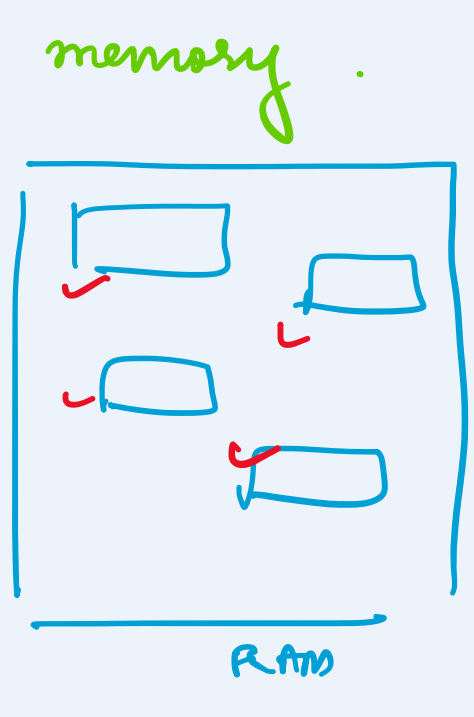
Friday, 13 August 2021 8:24 PM

Arrays

→ non-primitive / derived data types

50 students

50 variables ⇒ 50 placeholders in memory.




```
String name1 = "ganesh"
String name2 = "rahul"
   :
String name50 = "____";
```

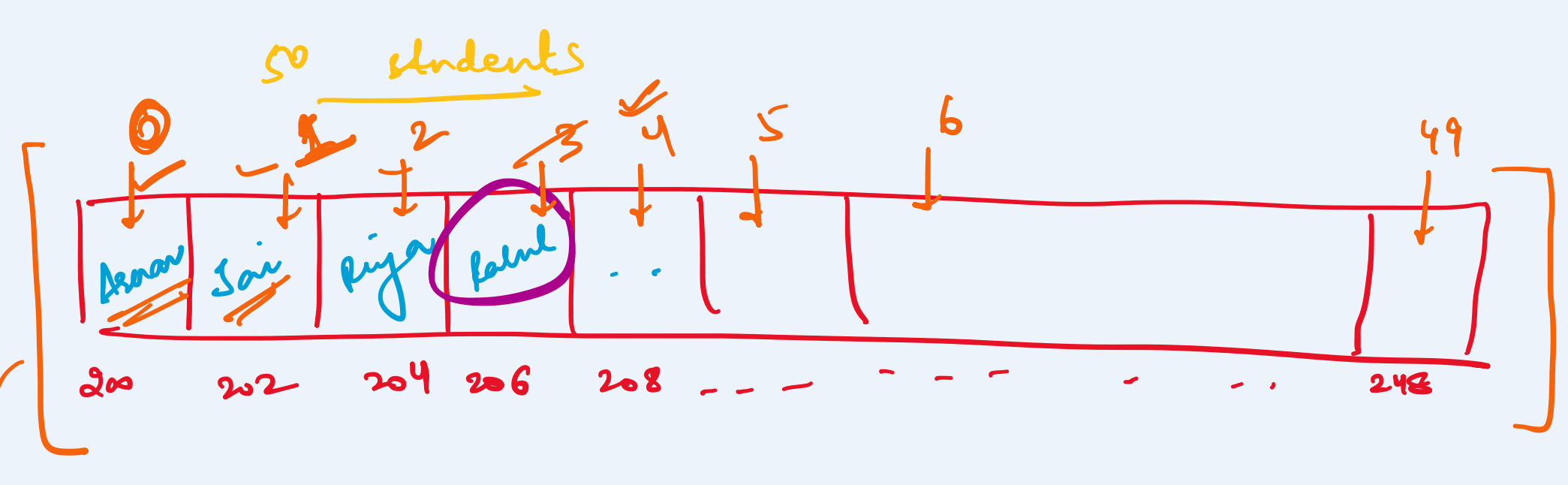
→ Homogeneous

→ keep track of 50 variables

30,000 students → 30 k variable names

- → homogeneous data
- → contiguous memory locations

50 students



students (string)

