Hashing	--> Unordered map
Heaps	--> ordered maps(map)
Priority queue	--> implemented using heap

unordered_map vs unordered_set :

In unordered_set, we have only key, no value, these are mainly used to see presence/absence in a set. For example, consider the problem of counting frequencies of individual words. We can't use unordered_set (or set) as we can't store counts.

unordered_map vs map :

map (like set) is an ordered sequence of unique keys whereas in unordered_map key can be stored in any order, so unordered.

Map is implemented as balanced tree structure that is why it is possible to maintain an order between the elements (by specific tree traversal). Time complexity of map operations is O(Log n) while for unordered_map, it is O(1) on average.

From <https://www.geeksforgeeks.org/unordered_map-in-cpp-stl/>

Difference :

	map	unordered_map
Ordering	increasing order (by default)	no ordering
Implementation	Self balancing BST like Red-Black Tree	Hash Table
search time	log(n)	O(1) -> Average O(n) -> Worst Case
Insertion time	log(n) + Rebalance	Same as search
Deletion time	log(n) + Rebalance	Same as search

Use std::map when

- You need ordered data.
- You would have to print/access the data (in sorted order).
- You need predecessor/successor of elements.
- See advantages of BST over Hash Table for more cases.

Use std::unordered_map when

- You need to keep count of some data (Example – strings) and no ordering is required.
- You need single element access i.e. no traversal.

Hashing:

//Can excess 2nd argument:

```
unordered_map<char,int> map;
for(auto it: map){
      if(it.second%2==1){
         odd++;
      }
   }

 for(auto s: map){
    ans.push_back(s.second);
 }
```

Important:

For erasing in Unordered list, erase using particular keys cause for the range we don't know which key is at what iterator

Heap-Sort:

Depth of a tree: floor(log2(n+1))

- To construct Heap(**Heapify: fn**. Used to maintain the heap property) **O(n)**:
  a. Shift up**: O(nlogn)**
  b. Shift down: **O(n)**
- **Then for sorting**: simply remove the root node **O(nlogn)**

Heap Data Structure is generally taught with Heapsort. Heapsort algorithm has limited uses because Quicksort is better in practice. Nevertheless, the Heap data structure itself is enormously used. Following are some uses other than Heapsort (Priority Queue).

Priority Queue:

Priority Queues: Priority queues can be efficiently implemented using Binary Heap because it supports insert(), delete() and extractmax(), decreaseKey() operations in O(logn) time. Binomial Heap and Fibonacci Heap are variations of Binary Heap. These variations perform union also in O(logn) time which is a O(n) operation in Binary Heap. Heap implemented priority queues are used in Graph algorithms like Prim's Algorithm and Dijkstra's algorithm.

Order statistics: The Heap data structure can be used to efficiently find the kth smallest (or largest) element in an array. See method 4 and 6 of this post for details.

Priority queues are a type of container adapters, specifically designed such that the first element of the queue is the greatest of all elements in the queue and elements are in non increasing order (hence we can see that each element of the queue has a priority {fixed order}).

Queue can have duplicates:

Max-heap for priority queue:

```
priority_queue<int> pq
```

Min-heap for priority queue:

```
priority_queue <int, vector<int>, greater<int>> g = gq;
```

Methods of priority queue are:

- priority_queue::empty() in C++ STL– **empty()** function returns whether the queue is empty.
- priority_queue::size() in C++ STL– **size()** function returns the size of the queue.
- priority_queue::top() in C++ STL– Returns a reference to the top most element of the queue
- priority_queue::push() in C++ STL– **push(g)** function adds the element 'g' at the end of the queue.
- priority_queue::pop() in C++ STL– **pop()** function deletes the first element of the queue.
- priority_queue::swap() in C++ STL– This function is used to swap the contents of one priority queue with another priority queue of same type and size.
- priority_queue::emplace() in C++ STL – This function is used to insert a new element into the priority queue container, the new element is added to the top of the priority queue.
- priority_queue value_type in C++ STL– Represents the type of object stored as an element in a priority_queue. It acts as a synonym for the template parameter.

Different ways to add element in priority queue:

1. Pq.push(); --> O(logn)
2. Passing data:
   priority_queue pq(A.begin(),A.end());

Map:

Auto iterator:
https://www.geeksforgeeks.org/traversing-a-map-or-unordered_map-in-cpp-stl/