How to build loc^n based Systems

⇒ Swiggy

↳ find restaurants near me

↳ find delivery partners near restaurant

⇒ Uber

↳ find drivers near me

⇒ Tinder

↳ find potential partners near you.

find restaurants near ⇒ static ⇒ Today

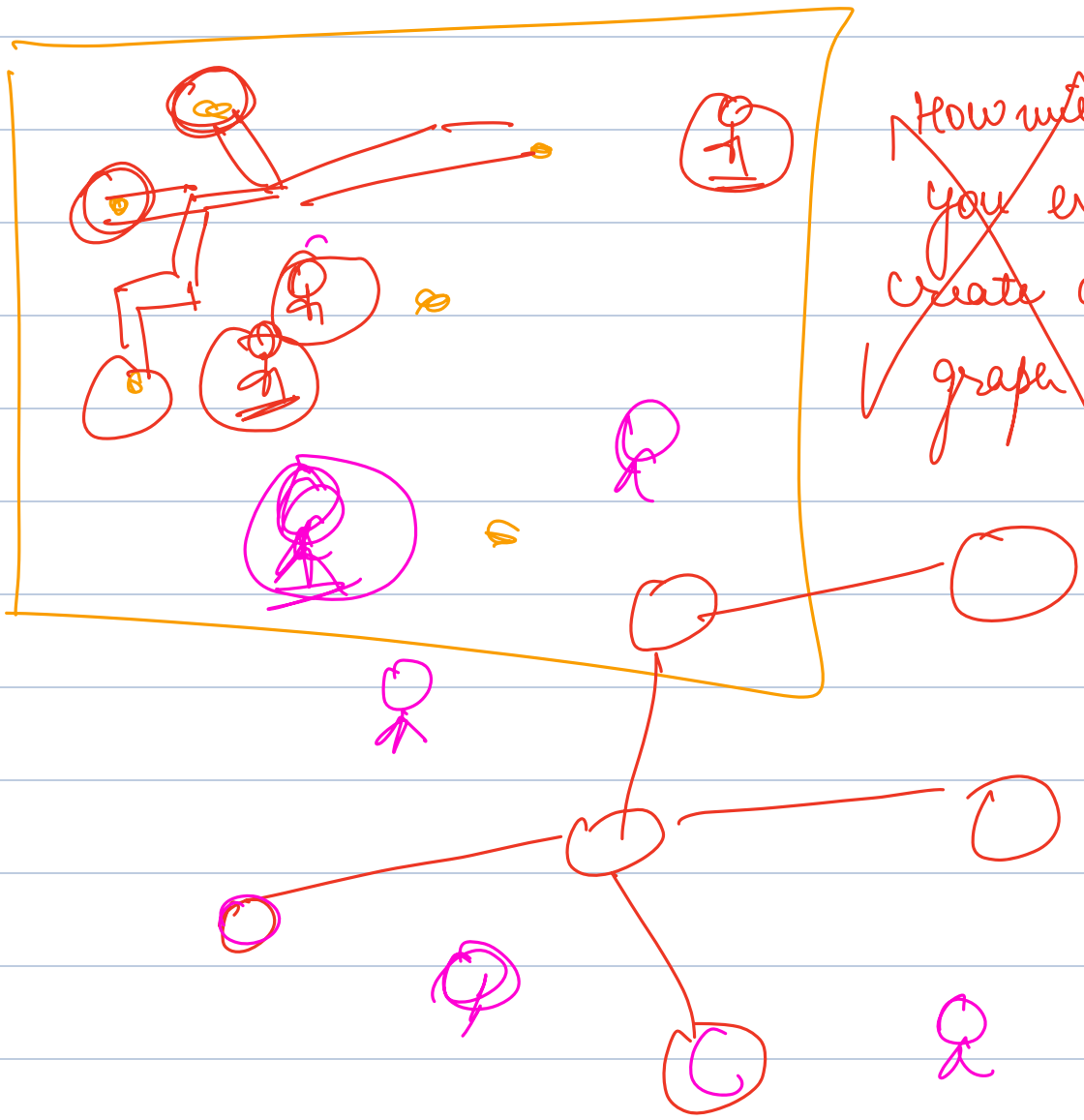
find drivers near me ⇒ dynamic ⇒ Next Class.

List<Restaurant> getRestaurants (loc^n)

$O(V+2E)$

quadratic

$\rightarrow O(N^2)$



How will you even create a graph

restaurants

id	name	loc^n

(x_1, y_1)

get 5 nearest restaurants to me.

getRestaurant(loc^n)

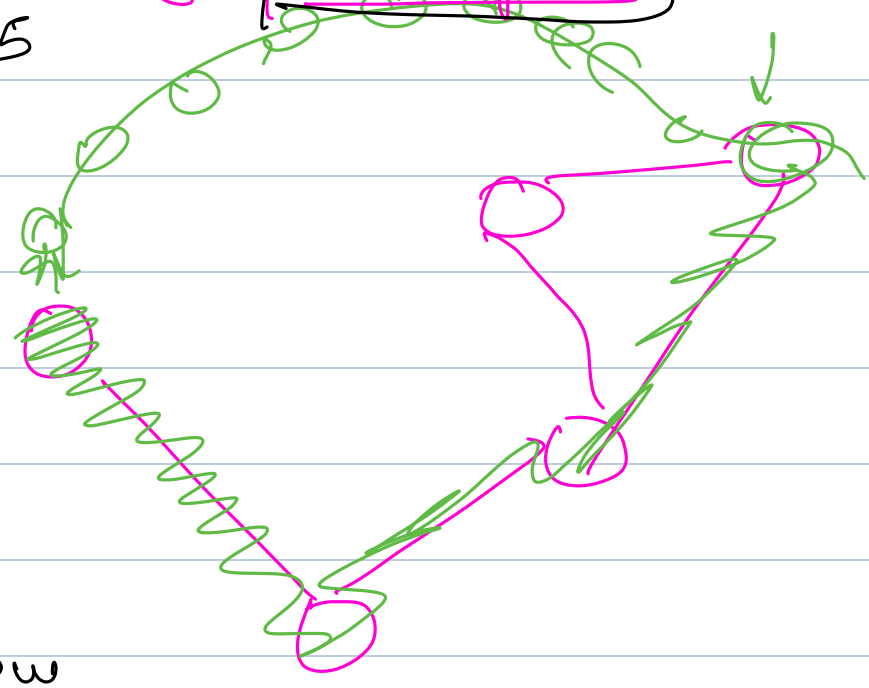
```
distances = [ ]
for each restaurant:
    dis = findDis(restaurant, human)
    distances.add(dis, restaurant)
distances.sort()
return top5
```

