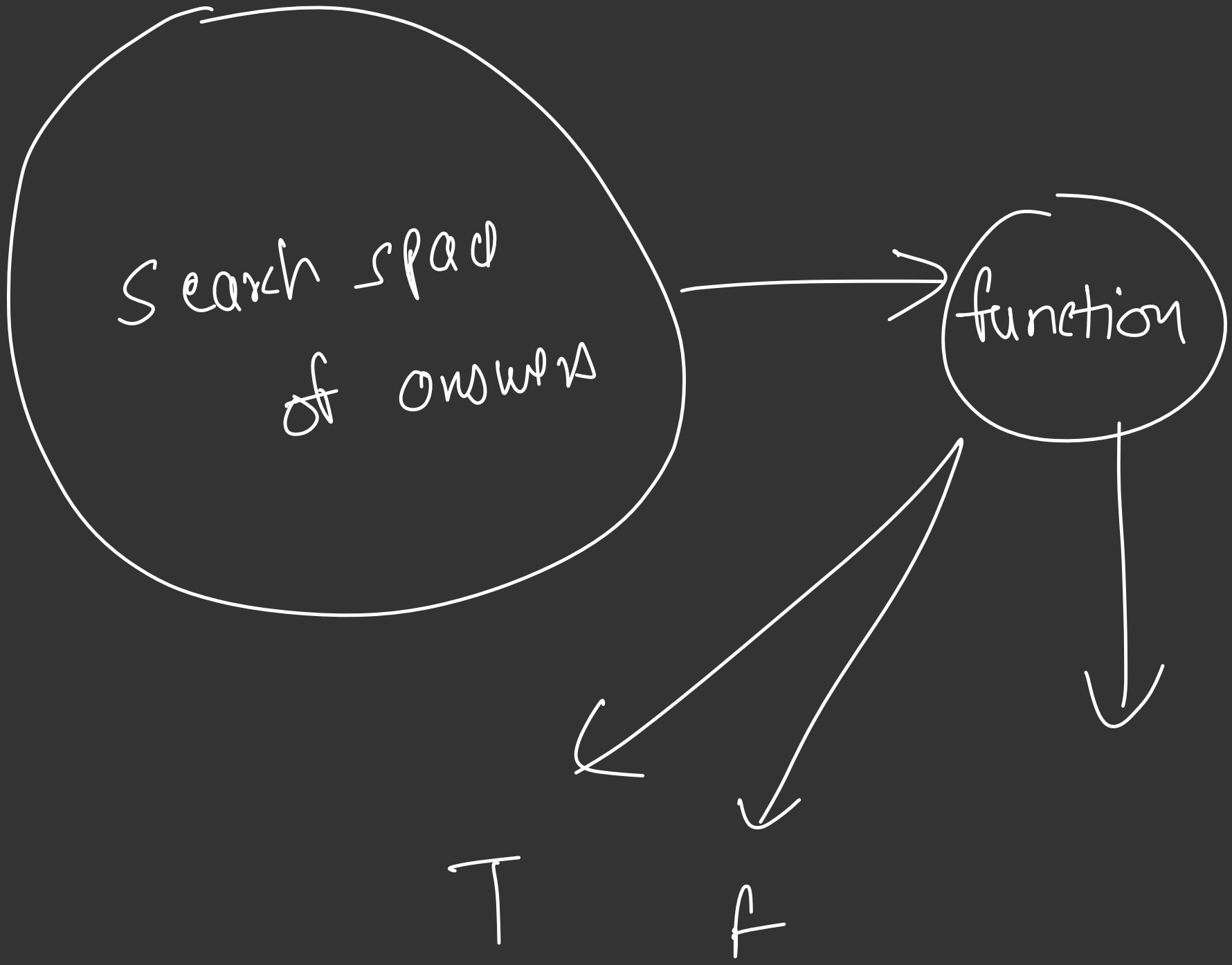
min possible answer

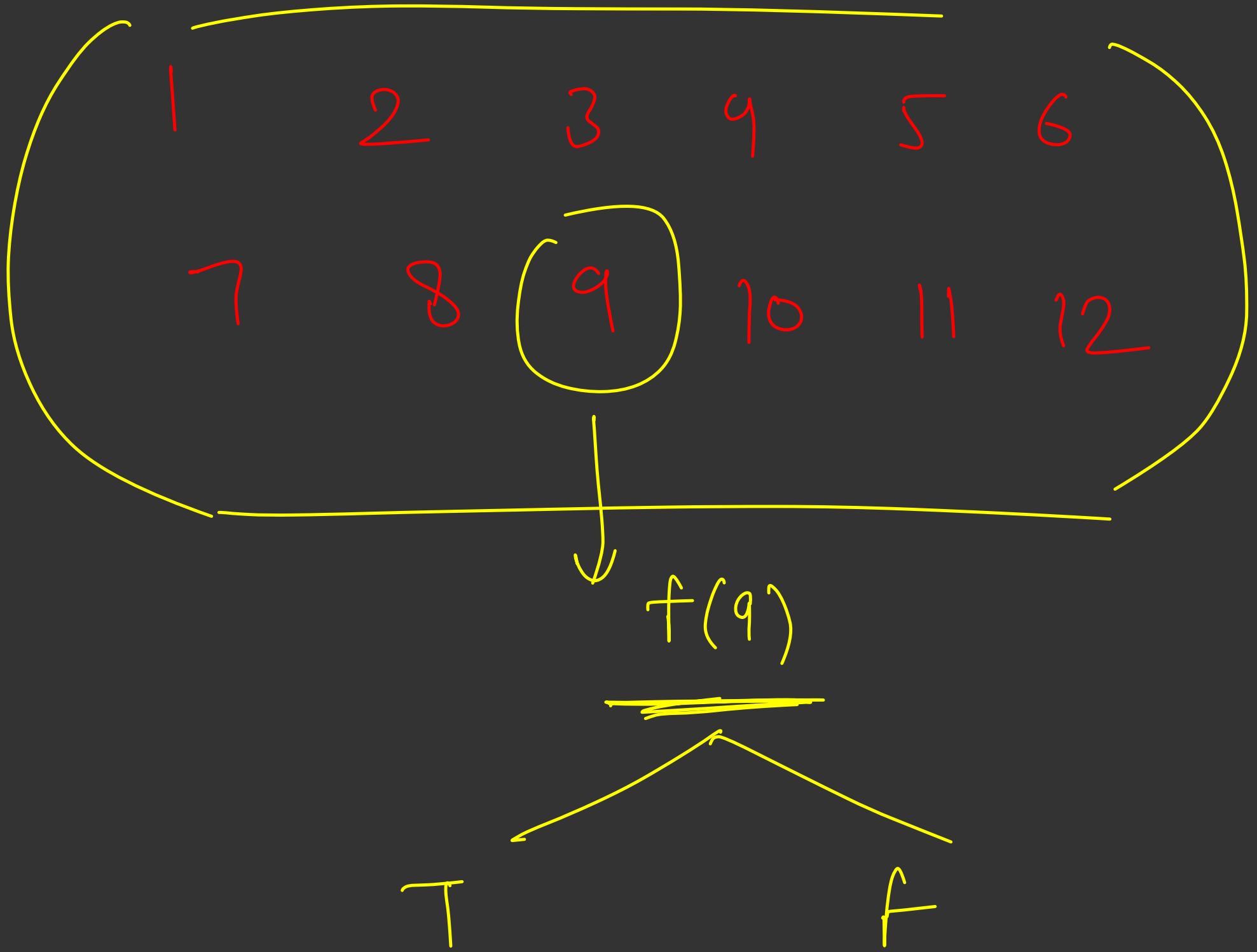






