* stacks - LIFO: like a stack of dishes
  + functions: push, pop (does not return anything, just removes top item), top, empty, size
* queues - FIFO: like a line at a theme park
  + functions: push, pop, front/back (read item w/o taking off the queue), empty, size
  + deques: can add and remove from anywhere in the deque w/ `.insert(int position)` (better than vectors at removing from front and back)
    - implemented using multiple arrays that can be scattered throughout memory
* priority queues - sorts elements as they are pushed on (default uses <, or you can define custom)
  + functions: push, pop, top (see top item), empty, size
* (linked) lists: similar to vectors & deques. better at add/remove from middle positions (ideal for sorting).
  + pro: fast adding and removing to any position. con: the index of an element does not give you quick access to it (have to iterate through until found)
  + made up of data structures called nodes that are linked together with pointers
  + push\_front, push\_back, pop\_front, pop\_back, front, back, insert(int position), erase, empty, size, sort
  + very little time to insert a value into the beginning. make a new node with new value, set the new node's pointer to point to the first node, then set the list's head pointer to point to the new node. removing is similar.
  + doubly-linked list can be traveresed in two directions head and tail
* sets: all items unique, no particular position, not index based; don't care where an item is, just whether it's in the set (finding things quickly is what sets are best at)
  + notation: 2 ∈ A (2 is in A), A ∪ B (A union B -- combined items from both sets), A ∩ B (A intersect B -- only items found in both sets), A - B (resulting set are members of A but no B). {1, 2, 4} ⊆ {0, 1, 2, 3, 4, 5} (1st set is subset of 2d if and only if every element of set A is also an element of set B
  + pair<iterator,bool> insert(<type> val), erase, clear, find, size, empty
  + iterating: set<string>::iterator it; for(it = words.begin(); it != words.end(); ++it) { cout << \*it << endl; }
  + unordered\_set: elements are iterated over in a seemingly random order (faster than set)
* maps: key-value pairs. if given a key, can find value v, v quickly
  + insert(key\_type key, value\_type value), erase(key\_type key), at(key\_type key), empty, size
  + iterating: map<int, string>::iterator it; for(it = items.begin(); it != items.end(); ++it) { it->first = key; it->second = value }
  + unordered\_map: main benefit is that it will find a key and access its associated value more quickly and efficiently than a map can
* Vector: Like an array but smarter because can resize. Good for accessing elements by index and inserting at the end. Not good to insert at the front/middle a lot. Stack: LIFO, good for reversing input, can only access top. Queue: FIFO, good for first-come-first-served ordering, add to back and remove from front. Deque: Like a vector but better at inserting in the front, more complex underneath the hood. Priority Queue: Elements sort themselves as you insert them, can only remove from front. List (Linked List): Uses a chain of "nodes" (tiny data structures that hold a single value) with pointers. Good at inserting and removing anywhere in the list and sorting. Not good at accessing elements by index. Set: Stores elements such that each element is unique. There are no indexes. Can find union, intersection, or difference between two sets. Can iterate over in sorted order. Good at finding elements very quickly. Unordered Set: Same as set but iterator returns elements in seemingly random order.
* recursion
* big O

|  |  |  |
| --- | --- | --- |
| O(1) | Constant | Inserting to the front of a linked list, determine if an element is in an unordered set, using a vector to access an element by index |
| O(log n) | Logarithmic | Determine if an element is in a set, binary search |
| O(n) | Linear | Inserting at the beginning of a vector, using a linked list to access an element by index, anything that requires a single for-loop to iterate over each element |
| O(n log n) | Linearithmic, loglinear, or ... | Quicksort, merge sort, heapsort (name is also quasilinear) |
| O(n2) | Quadratic | Sorting an array using selection sort, insertion sort, or bubble sort; if requires a double nested for-loop |
| O(n!) | Factorial | Brute-force solution to traveling s'man problem (finding the fastest route to go through multiple cities) |

* logarithms
  + Logs are used to solve problems of this format
* binary search
* bubble sort: worst case O(n^2) - After a single pass over the array, the greatest element in the array will be in the correct place at the end.
* selection sort
  + 1. Find smallest value in the unsorted part. 2. Swap value to index after the end of the sorted part. Repeat until unsorted part empty.
* insertion sort: like picking up hand of cards and sorting as you pick up
* merge sort:O(n log n).

if the size of the List is at least 2

split the List into the left half and the right half

recursively sort left

recursively sort right

merge left and right into one List

* quick sort