$\rightarrow O(N)$

$\rightarrow O(1)$

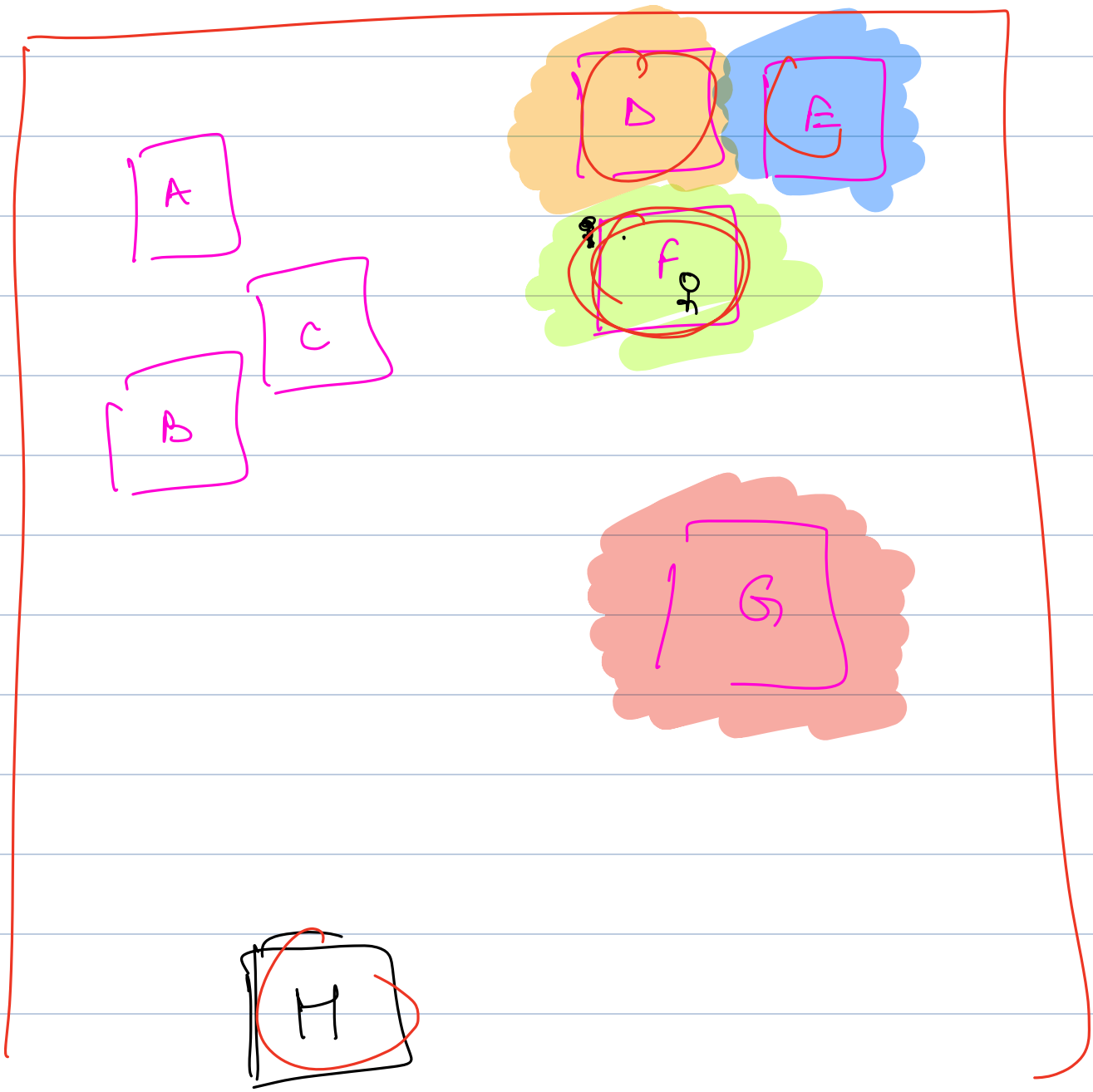
$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