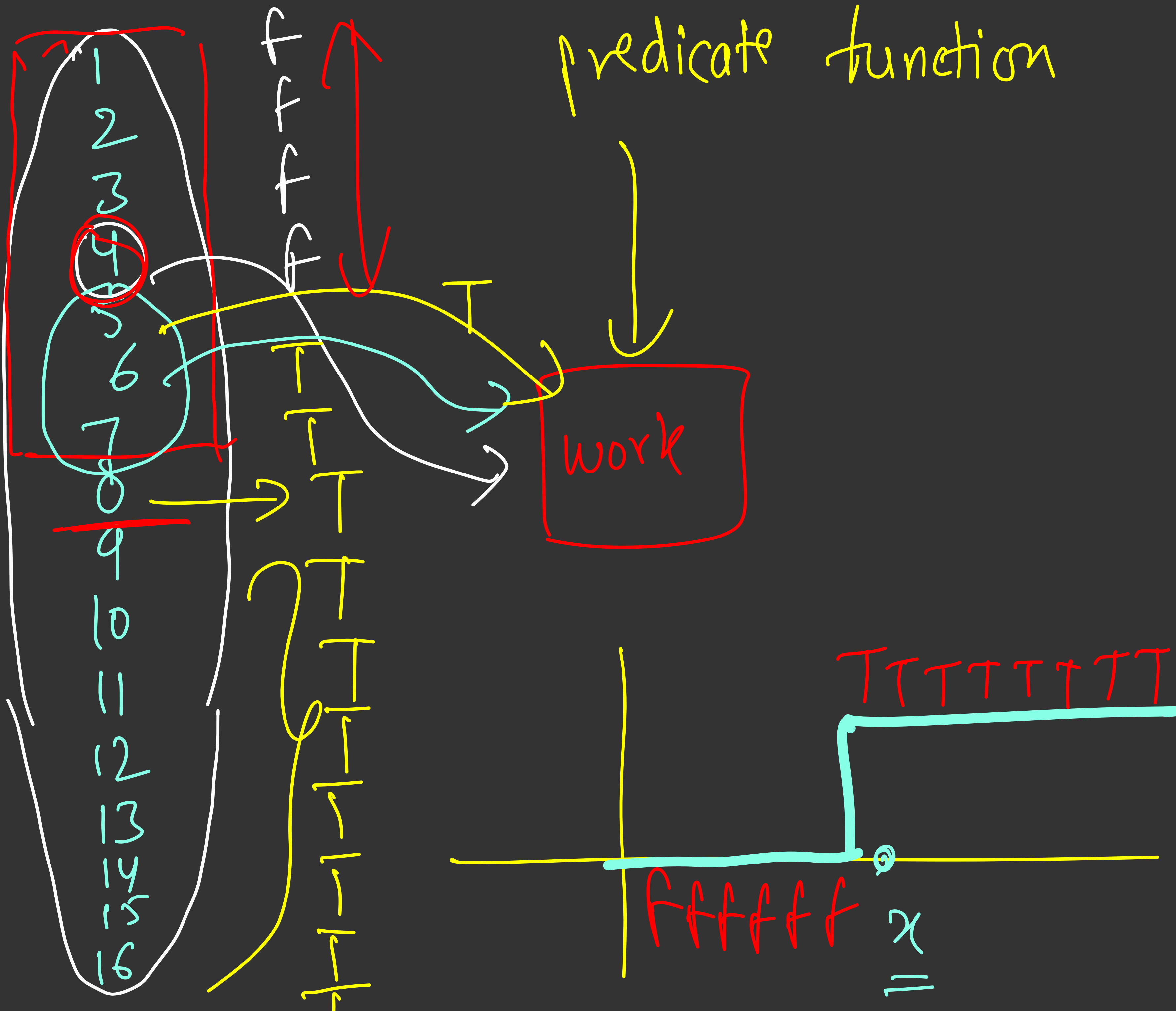
Given a task find out the min employees needed to complete the task


