

Indian Institute of Technology Madras
ZANZIBAR CAMPUS

Z5007 · PROGRAMMING AND DATA STRUCTURES

Merge Sort: Divide and Conquer

Lecture 16 · Wednesday 4 November 2026 · Canning, Broder & Lafore Ch 7; CLRS 2.3

Dr. Innocent Nyalala | MTech Semester 1 | IIT Madras Zanzibar

Today's Plan

IIT Madras Zanzibar

1. (0–5 min) Why we need a faster sort: n^2 against n log n in numbers
2. (5–15 min) The merge step: two sorted lists into one, finger by finger
3. (15–26 min) Merge sort: split tree, merge tree, the code, the order of the calls
4. (26–35 min) The cost: the recursion tree for $T(n) = 2T(n/2) + n$
5. (35–40 min) Stability, extra memory, where merge sort is used
6. (40–50 min) Live demo, practice questions, your questions

Learning Outcomes

IIT Madras Zanzibar

By the end of this session, you will be able to:

Merge	two sorted lists into one in O(n), keeping equal items in order
Trace	merge sort on a small list: the split tree going down and the merge tree coming up
Explain	why the running time is O(n log n), using the recursion tree
Compare	merge sort with insertion sort on time, memory and stability
Write	merge and merge_sort in Python (tomorrow's lab)

Key Terms for Today

IIT Madras Zanzibar

Term	Meaning	Example
Divide and conquer	split a problem into smaller copies, solve them recursively, combine	merge sort, quicksort
Merge	combine two SORTED lists into one sorted list	[3, 27] + [9] → [3, 9, 27]
Stable sort	items with equal keys keep their original order	two students with 85
In place	sorts using only O(1) extra memory	insertion sort: yes
Auxiliary memory	extra memory beyond the input itself	merge sort: O(n)
Recursion tree	a picture of all the calls, with the work done in each	levels of n/2, n/4, …
$\log_2 n$	how many times n can be halved before reaching 1	$\log_2 1024 = 10$

Merge Sort

IIT Madras Zanzibar

→ **Why We Need a Faster Sort**

The Merge Step

Merge Sort

Its Cost: n log n

Properties and Practice

Demo and Practice

Why We Need a Faster Sort

IIT Madras Zanzibar

n	$n^2 / 4$ (insertion sort, average)	$n \log_2 n$ (merge sort)	ratio
1,000	250,000	about 10,000	25×
100,000	2.5×10^9	about 1.7×10^6	about 1,500×
1,000,000	2.5×10^{11}	about 2×10^7	about 12,500×

n (demo race)	insertion (s)	merge (s)
2,000	0.111	0.007
4,000	0.459	0.013
8,000	1.877	0.027

1. Doubling n: insertion ×4.
2. Merge sort: just over ×2.

? **Think:** a sort takes 1 s for 10,000 items. About how long for 20,000 if $O(n^2)$? If $O(n \log n)$?

Divide and Conquer: Sorting a Pile of Exam Papers

IIT Madras Zanzibar

1 Divide

split the list into two halves: just compute mid

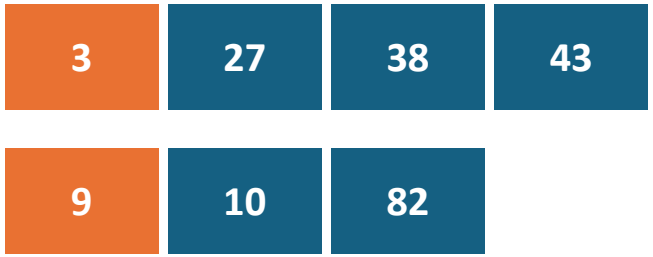
2 Conquer

sort each half recursively (trust the call)

3 Combine

merge the two sorted halves into one

Two sorted piles: look only at the top paper of each, take the smaller, repeat.



tops: 3 and 9 → take 3; then 27 and 9 → take 9; …

? **Think:** which step does the real work: dividing or combining?

