C++ STL Problem Solving Full Cheat Sheet

1. vector

```
Use cases: dynamic arrays, prefix sums, storing graph adjacency lists.
Example:
vector<int> v = {1,2,3};
v.push_back(5);
reverse(v.begin(), v.end());
```

2. pair and tuple

```
Use cases: returning multiple values, sorting by multiple keys.
Example:
pair<int,int> p = {3,4};
tuple<int,int,int> t = {1,2,3};
auto [a,b] = p; // structured binding
```

3. set / multiset

```
Use cases: unique elements, ordered queries, duplicates allowed in multiset.
Example:
set<int> s = {1,2,3};
s.insert(5);
auto it = s.lower_bound(3); // first >= 3
```

4. unordered_set / unordered_map

```
Use cases: hashing, frequency count, fast membership check.
Example:
unordered_map<int,int> freq;
for(int x: v) freq[x]++;
```

5. map

```
Use cases: ordered dictionary, range queries.
Example:
map<int,string> mp;
mp[1] = "A"; mp[2] = "B";
for(auto [k,v]: mp) cout << k << " " << v << "\n";
```

6. deque

```
Use cases: sliding window maximum/minimum, BFS (0-1 weights).
Example:
deque<int> dq;
dq.push_front(1);
dq.push_back(2);
dq.pop_front();
```

7. priority_queue

```
Use cases: heaps, Dijkstra, greedy selection.
Example:
// max heap
priority_queue<int> maxh;
// min heap
priority_queue<int, vector<int>, greater<int>> minh;
```

8. stack

```
Use cases: parentheses matching, monotonic stack, DFS.
Example:
stack<int> st;
st.push(1);
st.top();
st.pop();
```

9. queue

```
Use cases: BFS, simulation problems.
Example:
queue<int> q;
q.push(1);
q.front();
q.pop();
```

10. bitset

```
Use cases: fast bit manipulation, subset DP, sieve.
Example:
bitset<1000> b;
b.set(3);
b.reset(3);
b.flip(2);
cout << b.count();
```

11. DSU (Disjoint Set Union)

```
Use cases: union-find, connected components, Kruskal’s MST.
Example:
vector<int> parent(n), sz(n,1);
iota(parent.begin(), parent.end(), 0);
function<int(int)> find = [&](int x){
    return parent[x]==x ? x : parent[x]=find(parent[x]);
};
auto unite = [&](int a, int b){
    a=find(a), b=find(b);
    if(a!=b){
        if(sz[a]<sz[b]) swap(a,b);
        parent[b]=a;
        sz[a]+=sz[b];
    }
};
```

12. Other Handy Containers

```
array<int,N> → fixed size, faster than vector.
string → easy manipulation, supports iterators.
multimap → allows duplicate keys.
```

Common STL Tricks

```
1. Fast I/O:
ios::sync_with_stdio(false);
cin.tie(0);

2. Vector Shortcuts:
sort(v.begin(), v.end());
sort(v.rbegin(), v.rend()); // descending
auto it = lower_bound(v.begin(), v.end(), 3);
```

```
3. Priority Queue (Min/Max Heap):
priority_queue<int> maxh;
priority_queue<int, vector<int>, greater<int>> minh;

4. Maps & Sets:
set<int> s;
s.insert(5);
s.lower_bound(3);

5. String Tricks:
string s = "abc";
reverse(s.begin(), s.end()); // cba
stoi("123"); // 123

6. Algorithms:
int sum = accumulate(v.begin(), v.end(), 0);
int mx = *max_element(v.begin(), v.end());
int cnt = count(v.begin(), v.end(), 3);

7. Custom Sort:
sort(v.begin(), v.end(), [](auto &a, auto &b){
    return a.second > b.second;
});

8. Bit Tricks:
__builtin_popcount(x); // number of set bits
__builtin_clz(x);      // leading zeros
__builtin_ctz(x);      // trailing zeros

9. Debug Helper:
#define debug(x) cerr << #x << " = " << x << "\n";
```

Quick Problem Mapping

```
Sliding window → deque
Shortest path (unweighted) → queue
Shortest path (weighted) → priority_queue
Unique elements → set
Frequency counting → unordered_map
Subset / bitmask DP → bitset
Connected components / MST → DSU
Greedy selection → priority_queue
Parentheses / Next greater element → stack
```