1. Stack

Definition: Last In, First Out (LIFO) container.

- **Declaration**:

```
#include <stack>
stack<int> s;  // Stack of integers
```

- **Insert Element**: `s.push(x)` (Adds to the top).
- **Remove Element**: `s.pop()` (Removes the top element).
- **Access Top Element**: `s.top()` (Returns the top element).
- **Size Check**: `s.size()`.
- **Empty Check**: `s.empty()`.
- **Time Complexity**:
  - Push: O(1).
  - Pop: O(1).
  - Top: O(1).

---

2. Queue

Definition: First In, First Out (FIFO) container.

- **Declaration**:

```
#include <queue>
queue<int> q;  // Queue of integers
```

- **Insert Element**: `q.push(x)` (Adds element to the back).
- **Remove Element**: `q.pop()` (Removes the front element).
- **Access Front Element**: `q.front()`.
- **Access Back Element**: `q.back()`.
- **Size Check**: `q.size()`.
- **Empty Check**: `q.empty()`.
- **Time Complexity**:
  - Push: O(1).
  - Pop: O(1).
  - Front/Back Access: O(1).

---

3. Priority Queue

Definition: A queue where elements are arranged based on priority (default: max-heap).

- **Declaration**:

```
#include <queue>
priority_queue<int> pq;  // Max-heap priority queue of integers
priority_queue<int, vector<int>, greater<int>> pq_min;  // Min-heap
```

- **Insert Element**: `pq.push(x)`.
- **Remove Top Priority**: `pq.pop()`.
- **Access Top Element**: `pq.top()`.
- **Size Check**: `pq.size()`.
- **Empty Check**: `pq.empty()`.
- **Time Complexity**:
  - Push: O(log N).
  - Pop: O(log N).
  - Top Access: O(1).

---

4. Set

Definition: A collection of unique, ordered elements.

- **Declaration**:

```
#include <set>
set<int> s;  // Set of integers
```

- **Insert Element**: `s.insert(x)`.
- **Find Element**: `s.find(x)` (Returns iterator to element or `s.end()` if not found).
- **Erase Element**: `s.erase(x)` or iterator.
- **Access Min/Max**: `*s.begin()` (Min), `*s.rbegin()` (Max).
- **Size Check**: `s.size()`.
- **Empty Check**: `s.empty()`.
- **Time Complexity**:
  - Insert: O(log N).
  - Erase: O(log N).
  - Find: O(log N).

---

5. Unordered Set

Definition: A collection of unique elements with no particular order (uses hash table).

- **Declaration**:

```
#include <unordered_set>
unordered_set<int> us;  // Unordered set of integers
```

- **Insert Element**: `us.insert(x)`.
- **Find Element**: `us.find(x)`.
- **Erase Element**: `us.erase(x)` or iterator.
- **Size Check**: `us.size()`.
- **Empty Check**: `us.empty()`.
- **Time Complexity**:
  - Insert: O(1) (Average).
  - Erase: O(1) (Average).
  - Find: O(1) (Average).

---

6. Map

Definition: A collection of key-value pairs, ordered by keys.

- **Declaration**:

  ```
  #include <map>
  map<int, int> m;  // Map with integer keys and values
  ```

- **Insert Element**: `m[key] = value` or `m.insert({key, value})`.
- **Access Element**: `m[key]`.
- **Erase Element**: `m.erase(key)` or iterator.
- **Find Element**: `m.find(key)`.
- **Size Check**: `m.size()`.
- **Empty Check**: `m.empty()`.
- **Time Complexity**:
  - Insert: O(log N).
  - Erase: O(log N).
  - Access: O(log N).

---

7. Unordered Map

Definition: A collection of key-value pairs, stored with no specific order (uses hash table).

- **Declaration**:

  ```
  #include <unordered_map>
  unordered_map<int, int> um;  // Unordered map with integer keys and values
  ```

- **Insert Element**: `um[key] = value` or `um.insert({key, value})`.
- **Access Element**: `um[key]`.
- **Erase Element**: `um.erase(key)`.
- **Find Element**: `um.find(key)`.
- **Size Check**: `um.size()`.

- **Empty Check**: `um.empty()`.
- **Time Complexity**:
  - Insert: O(1) (Average).
  - Erase: O(1) (Average).
  - Access: O(1) (Average).

---

8. List

Definition: A doubly-linked list.

- **Declaration**:

  ```
  #include <list>
  list<int> l;  // Doubly-linked list of integers
  ```

- **Insert Element**:
  - `l.push_front(x)` (Adds to the front).
  - `l.push_back(x)` (Adds to the back).
  - `l.insert(iterator, x)` (Insert at specific position).
- **Remove Element**:
  - `l.pop_front()` (Removes front).
  - `l.pop_back()` (Removes back).
  - `l.erase(iterator)` (Erase specific element).
- **Access Elements**: Sequential access only (no random access).
- **Size Check**: `l.size()`.
- **Empty Check**: `l.empty()`.
- **Time Complexity**:
  - Push/Pop Front/Back: O(1).
  - Insert/Erase: O(N) (Worst case due to traversal).

This report covers all key STL components with only necessary details and time complexities.