The running time is $O(n)$ because it takes
$O(n)$ time to initialize the two arrays (buckets)
$O(n)$ time to copy from the remainder array to the quotient array
$O(n)$ time to retrieve from the quotient array and return.

Hence $O(n)+O(n)+O(n) = O(3n) = \underline{\underline{O(n)}}$

T(0) =

$r^0 b + a\frac{1-r^0}{1-r}$ which is b, so the formula is true when $n = 0$. Now assum

$$T(n-1) = r^{n-1}b + a\frac{1-r^{n-1}}{1-r}.$$

Then we have

$$\begin{aligned}
T(n) &= rT(n-1) + a \\
&= r\left(r^{n-1}b + a\frac{1-r^{n-1}}{1-r}\right) + a \\
&= r^n b + \frac{ar - ar^n}{1-r} + a \\
&= r^n b + \frac{ar - ar^n + a - ar}{1-r} \\
&= r^n b + a\frac{1-r^n}{1-r}.
\end{aligned}$$

Continue of 4a:

Array before sorting:

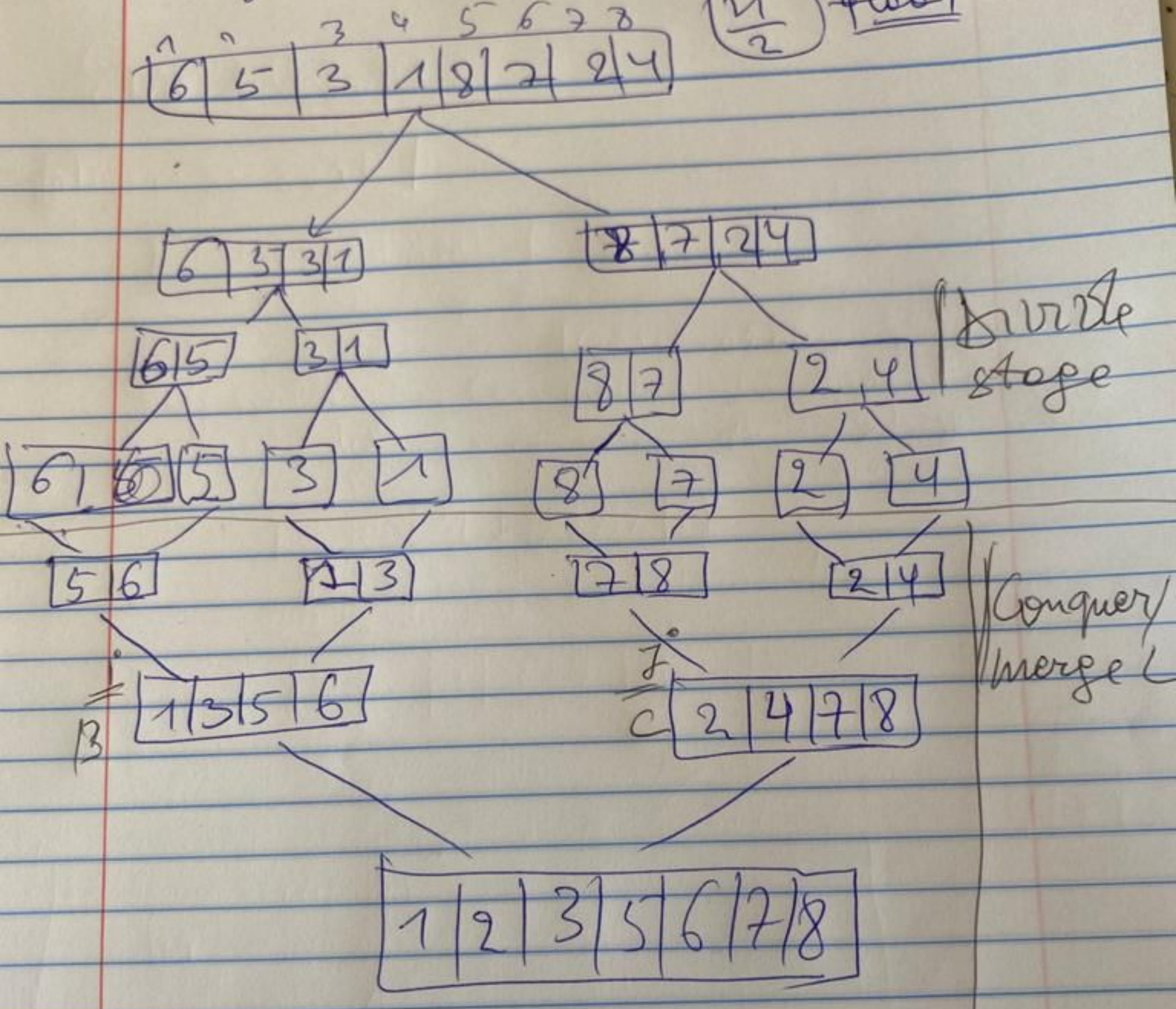
0	1	2	3
4_a	4_b	4_c	4_d

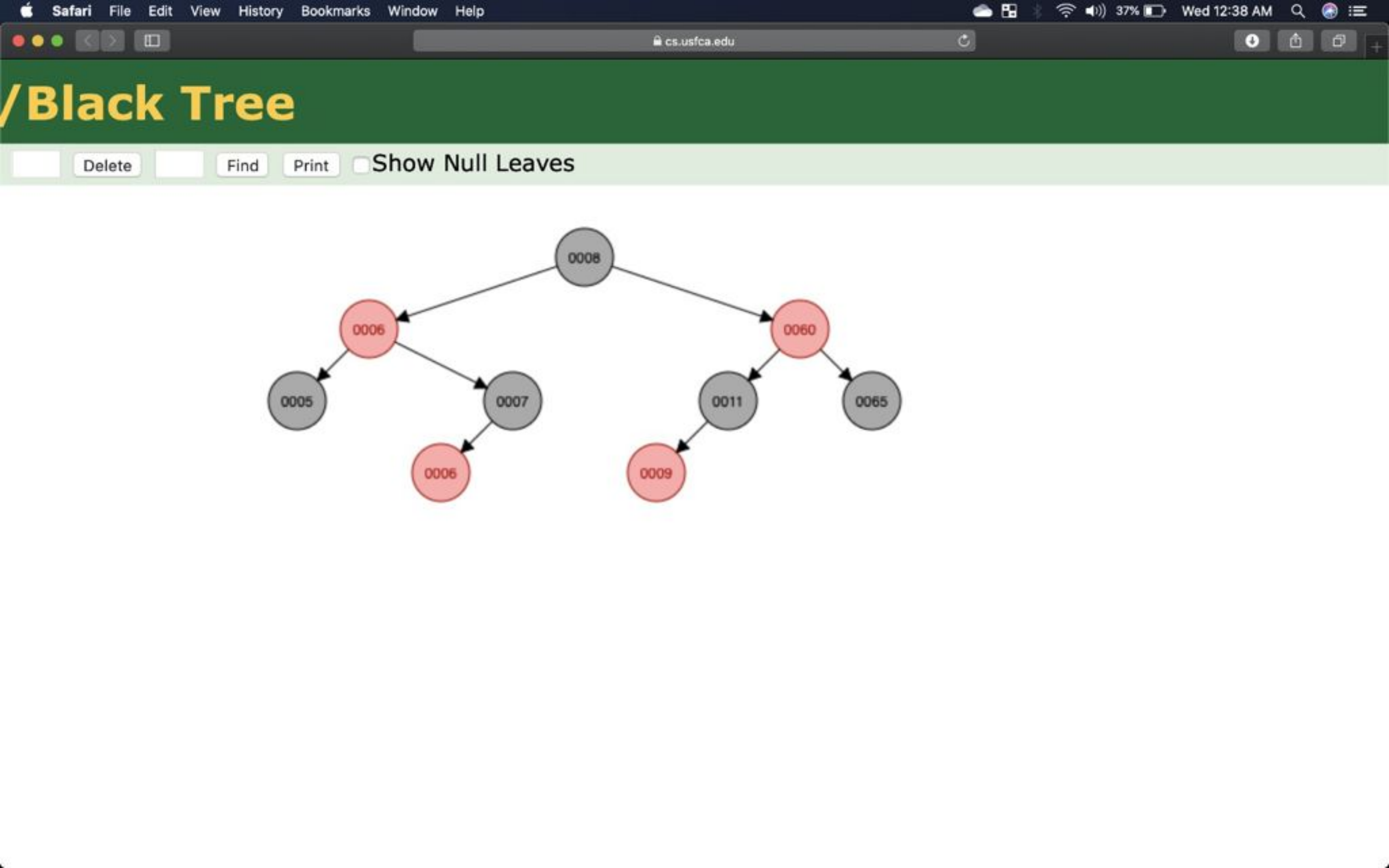
Array after sorting:

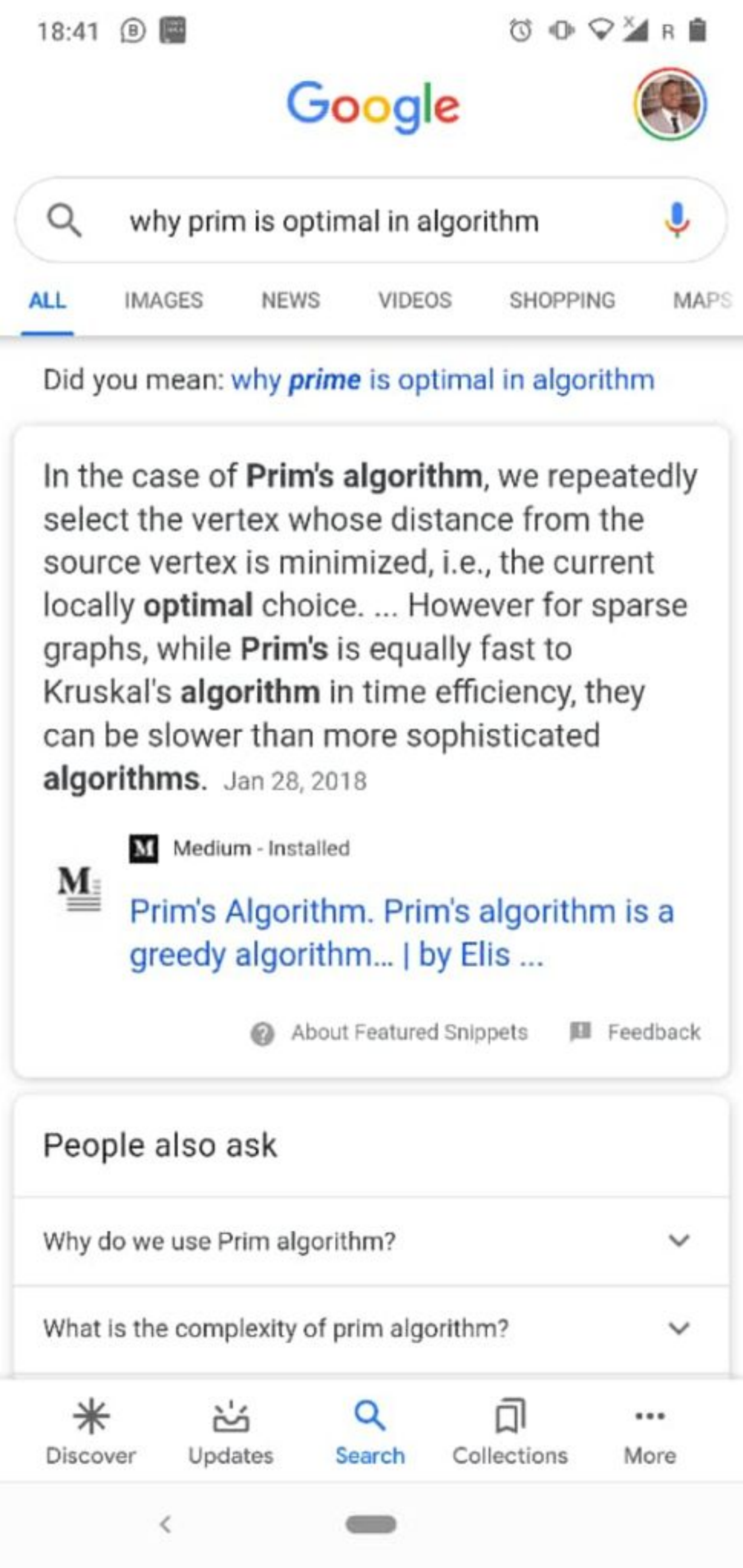
4_c	4_d	4_b	4_a

Obviously, elements were not kept in their places ~~atthougs~~ although they are equal. That's why quick sort is not stable.

