Lesson Plan: Zombie Survival

Year Group: 8 | **Duration:** 50 minutes | **Topic:** Decomposition

1. Overview

Core Concept: Decomposition — breaking a complex problem into smaller, manageable sub-problems that can each be solved independently.

Learning Objectives:

- Define decomposition and explain why it is useful
- Break a complex problem into sub-problems using a hierarchy tree
- Identify dependencies between sub-problems
- Relate decomposition to how programs are structured as functions and modules

Key Vocabulary:

Term	Definition
Decomposition	Breaking a complex problem into smaller, more manageable sub-problems
Sub-problem	A smaller part of a larger problem
Hierarchy	An arrangement where items are ranked, with the main problem at the top
Dependency	When one sub-problem must be solved before another can start
Top-down design	Starting with the main problem and breaking it into smaller parts

2. Before the Lesson

Print:

- ☐ `worksheet-hierarchy-tree.md` — 1 copy per student

Gather:

- ☐ Sticky notes (optional — students can write sub-tasks and physically rearrange them)
- ☐ Coloured pens (for drawing dependency arrows in a different colour)

Room Setup:

- Groups of 3–4 students
- Large desk space preferred (A3 worksheet if possible)

3. Timed Lesson Flow

0–5 min — Hook: Zombie Chaos

1. Set the scene dramatically: *"A zombie apocalypse has just begun. You have exactly 2 minutes to write down everything you need to do to survive. GO."*
2. Students brainstorm chaotically — don't structure it.
3. After 2 minutes: *"How many items did you get? Are they in any order? Could you hand this list to a stranger and have them survive?"*
4. Key point: **an undecomposed problem is chaos. Decomposition organises it.**

5–10 min — Introduce Decomposition

1. Draw the top of the tree: **SURVIVE THE ZOMBIE APOCALYPSE**
2. Ask: *"What are the biggest categories?"* Elicit: Food, Shelter, Safety, Communication, Medical
3. Draw Level 1 branches.
4. Key point: **each branch is a sub-problem we can tackle separately.**

10–15 min — Go Deeper

1. Take one branch (e.g., Food). Ask: *"What does 'get food' actually involve?"*
2. Elicit level-2 items: Find sources, Preserve food, Transport food, Purify water.
3. Go deeper: "Find sources" → Identify edible plants, Locate supermarkets, Set up traps.
4. Stress: **leaves of the tree should be specific, actionable tasks.**

15–35 min — Group Work: Build the Full Tree

1. Distribute `worksheet-hierarchy-tree.md`.
2. Groups fill in the full hierarchy, going at least 3 levels deep.
3. Every leaf node should be specific and doable.
4. Circulate and push groups who stay at 2 levels: *"What does that actually involve?"*

35–43 min — Identify Dependencies

1. Ask: *"Which sub-problems depend on each other?"* (You can't get food before having a safe route.)
2. Draw arrows between branches showing dependencies, in a different colour.
3. Label each arrow: WHY does this dependency exist?

43–48 min — CS Connection

1. *"How do programmers use decomposition?"*
2. Large programs are broken into functions and modules. Each function solves one sub-problem.
3. *"If you were writing a zombie survival app, what function names might you write?"* (e.g., `find_food()`, `check_water_safe()`, `alert_nearby_survivors()`)

48–50 min — Key Takeaway

---

4. Teacher Facilitation Notes

What to look for:

- Groups who stay at level 1 or 2 — push them deeper
- Groups who create a flat list — ask *"which of these is a sub-category of another?"*

- Groups missing important categories (medical, communication) — prompt: *"what if someone gets hurt?"*

Common misconceptions:

- Decomposition is just making a list — stress the HIERARCHY (tree structure)
- All sub-problems are equal — dependencies show some must happen first
- You need to solve everything at once — decomposition lets you focus on one part at a time

---

5. Extension / Early Finisher Tasks

1. **Parallel vs sequential:** Mark each dependency as "must happen before" or "can happen at the same time."
2. **Function naming:** Write a function name + one-line description for every leaf node.
3. **Complexity estimate:** Assign each leaf Easy/Medium/Hard. Which branch is hardest overall?

---

6. Key Takeaway

> **Decomposition breaks impossible problems into solvable ones. Every complex program is built by decomposing the problem into functions — each solving exactly one sub-problem.**