MERGE SORT:
DIVIDE AND CONQUER

Link to practice handout

https://bit.ly/Divide-and-Conquer-Practice

Divide and Conquer Algorithms

Algorithm Approach:

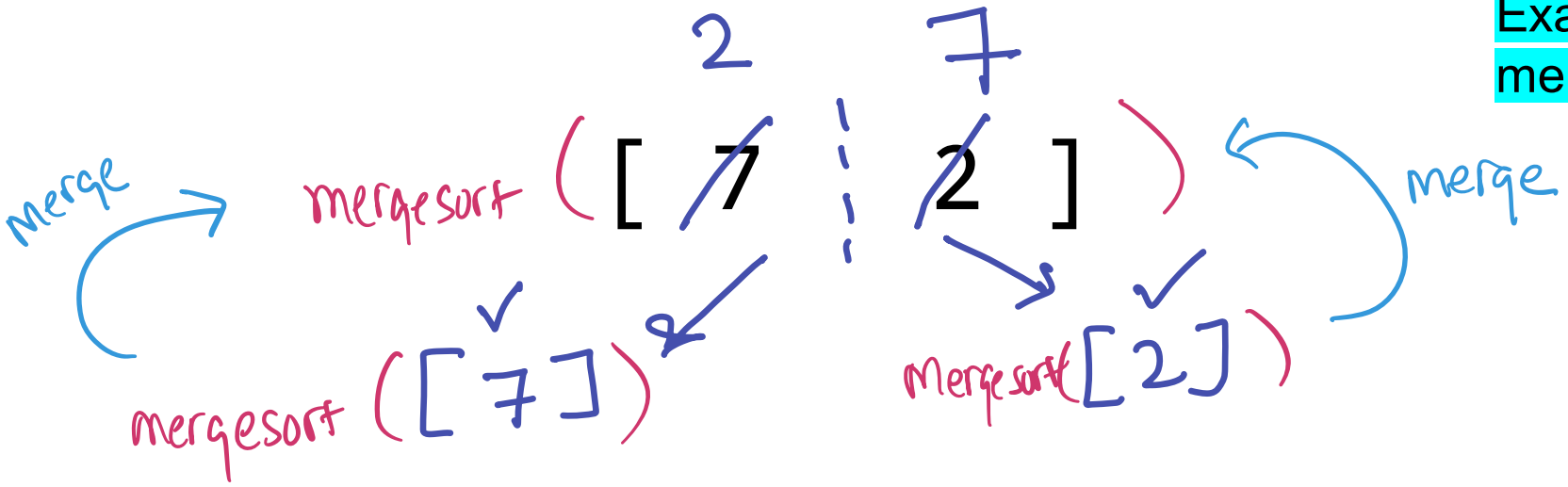
- **Divide** a large problem into sub-problems
- **Solve** each sub-problem
- **Combine** the solutions of sub-problems to obtain the solution for the original problem