Mergesort (Intuition)




Safari
File
Edit
View
History
Bookmarks
Window
Help
37%
Wed 12:38 AM
cs.usfca.edu
/Black Tree
Delete
Find
Print
□ Show Null Leaves
0008
0006
0060
0005
0007
0011
0065
0006
0009



why prim is optimal in algorithm

Did you mean: why *prime* is optimal in algorithm

In the case of **Prim's algorithm**, we repeatedly select the vertex whose distance from the source vertex is minimized, i.e., the current locally **optimal** choice. ... However for sparse graphs, while **Prim's** is equally fast to Kruskal's **algorithm** in time efficiency, they can be slower than more sophisticated **algorithms**. Jan 28, 2018

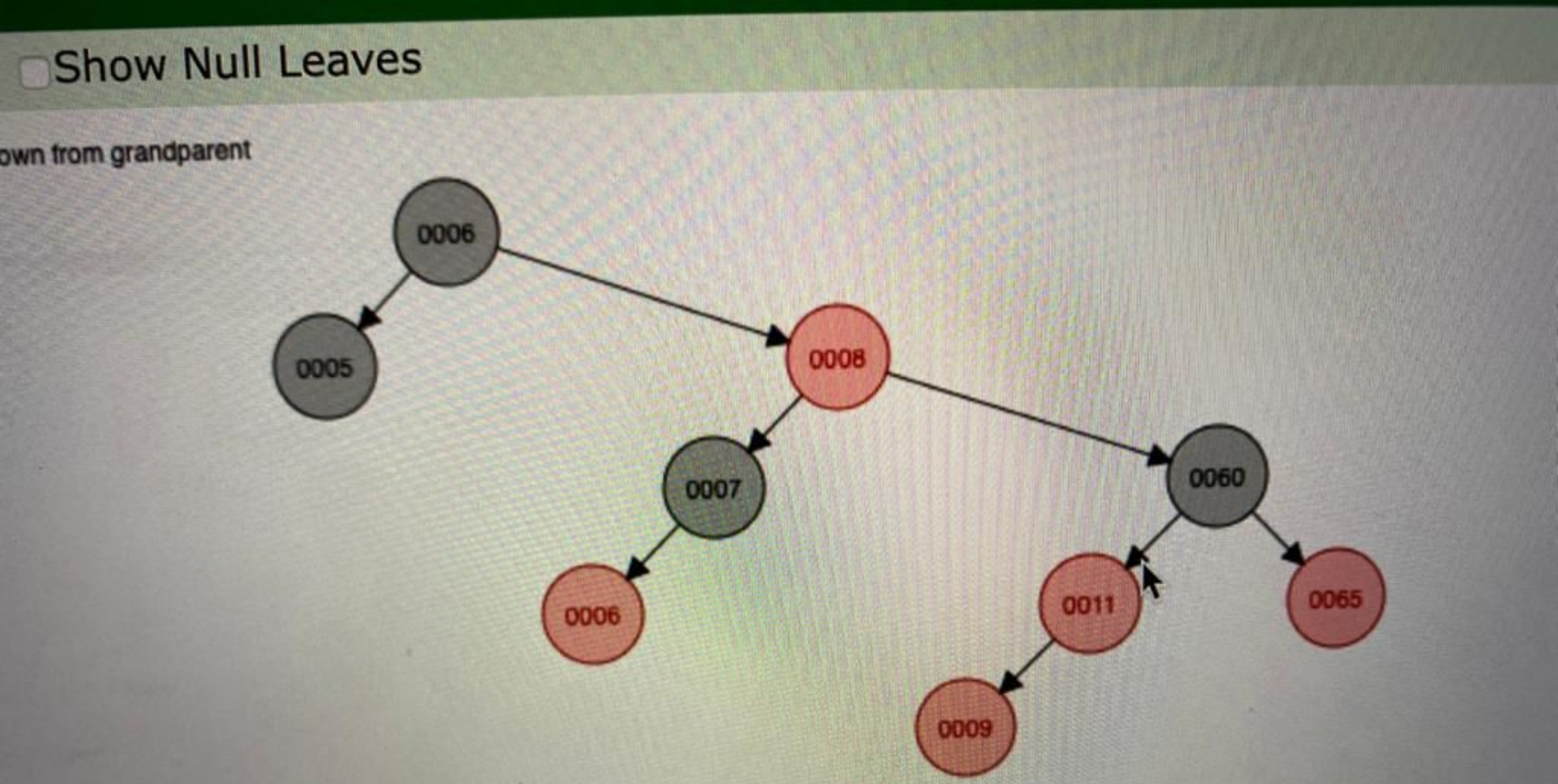
Medium - Installed

Prim's Algorithm. Prim's algorithm is a greedy algorithm... | by Elis ...

People also ask

Why do we use Prim algorithm?

What is the complexity of prim algorithm?



□Show Null Leaves

own from grandparent

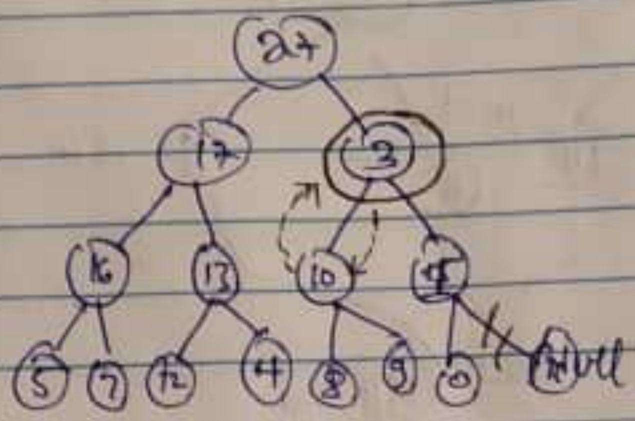
Solution To Lab 9

Names:

