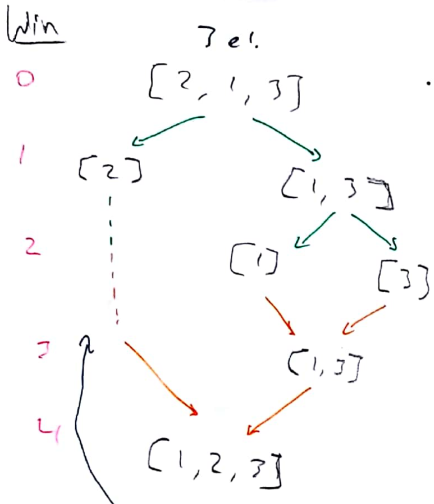
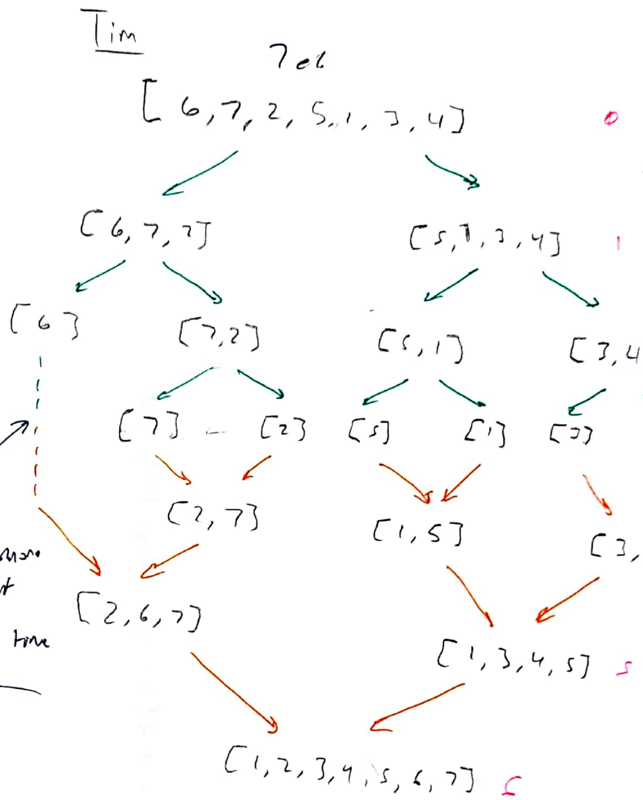


Notes

- 2 layers to split to single element arrs.



Seems like you could add 1 more element without gaining too much time

Notes

- 3 layers to split to single element arr.
- Pattern: total # of layers = 2 × the layers it takes to split

Scanned by CamScanner

Jeremy

8 el.