Merge Sort

IIT Madras Zanzibar

- Why We Need a Faster Sort
- → **The Merge Step**
- Merge Sort
- Its Cost: n log n
- Properties and Practice
- Demo and Practice

Merging Two Sorted Lists: Two Fingers



i=0 j=0	**3 ≤ 9: take 3 from L**	3
i=1 j=0	**27 > 9: take 9 from R**	3 9
i=1 j=1	**27 > 10: take 10 from R**	3 9 10
i=1 j=2	**27 ≤ 82: take 27 from L**	3 9 10 27
i=2 j=2	**38 ≤ 82: take 38**	3 9 10 27 38
i=3 j=2	**43 ≤ 82: take 43; L used up**	3 9 10 27 38 43
i=4 j=2	**copy the rest of R: 82**	3 9 10 27 38 43 82

1. Compare the two fronts.
2. Take the smaller; move that finger.
3. One side empty: copy the rest.

? **Think:** why is it enough to compare only the two FRONT items?

The merge Code, Line by Line

IIT Madras Zanzibar

```
 1  def merge(L, R):
 2      out, i, j = [], 0, 0
 3      while i < len(L) and j < len(R):
 4          if L[i] <= R[j]:
 5              out.append(L[i]); i += 1
 6          else:
 7              out.append(R[j]); j += 1
 8      out.extend(L[i:])
 9      out.extend(R[j:])
10      return out
```

Line by line

Line 2	empty output; fingers at the front of L and R
Line 3	loop while both lists have items
Line 4	compare the fronts (≤: ties go left)
Line 5	take from L and move i
Line 7	take from R and move j
Line 8–9	one list is empty: copy the rest of the other
Cost	each item handled once: O(len L + len R)

? **Think:** largest number of comparisons when merging lists of sizes 4 and 3?

Ties Go Left: Why Merge Sort Is Stable

IIT Madras Zanzibar

Before (roll order)	mark	half
Asha	85	left
Baraka	70	left
Juma	85	right
Neema	92	right

After, with ≤ (stable)	mark	
Baraka	70	
Asha	85	first
Juma	85	second
Neema	92	

Definition: **Stable sort:** a sort in which items with equal keys keep their original relative order.

Use: sort by roll number, then (stably) by mark: equal marks stay in roll-number order.

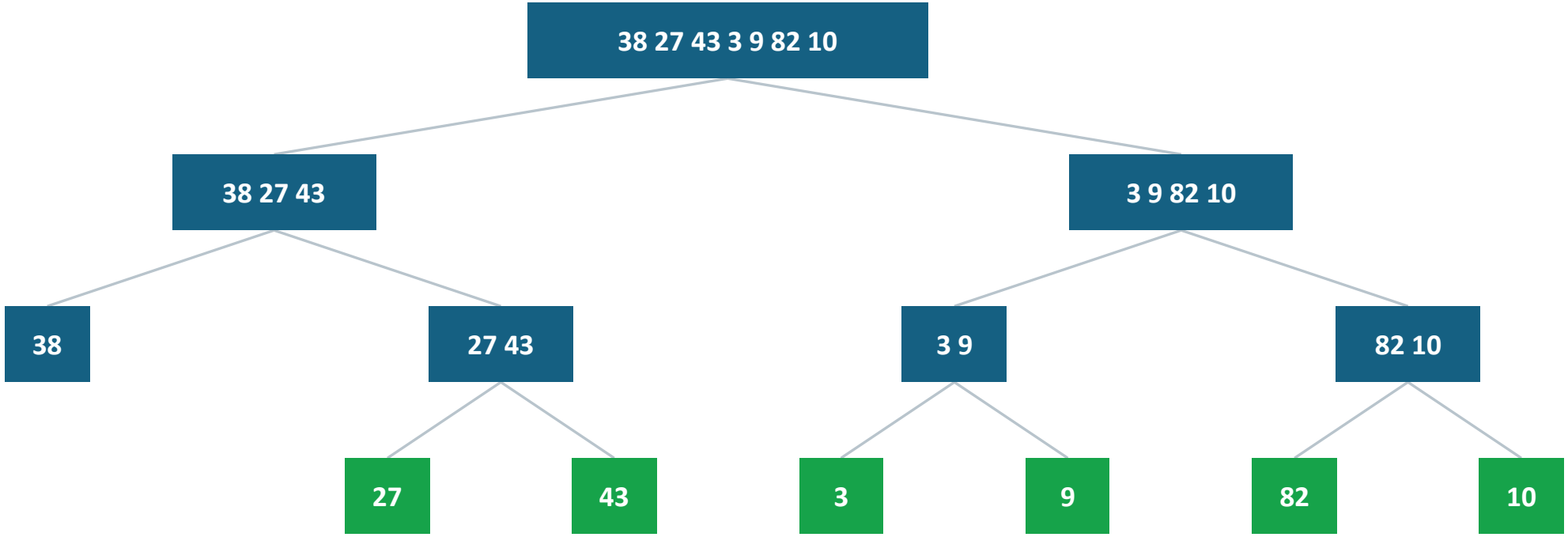
? **Think:** with < instead of ≤, who comes first among the 85s? Is the result still sorted?