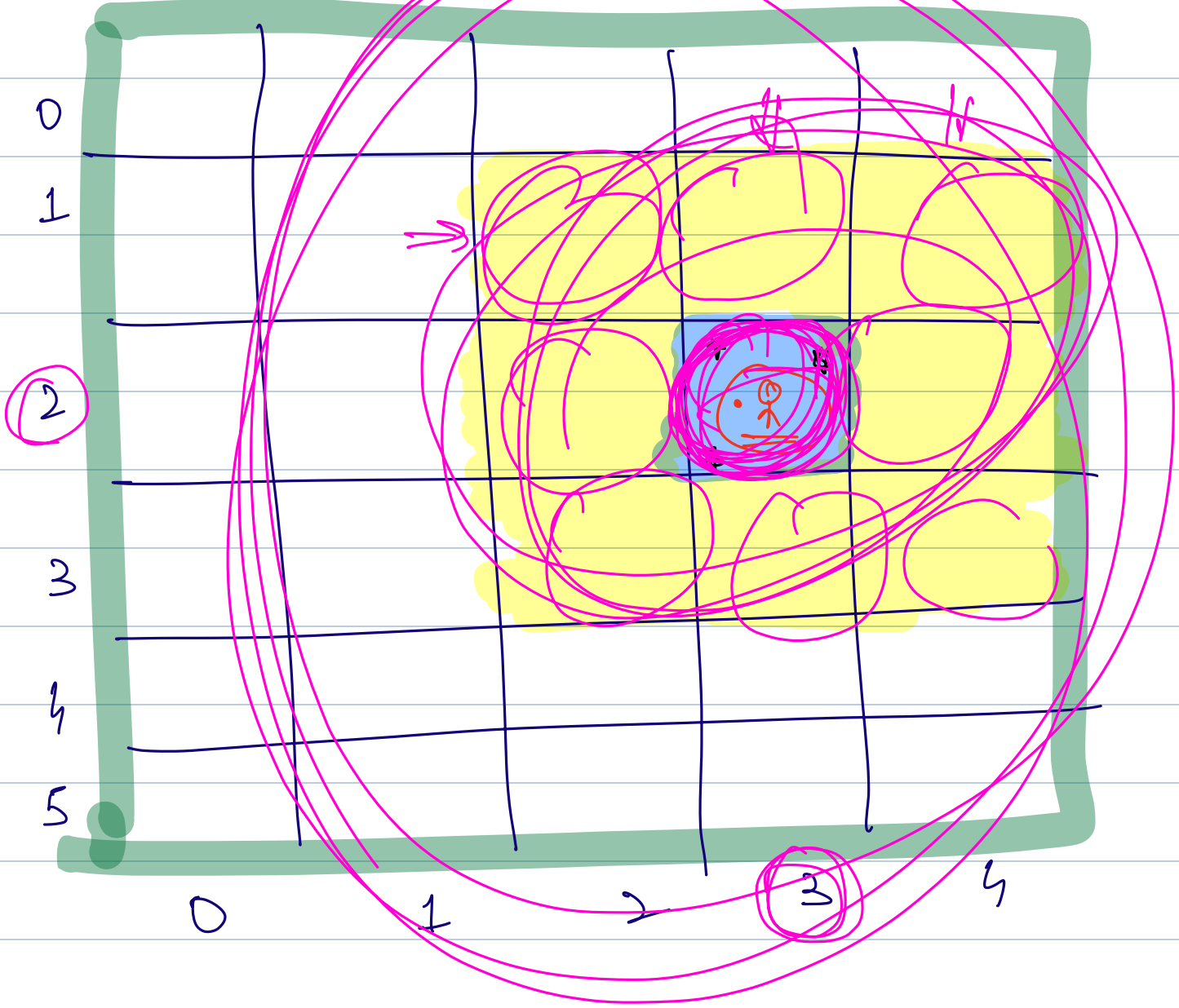
$\rightarrow O(N \log N)$



V.V. Slow

Assume we are living in a society





NEW COLUMN

restaurants

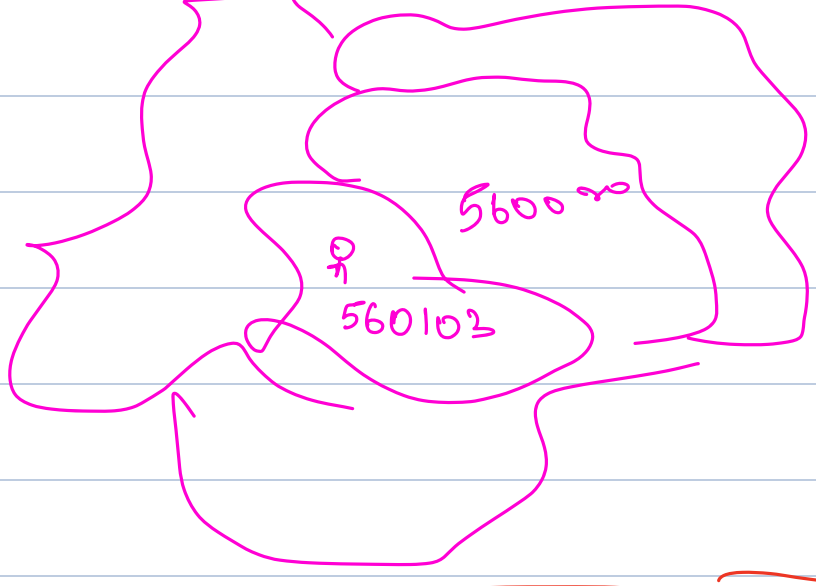
id	title	loc^n	grid-cell

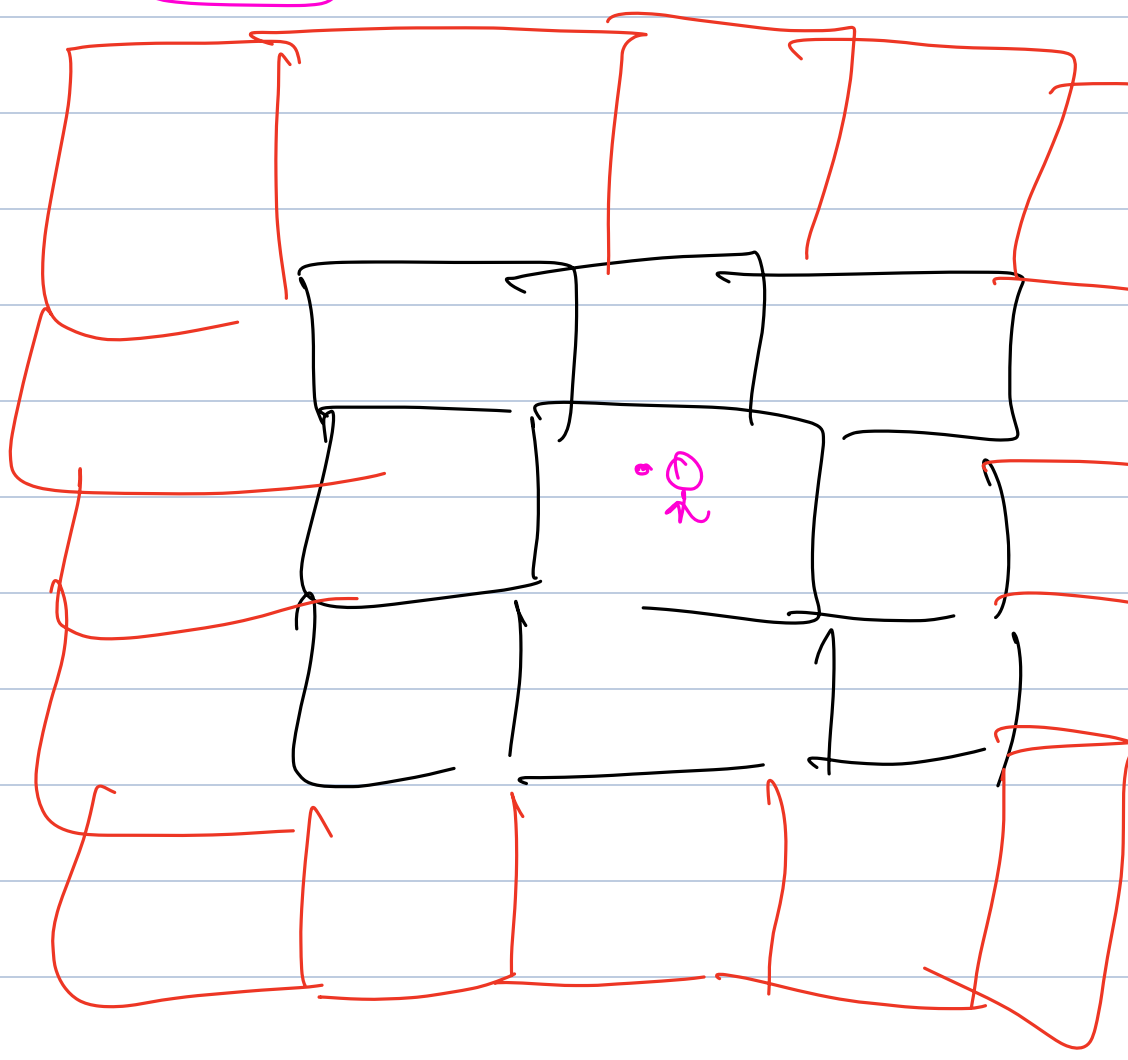
rests = Select * from restaurants where grid = (2, 3)

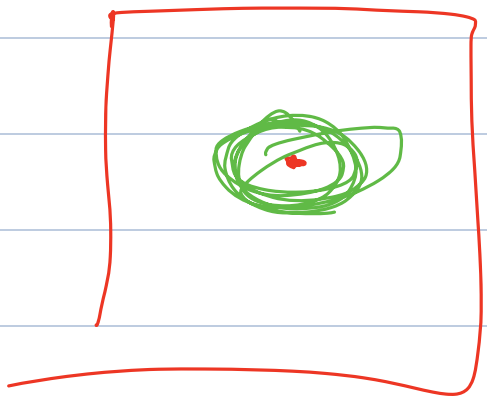
if rests.size() ≥ 100
Sort and return.