[6, 7, 2, 8, 5, 1, 3, 4]   0

[6, 7, 2, 8]   [5, 1, 3, 4]   1

[6, 7]   [2, 8]   [5, 1]   [3, 4]   2

[6]   [7]   [2]   [8]   [5]   [1]   [3]   [4]   3

[6, 7]   [2, 8]   [1, 5]   [3, 4]   4

[2, 6, 7, 8]   [1, 3, 4, 5]   5

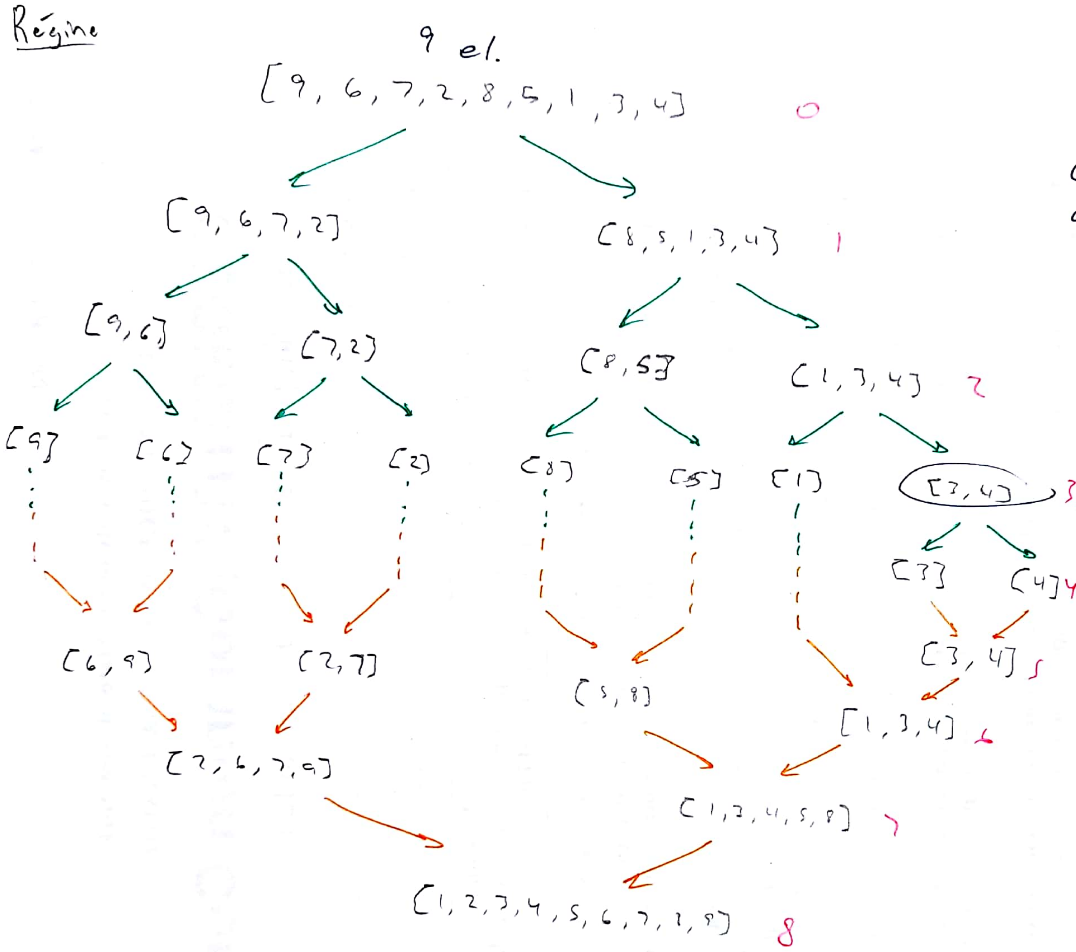
[1, 2, 3, 4, 5, 6, 7, 8]   6

Notes

- Previous pattern still appears to be true
- This array seems to be at capacity.

The total # of layers is also exactly $2\log_2(n)$...