Merge Sort

IIT Madras Zanzibar

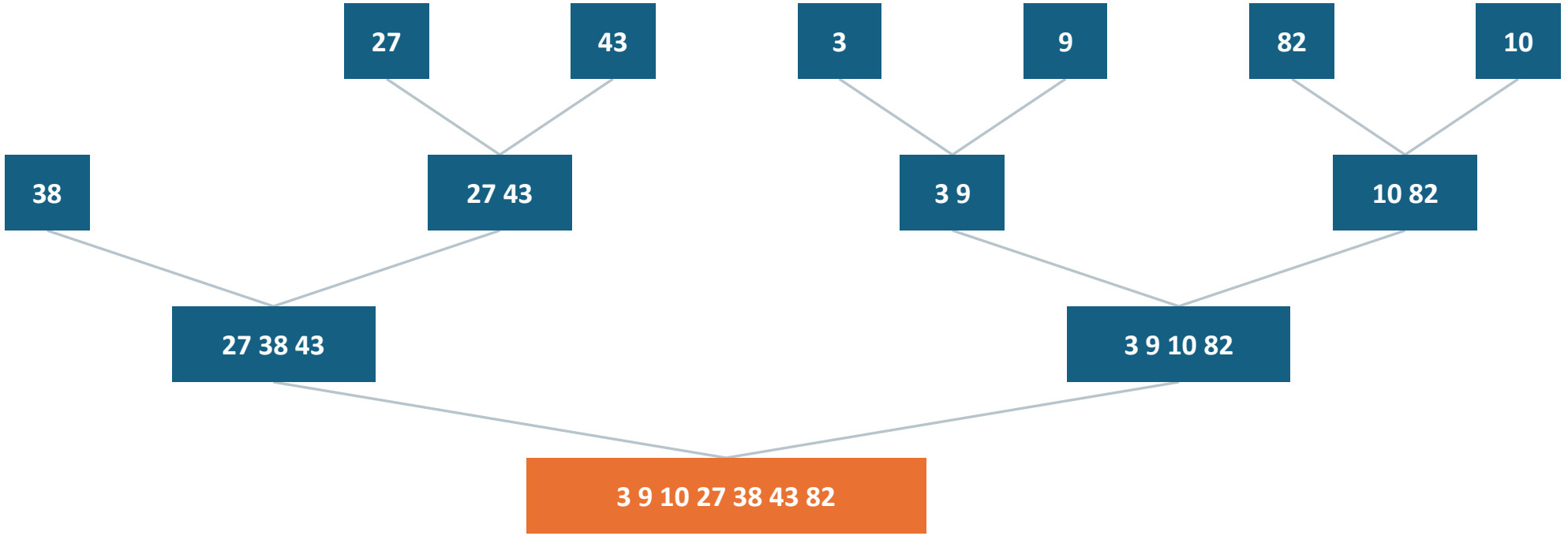
- Why We Need a Faster Sort
- The Merge Step
- → **Merge Sort**
- Its Cost: n log n
- Properties and Practice
- Demo and Practice

Going Down: the Split Tree



Think: how many levels of splitting for 8 items? For 1,024 items?