Select * from restaurants
where grid in (
(1, 2)
(1, 3)
(1, 4)
|
|
)

Now if total ≥ 100:
sort and return


5600 m
560103





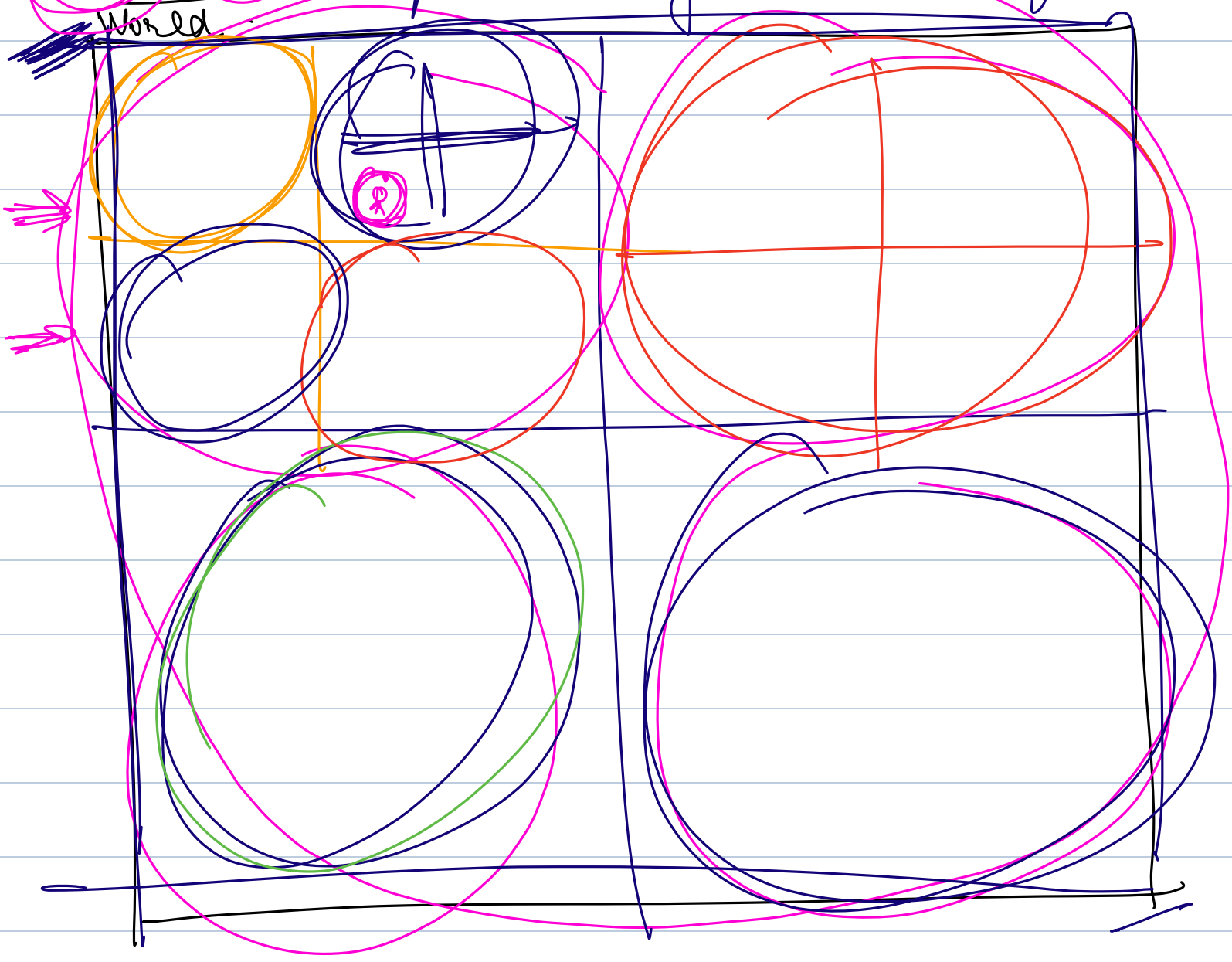
Above one is a great direction.
But, isn't equally performant
for everyone.

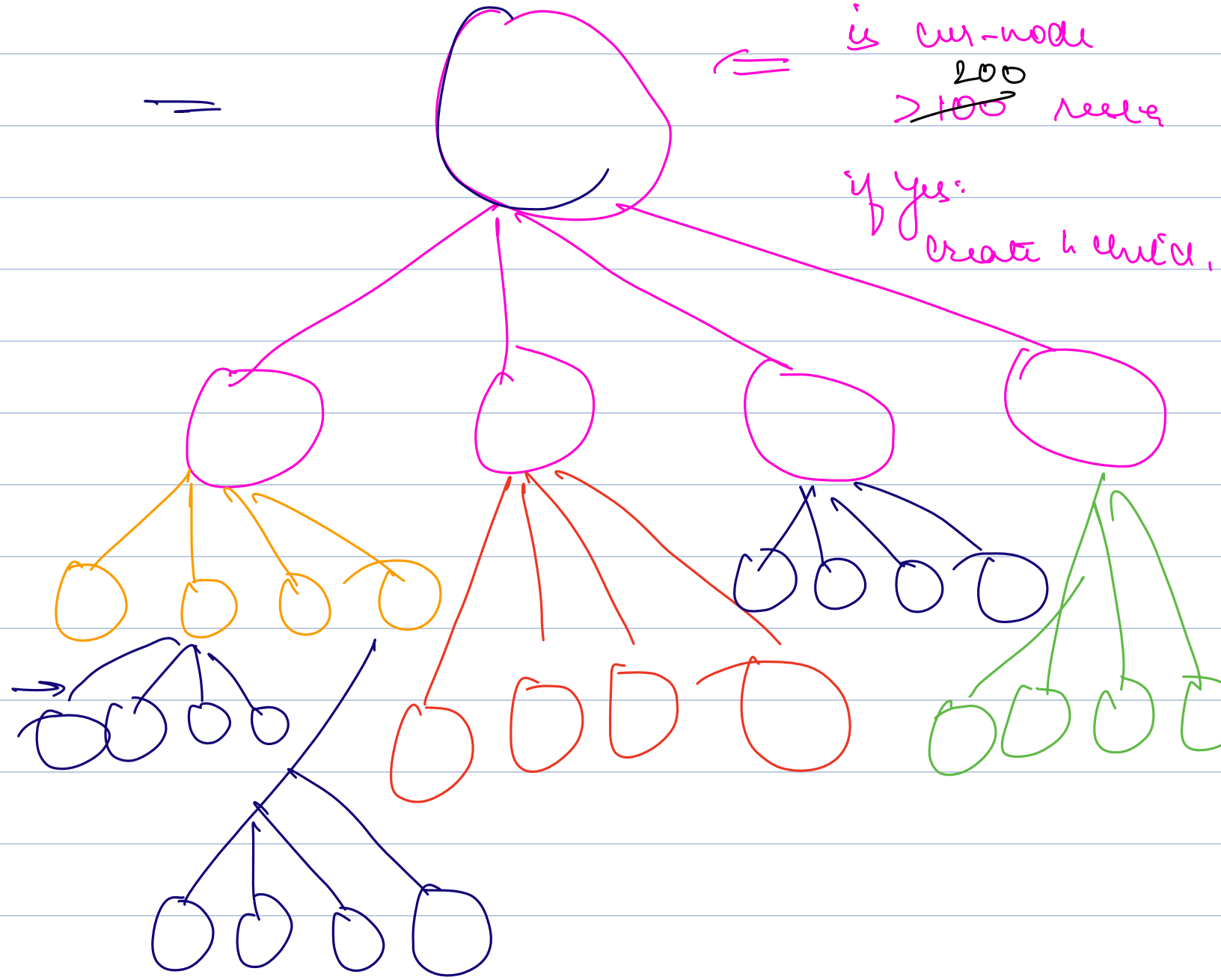
⇒ Is there a way in which I
can create grids of unequal
size?

