CPP STL DSA Guide

need complete sheet on cpp stl, its various data structures - their nature, their functions needed for dsa

Absolutely! Let's make a **complete C++ STL Sheet** that covers:

1. The **main data structures** in STL
2. Their **nature** (ordered/unordered, underlying structure)
3. Commonly used **functions and operations** relevant for **Data Structures & Algorithms (DSA)**

Let's go step by step.

📚 C++ STL Sheet for DSA

🧱 1. Containers: Nature + Functions

Container	Nature	Underlying Structure	Key Functions for DSA
Vector	Dynamic array, ordered	Array	`push_back()`, `pop_back()`, `size()`, `clear()`, `begin()`, `end()`, `insert()`, `erase()`, `sort()`, `reverse()`, `find()` (via `<algorithm>`), `lower_bound()`, `upper_bound()`
Deque	Double-ended queue	Array of blocks	`push_back()`, `push_front()`, `pop_back()`, `pop_front()`, `at()`, `size()`, `front()`, `back()`
List	Doubly linked list	Linked List	`push_back()`, `push_front()`, `pop_back()`, `pop_front()`, `insert()`, `erase()`, `sort()`, `reverse()`
Stack	LIFO	Adapter over `deque`	`push()`, `pop()`, `top()`, `size()`, `empty()`
Queue	FIFO	Adapter over `deque`	`push()`, `pop()`, `front()`, `back()`, `size()`, `empty()`
Priority Queue	Max-heap (default)	Adapter over `vector`	`push()`, `pop()`, `top()`, `size()`, `empty()` For min-heap: `priority_queue<type, vector<type>, greater<type>>`
Set	Ordered, unique elements	Balanced BST (Red-Black Tree)	`insert()`, `erase()`, `find()`, `count()`, `lower_bound()`, `upper_bound()`, `begin()`, `end()`
Multiset	Ordered, allows duplicates	Balanced BST	Same as `set`, but allows duplicates
Unordered Set	Unordered, unique	Hash Table	`insert()`, `erase()`, `find()`, `count()`

Container	Nature	Underlying Structure	Key Functions for DSA
Map	Ordered key-value pairs, unique keys	Balanced BST	`insert()`, `erase()`, `find()`, `count()`, `[]`, `at()`, `lower_bound()`, `upper_bound()`
Multimap	Ordered key-value pairs, allows duplicates	Balanced BST	Same as `map`, allows duplicate keys
Unordered Map	Unordered key-value pairs	Hash Table	`insert()`, `erase()`, `find()`, `[]`, `at()`

🧮 2. Algorithms

Header: `<algorithm>`

Algorithm	Usage
`sort(start, end)`	Sort in ascending order (default `<`)
`sort(start, end, comp)`	Sort using custom comparator
`reverse(start, end)`	Reverse elements
`min_element(start, end)`	Iterator to min element
`max_element(start, end)`	Iterator to max element
`accumulate(start, end, init)`	Sum of elements (needs `<numeric>`)
`count(start, end, val)`	Count occurrences of `val`
`find(start, end, val)`	Iterator to `val` or `end` if not found
`binary_search(start, end, val)`	Returns bool: found or not
`lower_bound(start, end, val)`	First element >= `val` (binary search)
`upper_bound(start, end, val)`	First element > `val` (binary search)
`next_permutation(start, end)`	Next lexicographic permutation
`prev_permutation(start, end)`	Previous lexicographic permutation
`rotate(start, mid, end)`	Rotates elements (useful for array rotations)
`unique(start, end)`	Removes consecutive duplicates (need `erase` for final cleanup)

📊 3. Iterators

Function	Usage
`begin()`	Iterator to first element
`end()`	Iterator to one past the last element
`rbegin()`	Reverse iterator to last element
`rend()`	Reverse iterator to one before first element
`cbegin()`, `cend()`	Constant iterators (read-only)
`auto it = container.begin()`	Type inference for iterators
`for(auto it: container)`	Range-based loops

🏹 4. Pairs

Function	Usage
`` `pair<type1, type2> p` ``	Create a pair
`` `make_pair(a, b)` ``	Create a pair without specifying types
`` `p.first`, `p.second` ``	Access elements

🔥 5. Custom Comparators for Sorting

```cpp
// Ascending order by second element, if tie then first element
bool cmp(pair<int, int>& a, pair<int, int>& b) {
    if (a.second == b.second)
        return a.first < b.first;
    return a.second < b.second;
}

// Usage
sort(v.begin(), v.end(), cmp);
```

⚡ 6. Time Complexities (Approximate)

Container	Insert	Search	Delete
`` `vector` ``	Amortized O(1)	O(N)	O(N)
`` `deque` ``	Amortized O(1)	O(N)	O(N)
`` `list` ``	O(1) (with iterator)	O(N)	O(1) (with iterator)
`` `stack` ``	O(1)	O(1)	O(1)
`` `queue` ``	O(1)	O(1)	O(1)
`` `priority_queue` ``	O(logN)	O(1) (top)	O(logN)
`` `set/map` ``	O(logN)	O(logN)	O(logN)
`` `multiset/multimap` ``	O(logN)	O(logN)	O(logN)
`` `unordered_set/unordered_map` ``	O(1) average, O(N) worst	O(1) avg	O(1) avg

Would you like this in a **PDF** or **Markdown** format too? Let me know if you want a **DSA-focused cheat sheet** with problem patterns too! 🚀