Indexing

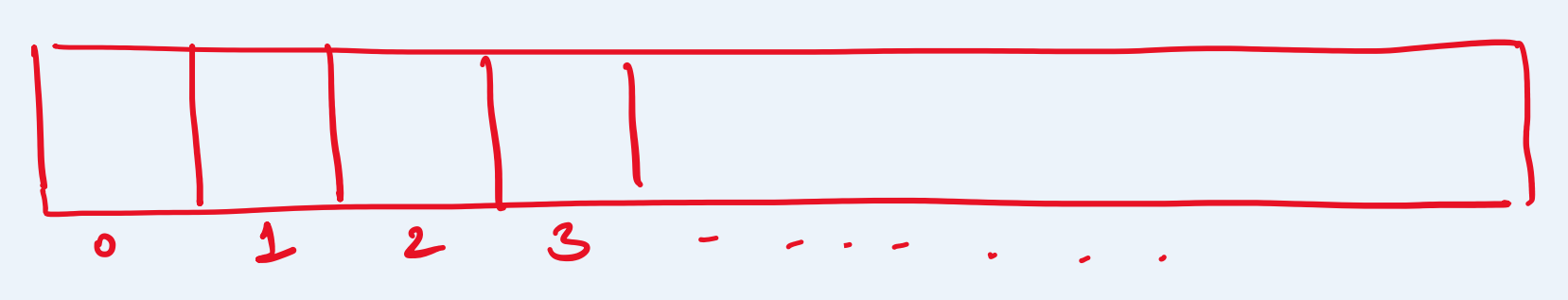
50 students ⟶ [0, ⟶ 49]

students[30] = "Rishav"

→ whenever we are declaring an array, we need to declare its size in the start itself.

Arrays / 1-D Arrays

X-axis



Array → fixed size

Syntax

```
int [] arr = new int [size];
```

int → data type; arr → name of the array; int (after new) → datatype

[I@6996db8 → [I → array is of Integers; 6996db8 → address of memory location; arr is pointing to an array

Q Given an array, print the max ele of the array.

32	43	6	91	7	22

output → 91

-2, -3, -4

max = -∞ / arr[0]

```
for (int i=0; i< arr.length; i++) {
    if (arr[i] > max) {
        max = arr[i];
}
```

Q Reverse the array

Eg: 10 | 20 | 30 | 40 | 50

ans → 50 | 40 | 30 | 20 | 10

left, right

~~10~~ 50, ~~20~~ 40, 30, ~~40~~ 20, ~~50~~ 10

Two pointer

```
while (left <= right) {
    swap(left, right)
    left++;
    right--;
}
```