"Segment Trees"

Quad Tree (depends only on location of rectangle)

World





5 mins

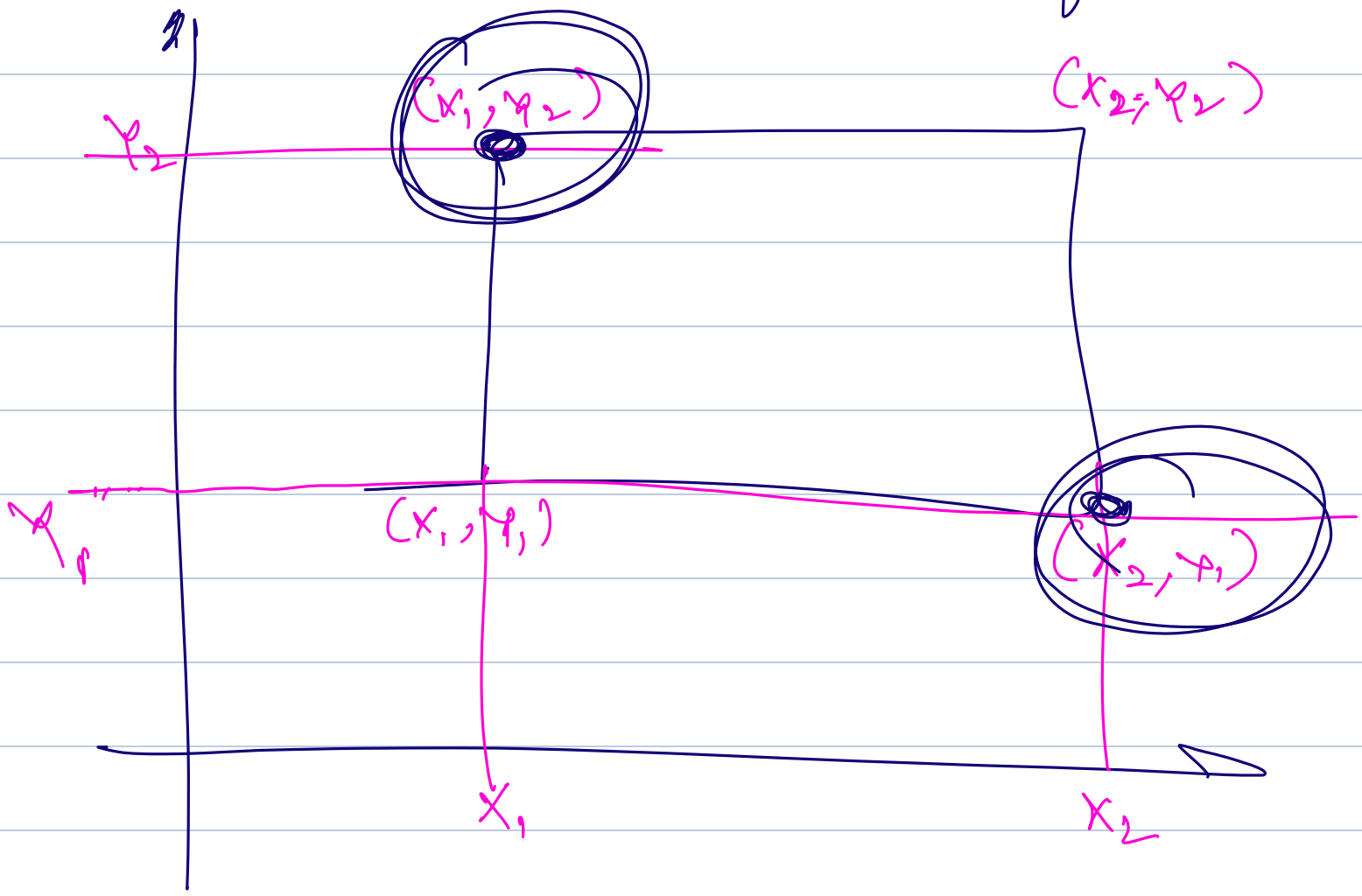
```
Node {
    top-left-coord:
    bottom-right-coord:
    List<Node> children;
    List<Restaurant>
}
```

area being represented

its children

only in the

leaf node.



