DIVIDE AND CONQUER: MERGESORT

Link to practice handout
https://bit.ly/Divide-and-Conquer-Practice

Divide and Conquer Algorithms

Algorithm Approach:

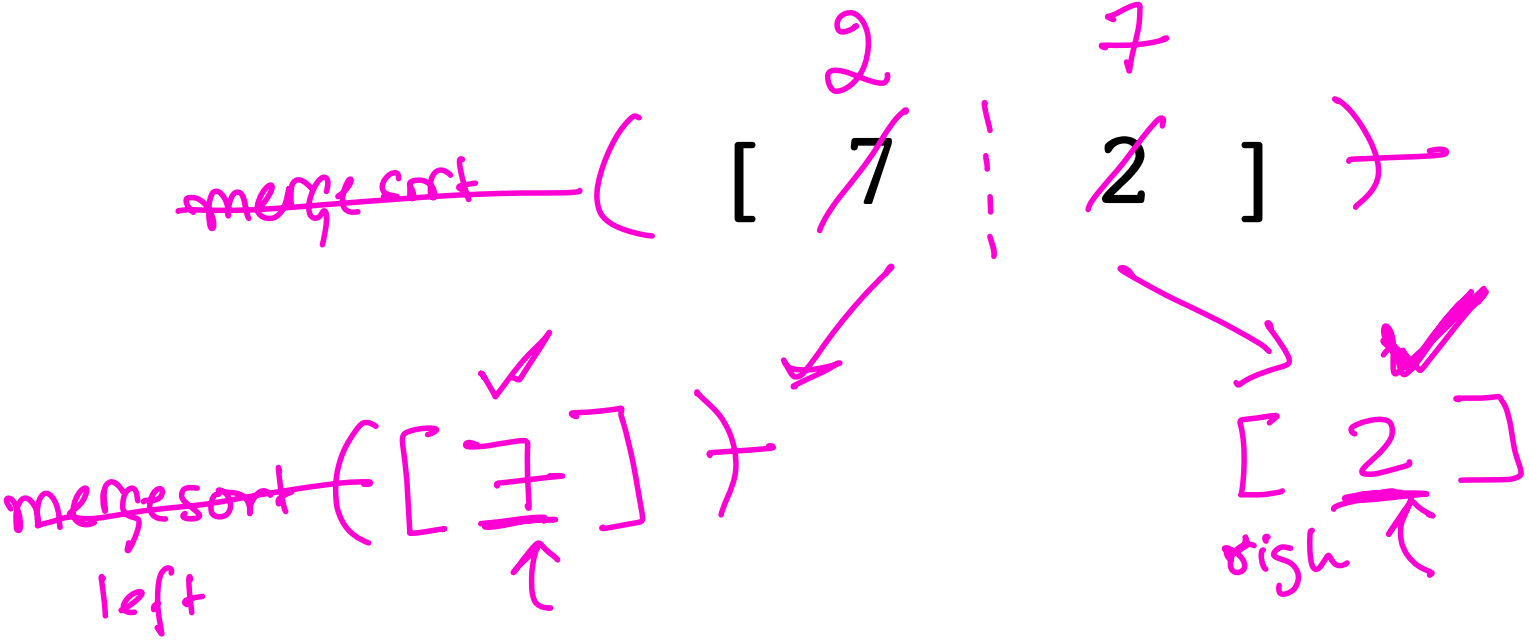
- **Divide** a large problem into sub-problems
- **Solve** each sub-problem
- **Combine** the solutions of sub-problems to obtain the solution for the original problem