Search space
of answers
function
T
f

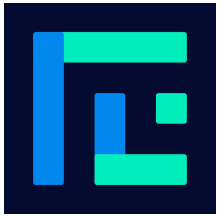

1 2 3 4 5 6
7 8 9 10 11 12
f(9)
T
f

Task T ⟶ min no. of employees required to get it done = 5

16 employees


predicate function
work
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
f f f f
T T T T T T T T T T T T
TTTTTTTT
ffffff
x

Monotonic Predicate Functions



1. TTTTTTTTTFFFFFFFF

T	T	T	T	F	F	F	F	F

2. FFFFFFFFTTTTTTTTT

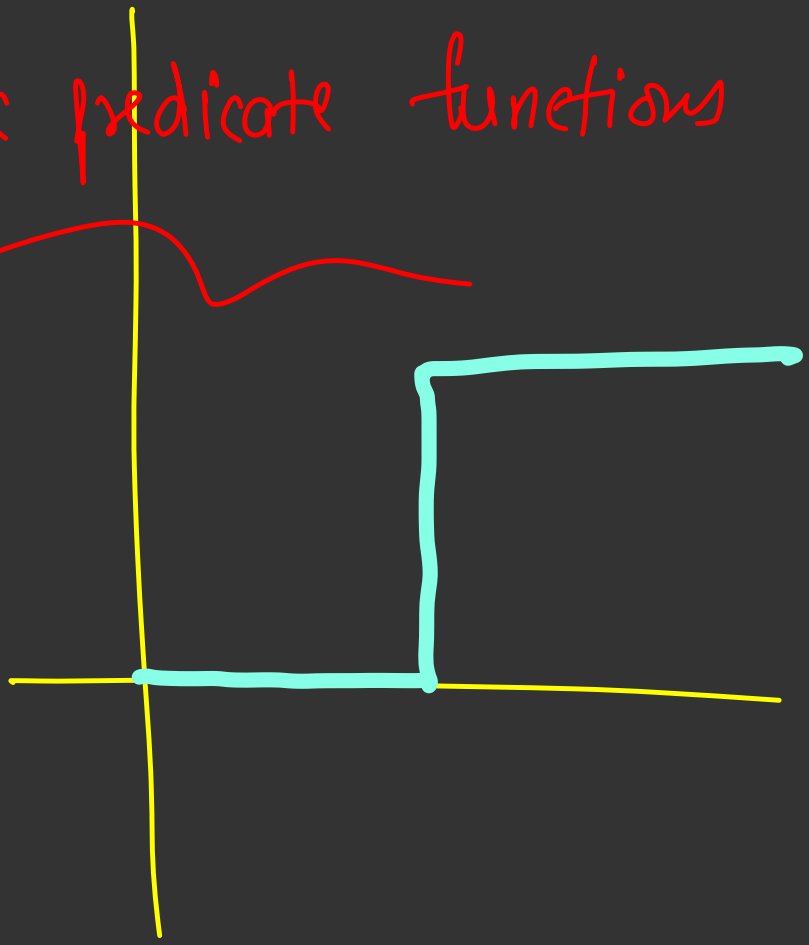
F	F	F	F	F	F	T	T	T

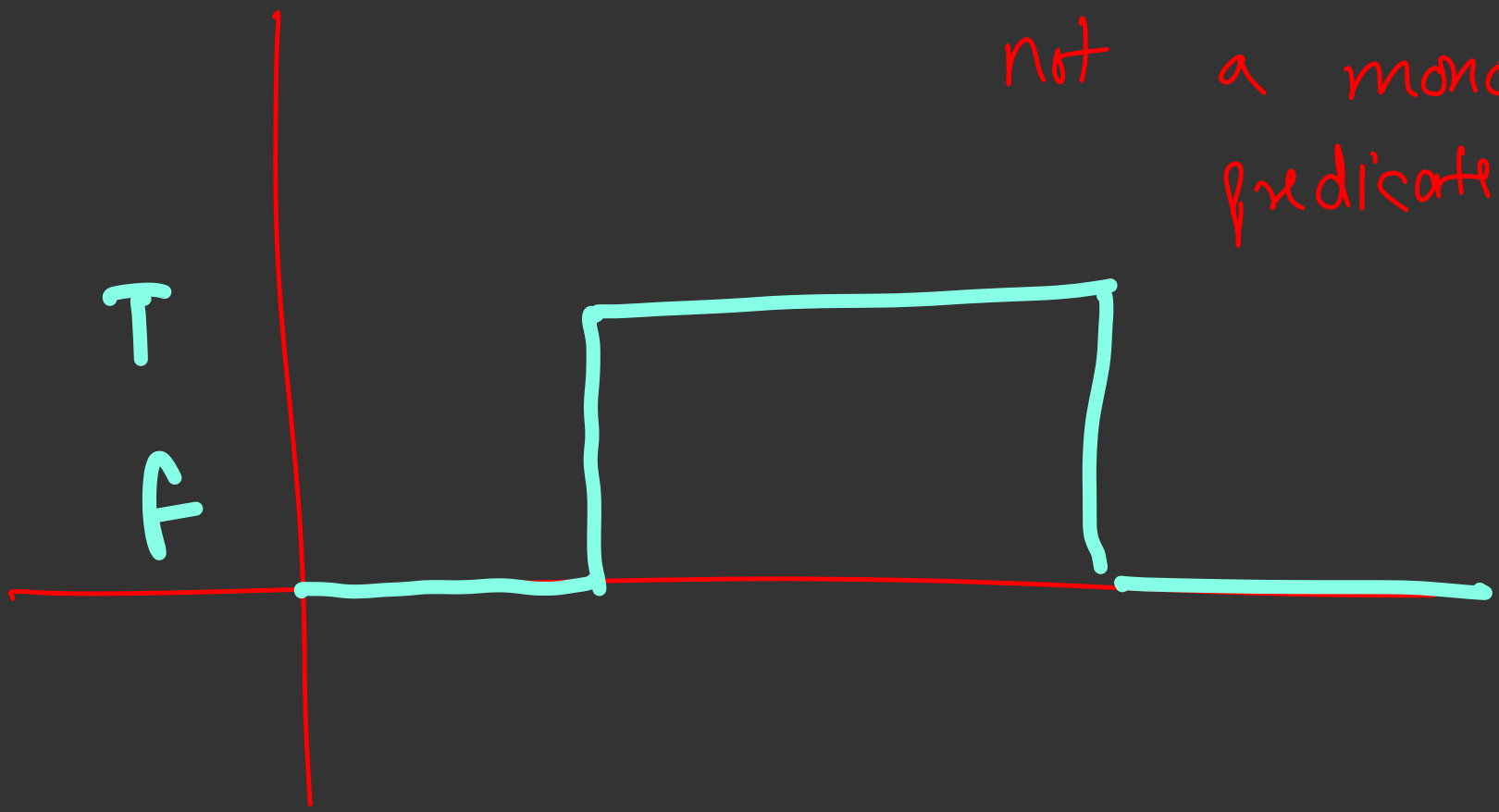
monotonicity

predicate function

True = 1 , false = 0

monotonic predicate functions

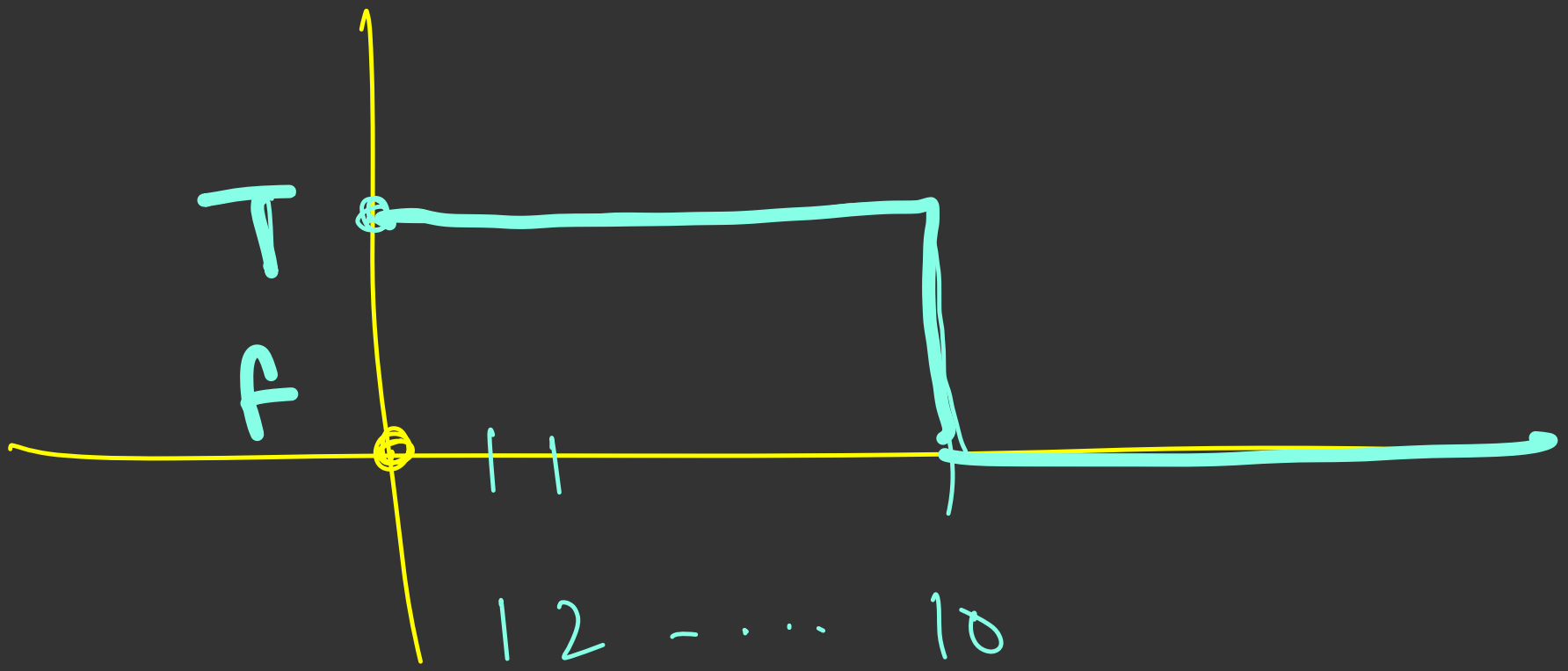




