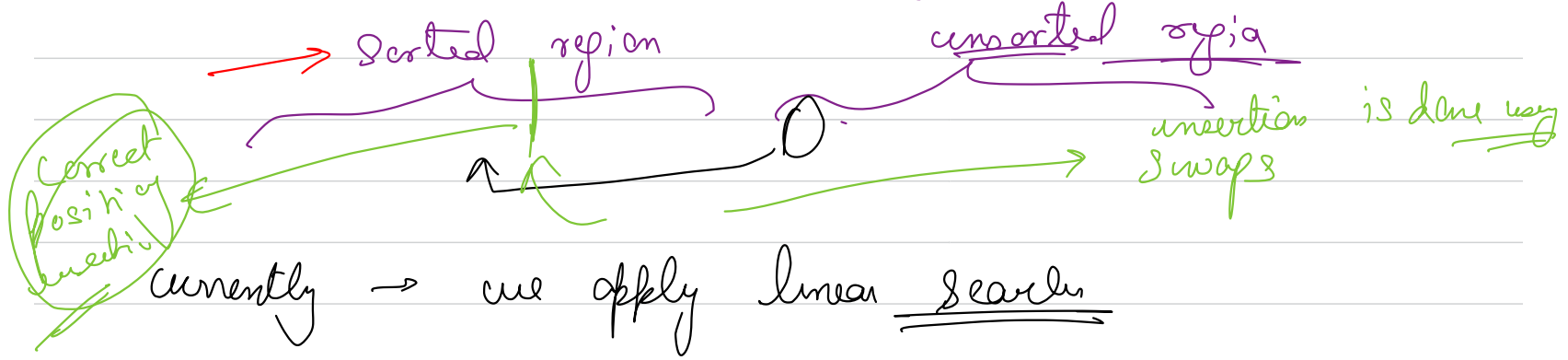
⇒ Insertion Sort → from the unsorted region, pick any element & search for it's best position in the sorted region & insert.




currently → we apply linear search

↳ optimize it by binary search ??

a) $n$

b) $n \log n$

c) $n^2$ ✓

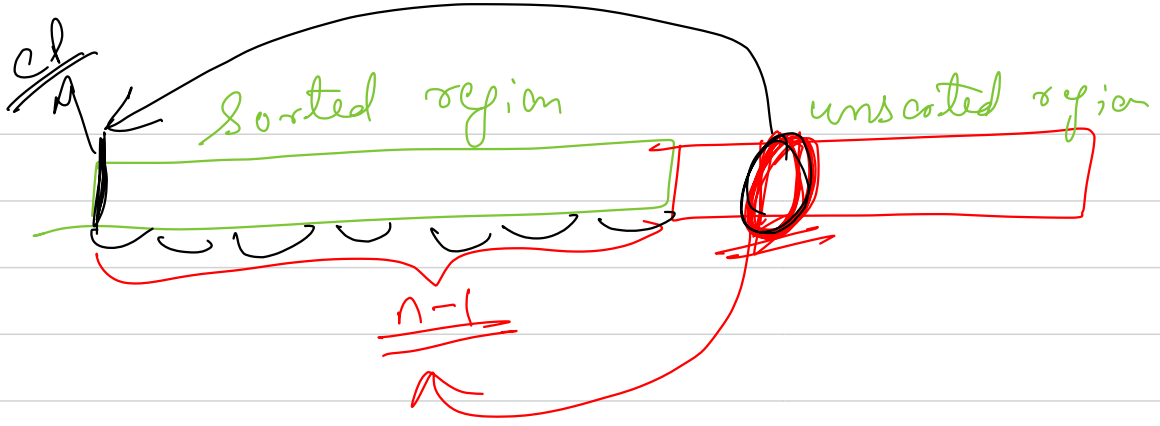
d) $n^2 \log n$

$O(n^2)$

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

$O(n \log n + n^2)$

$O(n^2)$




$O(n(\log n + n)) \approx O(n^2)$

search the correct pos

actually insert by swapping

Q → Given a sorted list and an integer $x$. Find the first index / value in the list which is greater than or equal to $x$.

→ [2, 3, 5, 6, 8, 10] n

$x = 4$

$a[idx] \geq x$ → first value

$n \leq 10^7$

ans ⇒ 5

Linear Search → $O(n)$

if $a[mid] \geq X$ → then there is a possibility $a[mid]$ can be or cannot be the first value.