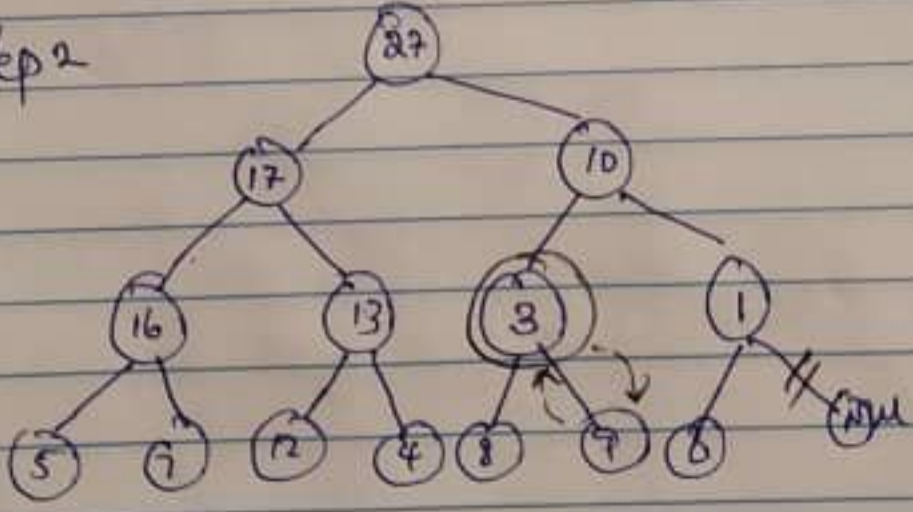
EMILE KARAMUTSA 611072
BRIAN NKURUNUNGI 611061

Qn1. Max_Heapify (A, 3)
A = ⟨27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0⟩

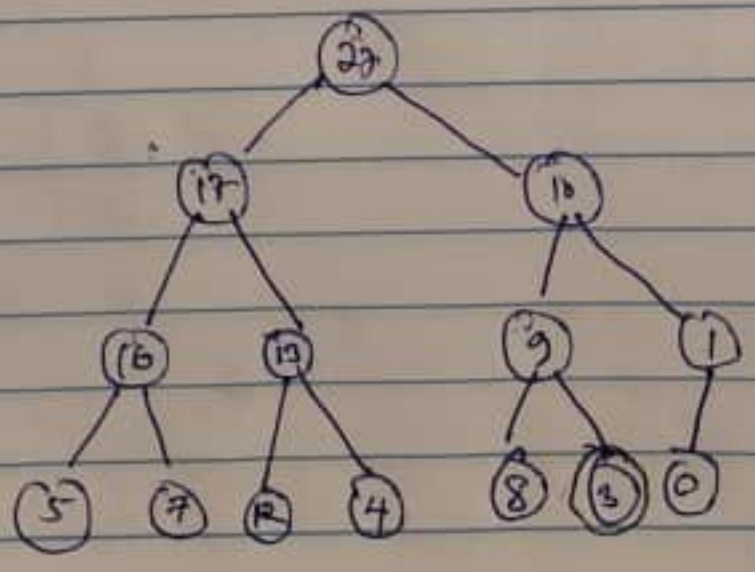
Step 1



Step 2



Step 3

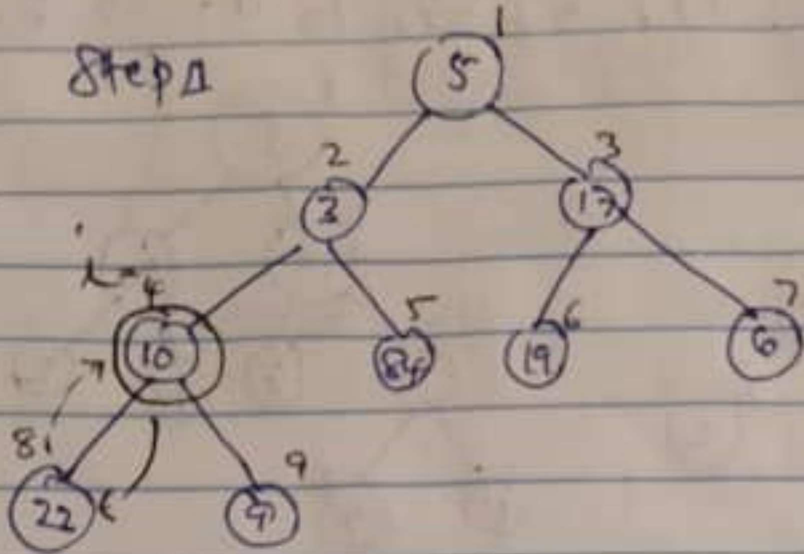


Q2 A= (5, 3, 17, 10, 84, 19, 6, 22, 9)

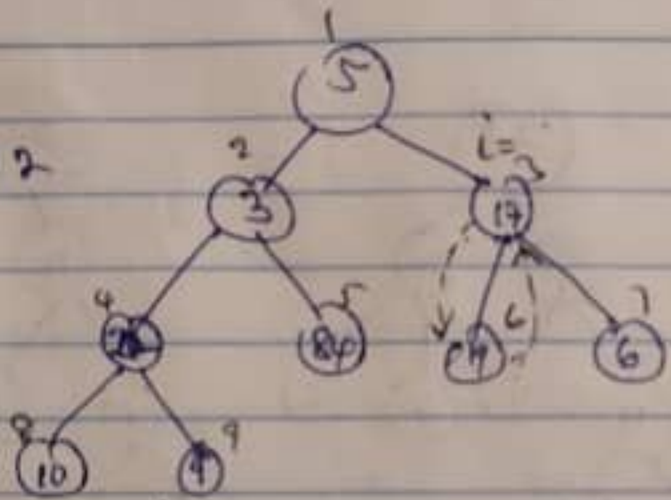
The Initial i (= $\left\lfloor \frac{n}{2} \right\rfloor$