not a monotonic predicate function

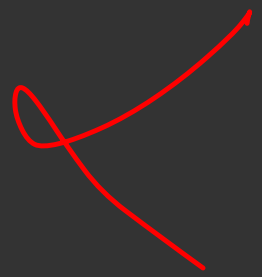
$f(x)$ = True if $x^2 \leq 100$

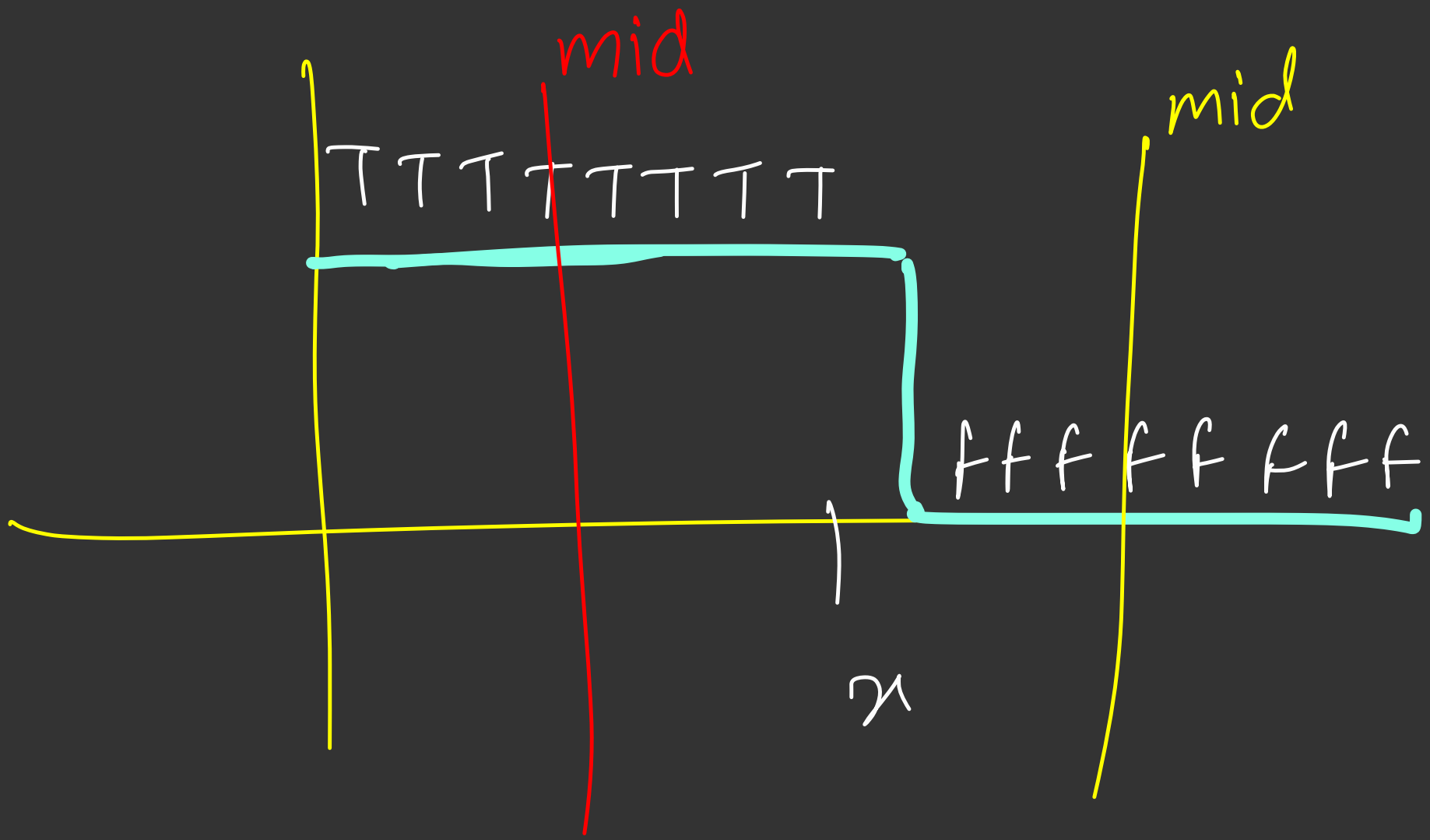
and false o/w



$f(x)$ = True if x is prime

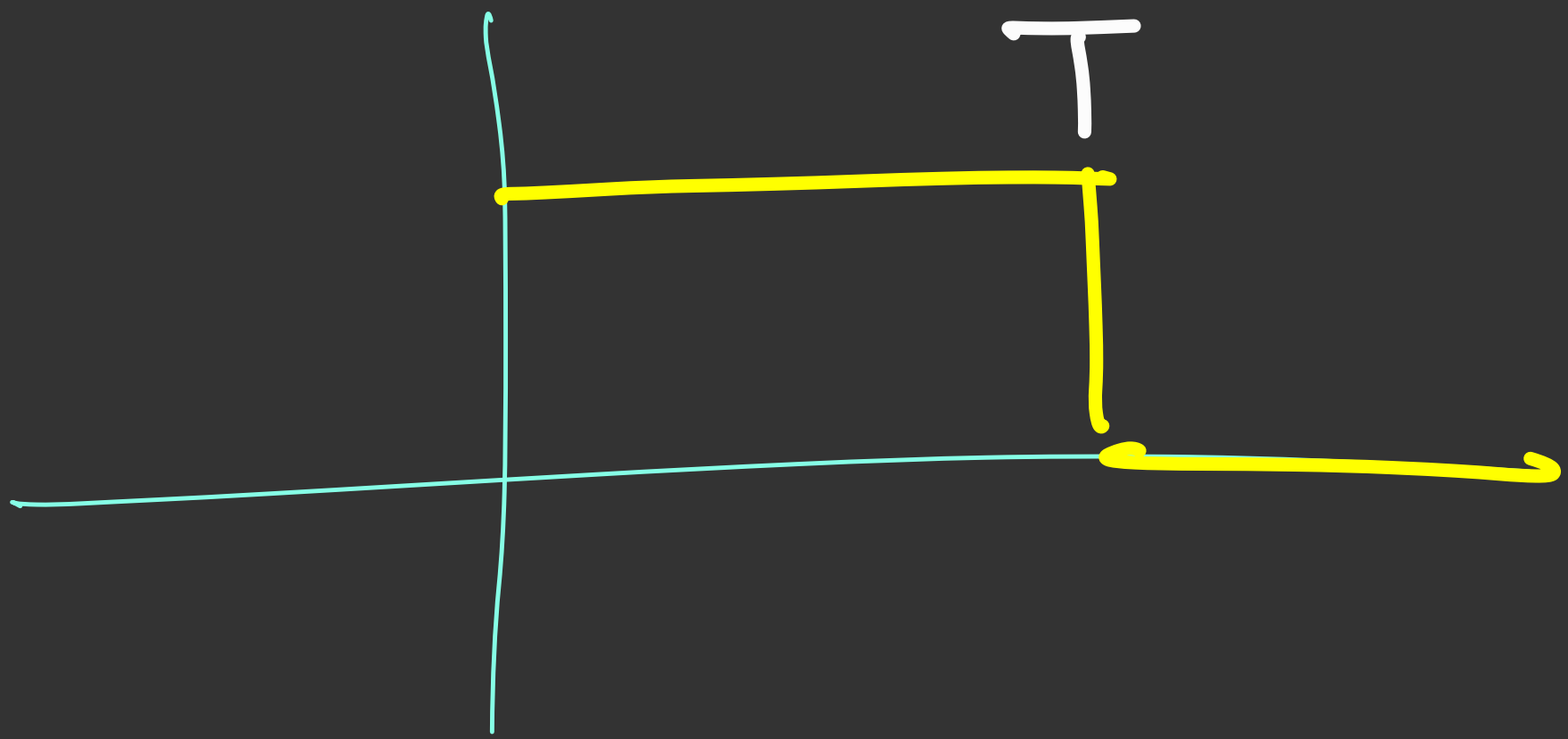
and false o/w



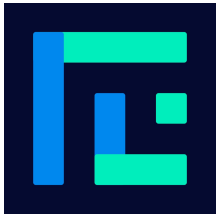


no. of elements in array $\leq x$