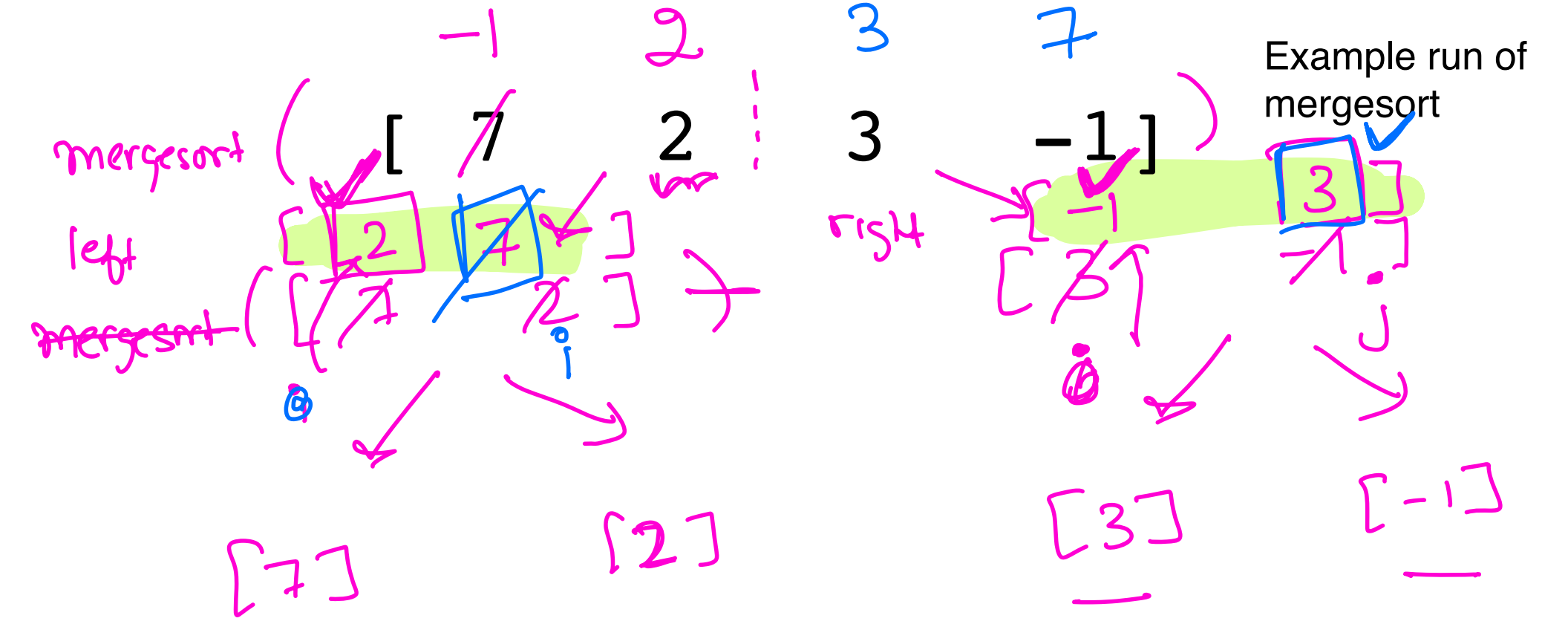
Merge Sort Algorithm

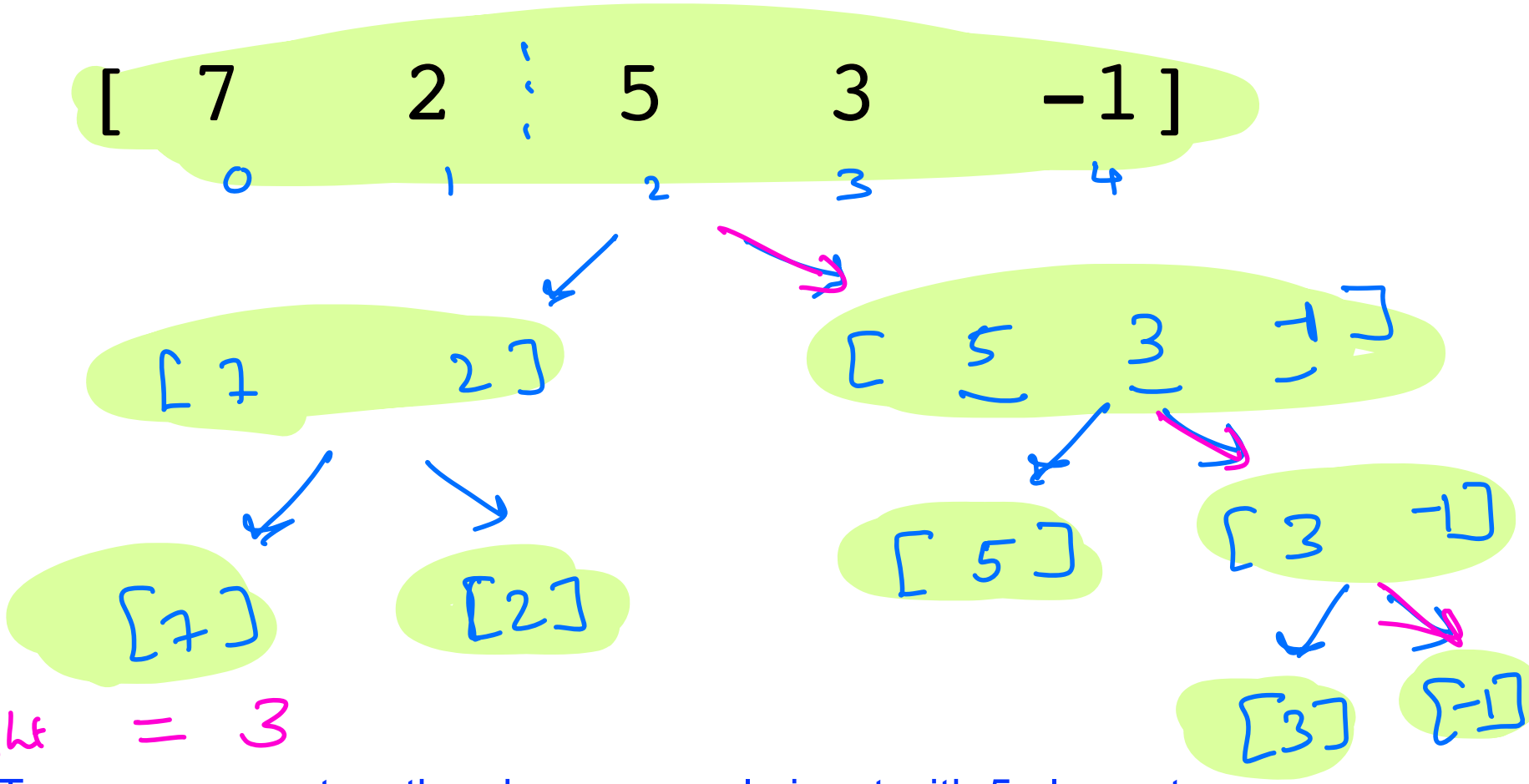
MergeSort(vector v)

- **Divide v into left half and right half**
- **Sort the left half, then sort the right half**
- **Combine (merge) the two sorted halves**

