Cheatsheet

Lists

Implementation	Append	Concat	Prepend	Random Access	Insert/remove After Ptr	Insert/remove Before Ptr
`std::vector` (dyn array)	O(1)	O(n)	O(n)	O(1)	O(n)	O(n)
`std::list` (dbl-LL)	O(1)	O(1)	O(1)	O(n)	O(1)	O(1)
`std::forward_list` (LL)	O(n)	O(1)	O(1)	O(n)	O(1)	O(1)
`std::deque` (ring buffer)	O(1)	O(n)	O(1)	O(1)	O(n)	O(n)

Considerations

- **Memory Overhead**
  - `std::vector` has the smallest memory overhead since the only accounting required is the capacity and the length of the vector.
  - `std::list` has the most memory overhead since two pointers must be stored alongside each piece of data.
  - `std::deque` has only slightly larger memory overhead to `std::vector`, but the computations required to calculate an index are more involved.
- **Locality**
  - `std::vector` and `std::deque` have the best cache locality since they store the data contiguously.
  - `std::list` and `std::forward_list` have the worst cache locality since pointer traversals are required for every element.

Dictionaries (Maps) and Sets

Implementation	Get	Set	Remove	Contains
`std::map` (tree*)	O(log n)	O(log n)	O(log n)	O(log n)
`std::unordered_map` (hashmap)**	O(1)	O(1)	O(1)	O(1)

* normally a Red-black tree

** complexities for `std::unordered_map` are on average, and assume a sufficiently random hash function

Considerations

- Worst-case complexities for all operations on hashmaps is O(n)
- **Memory Overhead**: `std::map` requires more accounting due to needing to maintain left/right pointers from each tree node.

- All of the dictionary data structures have corresponding set data structures with equivalent properties.

Sorting

Function	Complexity	Stable	Implementation
C++ `std::sort`*	O(n log n)	No	Likely Introsort.
C++ `std::stable_sort`*	O(n (log n)^2), if additional memory available, then O(n log n)	Yes	Likely mergesort.
C++ `std::qsort`*	No guarantee	No guarantee	Likely quicksort
Python `list.sort` and `sorted`	O(n log n)	Yes	Timsort
Rust `Vec::sort`	O(n log n)	Yes	A variant of Timsort
Rust `Vec::sort`	O(n log n)	No	A pattern-defeating quicksort

* The C++ specification dos not require a specific sorting algorithm to be used to implement the sort functions.