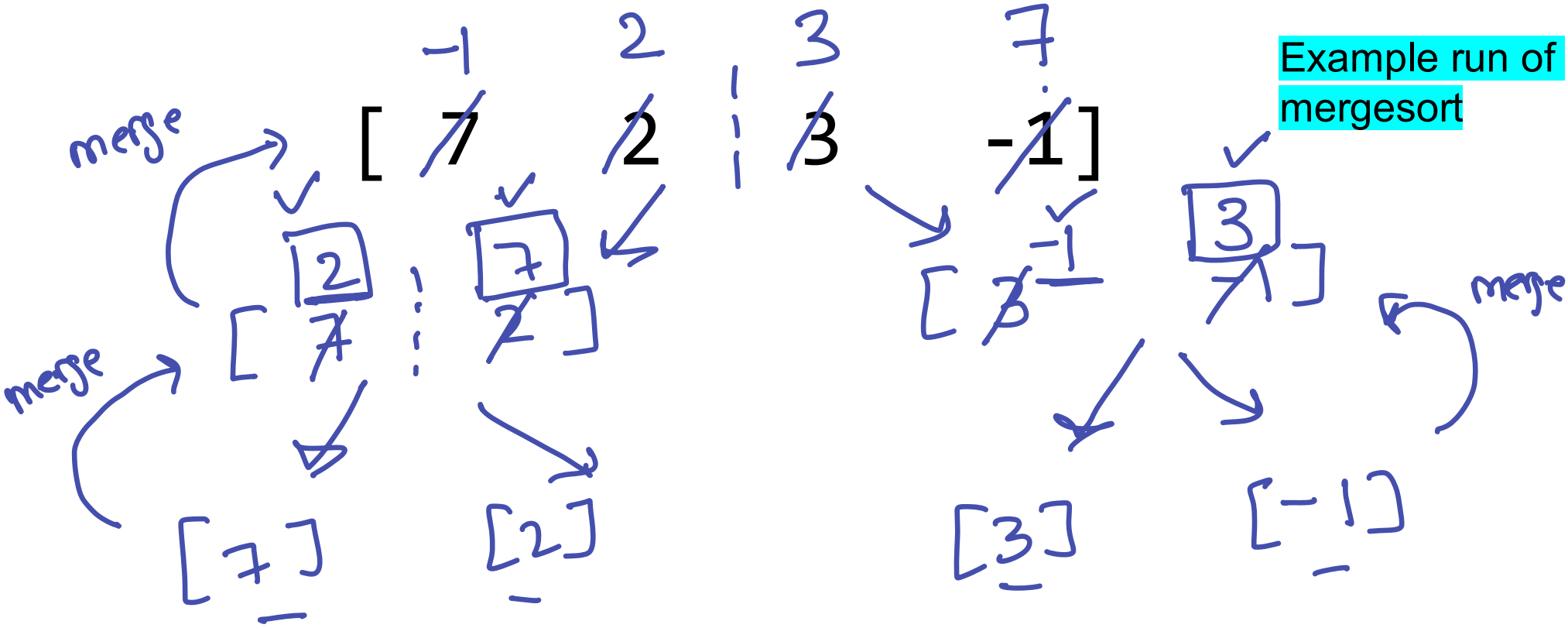
Merge Sort Algorithm

MergeSort(vector v)

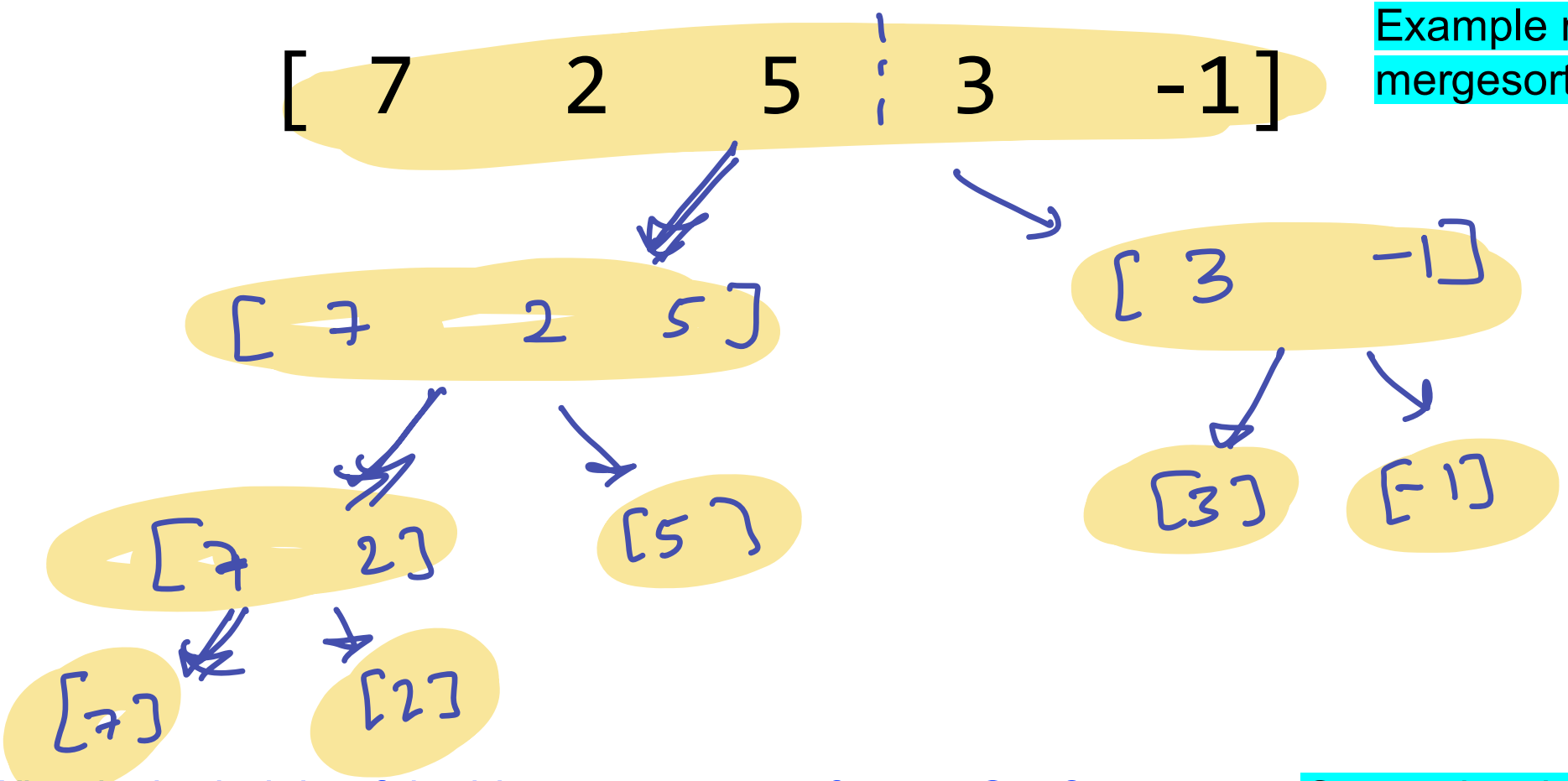
- **Divide v into left half and right half**
- **Sort the left half, then sort the right half**
- **Combine (merge) the two sorted halves**