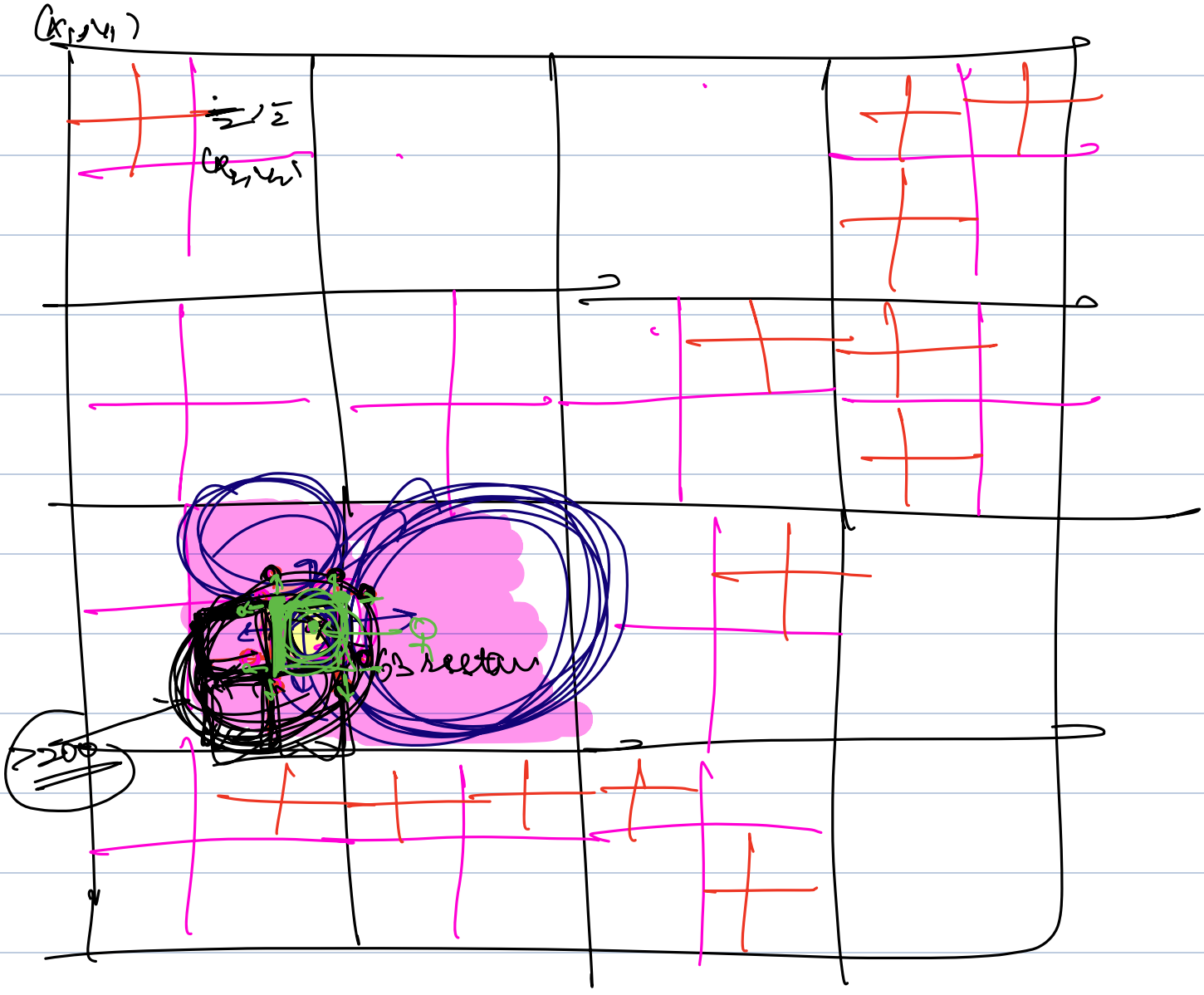
Only 2 points

Assume this tree exists

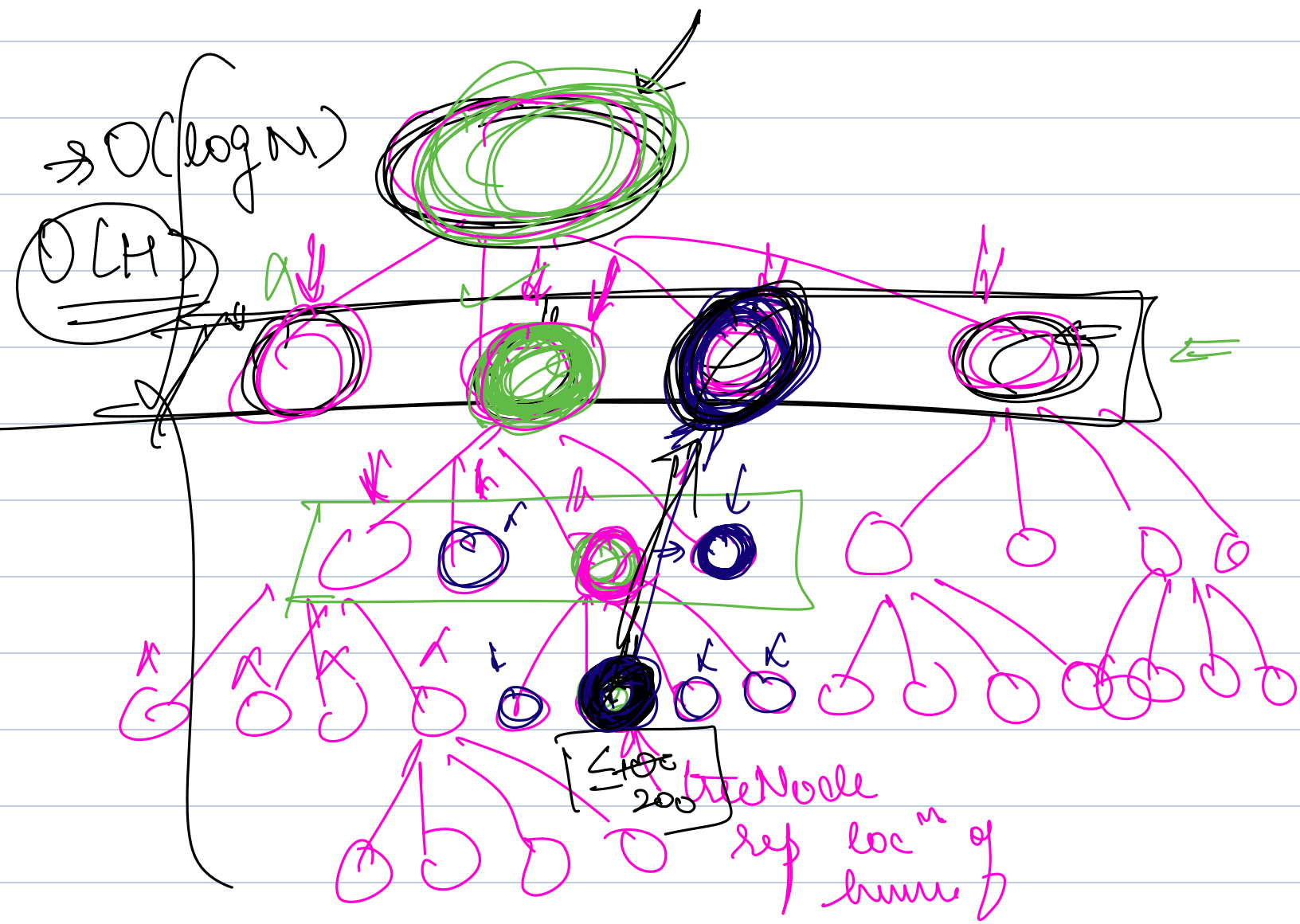
List <Restaurant> getRestaurants (loc^n) {