~~10~~ 60, ~~20~~ 50, ~~30~~ 40, ~~40~~ 30, ~~50~~ 20, ~~60~~ 10

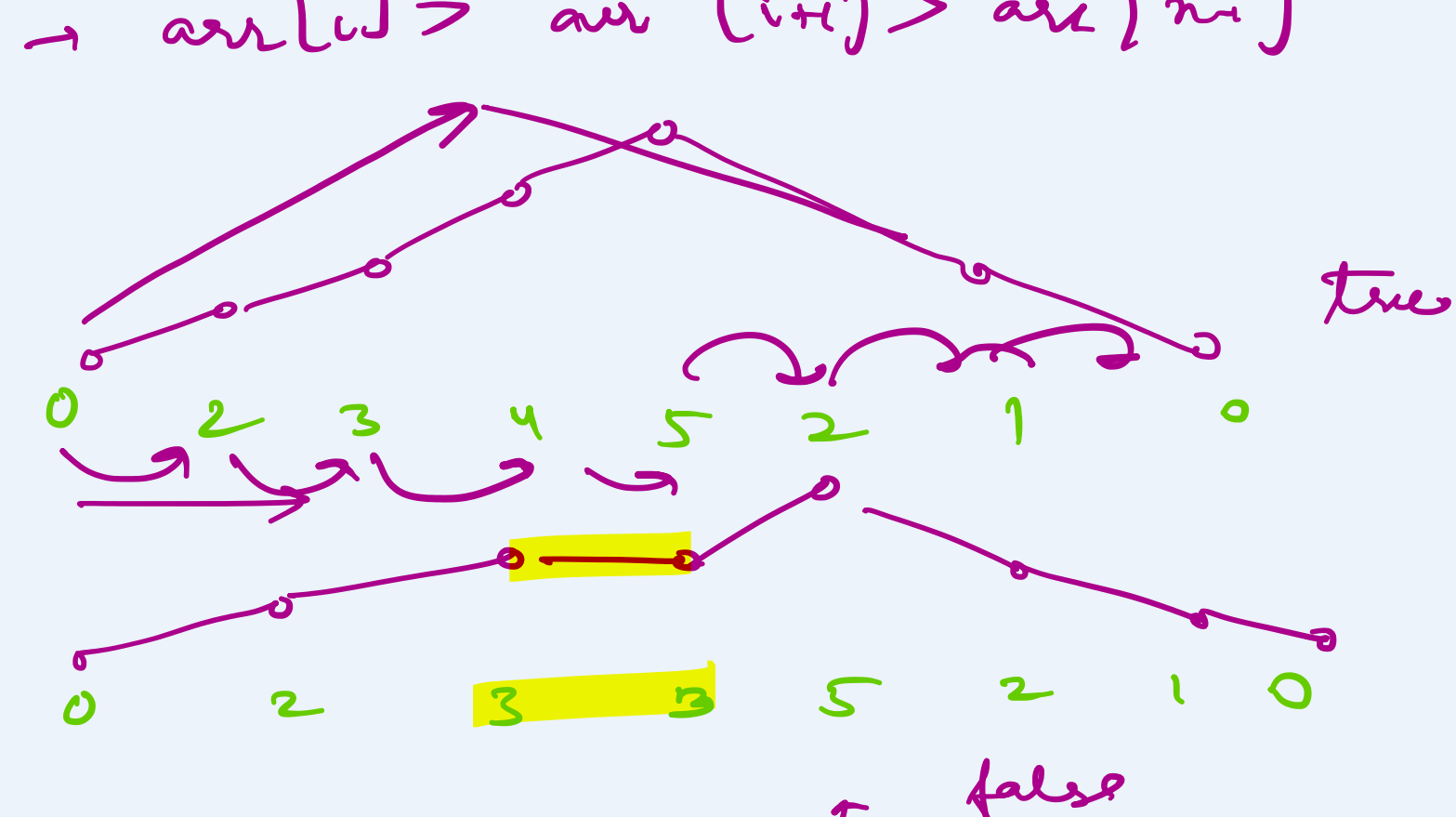
0 1 2 3 4 5

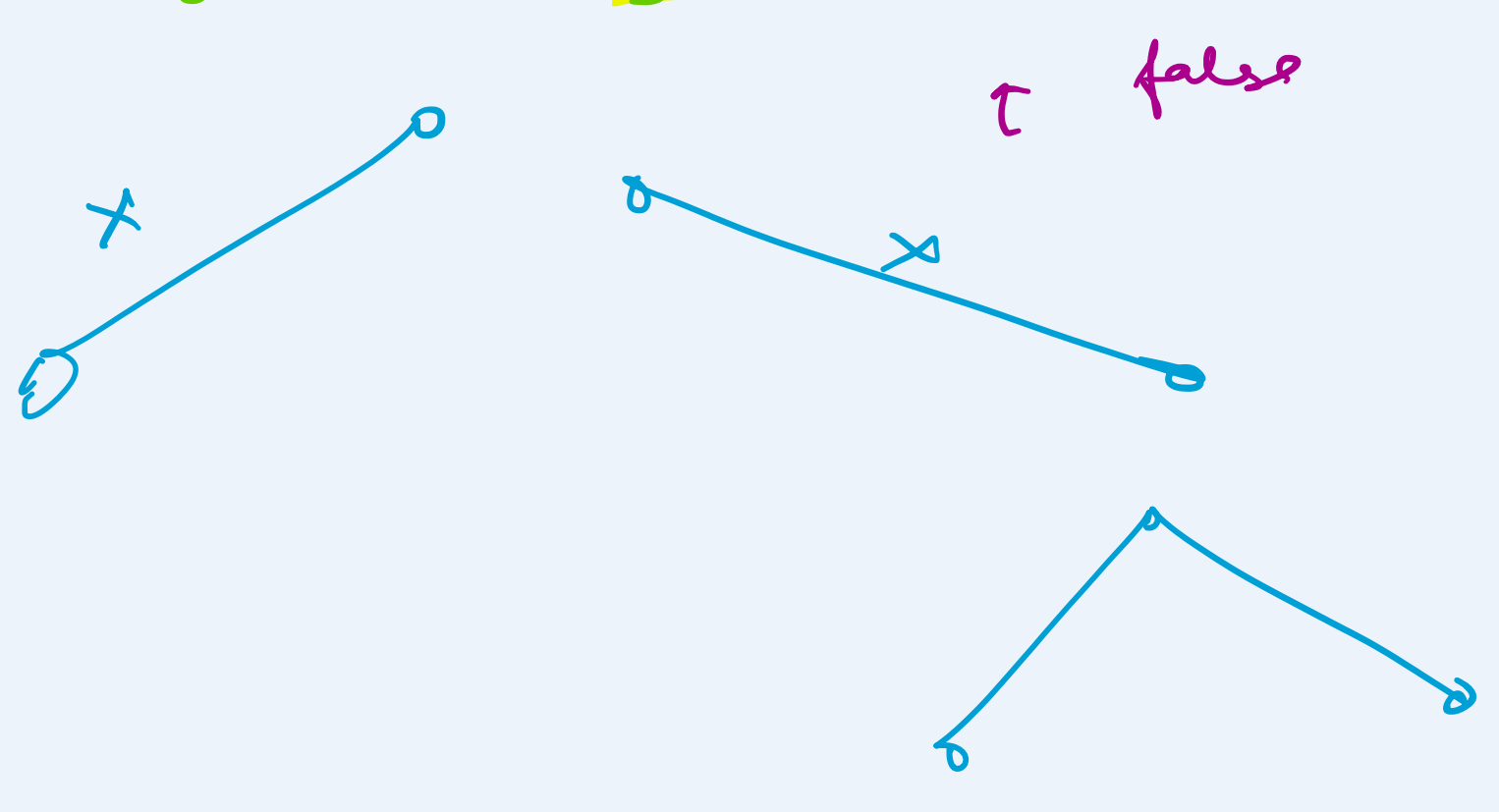
