



tracing insertion sort and binary search, step by step, by hand

From class 11: two traces you already ran

- **linear search** checks each element in turn from the start, and stops the moment it finds a match or runs out of elements to check
- **bubble sort** repeatedly walks the array comparing neighbours, swapping any pair that is out of order, so the largest unsorted value bubbles to the end each pass
- this chapter adds two more traces on top of those: **insertion sort** and **binary search**, both worked by hand exactly the same way
- how FAST each one runs as the array grows is a separate question, answered by the Complexity and Big O chapter, this chapter is only about tracing the steps correctly

Insertion sort, the idea

- the array is treated as a **sorted part on the left** and an **unsorted part on the right**, one element wide to start
- each pass takes the next unsorted element, called the **key**, and slides it left past every already sorted value that is bigger than it
- sliding stops the moment a smaller value is reached, or the start of the array is reached, and the key is dropped into that gap
- after n minus 1 passes the whole array is the sorted part, which is why it needs no extra array, the shifting happens in place

Insertion sort on 5, 2, 4, 6, 1, 3

Q. Trace insertion sort on the array 5, 2, 4, 6, 1, 3, showing the array after each key is placed.

- 1 key 2 (index 1): 2 is less than 5, so 5 slides right and 2 drops into index 0, array becomes 2, 5, 4, 6, 1, 3
- 2 key 4 (index 2): 4 is less than 5 so 5 slides right, 4 is not less than 2 so sliding stops, 4 drops into index 1, array becomes 2, 4, 5, 6, 1, 3
- 3 key 6 (index 3): 6 is not less than 5, sliding never starts, array stays 2, 4, 5, 6, 1, 3
- 4 key 1 (index 4): 1 is less than 6, 5, 4 and 2 in turn, all four slide right, 1 drops into index 0, array becomes 1, 2, 4, 5, 6, 3
- 5 key 3 (index 5): 3 is less than 6, 5 and 4 in turn, those three slide right, 3 is not less than 2 so sliding stops, 3 drops into index 2, array becomes 1, 2, 3, 4, 5, 6

sorted 1 2 3 4 5 6

the array once every key has been placed, fully sorted after five passes

Answer the sorted array is 1, 2, 3, 4, 5, 6

A first look at divide and conquer *

Split

an unsorted array is cut into two halves, each handled the same way

Conquer

each half is sorted on its own, recursively, until a half is down to one element and is trivially sorted

Combine

6 unsorted values



the two sorted halves are merged back together into one sorted array, comparing their fronts one pair at a time

Not traced here

merge sort is only introduced as an idea in this chapter, insertion sort and binary search are the two traces expected in full