$=$ index of highest element $\leq x$


T

FairWorkload Problem



Given an array of workloads, split it among **K** workers, such that the maximum work that any worker has to do is minimised (can't change order of workloads).

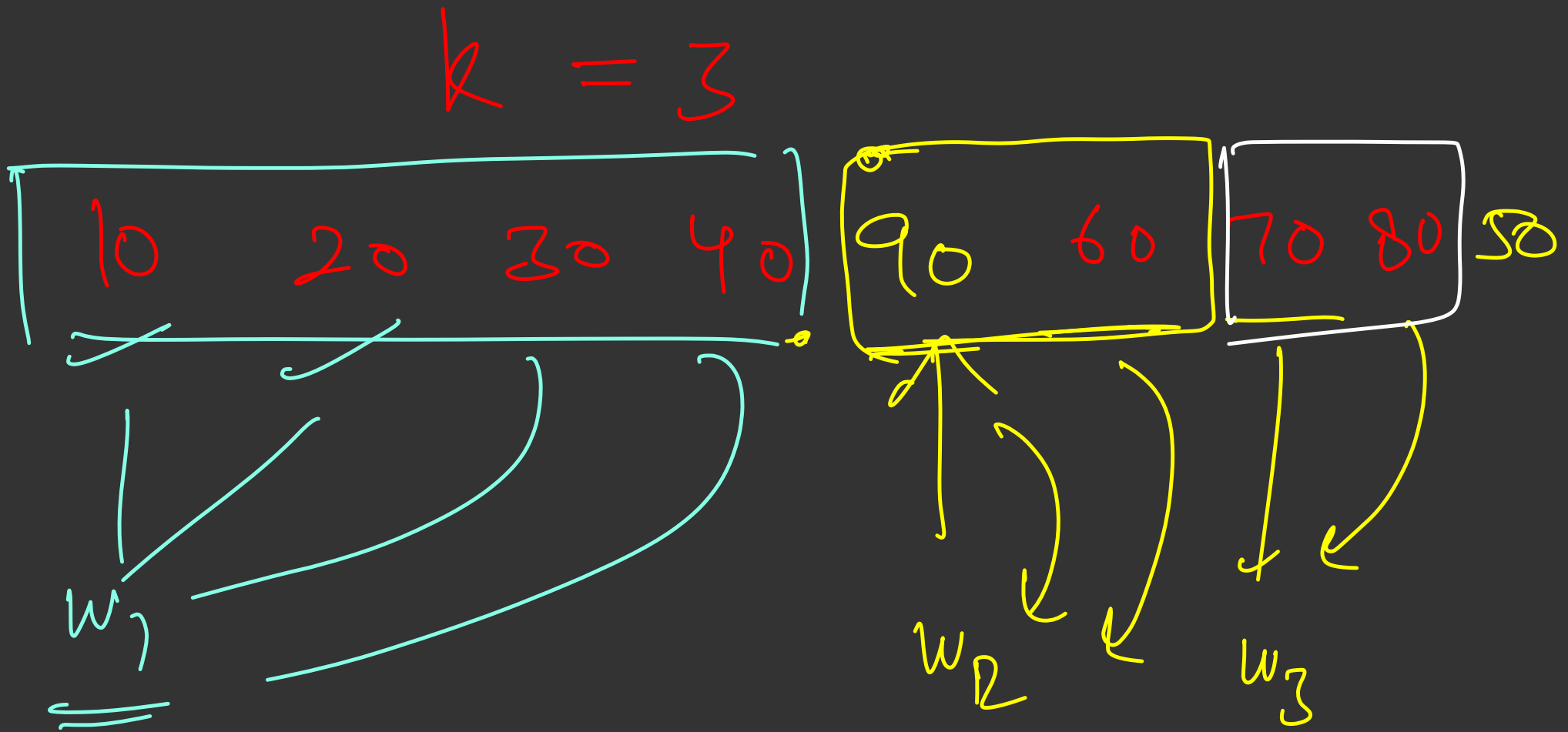
Eg. [10, 20, 30, 40, 50, 60, 70, 80, 90]