$= \left\lfloor \frac{9}{2} \right\rfloor = 4$

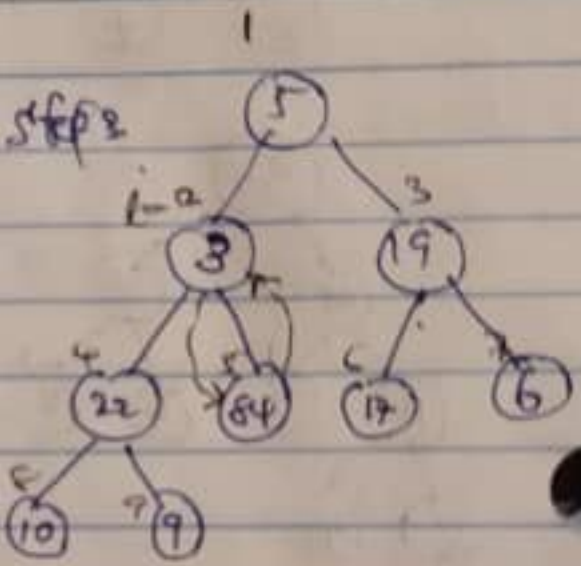
Step 1



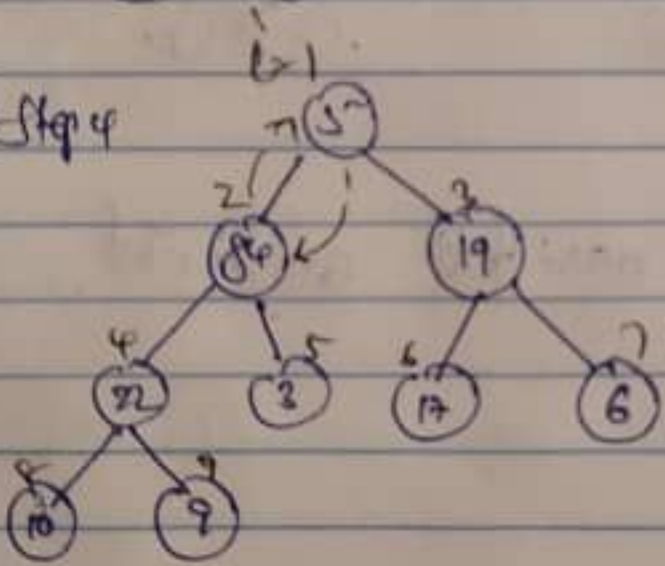
Step 2



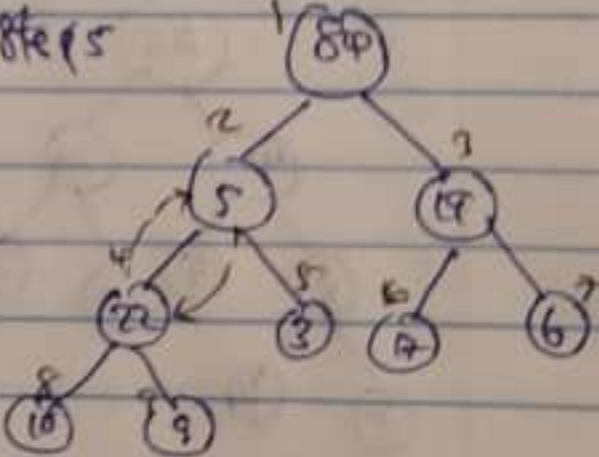
Step 3



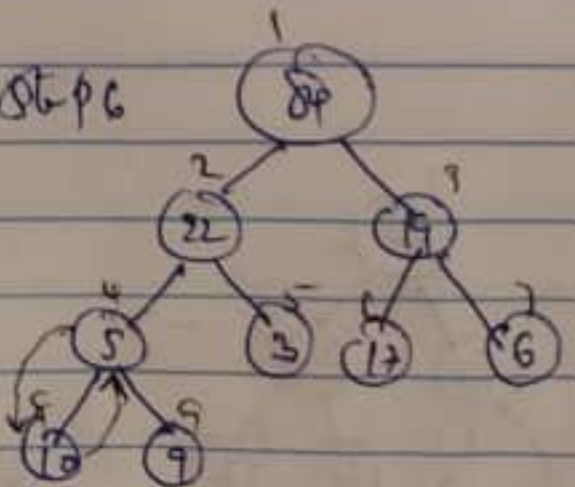
Step 4



