Divide & Conquer

10.020 Data Driven World

Natalie Agus

Learning Objectives

- **Recursion**:
  - Solve problems using recursion.
  - Identify problems that has recursive solutions.
  - Identify **base case** and **recursive case** in a recursive problem and its solution.
  - Explain and implement the recursive solution of Tower of Hanoi.
  - Derive solution for **recurrence of Tower of Hanoi** using recursion-tree method
- **Merge-Sort**:
  - Explain and **implement** merge sort algorithm.
  - Derive solution of recurrence of merge sort using **recursion-tree method**.
  - Measure computation time of merge sort and compare it with the other sort algorithms.

Recursion

Definition

- It is a function that **calls itself** to solve **smaller problems**
- Requires a **base case** to stop recursion and prevent infinite loops, then it has **recursive case to solve repetitive substructure**
- Often used for problems that have **repetitive substructure** (e.g., factorial, Fibonacci, tree traversal, Tower of Hanoi)

Identify the base case and recursive case

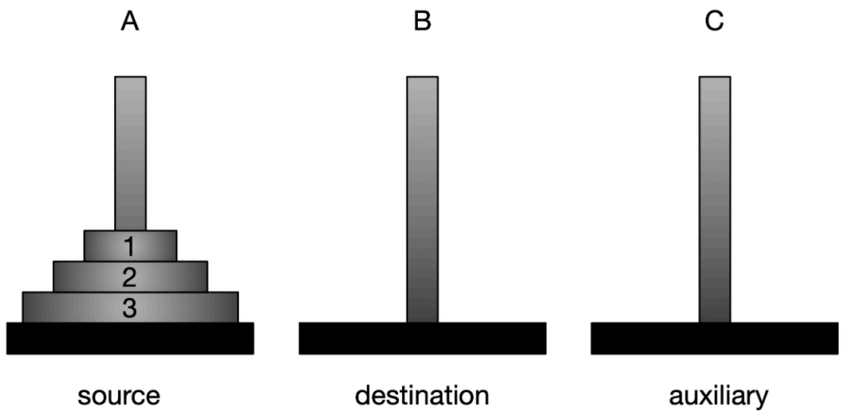
```
def factorial(n):
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result

```

Identify the base case and recursive case

```
1 def sum(array):
2     result = 0
3     for number in array:
4         result += number
5     return result
6
7 input_array = [4, 3, 2, 1, 7]
8 print(sum(input_array))
```

Identify the base case and recursive case



Show Pseudocode

```
Input:
  - n, number of disks
  - source tower
  - destination tower
  - auxiliary tower
Output:
  - sequence of steps to move n disks from source to destination tower using auxiliary to
Steps:
1. if n is 1 disk:
    1.1 Move the one disk from source to destination tower
2. otherwise, if n is greater than 1:
    2.1 Move the first n - 1 disks from source to auxiliary tower
    2.2 Move the last disk n from source to destination tower
    2.3 Move the first n - 1 disks from the auxiliary tower to the destination tower
```

Complexity

Time Complexity (there's also space complexity)

Factorial?

Simplify & draw the recurrence tree

Complexity

Time Complexity (there's also space complexity)

Sum?

Simplify & draw the recurrence tree

Complexity

Time Complexity (there's also space complexity)

Tower of Hanoi?

