CBS Department

Sorting Algorithms – Merge Sort

Soma Soleiman Zadeh
Data Structures & Algorithms (CBS 216)
Spring 2025 - 2026
Week 13
April 29-30, 2026

Outline

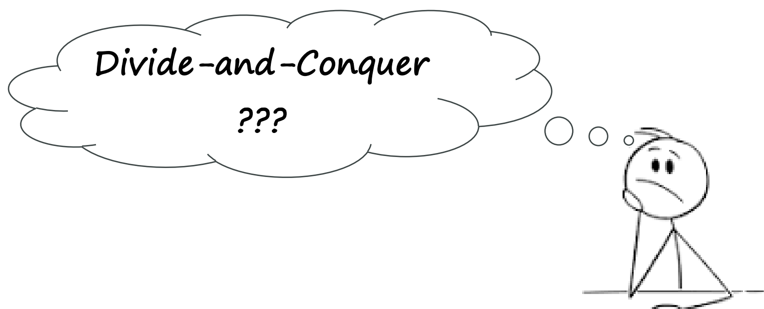
- **Merge Sorting** Algorithm
- **Divide-and-Conquer** Approach
- When to Use **Merge Sort**?
- Implementation of **Merge Sort** in Python
- Time Complexity and Space Complexity

Time Complexity & Space Complexity

Sorting Algorithm	Time Complexity			Space Complexity
	Best case	Average case	Worst case	Worst case
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Bubble	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Quick	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(n)$
Merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$

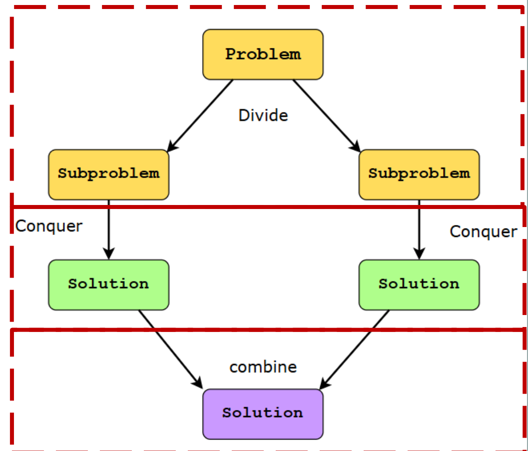
Merge Sort

- The **Merge Sort** is an efficient sorting algorithm that is faster than sorting algorithms like **Bubble sort** and **Insertion sort**.
- The **Merge Sort** follows the **Divide-and-Conquer** approach.



Divide-and-Conquer Approach

- The **Divide-and-Conquer** approach is a problem-solving technique that solves a complex problem in three main steps:
    1. **Divide:** splitting the main problem into smaller sub-problems.
    2. **Conquer:** solving each sub-problem recursively.
    3. **Combine:** merging the solutions of sub-problems to form the solution to the original problem.



How Merge Sort Works?

- Merge sort works by recursively dividing the unsorted list into smaller lists until each sub-list contains only one element.
- Then, it merges these sub-lists back into a sorted list. The basic steps are:
    - **Split** the unsorted list into two halves.
    - **Recursively sort each half**.
    - **Merge** the sorted halves into a single sorted list.

How Merge Sort Works?

Unsorted Data

8	2	6	4	5

Goal

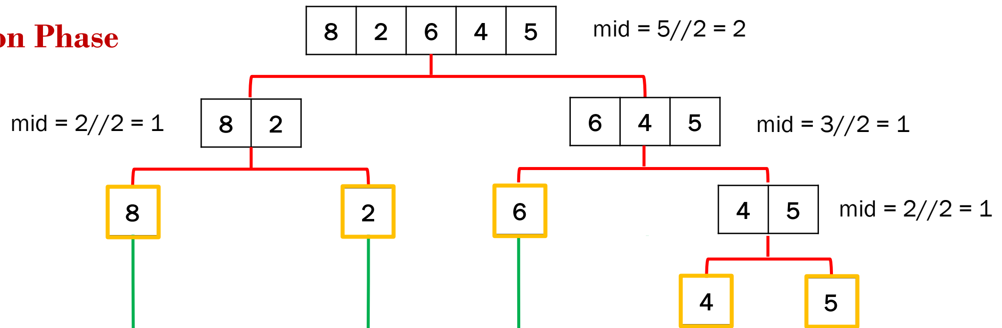
Sorted Data

2	4	5	6	8

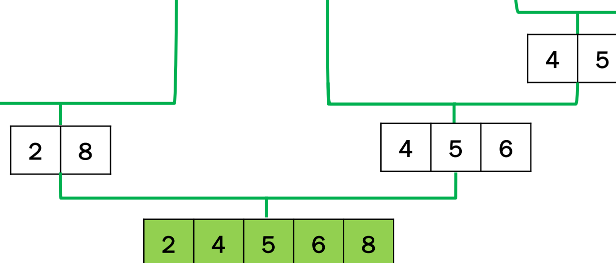
We can show the process of **Merge Sort** algorithm in two phases:

- **Division** Phase
- **Merging** Phase

Division Phase



Merging Phase



2	4	5	6	8

When to Use Merge Sort?

- **Merge Sort** is an efficient sorting algorithm where:
  - **Large Datasets** need to be sorted efficiently.
  - **External sorting** is necessary, such as when data is too large to fit into memory (such as RAM).
- Merge sort performs far better than Bubble Sort and Insertion Sort.

Merge Sort Implementation

```
################## Merge Sort Function ###################

def mergeSort(data):
    # Base Case
    if len(data)<=1:
        return data

    # Dividing the dataset to Two Halves (Left and Right)
    mid = len(data)//2
    left = data[:mid]
    right = data[mid:]

    # Recursion (Calling the algorithm for each half)
    mergeSort(left)
    mergeSort(right)

    # Merging Two Halves
    i = 0
    j = 0
    k = 0
    while i<len(left) and j<len(right):
        if left[i]<= right[j]:
            data[k] = left[i]
            i += 1
        else:
            data[k] = right[j]
            j += 1
        k += 1

    # Handling the Remaining Items in Both Halves
    while i<len(left):
        data[k] = left[i]
        i += 1
        k += 1
    while j<len(right):
        data[k] = right[j]
        j += 1
        k += 1
    return data
```

Merge Sort Implementation (Step 1)

```
################### Merge Sort Function ####################

def mergeSort(data):
    # Base Case
    if len(data)<=1:
        return data

    # Dividing the dataset to Two Halves (Left and Right)
    mid = len(data)//2
    left = data[:mid]
    right = data[mid:]

    # Recursion (Calling the algorithm for each half)
    mergeSort(left)
    mergeSort(right)
```

Merge Sort Implementation (Step 2)

```
# Merging Two Halves

i = 0
j = 0
k = 0
while i<len(left) and j<len(right):
    if left[i]<= right[j]:
        data[k] = left[i]
        i += 1
    else:
        data[k] = right[j]
        j += 1

    k += 1
```

Merge Sort Implementation (Step 3)

```
# Handling the Remaining Items in Both Halves

while i<len(left):
    data[k] = left[i]
    i += 1
    k += 1

while j<len(right):
    data[k] = right[j]
    j += 1
    k += 1

return data
```

Merge Sort: Time and Space Complexity

Sorting Algorithm	Time Complexity (Worst Case)	Space Complexity	Stability
Merge Sort	$O\ (n\ \log n)$	$O\ (n)$	**Stable**

- **Efficient (Fast) Sorting Algorithm**
- **Good for Sorting Large Datasets**
- **Needs Extra Memory (Out-of-Place Sorting Algorithm)**