Thought for 7 seconds

1.0a

Space complexity of an algorithm is the total amount of memory it requires as a function of input size. It consists of:

* **Fixed space** for instructions, constants and any variables whose size does not depend on n.
* **Variable space** for input-dependent data structures (e.g., arrays of size n) and the call/recursion stack.
* **Return space** to hold the output before it is returned.

For example, Merge Sort uses O(n) auxiliary array space (variable) atop its O(1) fixed working memory and O(1) return buffer .

1.0b

```
FIND(x):
if parent[x] ≠ x:
parent[x] = FIND(parent[x]) // path compression
return parent[x]
```

Path compression flattens the tree by making every node on the search path point directly to the root. In a graph-connectivity check, this means subsequent FIND calls on those nodes complete in near-constant time .

1.0c

Divide-and-Conquer is a design paradigm that (1) **divides** the problem into smaller subproblems of the same form, (2) **conquers** each subproblem (typically

via recursion), and (3) **combines** the sub-solutions into a final answer. For example, Quick Sort divides the array around a pivot, recursively sorts each part, then concatenates the results .

2.0b

Disjoint Sets (Union-Find) maintain a collection of non-overlapping sets supporting two operations:

1. **Find** locates the set representative (root) of element x. With **path compression**, each visited node on the way to the root is re-linked directly to the root, flattening the tree.
2. **Union** merges two sets by linking one root under the other. With **union by rank (or size)**, the shallower tree attaches under the deeper, keeping overall height small.

Together, these optimizations drive the amortized cost per operation to $O(\alpha(n))$, where α is the inverse Ackermann function that grows extremely slowly. In practice ——even for millions of elements——each UNION or FIND runs in effectively constant time. For instance, Kruskal's Minimum Spanning Tree algorithm uses Union-Find with path compression and union by rank to process edges in near-linear time. Similarly, dynamic connectivity in social-network clustering relies on these improvements to merge communities and answer connectivity queries in microseconds per operation.