ans

possible candidate

if $a[mid] < X$

→ Neither mid nor left of mid will be ans ever

bisect library in python → to use binary search

- bisect_left(t) → the leftmost possible index to insert in the list such that list is still sorted.
- bisect_right(t) → But for rightmost possible index

t → element

Q. Given an unsorted list, find any element $a[i]$ such that $a[i] > a[i-1]$ and $a[i] > a[i+1]$

if $(i+1)$ → out of bound

There can be multiple $a[i]$, find any one of them

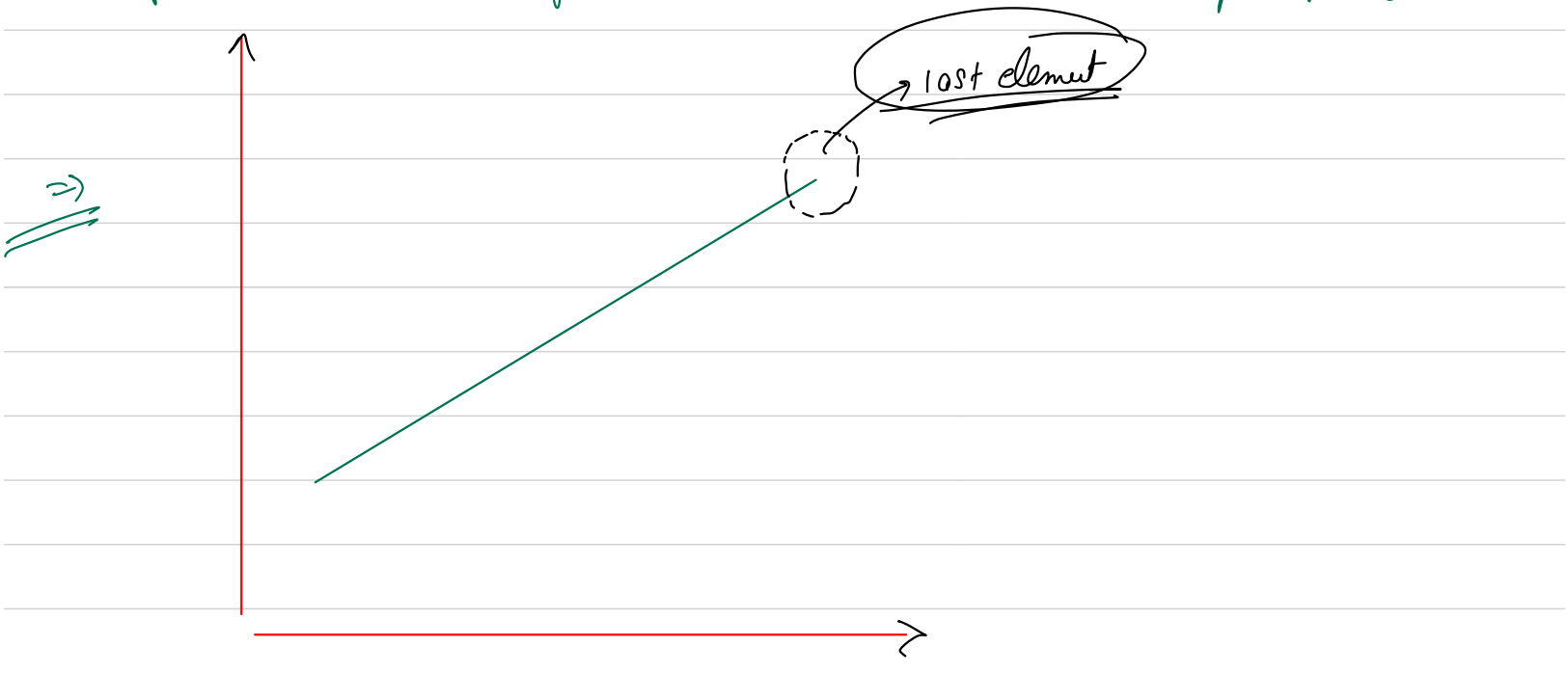
Return the index

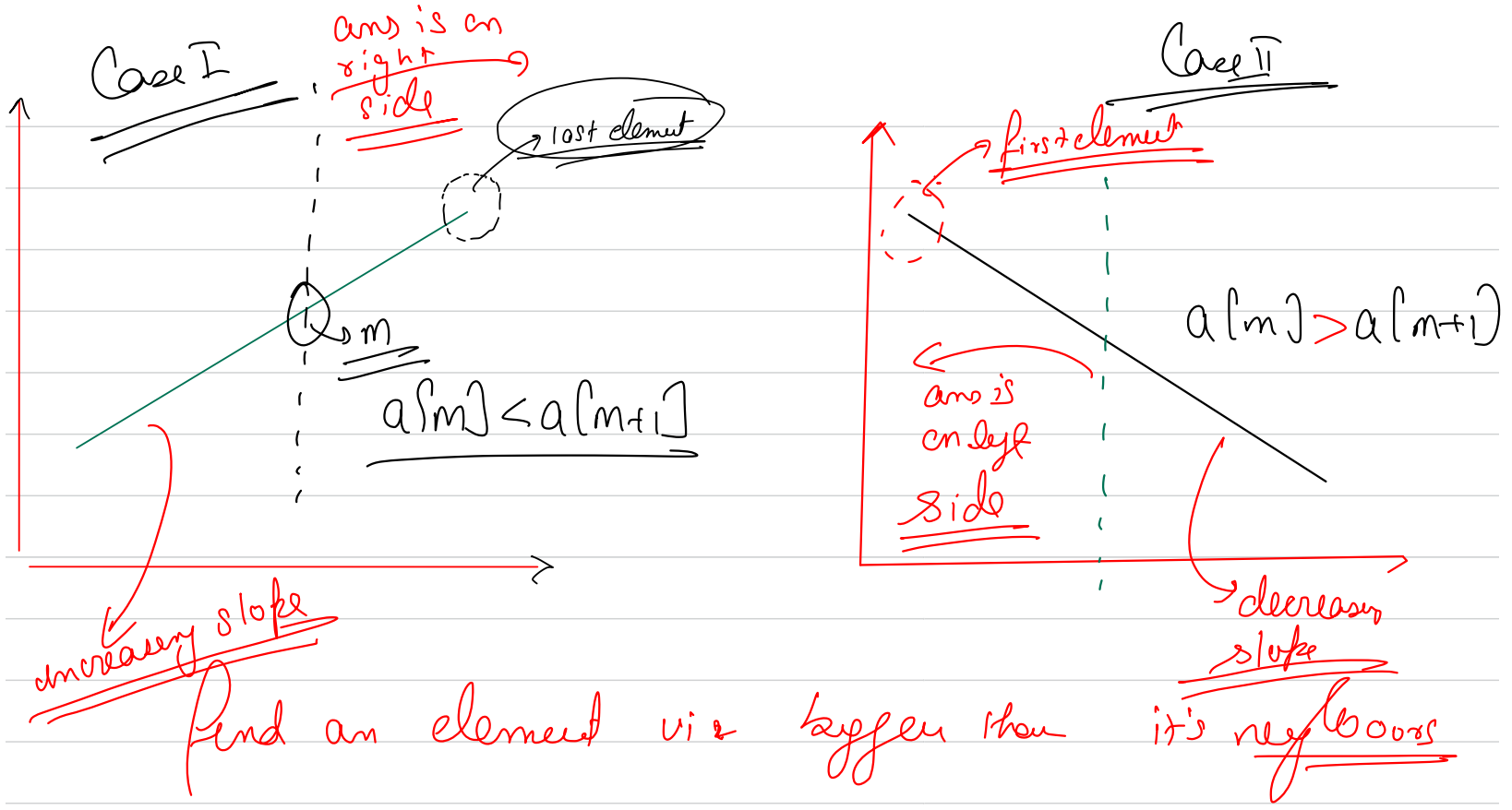
