Merging Two Sorted Lists in Python — Complete Illustrated Guide

A Light■Theme Illustrated PDF explaining multiple merge techniques, step■by■step traces, unequal list handling, and performance insights.

■ Why Merge Sorted Lists?

Merging sorted lists is a classic interview problem and a fundamental part of sorting algorithms like Merge Sort. When both input lists are sorted, we can merge them in linear time O(n + m).

■ Method 1 — Two■Pointer Merge

Two indices traverse each list, appending the smaller element first. When one list ends, append the rest.

```
def merge_sorted_lists(a, b):
    i = j = 0
    merged = []
    while i < len(a) and j < len(b):
        if a[i] < b[j]:
            merged.append(a[i])
            i += 1
        else:
            merged.append(b[j])
            j += 1
    merged.extend(a[i:])
    merged.extend(b[j:])
    return merged

a = [1, 3, 5, 7]
b = [2, 4, 6, 8]
print(merge_sorted_lists(a, b))
# [1, 2, 3, 4, 5, 6, 7, 8]
```

■ Visual Trace — Different Sized Lists

Step	a[i]	b[j]	Comparison	Appended	i	j	Merged
1	1	2	1 < 2	1	1	0	[1]
2	3	2	3 > 2	2	1	1	[1,2]
3	3	3	3 == 3	3	2	2	[1,2,3]
4	5	–	b done	extend a[i:]	→	→	[1,2,3,5,9,15,20]

```
# Unequal size example
a = [1, 4, 9, 15, 20]
b = [2, 3]
print(merge_sorted_lists(a, b))
# Output: [1, 2, 3, 4, 9, 15, 20]
```

■■ Method 2 — Using heapq.merge()

`heapq.merge()` merges multiple sorted iterables into a sorted stream (iterator) efficiently, ideal for big data or streaming merges.

```
import heapq
a = [1, 3, 5, 7]
b = [2, 4, 6, 8]
merged = list(heapq.merge(a, b))
print(merged)
# [1, 2, 3, 4, 5, 6, 7, 8]
```

■ Method 3 — Using sorted(a + b)

Concatenate both lists, then sort them again. Simpler syntax but O((n+m) log(n+m)) time complexity.

```
a = [1, 3, 5]
b = [2, 4, 6]
print(sorted(a + b))
# [1, 2, 3, 4, 5, 6]
```

■ Method 4 — Merge Multiple Lists

`heapq.merge()` can accept multiple lists and maintain overall sorted order efficiently.

```
import heapq

list1 = [1, 4, 7]
list2 = [2, 5, 8]
list3 = [0, 3, 6]
merged = list(heapq.merge(list1, list2, list3))
print(merged)
# [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

■ Summary Comparison

Method	Time	Space	Best Use Case
Two■Pointer	O(n+m)	O(n+m)	Classic array merge, interviews.
heapq.merge()	O(n+m)	O(1) iterator	Large sorted data or file streams.
sorted(a+b)	O((n+m)log(n+m))	O(n+m)	Quick one-liners, small data.
heapq.merge(*lists)	O(total elements)	O(1)	Merging multiple lists.

■ Edge Cases to Test

```
print(merge_sorted_lists([], []))             # []
print(merge_sorted_lists([1], [2,3,4]))       # [1,2,3,4]
print(merge_sorted_lists([10,20,30], [5,15])) # [5,10,15,20,30]
```