Example run of mergesort

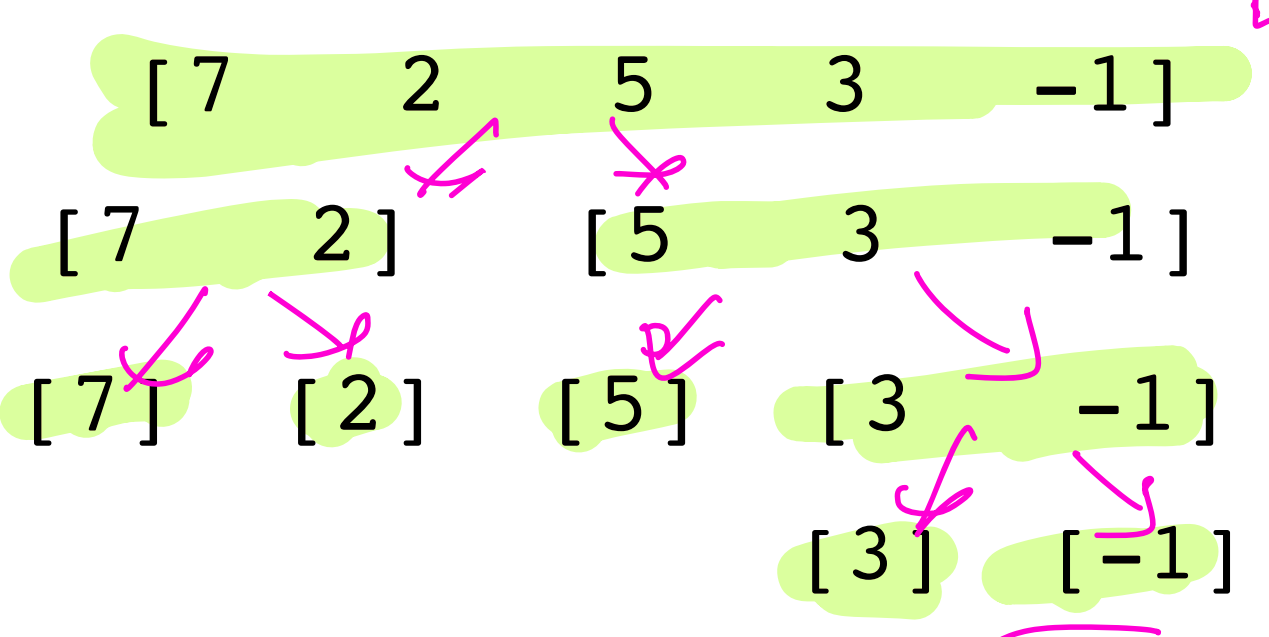



Example run of mergesort
mergesort
left
mergesort
right
[7 2 3 -1]
[7]
[2]
[3]
[-1]



Height = 3

Activity 1: Trace merge sort on the above example input with 5 elements
Draw the final binary tree representation of the trace.
What is the height of the resulting binary tree?



Level:

0

1 $O(n)$

2 $O(n)$

3 $O(1)$

Running time

$= \log(n)\,(O(n) + O(n)) + 2n-1$

$= O(n \log n)$

Running Time Analysis

T(n) = # copy operations to split lists +
comparisons to merge lists +
function calls

