Algorithms & Data Structures Library

Array Libraries

- Know the syntax for allocating and instantiating an array or a vector, i.e., array<int, 3> A = {1, 2, 3}, vector<int> A = {1, 2, 3}. To construct a subarray from an array, you can use vector<int> subarray_A(A.begin() + i, A.begin() + j) – this sets subarray_A to be A[i:j-1].
- Understand how to instantiate a 2D array – array<array<int, 2>, 3> A = {{1,2}, {3,4}, {5,6}}, and vector<vector<int>> A = {{1,2}, {3,4}, {5,6}} both create an array which will hold 3 rows where each column holds 2 elements.
- Since vector is dynamically sizable, push_back(42) (or emplace_back(42)) are frequently used to add values to the end.
- Understand what "deep" means when checking equality of arrays, and hashing them.
- Key methods in algorithms include: binary_search(A.begin(), A.end(), 42), lower_bound(A.begin(), A.end(), 42), upper_bound(A.begin(), A.end(), 42), fill(A.begin(), A.end(), 42), swap(x, y), min_element(A.begin(), A.end()), max_element(A.begin(), A.end()), reverse(A.begin(), A.end()), rotate(A.begin(), A.begin() + shift, A.end()), and sort(A.begin(), A.end()).
- Understand the variants of these methods, e.g., how to create and copy of a subarray.

String Libraries

- The basic methods are append("James"), push_back('a'), pop_back(), insert(s.begin() + shift, "James"), substr(pos, len), and compare("James").
- Remember a string is organized like an array. It performs well for operations from the back, e.g., push_back('a') and pop_back(), but poorly in the middle of a string, e.g., insert(A.begin() + middle, "James").
- The comparison operators <, <=, >, >=, and == can be applied to strings, with == testing logical equality, rather than pointer equality.

Linked List Libraries

For doubly-linked lists (list), here are the functions to know:

- The functions to insert and delete elements in list are push_front(42), pop_front(), push_back(), and pop_back().

- The splice(L1.end(), L2), reverse(), and sort() functions are analogous to those on forward_list.

For singly-linked lists (forward_list), here are the functions to know:

- The functions to insert and delete elements in list are push_front(42), pop_front(), insert_after(L.end(), 42), and erase_after(A.begin()).
- To transfer elements from list to another used splice_after(L1.end(), L2).
- Reverse the order of the elements with reverse().
- Use sort() to sort lists, and save yourself a great deal of pain.

Stack Libraries

The key functions in the stack class are top(), push (42), and pop(). When called from an empty stack, top() and pop() throw exceptions.

- Push(e) pushes an element onto the stack. Not much can go wrong with a call to push.
- Top() will retrieve, but does not remove, the element at the top of the stack.
- Pop() will remove and the element at the top of the stack but does not return. To avoid the exception, first test with empty().
- Empty() tests if the stack is empty.

Queue Libraries

The key functions in the queue class are front(), back(), push(42), and pop(). When called on an empty queue, front(), back(), and pop() throw exceptions.

- Push(e) pushes an element onto the queue. Not much can go wrong with a call to push.
- Front() will retrieve, but does not remove, the element at the front of the queue. Similarly, back() will retrieve, but also does not remove, the element at the back of the queue.
- Pop() will remove the element at the top of the queue but does not return.

Heap Libraries

The implementation of heaps in C++ is referred as a priority queue; the class is priority_queue. The key functions are push("James"), top(), and pop(). Calling top() and pop() on an empty stack throws an exception. It is possible to specify a custom comparator in the heap constructor.

Searching Libraries

Searching is a very broad concept, and it is present in many data structures. For example, find(A.begin(), A.end(), target) in algorithm header finds the first element in a STL container. Here we focus on binary search in a sorted STL container:

- To check a targeted value is presented, use binary_search (A.begin(), A.end(), target). Note that it returns a boolean about the status of existence instead of the location.
- To find the 1st element that is not less than a targeted value, use lower_bound(A.begin(), A.end(), target). In other words, it finds the first element that is greater than or equal to the targeted value.
- To find the 1st element that is greater than a targeted value, use upper_bound(A.begin(), A.end(), target).

All 3 functions above have a time complexity of O(n log n) in a sorted STL container containing n elements. In an interview, if it is allowed, use the above functions instead of implementing your own binary search.

Hash Table Libraries

There are 2 hash table-based data structure commonly used in C++ - unordered_set and unordered_map. The difference between the 2 is that the latter stores key-value pairs, whereas the former simply stores keys. Both have the property that they do not allow for duplicate keys, unlike, for example, list and priority_queue.

The most important functions for unordered_set are insert(42), erase(42), find(42), and size().

- Insert(val) inserts new element and returns a pair of iterator and Boolean where the iterator points to the newly inserted element or the element whose key is equivalent, and the Boolean indicating if the element was added successfully, i.e., was not already present.
- Find(k) returns the iterator to the element if it was present; otherwise, a special iterator end() is returned.
- The order in which keys are traversed by the iterator returned by begin() is unspecified; it may even change with time.

The most important methods for unordered_map are insert({42, "James"}), erase(42), find(42), and size(). Those functions are analogous to the ones in unordered_set. The pair<key, value> type is a key-value pair that's useful when iterating over the map. The iteration order is not fixed, though iterations over the entry set, the key set, and the value set do agree.

Sorting Libraries

To sort an array, use sort() in the algorithm header, and to sort a list use the member function list::sort().

- The time complexity of sorting is O(n log n), where n is the length of the array. The standard gives no guarantees as to the space complexity. In practice, it's most commonly a variant of quicksort, which does not allocate additional memory, but uses O(log n) space on the function call stack.
- Both sort() in algorithm and list::sort() operate on arrays and lists of objects that implement operator<().
- Both sort() in algorithm and list::sort() have the provision of sorting according to an explicit comparator object.

Binary Search Tree Libraries

There are 2 BST-based data structures commonly used in C++ - set and map. Set stores keys, and map stores key-value pairs. Below I describe the functionalities added by set that go beyond what's in unordered_set. The functionalities added by map are similar.

- The iterator returned by begin() traverses keys in ascending order (To iterate over keys in descending order, use rbegin()).
- *begin() / *rbegin() yield the smallest and largest keys in the BST.
- Lower_bound(12) / upper_bound(3) return the first element that is greater than or equal to/greter than the argument.
- Equal_range(10) return the range of values equal to the argument.