Simplify & draw the recurrence tree

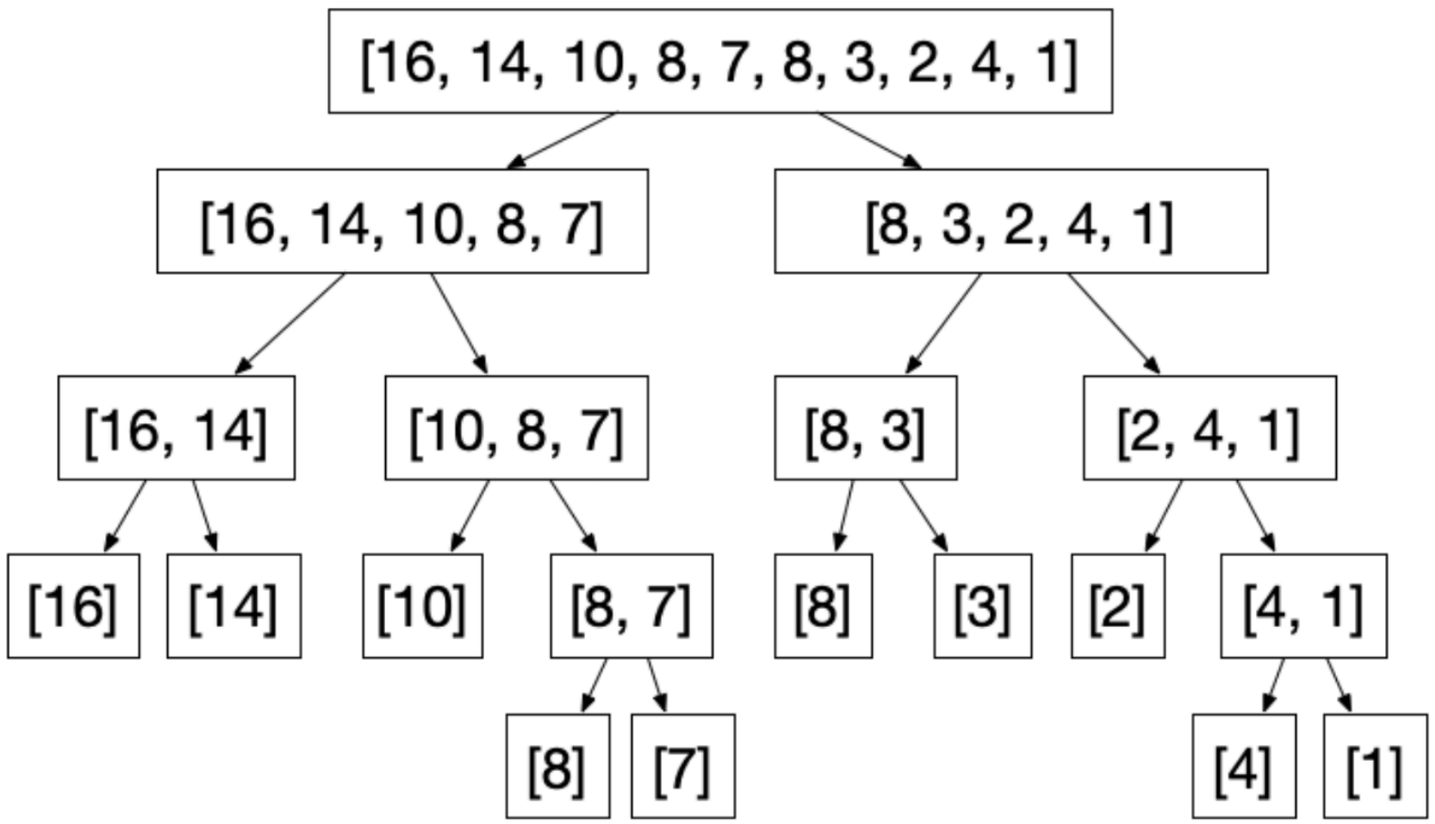
Merge Sort

A sorting algorithm that implements the principle of divide and conquer

```
[16, 14, 10, 8, 7, 8, 3, 2, 4, 1]
```