copy operations $\leq \log(n) \cdot O(n)$

comparisons to merge $\leq \log n \; O(n)$

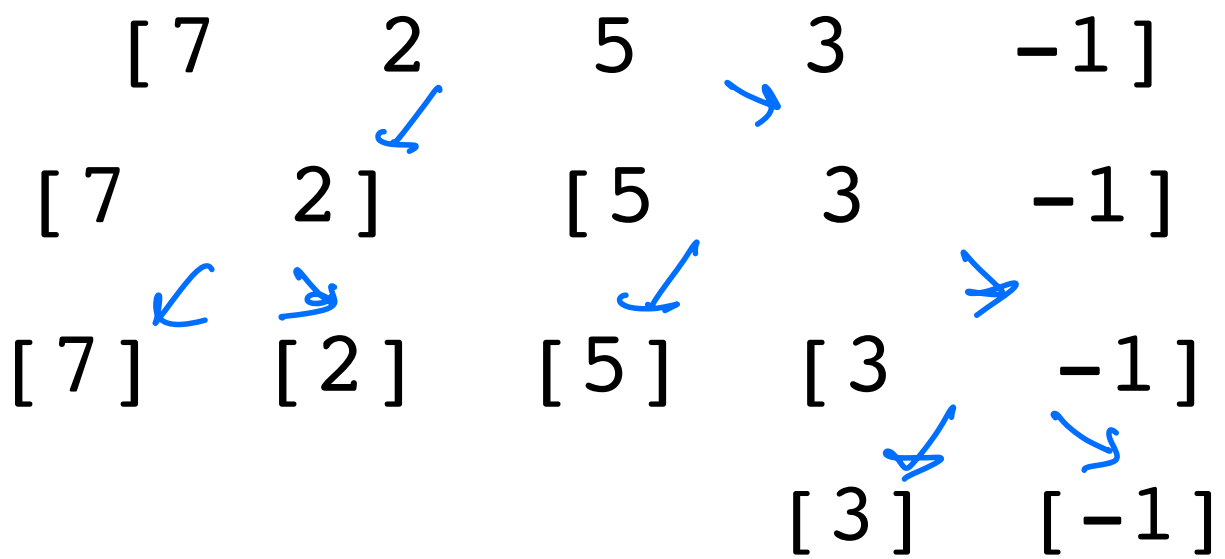
function calls

$\leq 2^0 + 2^1 + 2^2 + \cdots 2^{\log n}$

$= 2^{\log n + 1} - 1$

$= 2 \cdot 2^{\log n} - 1$

$= 2n - 1$



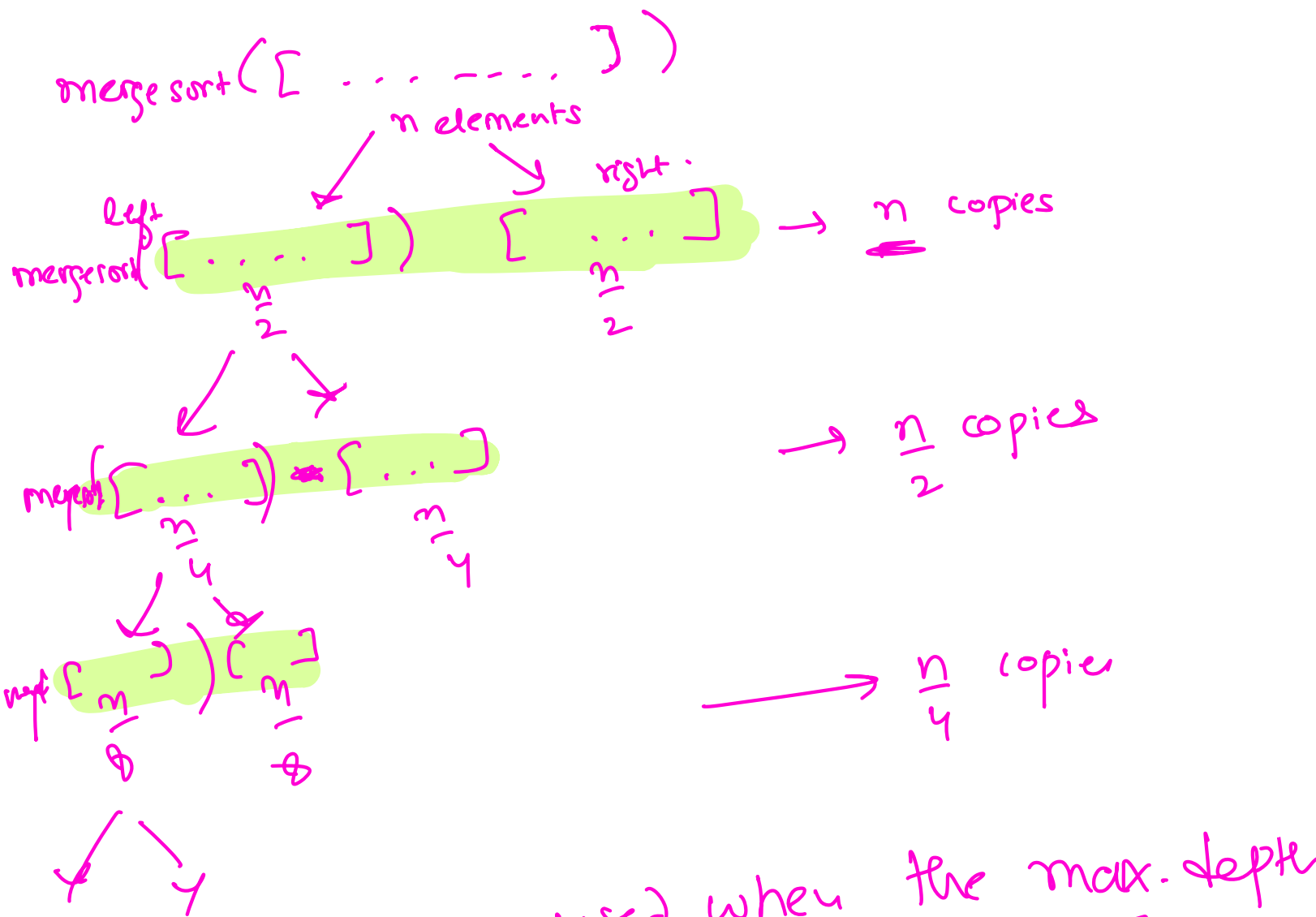
Space Complexity Analysis

How much additional space is used by the time mergsort reaches the base case?