Régine



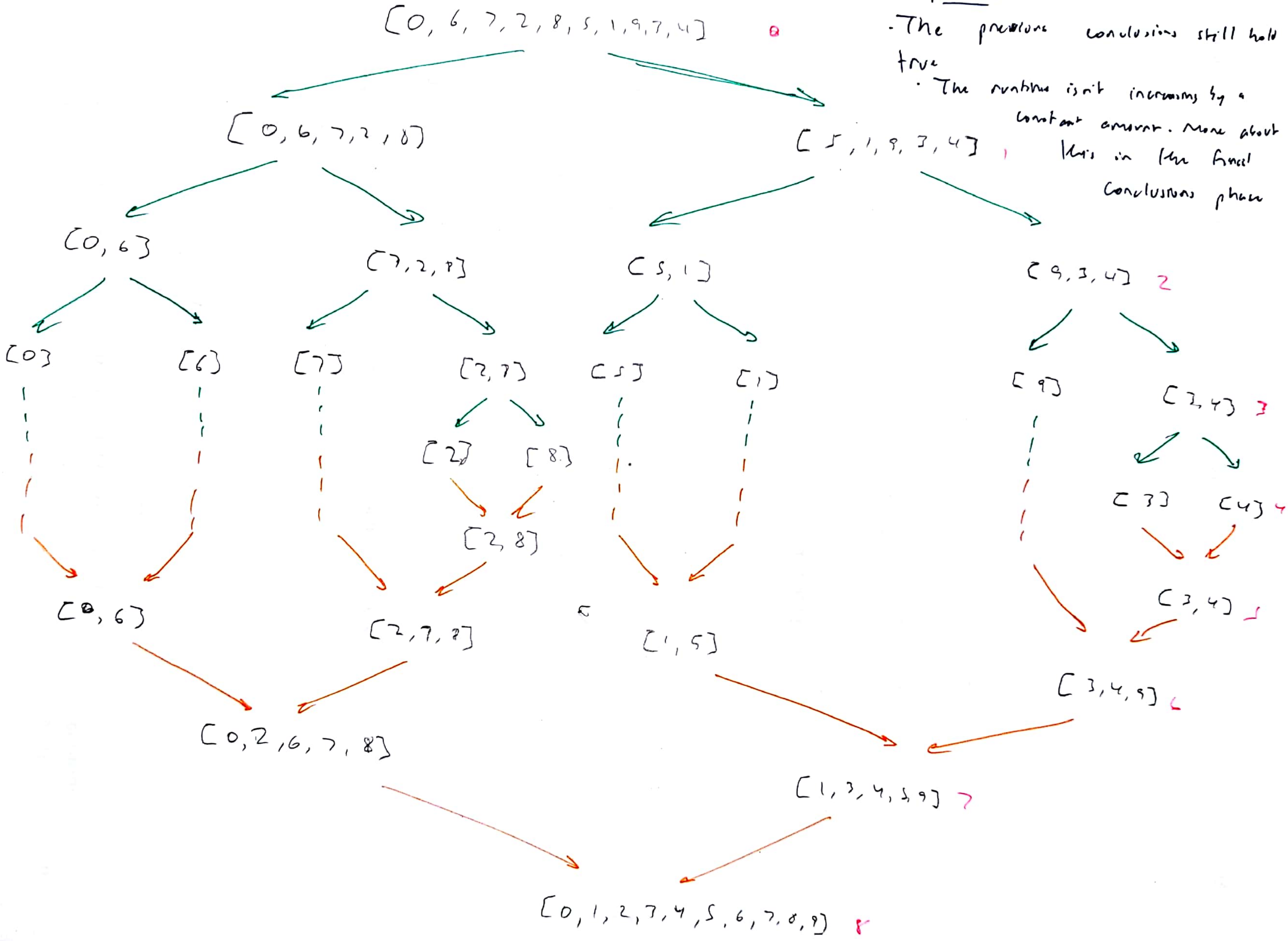
Notes

Because there are a ceil($2\log_2 n$) # of layers and n elements are copied per layer, it's safe to say runtime is $2n\log_2 n$ ish, (plus some constants) or $O(n\log n)$. We'll see if this pattern persists...

This 1:1 dude hurts... It's clear that going one element above a power of 2 will kill your runtime.