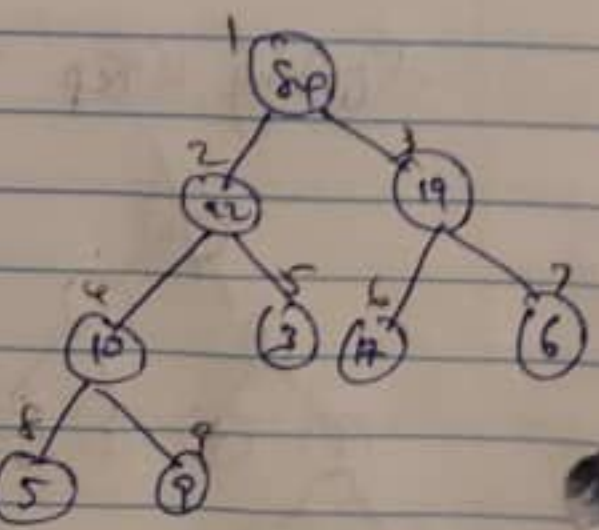
Step 5



Step 6



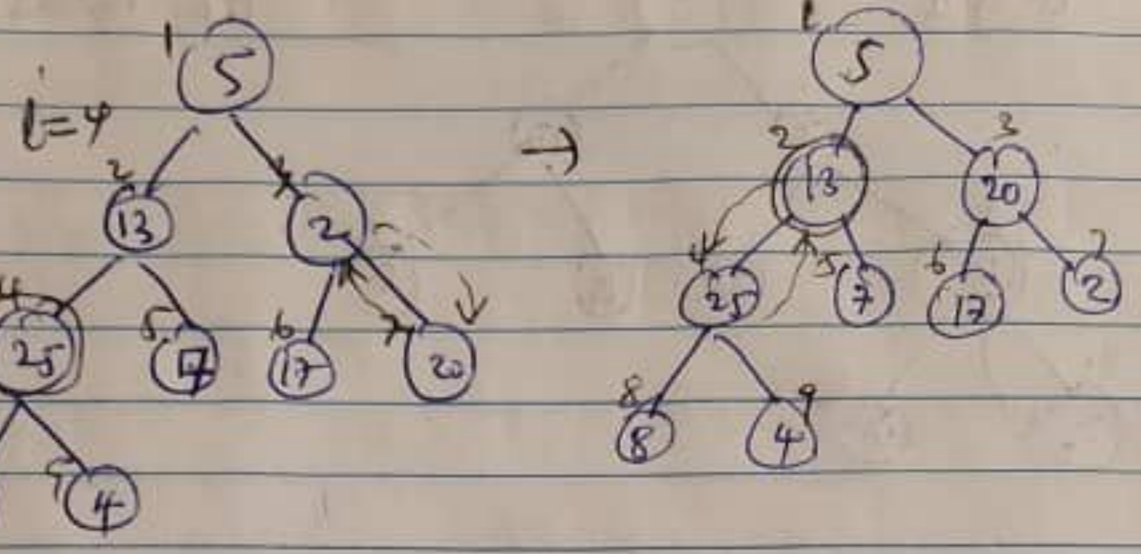
Step 7

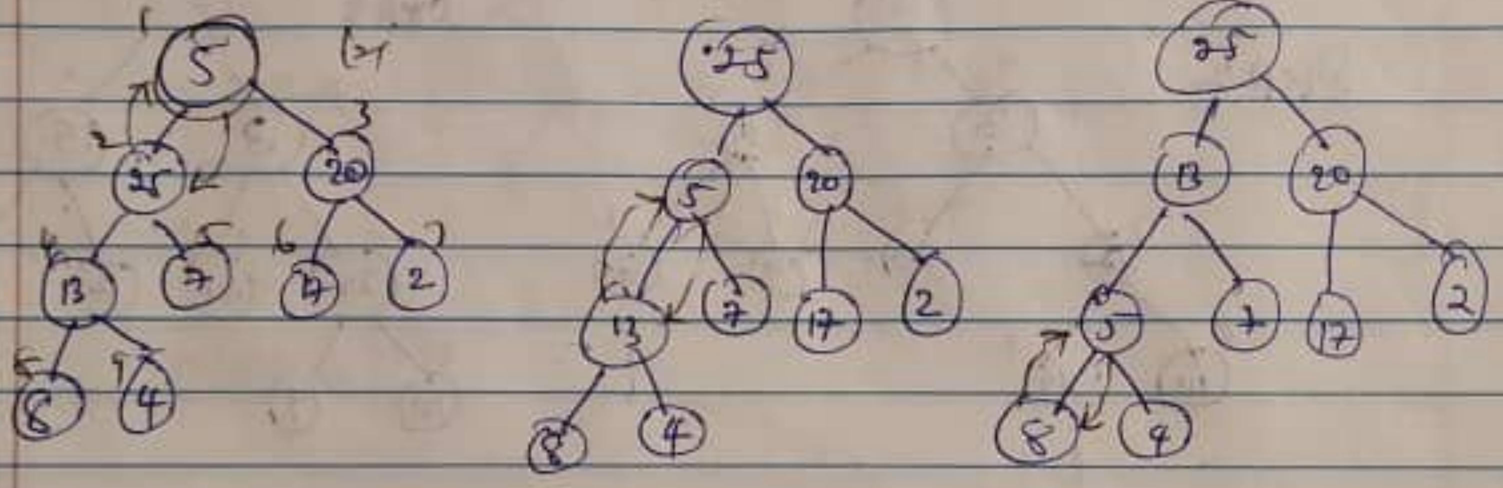


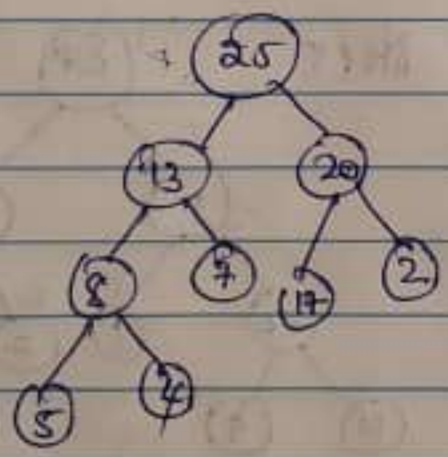
Q:3 Operations of Heapsort

A = ⟨5, 13, 2, 25, 7, 17, 20, 8, 4⟩

Step 1
Build the heap.

