Data structures in C++ STL library

Array

contiguous memory block
fixed size

Vector

* Contiguous memory block
* Dynamic array
  ↳ Starts with initial capacity (just enough)
  ↓
  when vector exceeds capacity: reallocates memory
  ↓
  (Doubles the size)

  (But can reserve memory initially if needed to avoid reallocations)

* Delete or insert at the end (or otherwise elements should be shifted)

Deque

* Non-contiguous (Multiple contiguous fixed-size blocks)
  ↓
  Internally maintains an array of pointers to these blocks of memory
  (Control Block / block pointer array)
  ↓
  might occasionally require resizing
* Accessing involves looking up the block-pointer-array and finding the appropriate block (slightly slower compared to direct memory access like in vector)
* Can insert/remove at both front & end.

- Random-Access:

  Address = Base Address + (Index x Element size)

List

* Doubly-linked list: (Each element contains a value and two pointers)
  - ∴ Non-contiguous memory
  - ∴ No random access (Need to traverse from start or end)
  - → to previous & next elements

* Best for inserting/deleting in the middle

Set / map

→ No duplicates

* implemented as balanced binary search trees
  - Tree needs to be balanced after insertion /deletion (still faster than lists because traversing is quick)
  - Searching is fast (Already sorted)
  - No random access (But faster than 'list')

Unordered_set / Unordered_map

* Implemented as hash-table

  ↓

  ∴ No-random access

* Faster than sets → can be slower than sets when handling collisions etc.
* Uses more memory than sets (For hash table)