Richard

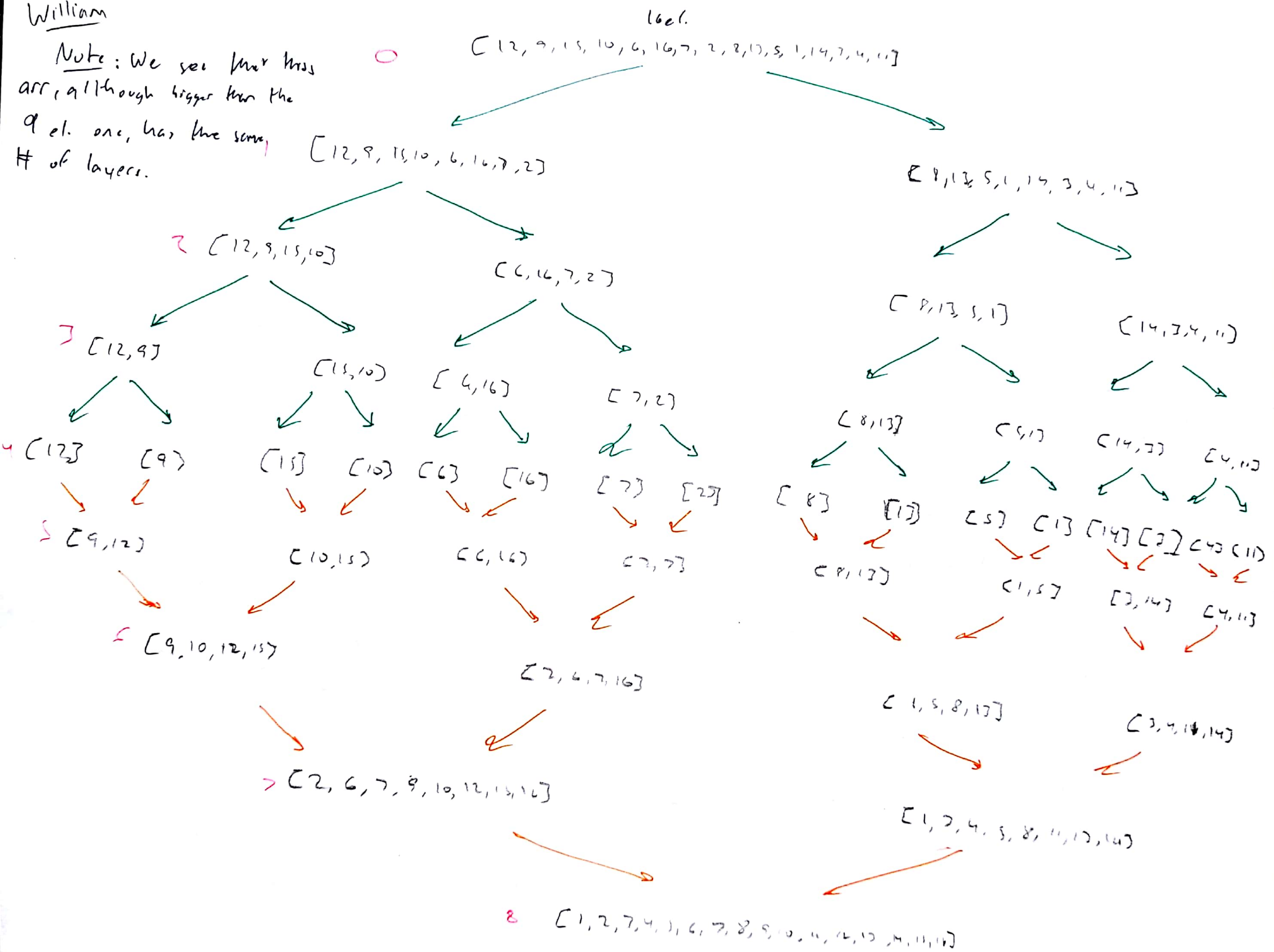
10 el.



Notes.
- The previous conclusions still hold true
- The runtime isn't increasing by a constant amount. More about this in the final conclusions phase

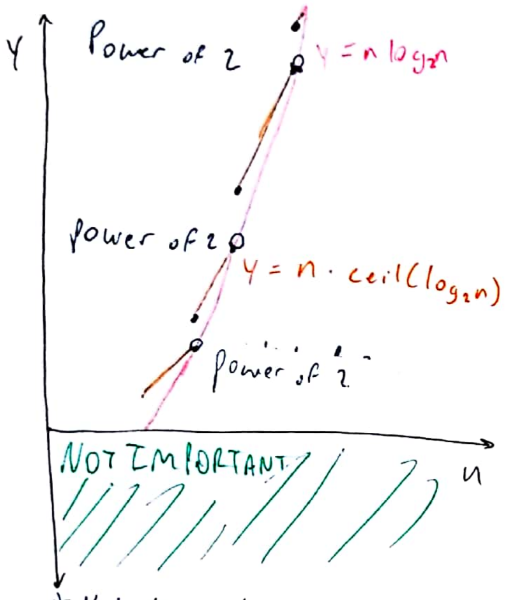
William

Note: We see that this arr, although bigger than the 9 el. one, has the same # of layers.



Conclusions Drawn

It is clear from the traces that mergeSort runs in O(nlogn) time. There are logn layers and each layer does n things (plus some constant). However, we also know that logn is rounded up due to the fact that a new layer is added at every power of 2. I wanted to see how this compared to a regular nlogn graph



* Not drawn to scale

From the 2 graphs, we can see that the n ceil(log₂n) graph spikes at every power of 2. This is because, at every power of 2, you're multiplying by a greater n and a greater logn. Thus, we can conclude that merge sort works best if done on an array of $2^x - 1$, where x is an integer >0.

Scanned by CamScanner