Coming Back Up: the Merge Tree



Think: at each level, how many items are merged in total?

The merge_sort Code, Line by Line

IIT Madras Zanzibar

```
1  def merge_sort(a):
2      if len(a) <= 1:
3          return list(a)
4      mid = len(a) // 2
5      left = merge_sort(a[:mid])
6      right = merge_sort(a[mid:])
7      return merge(left, right)
```

Line by line

Line 2–3 base case: 0 or 1 item is sorted

Line 4 the middle index

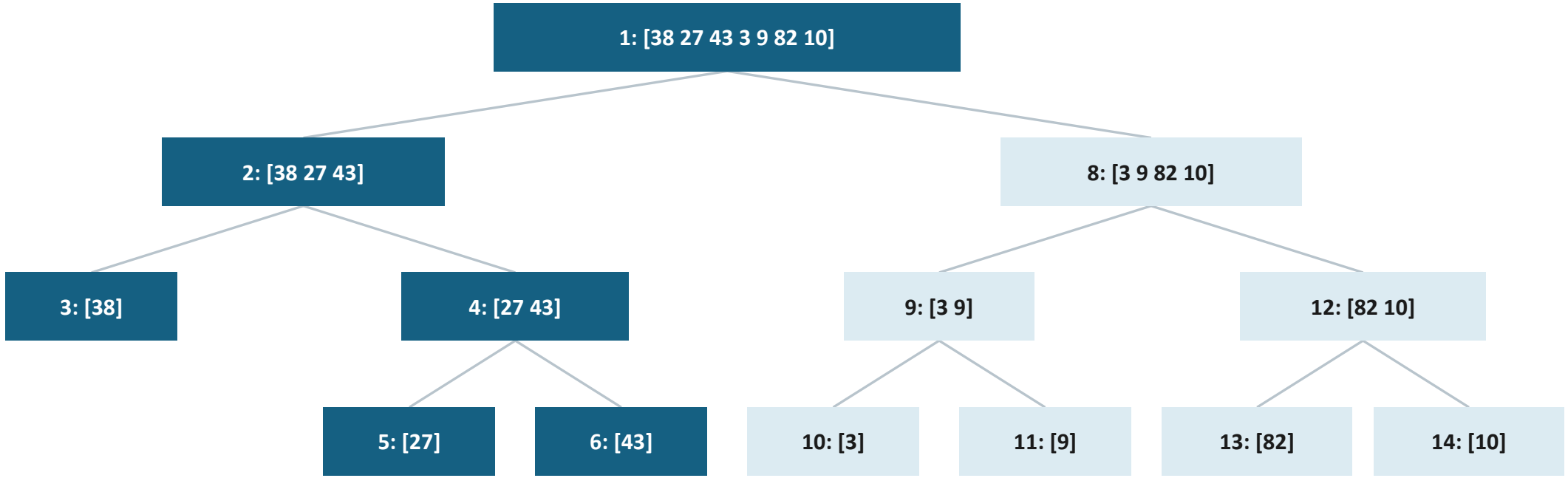
Line 5 sort the left half (trust it)

Line 6 sort the right half (trust it)

Line 7 combine: merge the two sorted halves

? **Think:** why len(a) <= 1 and not len(a) == 1?

In What Order Do the Calls Happen?