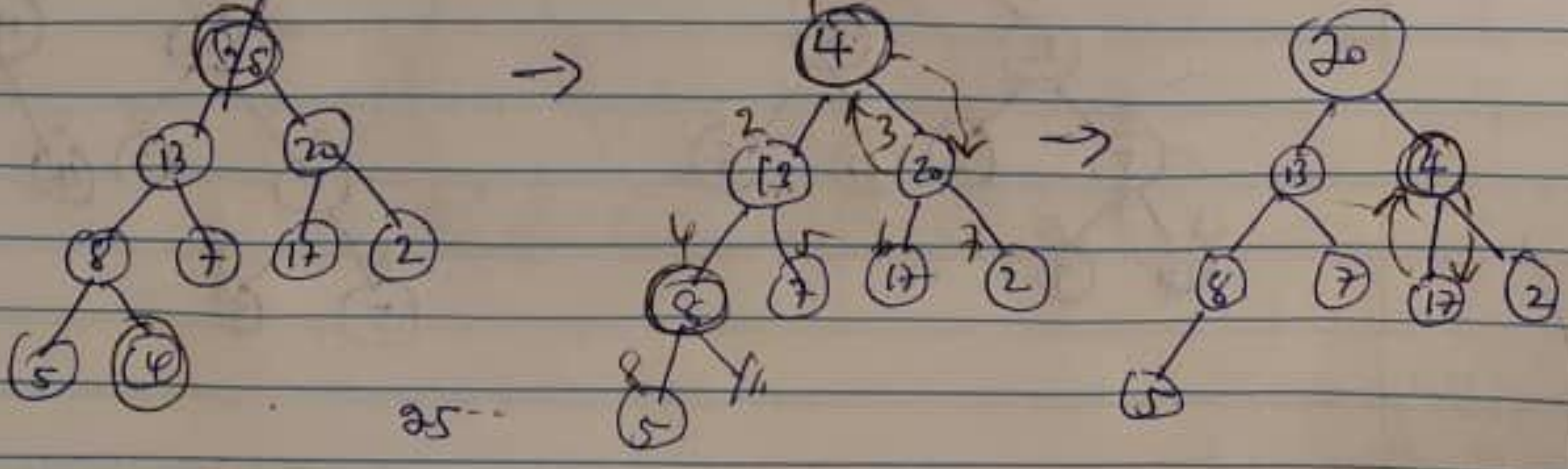


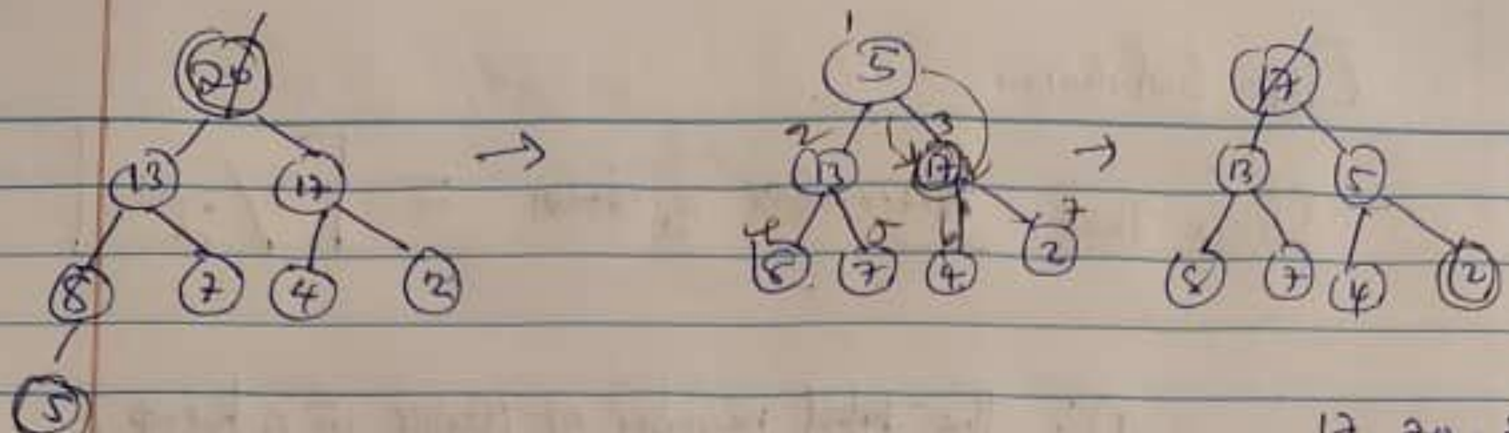




The max-Heap constructed.

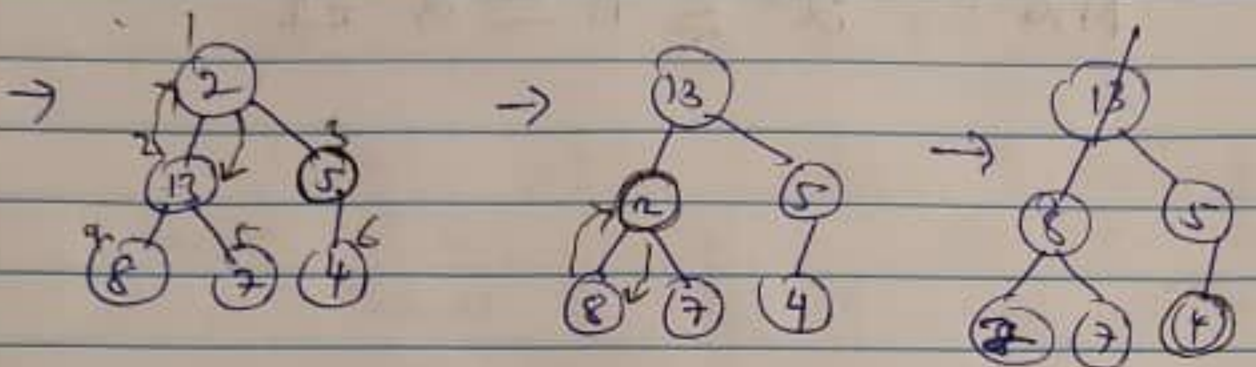
Sorting Step :



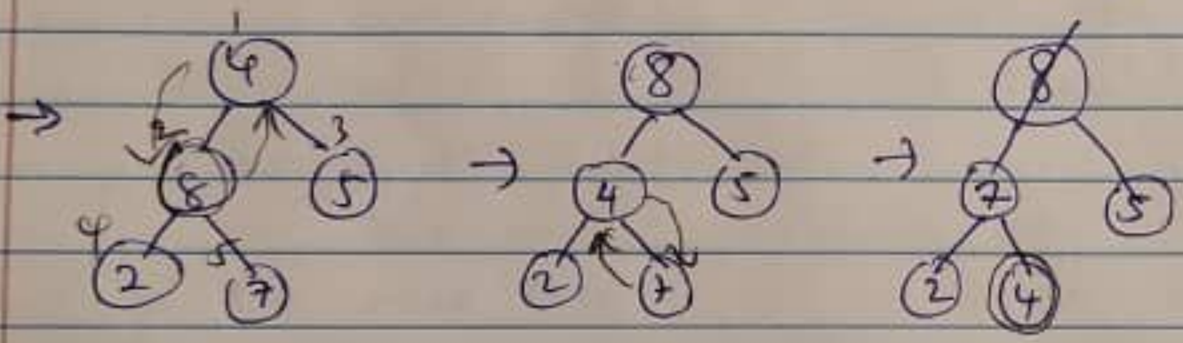


20, 25

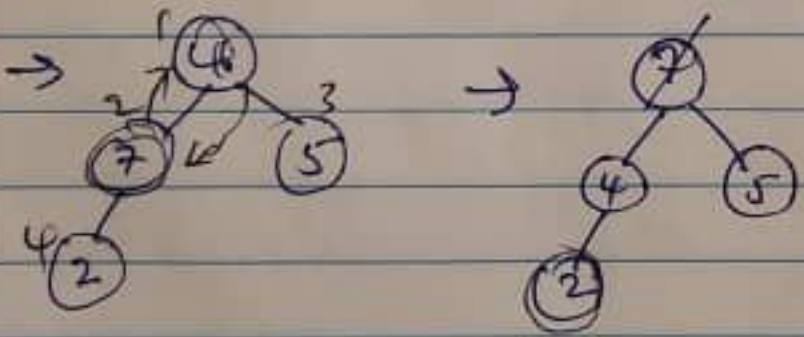
17, 20, 25



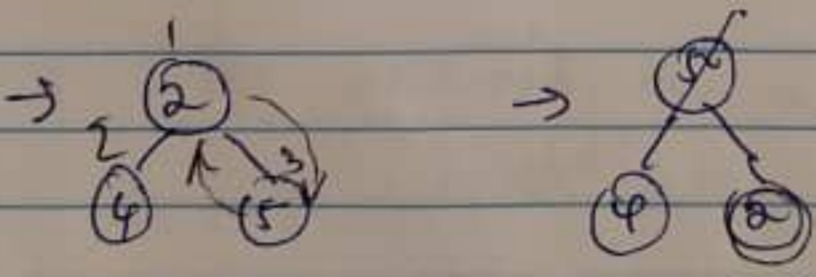
13,17; 20, 25



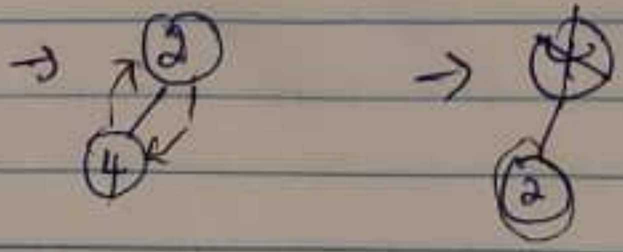
8, 13, 17, 20, 25



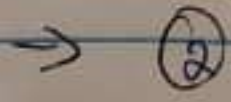
7, 8, 13, 17, 20, 25



5 7, 8, 13, 17, 20, 25



4, 5, 7, 8, 13, 17, 20, 25



2, 4, 5, 7, 8, 13, 17, 20, 25

Qn 4 Solution.

the number of leaves in a heap: $\lceil \frac{n}{2} \rceil$

By induction, on h.