- Solution : 10 20 30 40 50 **|** 60 70 **|** 80 90

  First worker - 150, Second worker - 130, Third worker - 170

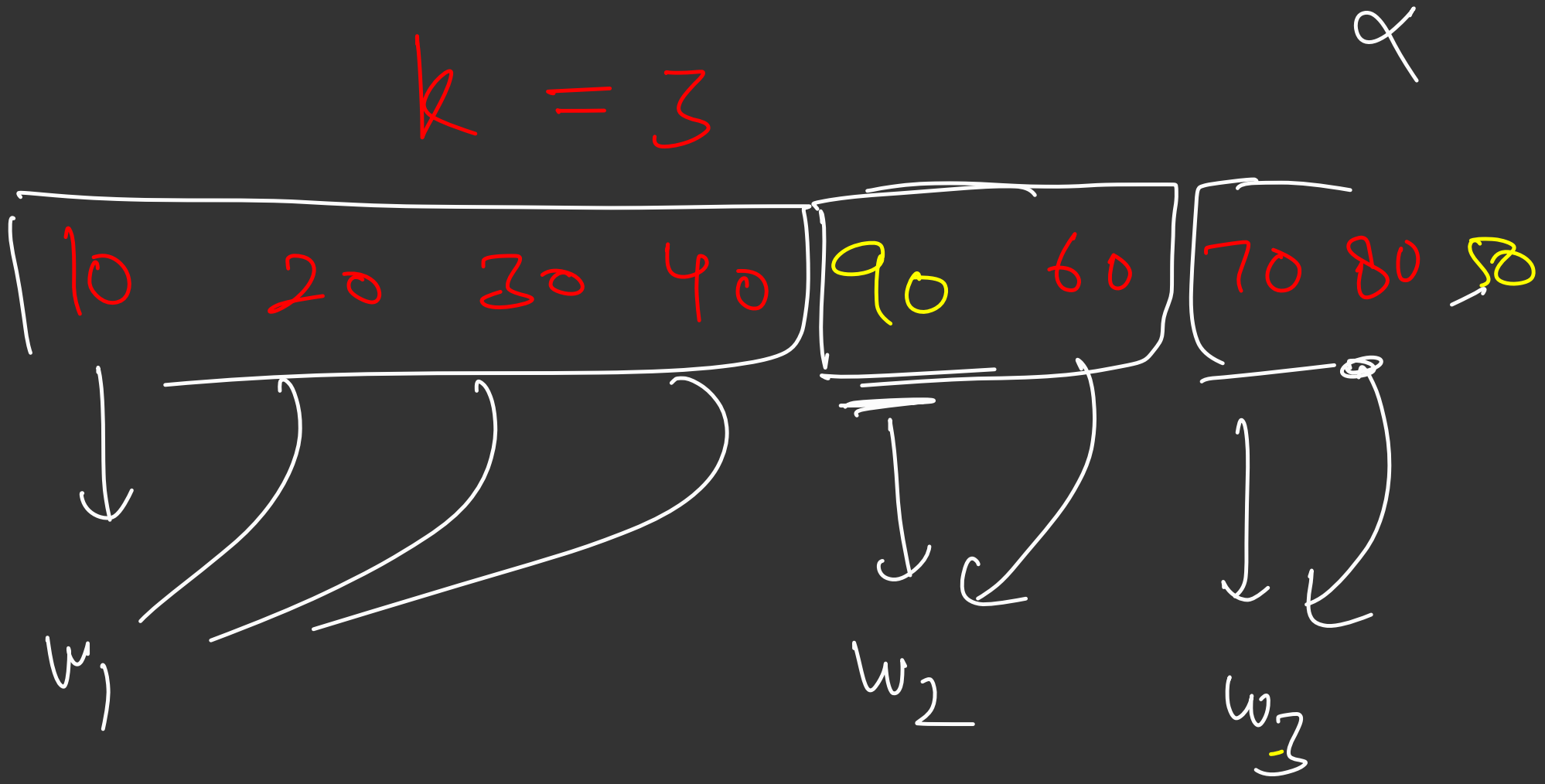
Is it possible to partition workload in a way that the highest workload of any worker is less than 170?