A. n* log(n)
B. n + n/2 + n/4 + n/8 +….+ 1
C. n + n/2 + n/4 + n/8 +….+ 1 + log(n)
D. Something else

Analysis of space complexity of mergesort.

Main insight: Not all recursive calls are active at the same time



Consider space used when the max. depth of the recursion is reached. This is the peak auxilliary space usage

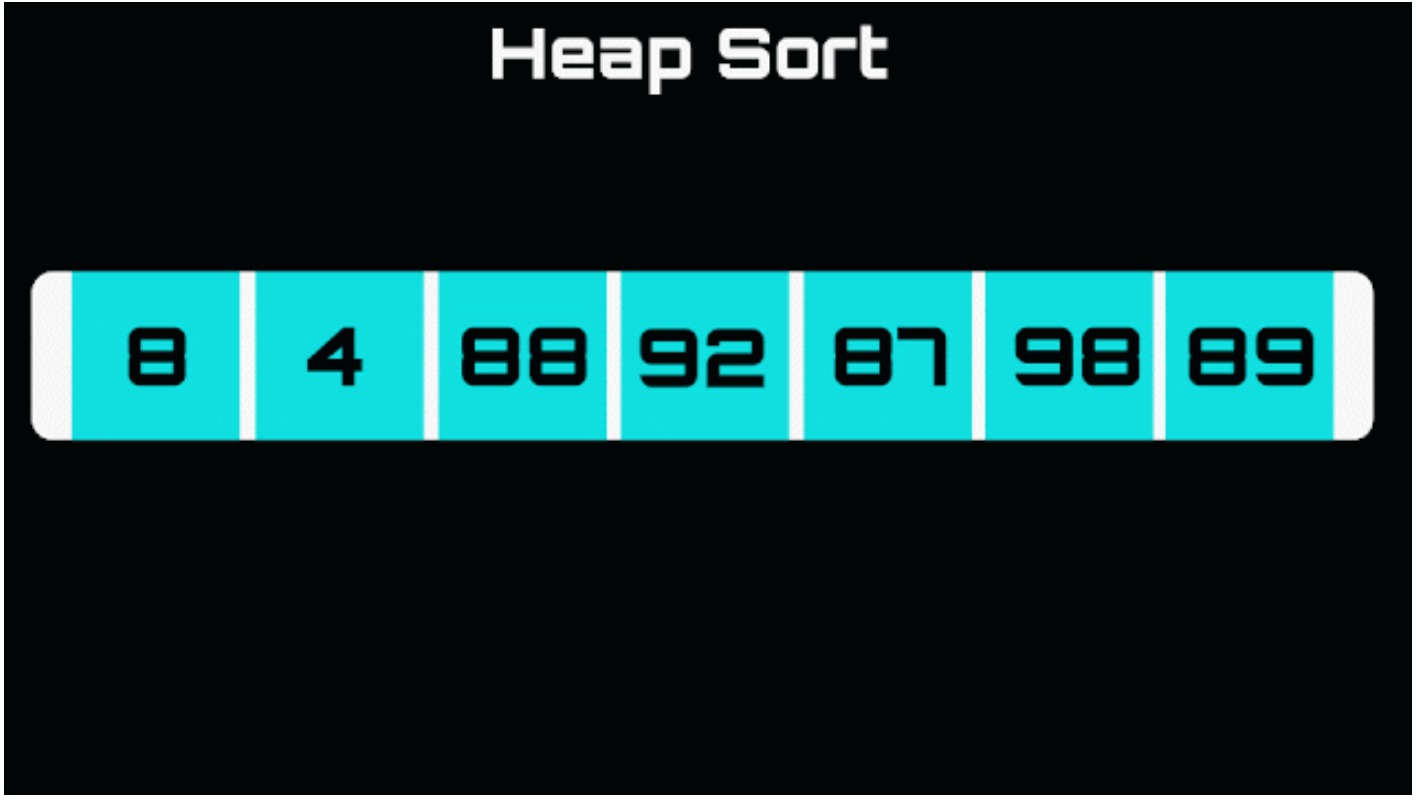
$$S(n) = n + \frac{n}{2} + \frac{n}{4} + \cdots + \frac{n}{n}$$