(x_1, y_1)

If (x_1, y_1)



→ { go To Node containing area where the human is :



$O(\log_4 N)$

→ $\boxed{O(\log N)}$

~~$O(\log 4)$~~

Cons

getNode (loc^n) {

ptr = root → coord of tumer

while (! ptr = child is empty) {

ptr = correct Child (loc^n)

}

return ptr

$O(\log N)$

}

}

$O(N)$ v/s $O(N^2)$

if >100:
divide

getNode (box^n) {

getRestaurants (loc^n) {

node = goToNode (loc^n)

if (node.restaurants.size ≥ 100)

sort

return;

50%

Guaranteed to have atleast 100

siblings = getSiblings (loc^n)

sort and return;

}

get K

Keep nodes of size 2k

⇒ Now leaf Node

→ 200 Restaurants

→ ≈ 50% probability we are done

⇒ leaf has
≥ 100

0 ; 1, 2, ——— , 200

100, 101, — 200

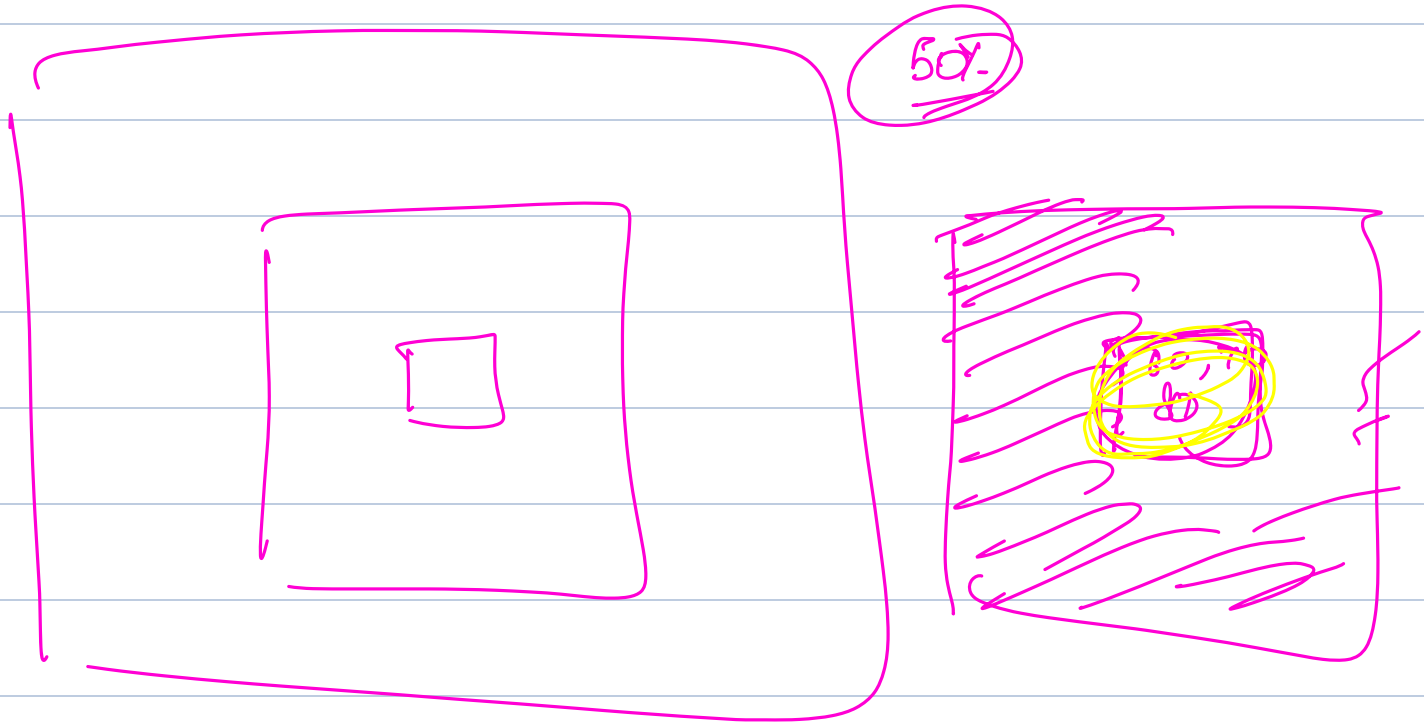
< 100

≥ 100

501

unique ($\&^{x}$ getNode (leaf)) ⇒ $O(\log N)$

⇒ $O(\log N)$



CON

⇒ How will this work in practice?

tree can't be stored in DB

⇒ we will have to store in RAM

⇒ how will I store such.