Q Valid mountain array

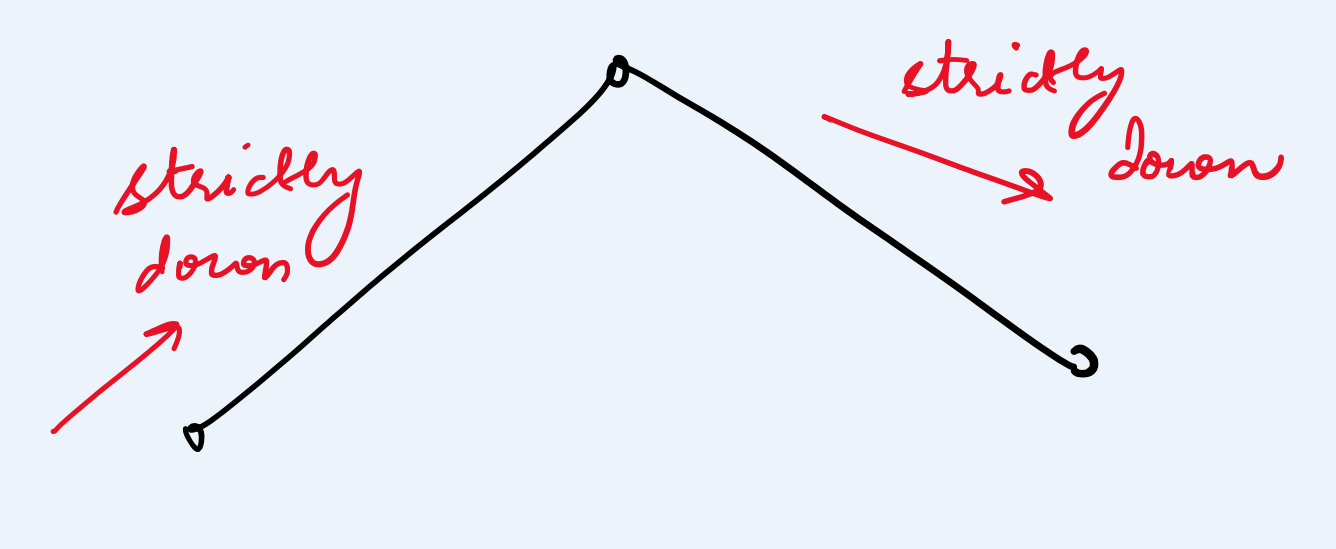
Given an array of integers, return true, if and only if it is a valid mountain array.