$$= n\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots\right)$$

Since the denominator for each term grows exponentially the summation converges to a constant

$$S(n) = n \cdot \text{constant}$$

$$= O(n)$$



Source: https://cosmictechie.hashnode.dev/heap-sort

HeapSort Algorithm

```
HeapSort(vector v)
    for i from n/2-1 down to 0:    // heapify: build max heap
      bubble_down(v, n, i)

    for i from n-1 down to 1:      // sort invariant:
      swap(v[0], v[i])                heap: v[0..i-1]
      bubble_down(v, i, 0)            sorted: v[i..n-1]
```

O(n)

O(n lg n)

Overall O(n log n)

Heapsort: https://visualgo.net/en/heap

Trace heapsort on input vector: 8, 4, 88, 92, 87, 98, 89

0 1 2 3 4 5 6

Step 1: Heapify

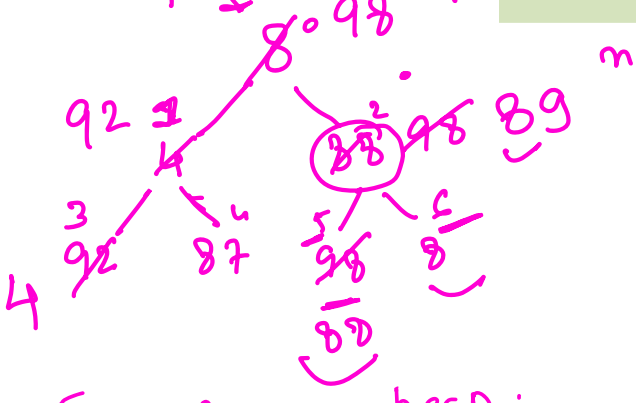
```
for i from n/2-1 down to 0:    // heapify: build max heap
  bubble_down(v, n, i)
```

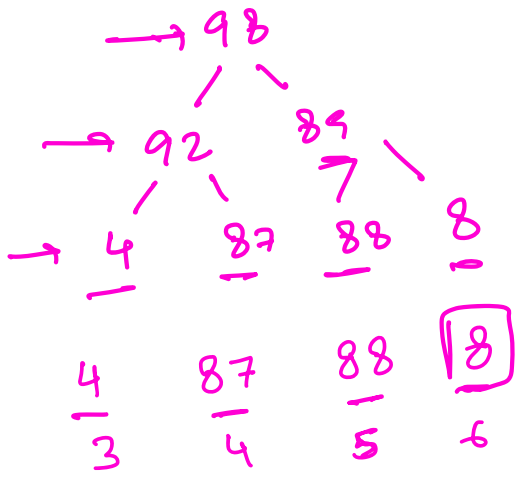
non-leaf nodes

O(n)

n = 7

$\frac{n}{2} - 1 = 2$





Expected max-heap:

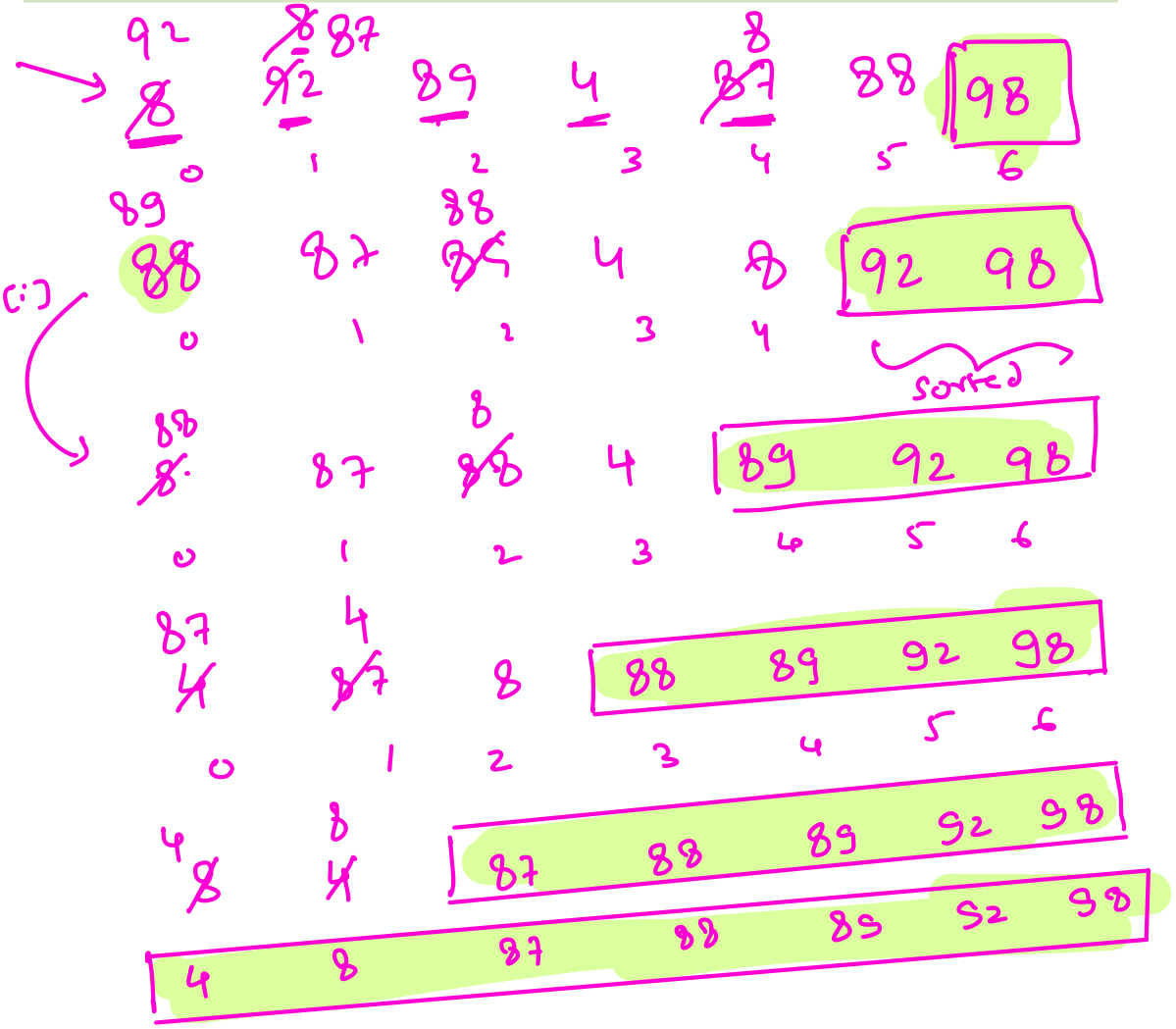
max: heap

0	1	2	3	4	5	6
98	92	89	4	87	88	8

Step 2:

```
for i from n-1 down to 1:      // sort invariant:
  swap(v[0], v[i])                heap: v[0..i-1]
  bubble_down(v, i, 0)            sorted: v[i..n-1]
```

Swap v[0], v[i]
Bubble down to fix heap property



Why Mergesort over Heapsort?

- Same O(n log n) time, but heapsort has poor cache performance - why?
- Mergesort is **stable**: equal elements keep their original relative order
- Mergesort parallelizes naturally