Base case, when $h = 0$.
The number of leaves $= \lceil n/2 \rceil =$

$$\lceil n/2^{0+1} \rceil .$$

Step: Let us assume it holds for nodes of height $h-1$. Let us take a tree and remove all its leaves. We get a tree with $n - \lceil \frac{n}{2} \rceil = \lfloor n/2 \rfloor$ elements. Note that the nodes with height h in the old tree have height of $(h-1)$ in a new one.

We will calculate the number of such nodes in the new tree. By inductive assumption we have that T, the number of nodes with height $(h-1)$ in the the new tree, is

$$T = \left\lceil \lfloor n/2 \rfloor / 2^{h-1+1} \right\rceil < \left\lceil \left(\frac{n}{2}\right) / 2^{h} \right\rceil = \left\lceil \frac{n}{2^{h+1}} \right\rceil .$$

As mentioned, this also the number of nodes with height h.

The running time is $O(n)$ because it takes

$O(n)$ time to initialize the two arrays (buckets)

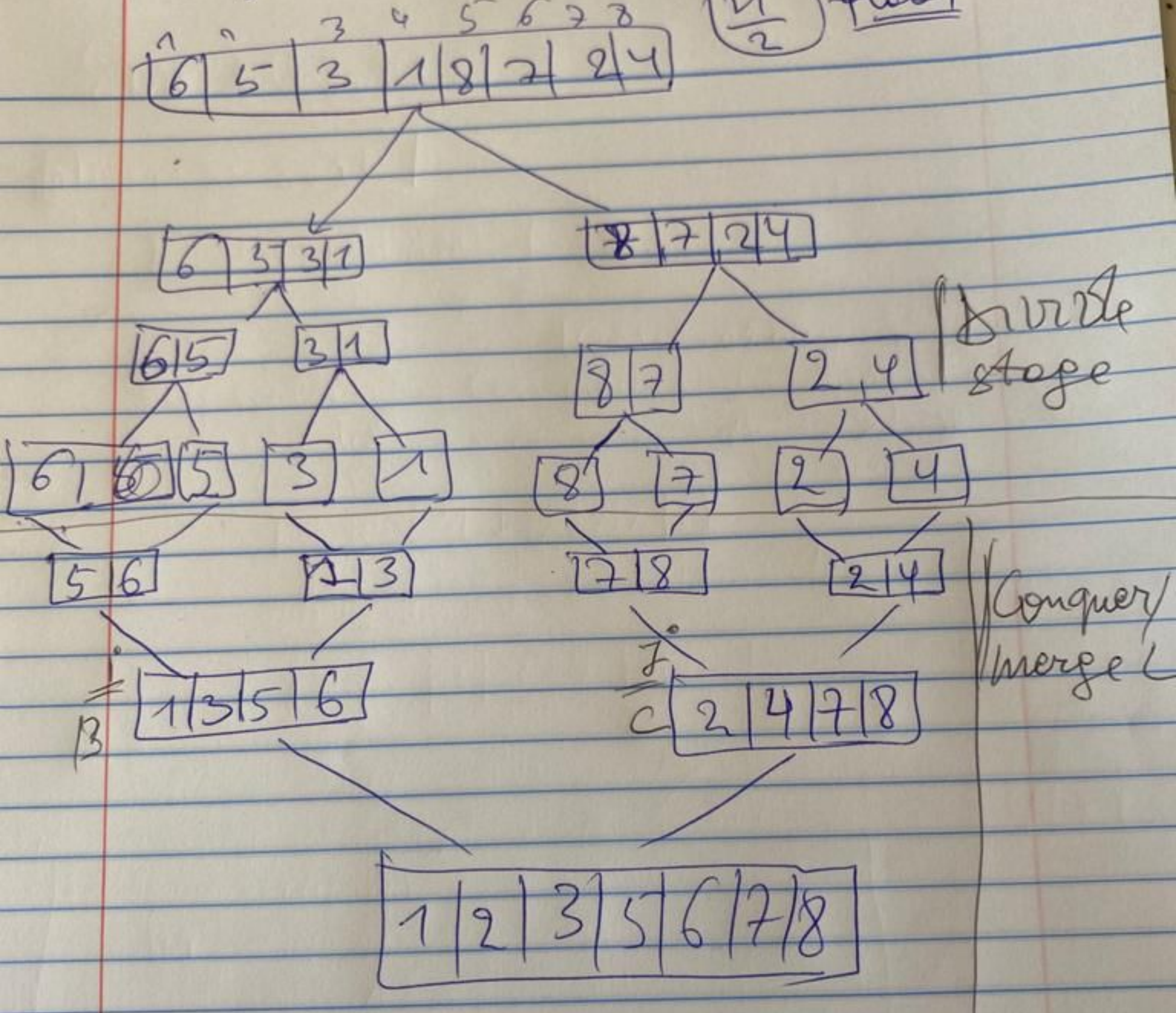
$O(n)$ time to copy from the remainder array to the quotient array

$O(n)$ time to retrieve from the quotient array and return.

Hence $O(n)+O(n)+O(n) = O(3n) = \underline{\underline{O(n)}}$

Mergesort (illustration)

$\frac{n}{2}$ floor


1 2 3 4 5 6 7 8
6 5 3 1 8 7 2 4
6 3 3 1
8 7 2 4
6 5
3 1
8 7
2 4
6 6 5
3
1
8
7
2
4
Divide stage
5 6
1 3
7 8
2 4
B
1 3 5 6
C
2 4 7 8
Conquer/merge
1 2 3 5 6 7 8