Example run of mergesort




Example run of mergesort
merge
merge
merge
[7 2 3 -1]
[7 2]
[3 -1]
[7]
[2]
[3]
[-1]

Example run of mergesort



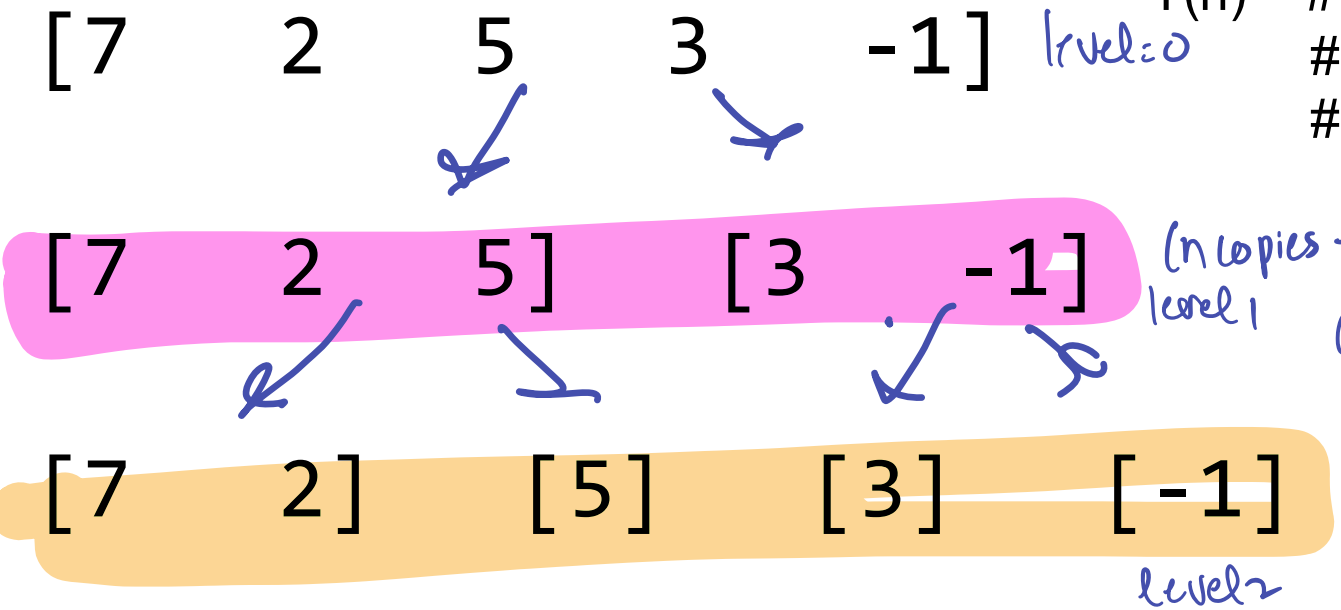
What is the height of the binary tree trace of mergeSort?