1. Left half completely first (calls 2–7), then the right half (calls 8–14). Stack depth ≤ about $\log_2 n$.

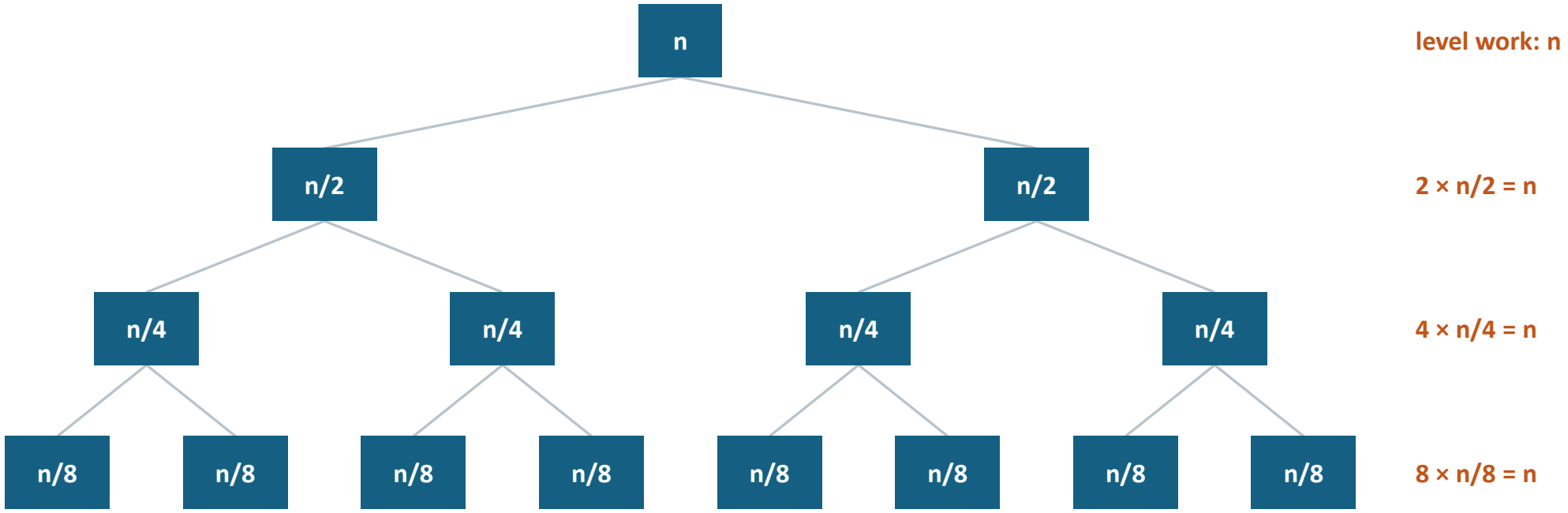
? **Think:** which merge happens first?

Merge Sort

IIT Madras Zanzibar

- Why We Need a Faster Sort
- The Merge Step
- Merge Sort
- → **Its Cost: n log n**
- Properties and Practice
- Demo and Practice

The Recursion Tree: n Work on Each of $\log_2 n$ Levels



… down to n leaves of size 1 …

Total: n per level × $\log_2 n$ levels = n $\log_2 n$ → O(n log n)

? **Think:** n = 1,024: how many levels? About how much total work?

Every Input Costs the Same

Input	Insertion sort	Merge sort
random	$O(n^2)$	O(n log n)
already sorted	O(n): best case	O(n log n)
reversed	$O(n^2)$: worst case	O(n log n)

1. Same tree for every input.
2. No bad input: O(n log n) guaranteed.
3. Insertion sort wins only on nearly sorted data.

? **Think:** what does merge sort do on an already sorted list?

Merge Sort

IIT Madras Zanzibar

Why We Need a Faster Sort

The Merge Step

Merge Sort

Its Cost: n log n

→ **Properties and Practice**

Demo and Practice

The Price: O(n) Extra Memory

IIT Madras Zanzibar

Sort	Best	Average	Worst	Extra memory	Stable?
Bubble	O(n)	$O(n^2)$	$O(n^2)$	O(1)	yes
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	no
Insertion	O(n)	$O(n^2)$	$O(n^2)$	O(1)	yes
Merge	O(n log n)	O(n log n)	O(n log n)	O(n)	yes

? **Think:** why can't merge simply write its output back into the input list as it goes?

Where Merge Sort Is Used

IIT Madras Zanzibar

1 **Python: sorted() and list.sort()**

Timsort: merge sort on natural runs, insertion sort for short runs; stable

2 **Java: sorting objects**

a merge-sort variant, because stability is required

3 **External sorting**

data larger than memory: sort chunks, write them to disk, merge the chunks

4 **Linked lists**

merging needs only the fronts: no random access required

? **Think:** why is merging well suited to data stored on disk?