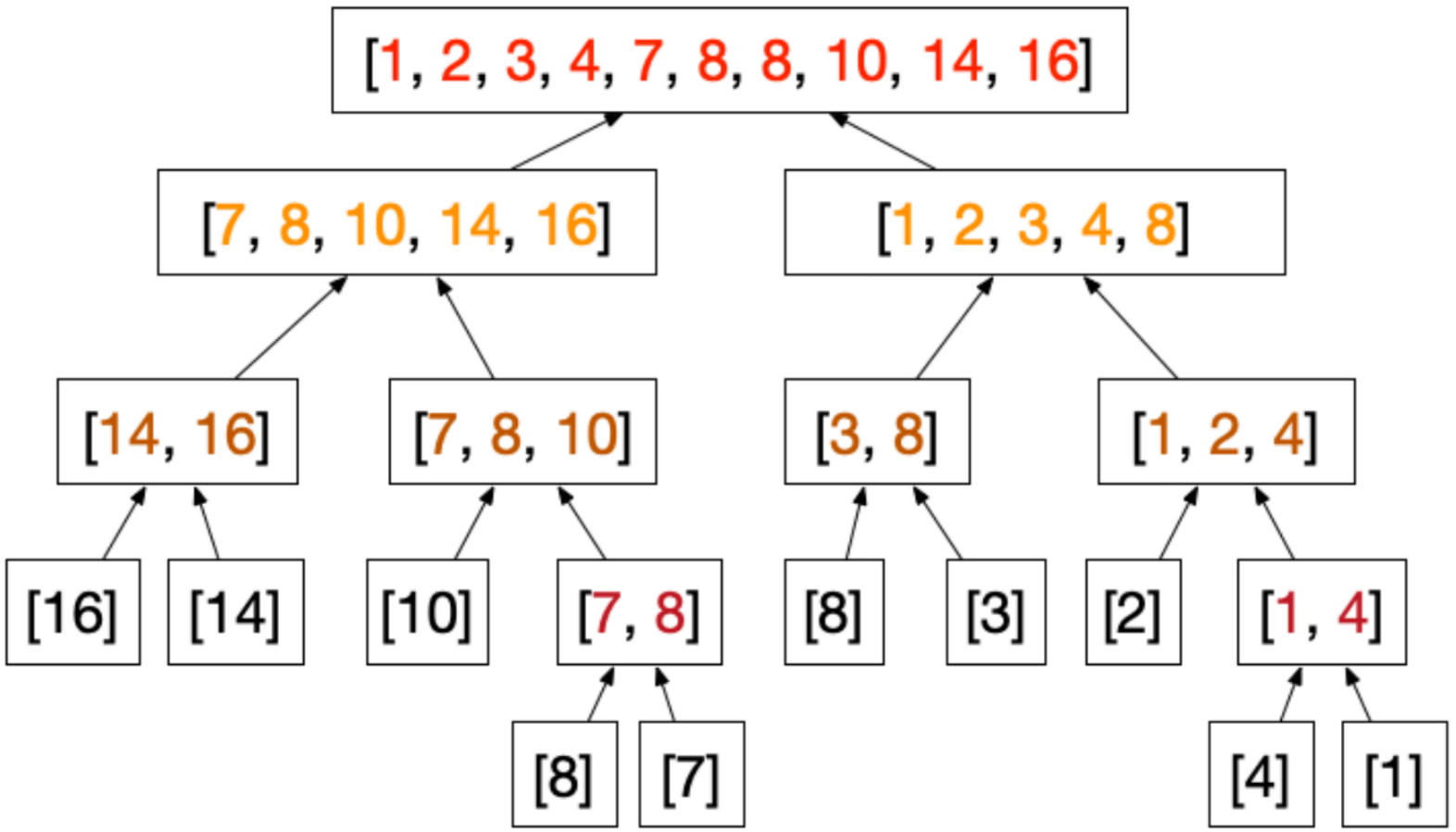
Step 1: Split

Stop until you just have 1 element



Step 2: Merge

Compare and merge two arrays



Complexity

Time complexity of merge-sort algorthm

Simplify & draw the recurrence tree

Complexity Summary Table

Algorithm	Time Complexity	Space Complexity
Sum (recursive)	O(n)	O(n)
Sum (iterative)	O(n)	O(1)
Factorial (recursive)	O(n)	O(n)
Factorial (iterative)	O(n)	O(1)
Tower of Hanoi	$O(2^n)$	O(n)
Merge Sort	O(n log n)	O(n)

Why Recursion?

We can use iterative method + some helper data structure

- However, recursion is **superior** when the problem has a **self-similar structure** **(can divide & conquer)**: trees, nested data

- Recursive calls mirror the problem shape ***naturally***
  - Back-tracking and divide and conquer tasks
  - Iterative solutions require manual stacks and control flow

- Recursive functions are **modular** and **composable**, making them easier to **read**, **test**, and **reuse** than loop-based or stack-simulated alternatives

Summary

- **Recursion**:
  - Solve problems using recursion.
  - Identify problems that has recursive solutions.
  - Identify **base case** and **recursive case** in a recursive problem and its solution.
  - Explain and implement the recursive solution of Tower of Hanoi.
  - Derive solution for **recurrence of Tower of Hanoi** using recursion-tree method
- **Merge-Sort**:
  - Explain and **implement** merge sort algorithm.
  - Derive solution of recurrence of merge sort using **recursion-tree method**.
  - Measure computation time of merge sort and compare it with the other sort algorithms.