$[1, 2, 3, 1]$

ans → 2 ← index

do something better than $O(n)$

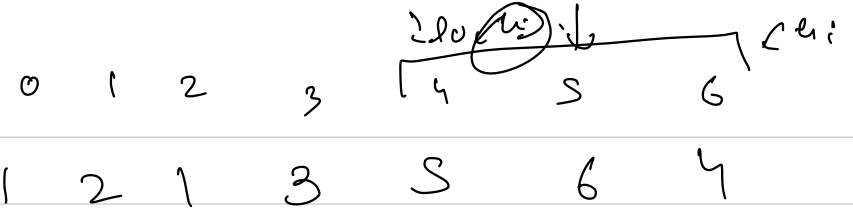
Whether we are able to divide our search space into 2 parts based on some property or not.






Find an element which is bigger than it's neighbours

$O(\log_2 n) \approx O(\log n)$




lo = 0, hi = 6

mid → 3 → lo = 4 hi = 6

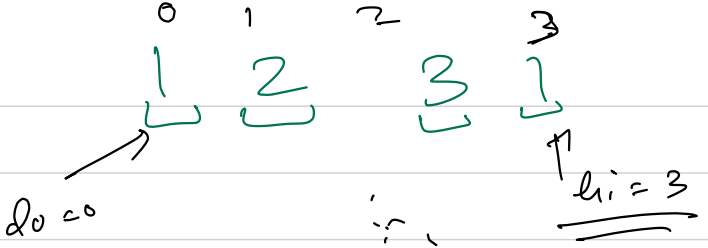
mid → 5 → lo = 4 hi = 5

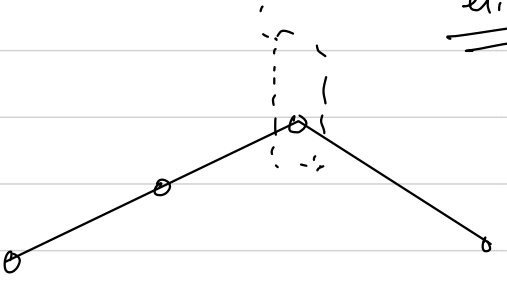
mid → 4 → lo = 5, hi = 5

hi = mid − 1 → hi = mid

lo → mid

hi → mid-1




mid → 1 → lo = 1 hi = 3

mid → 2 lo = 1 hi = 2

mid → 1 lo = 01 → hi = 2