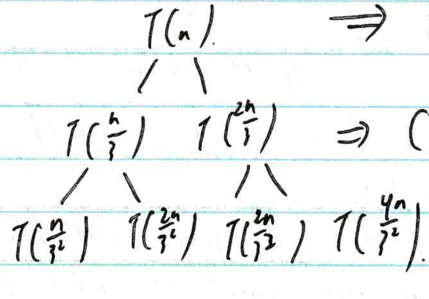
1. $T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + n^2$ is in $O(n^2 \log_2 n)$.



$$\Rightarrow n^2.$$

$$\Rightarrow (\frac{n}{3} + \frac{2n}{3})^2 = n^2$$

$$\Rightarrow (\frac{n}{3^2} + \frac{2n}{3^2}) \Rightarrow \frac{3n}{9} \Rightarrow \frac{n}{3}$$

$$(\frac{2n}{3^2} + \frac{4n}{3^2}) \Rightarrow \frac{6n}{9} \Rightarrow \frac{2n}{3}$$

by the tree we can get the right side work must be more than left side work so that is a unbalance tree. so that be $O(\log_2 n)$.

~~So for the left side~~ [illegible] ~~right side~~ [illegible] ~~$= \log_3 n$~~

~~because the~~ ~~shortest path are being being divided by 3. so the path length will equal to $\log_3 n$.~~

and because the ~~deep is~~ depth is $O(n^2)$.

So complexity is $O(\text{depth} \cdot \text{height}) = O(n^2 \log_2 n)$

2. I think the first one must be all element are same.

① same

like [10, 10, 10, 10, 10, 10, 10, 10, 10, 10]

② biggest on the right

the biggest one and smallest one is one the right side will be

① smallest on the right

bad too like [0, 5, 7, 4, 3, 2. 5. 10, 40, 90]

④ \ pivot number choose
⑤ /

[4, 2, 1, 9. 50, 4, 3. 7, 6. 0]

and. the pivot number will be the worse-case too like.

when we choose 10 to both

⑤ [10, 20. 30, 40, 50, 60, 70, 80. 90. 100]

④ [100, 90. 80, 70, 60. 50, 40. 30, 20. 10]

sort that will use time because that are unbalance sort.

The operating logic of max priority queue is mainly based on another value carried by the value. I named this value here as key.

The stack data structure is LIFO, so we only need to ensure that the last in key is the maximum to ensure that it runs first.

{5(0),6(1),8(2),4(3)} () is the key

For queue, we only need to ensure that the key of the most advanced value is the largest, then the value of First in will be run first.

{5(3),6(2),8(1),4(0)} () is the key

1. { stack last in, first one (LIFO)
{ Queue first in, first one (FIFO)

the max priority queue. is implementation by the key

2 means. when you add a element stack. you need two. element key and value.

{ key is the max, the key to make sure which one one.
{ value is the value of you want to add to your stack.

① stack (key, value). when you add a value.
~~value = 0~~ at the begin. key = 0.

when everytime push(), (key ++).
the end of one value's key will be the maximum one.
so the last one will be pop first.

② 1. for the max priority queue to implementation to queue_
the main point is sure with stack the is the value of key

because max priority queue is pop by key first.
like when we push in $10.\ 9.\ 7,\ \underset{①}{6},\ 5.\ \underset{①}{6}$

$\Downarrow$

when they have two. after max priority
same value $10.\ 9.\ 7.\ \underset{②}{6},\ \underset{①}{6}.\ 5.$
like we have two 6 at here.

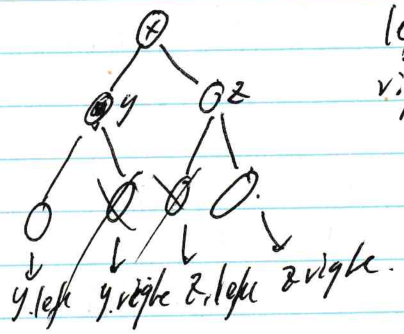
$\Downarrow$ output will be

$\underset{②}{6}.\ \underset{①}{6}$ $10.\ 9.\ 7.\ \underset{②}{6},\ \underset{①}{6}.\ 5$

↓ ↓
first in next in.

we find ② key is max so we move one $\underset{②}{6}$ first. and next
will be $\underset{①}{6}$.

4.



left predecessor maximum

right successor minimum

So at the first if y have y.right

$y[value] <= y.right <= x[value]$

if look like that the.

y[value] ⇒ y.left.

change to locate

y.right ⇒ y.

x ⇒ x.

if z have z.left.

$x[value] \le z.left <= z.value$.

x ⇒ x.

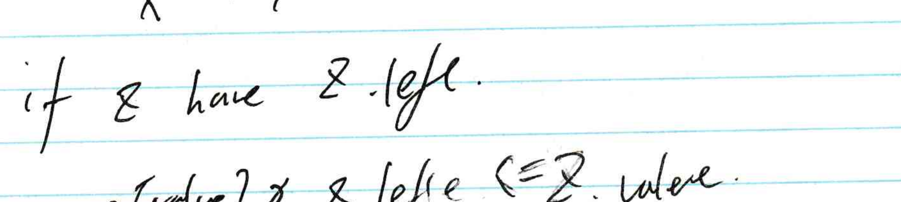
z.left ⇒ z.

z. ⇒ z.right

for example 1. 2. 3. 4. 5

↙ if

↘ actually


1
2
3
5
4
2
3
4
1
5