A. 1. B. 2 C. 3 D. 4 E. 5

Generalize the answer for an input vector of size n

Running Time Analysis

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



(n copies + n comparisons)

Observe that at each level, merge sort needs to copy n elements to split lists and n comparisons.

max number of levels, $H = \log n$

Function calls $\leq 2^0 + 2^1 + \dots 2^H$

$= 2^{H+1} - 1$

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

$= 2n - 1$

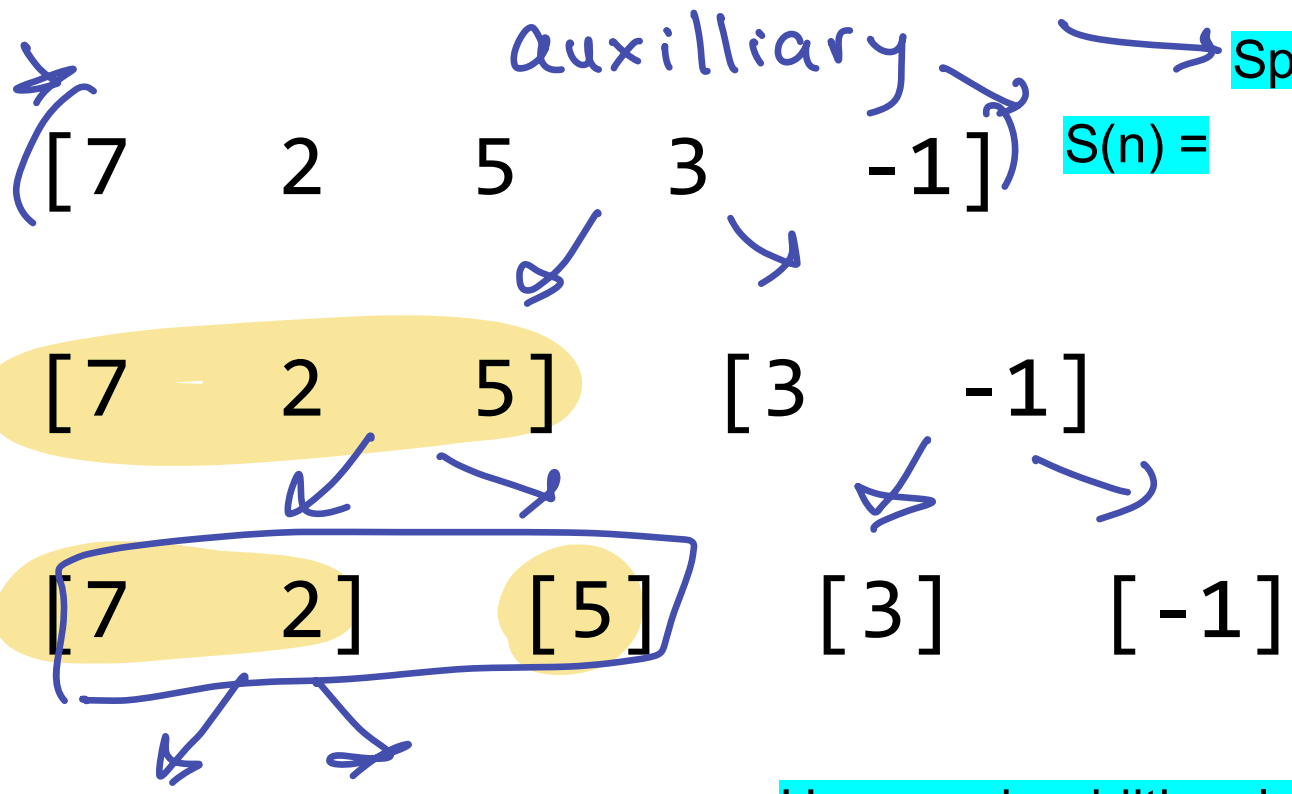
$T(n) = \log n * (O(n) + O(n)) + 2n + 1$