Let's assume the maximum work that anybody has to do = 150

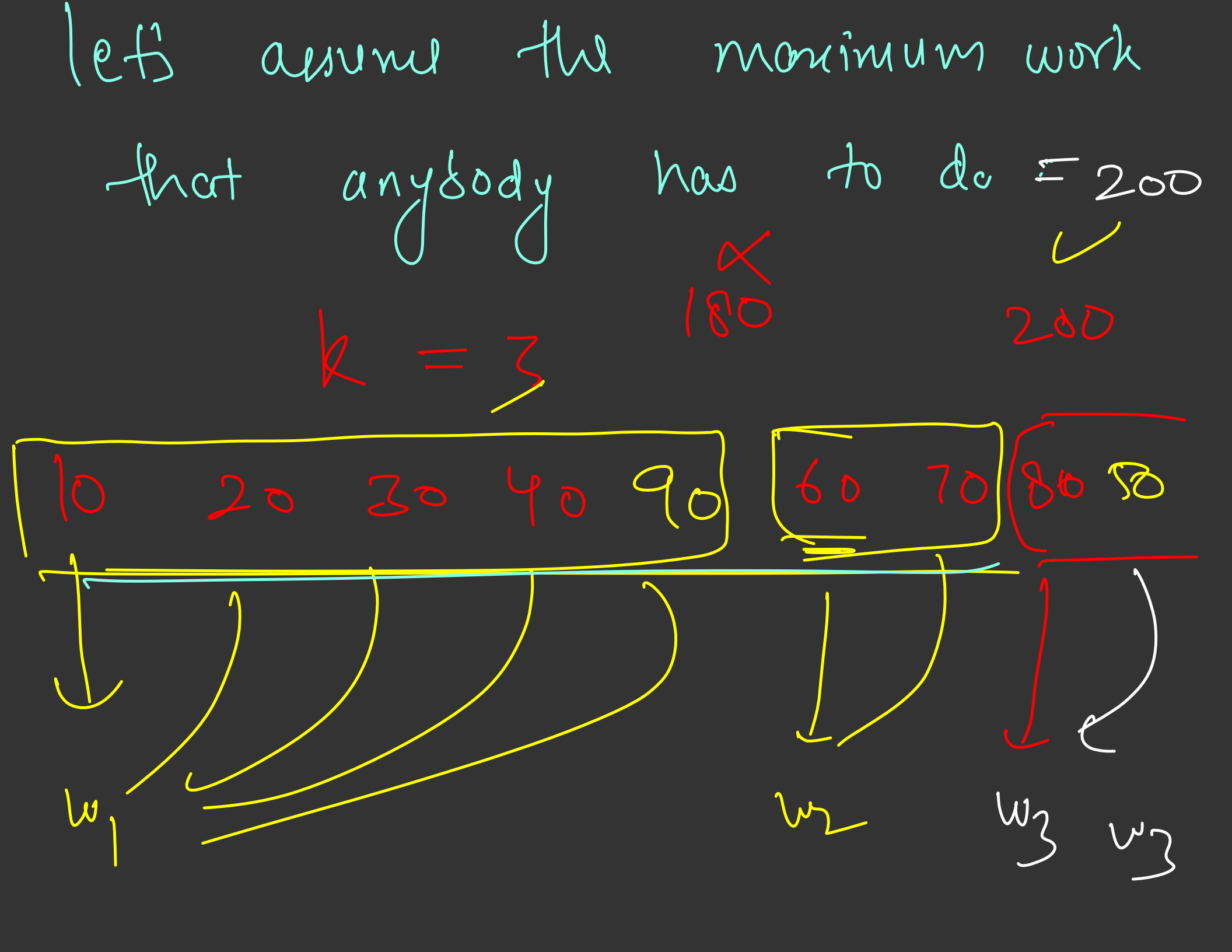


Let's assume the maximum work that anybody has to do = 180

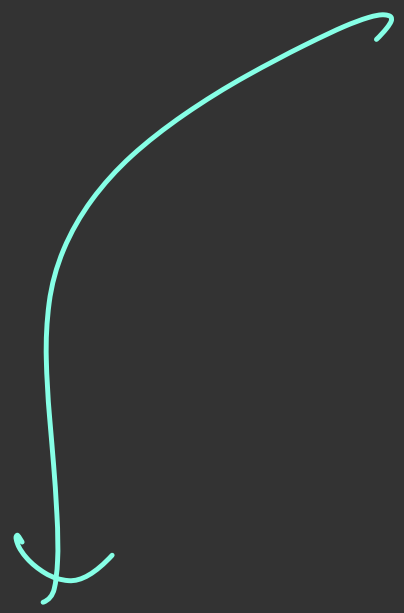
$k = 3$



Let's assume the maximum work that anybody has to do = 200



Time Complexity for Binary Searching problems



$O(\log_{\text{search space}} \times \text{Time to test 1 candidate})$

$O\left(\log(\text{sum of all elements}) \times O(n)\right)$

sum of all elements — max in the array

10 20 40 90 80

min → max of array

Max → sum of all elements