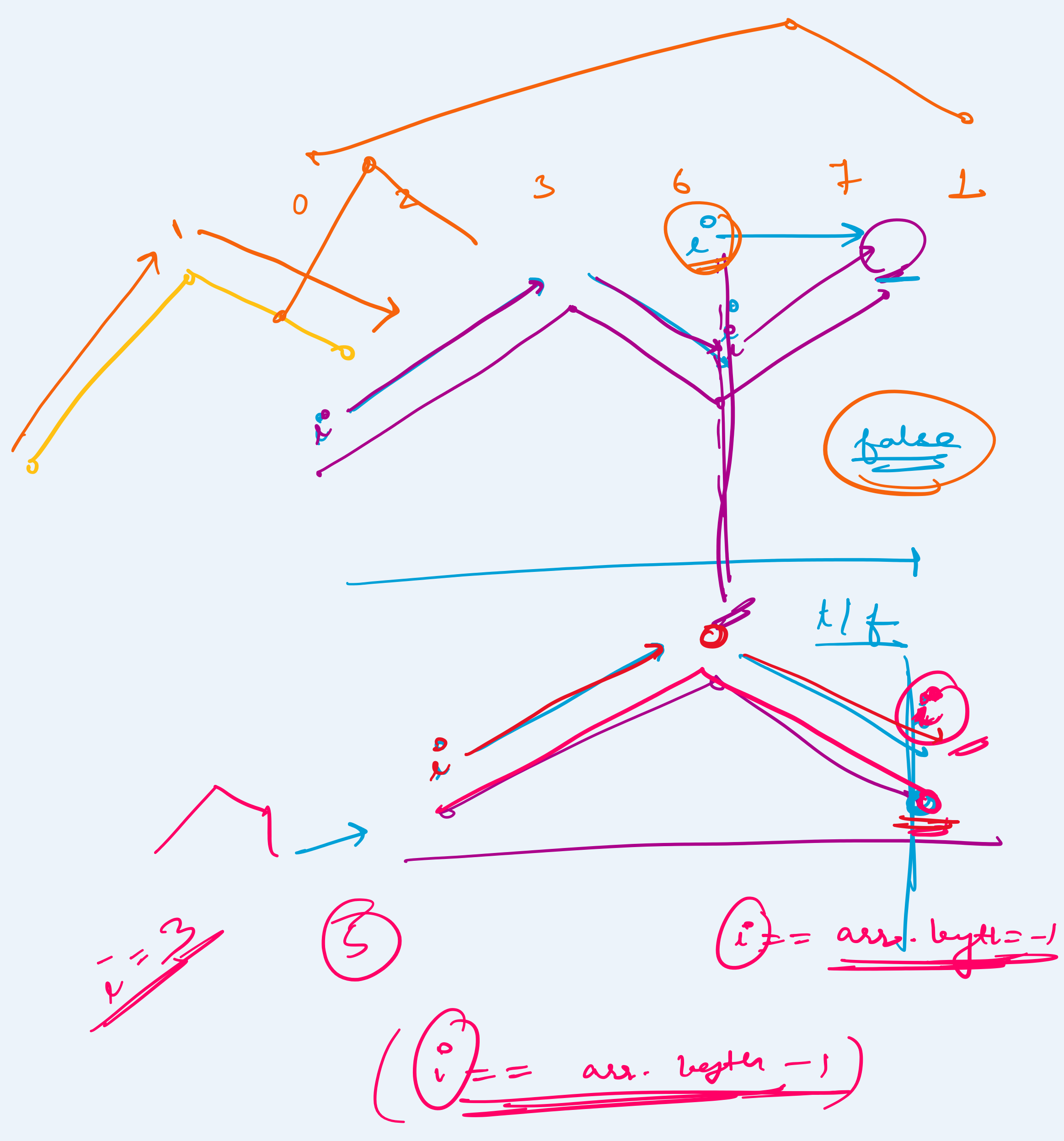
arr is a mountain array if.

1. arr.length >= 3
2. There exists some i for
   - → arr[0] < arr[1] < arr[2] ... arr[i-1] < arr[i]
   - → arr[i] > arr(i+1) > arr[n-1]









(i == arr.length - 1)

Q Given an array, return the answer array such that ans[i] is equal to the product of all elements of arr except arr[i].

Eg: arr = [1 2 3 4]

output = [24, 12, 8, 6]

24 → 2×3×4; 12 → 1×3×4; 8 → 1×2×4; 6 → 1×2×3

→ %, /, operator ×