bound on the number of function calls

max depth of recursion

copies per level

comparisons per level



Space Analysis

S(n) =

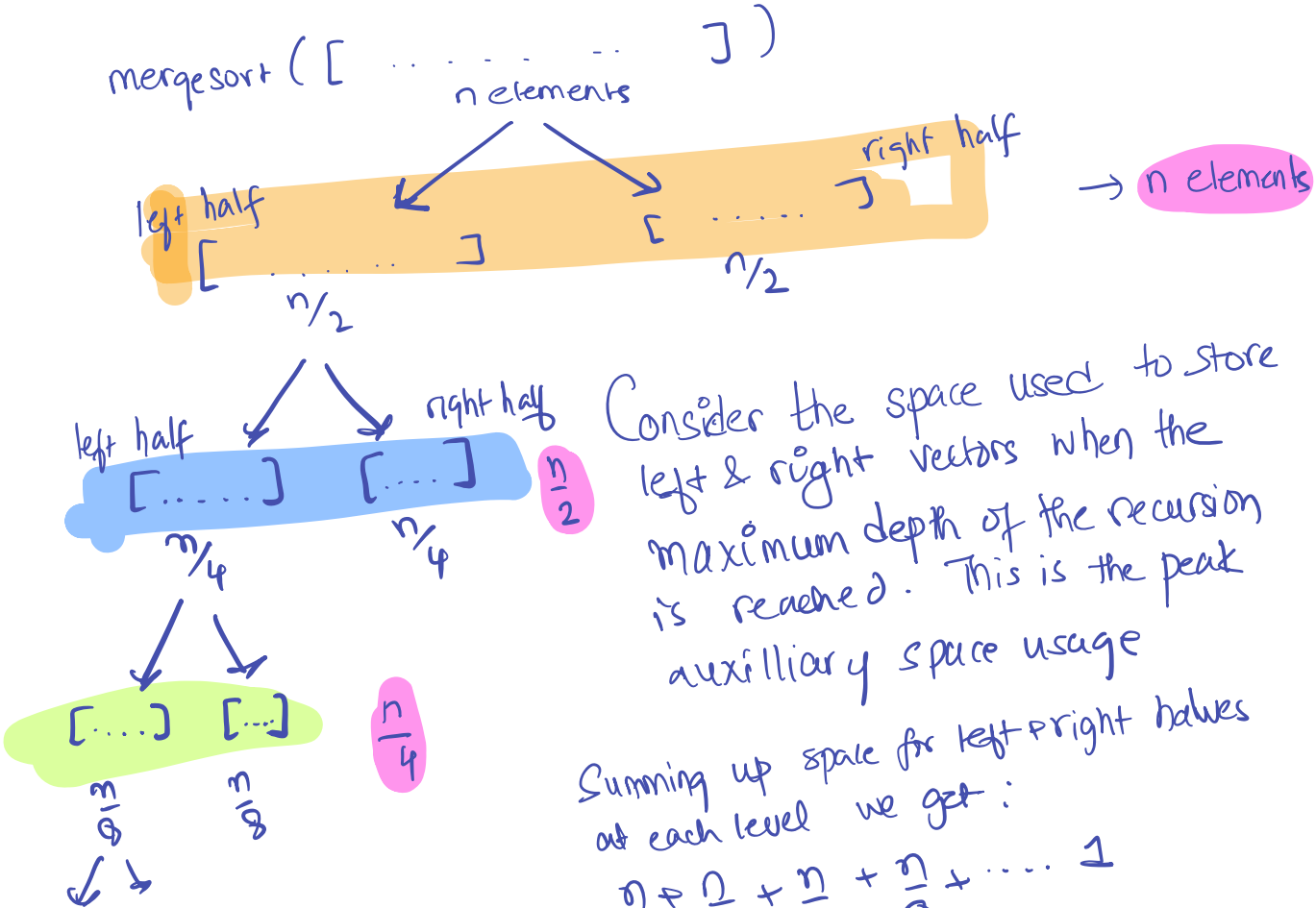
How much additional space is used by the time mergsort reaches the base case?