Node {

topLeft (x, y) → 16B

bottomRight → 16B

list <child> → 32B

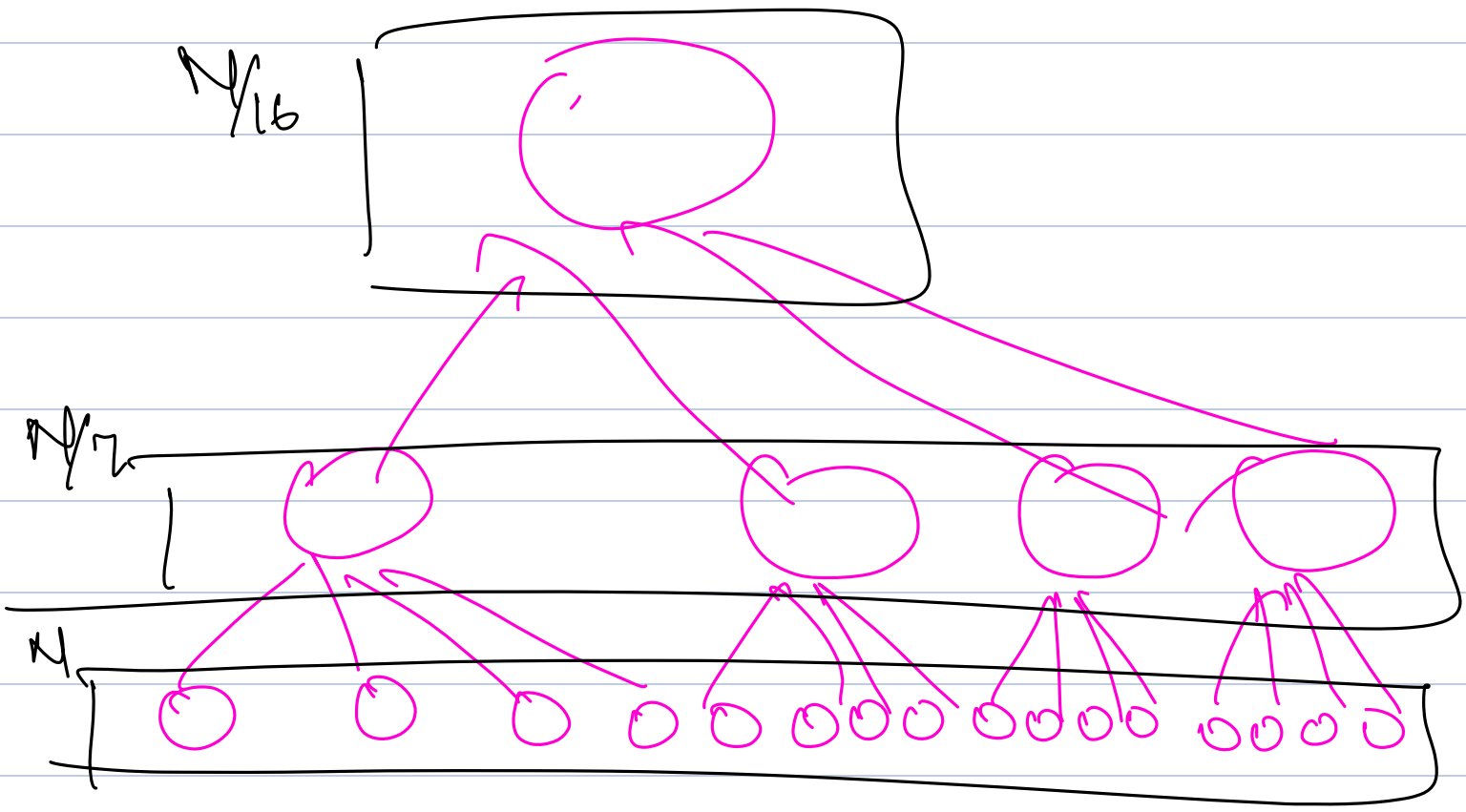
list <Restaurant>

} 64B

leafNode of average 100 restaurants

→ 10M restaurants

≈ leaf Node ⇒ 100K leafNodes



$$100k + \frac{100k}{4} + \frac{100k}{16} + \frac{100k}{64} + \frac{100k}{256}.$$

$$100k \left(1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \cdots\right)$$

$\approx 200k$ Nodes

100

64 B Node

200k nodes

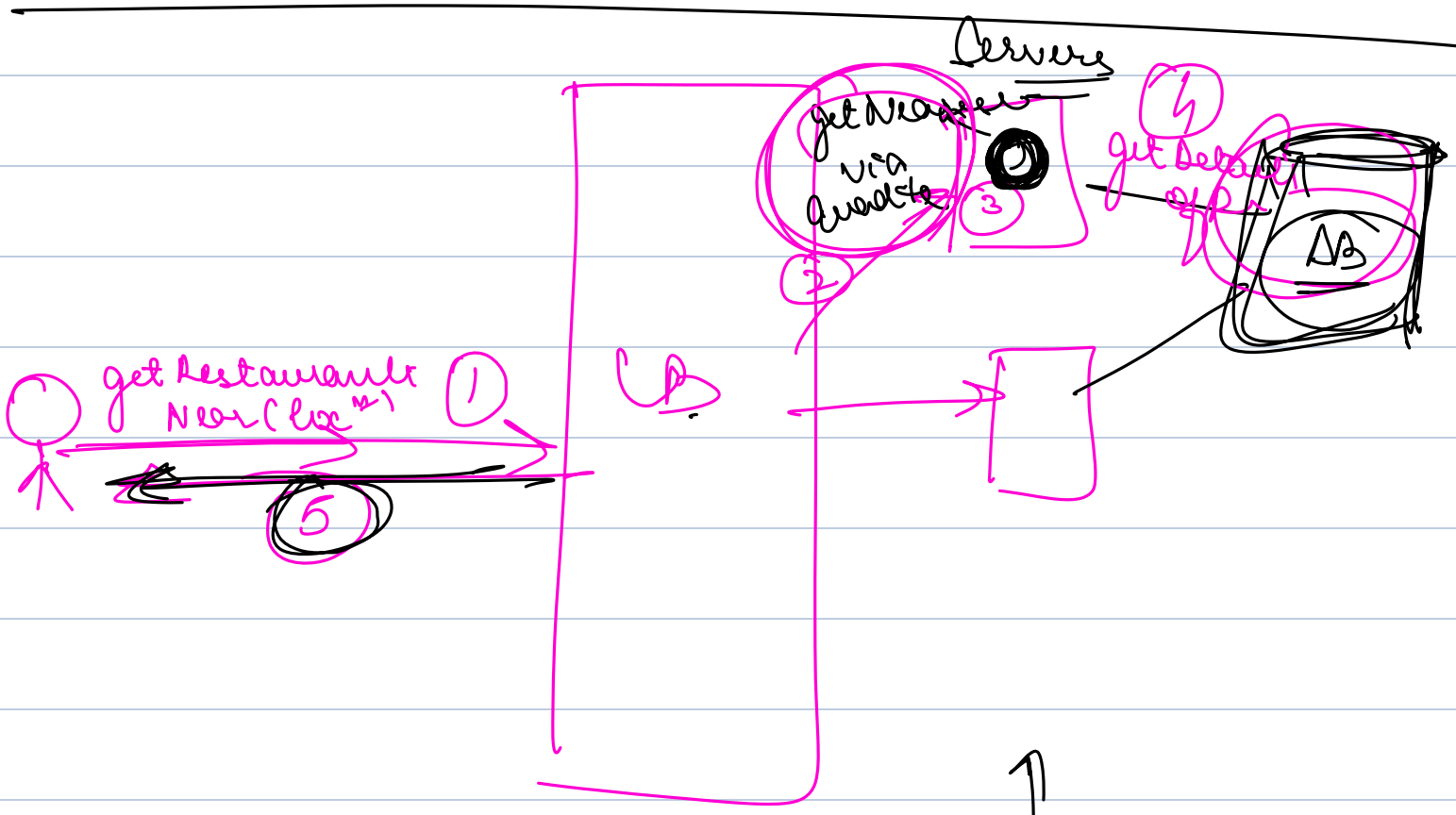
$\rightarrow 200 \times 10^3 \times 100\ B$

$\rightarrow 20 \times 10^6\ B \rightarrow 20MB$

can be stored in RAM

$+ 8B \times 10 \times 10^6$

$\rightarrow 20MB + 80 \times 10^6\ B \rightarrow 100MB$



every day at Midnight the Quad Tree is created.

and put on servers

GeoHashing → in side Redis, Elastic Search