Merge Sort

IIT Madras Zanzibar

Why We Need a Faster Sort

The Merge Step

Merge Sort

Its Cost: n log n

Properties and Practice

→ **Demo and Practice**

Live Demo

IIT Madras Zanzibar

```
cd ~/PDS/Class_Demos/Week_05
python3 W05_Wed_Nov04_merge_sort.py                # 38 27 43 3 9 82 10
python3 W05_Wed_Nov04_merge_sort.py merge 3 27 38 43 / 9 10 82
python3 W05_Wed_Nov04_merge_sort.py race
python3 W05_Wed_Nov04_merge_sort.py 5 2 4 7 1 3 2 6
```

What to watch

1. Indentation = depth; compare with the split and merge trees.
2. Race: insertion ×4 per doubling, merge sort just over ×2.

In-Class Exercise (6 minutes, in pairs)

IIT Madras Zanzibar

IN-CLASS TASK: draw first, then answer

1. Merge [2, 5, 9] and [1, 5, 6] step by step. Which 5 goes into the output first, and why?
2. Draw the split tree and the merge tree for [5, 2, 4, 7, 1, 3, 2, 6].
3. For n = 1,024: how many levels does the recursion tree have? About how many steps in total?
4. A friend's 'merge sort' splits off ONE item: sort a[1:], then merge [a[0]] with it. Recurrence? Cost?
5. Merge sort uses O(n) extra memory. Where is it used, and why can't the merge avoid it simply?

Solution

IIT Madras Zanzibar

1 **1: [1, 2, 5, 5, 6, 9]**

on the tie 5 = 5 the LEFT 5 is taken first (≤): the sort stays stable.

2 **2: split, then merge**

5 2 | 4 7 | 1 3 | 2 6 → 2 5, 4 7, 1 3, 2 6 → 2 4 5 7, 1 2 3 6 → 1 2 2 3 4 5 6 7.

3 **3: 10 levels**

1,024 merged per level × 10 levels ≈ 10,240 steps.

4 **4: $T(n) = T(n - 1) + n \rightarrow O(n^2)$**

no halving: n + (n − 1) + … + 1. It is insertion sort in disguise.

5 **5: O(n) memory**

the output list; merging into the input would overwrite uncompared left-half items.

Common Mistakes

IIT Madras Zanzibar

1 **< instead of ≤ in merge**

ties taken from the right: the sort is no longer stable

2 **Forgetting the leftovers**

after the loop, extend with L[i:] and R[j:]

3 **Base case len(a) == 1**

the empty list never stops: use len(a) <= 1

4 **Not halving**

splitting off one item gives $T(n) = T(n - 1) + n$: $O(n^2)$

Summary and Exam Checkpoints

IIT Madras Zanzibar

Merge

two fingers; take the smaller front; ties go left; O(n)

Merge sort

split in half, sort both halves recursively, merge

Cost

n work per level × $\log_2 n$ levels = O(n log n), for every input

Price

O(n) extra memory; stable; Python's sort is a merge sort

Next

Thu 5 Nov, 10:00: Quicksort and quickselect

Thu 5 Nov, 2:00: ICA 2 (2:00–2:45), then Lab 4: recursion, merge sort, quicksort

Fri 6 Nov: Recurrences and the Master Theorem

References and Further Reading

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2022). Introduction to Algorithms, 4th ed. MIT Press. Section 2.3 (merge sort and its analysis).

[2] Canning, J., Broder, A. & Lafore, R. (2022). Data Structures & Algorithms in Python. Addison-Wesley. Chapter 7 (Advanced Sorting: mergesort).

[3] Peters, T. (2002). listsort.txt: description of Timsort, in the CPython source repository (Objects/listsort.txt).

[4] Knuth, D. E. (1998). The Art of Computer Programming, Vol. 3: Sorting and Searching, 2nd ed. Addison-Wesley. Section 5.2.4 (merging).