A. n* log(n) ← Assume all recursive calls are active simultaneously
B. n + n/2 + n/4 + n/8 +….+ 1
C. n + n/2 + n/4 + n/8 +….+ 1 + log(n)
D. Something else

S(n) = O(n) + O(log n) = O(n)

Analysis of Space Complexity of MergeSort.

main insight: Not all recursive calls are active simultaneously.



Consider the space used to store left & right vectors when the maximum depth of the recursion is reached. This is the peak auxilliary space usage

Summing up space for left+right halves at each level we get:

$$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots 1$$

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

converges to a constant

$$= O(n)$$

Space required for function calls alone until max depth of recursion = $O(\log(n))$

$$S(n) = n\left(1 + \frac{1}{2} + \cdots \frac{1}{n}\right) + \log(n) = O(n) + O(\log n) = O(n)$$

Path: a sequence of nodes in which each node is connected by an edge to the next.

3 → 5 → 6

Ancestor(u): any node that is on a path ending in u

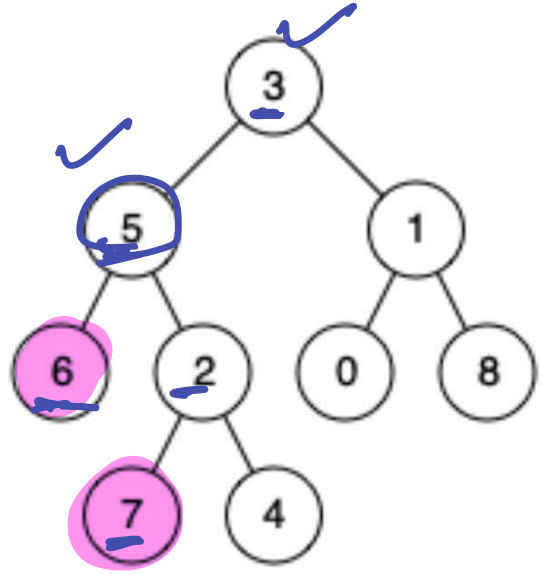
u = 5: 3 → 5 → 6

Descendant(v): any node that is on a path starting from v

v = 5: 3 → 5 → 2 → 4

Common ancestor(u, v): any node that is the ancestor of both u and v

Lowest Common ancestor(u, v): deepest node in the tree that is a common ancestor of u and v



https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/

Approach: Divide and Conquer

What is the LCA of each of the following?

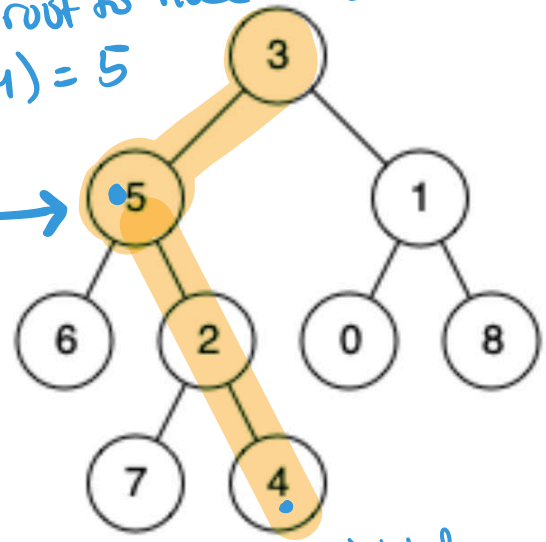
5 and 1: 3

5 and 4: 5

6 and 7: 5

Finding LCA(5,4)
Path from root to node 5: 3 → 5
Path from root to node 4: 3 → 5 → 2 → 4
LCA(5,4) = 5

LCA(5,4): deepest node in the tree that is common ancestor of 5 & 4



Base case:
if current node is either value or null return current node

Discuss how you would solve the problem with your neighbor, trace your solution, describe in words, then implement Leetcode or handout

Outline:
Subproblems to solve for divide & conquer
- Find LCA on left subtree
- Find LCA on right subtree
- Merge results to get LCA of root tree

Consider base cases

https://bit.ly/Divide-and-Conquer-Practice

https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/