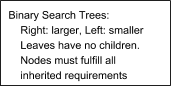
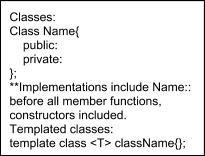
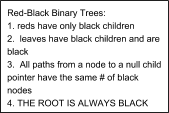
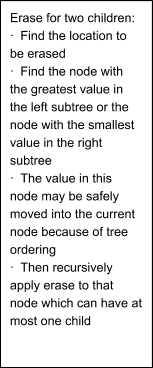
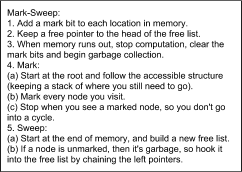
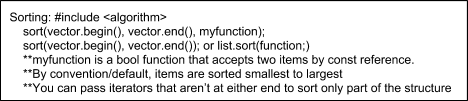
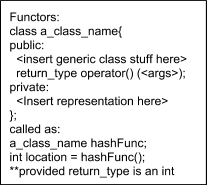
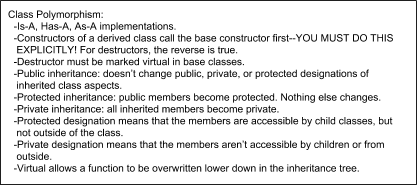
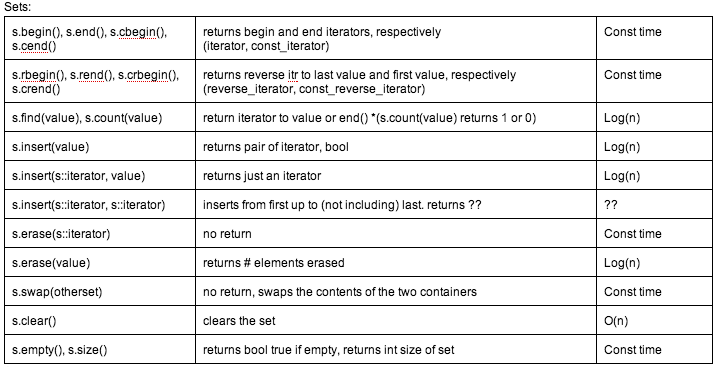
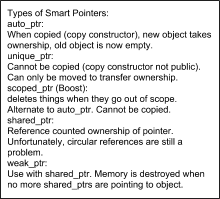
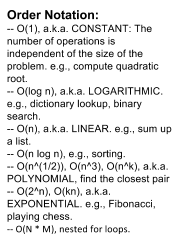
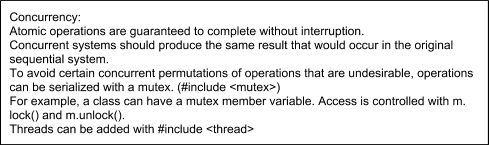
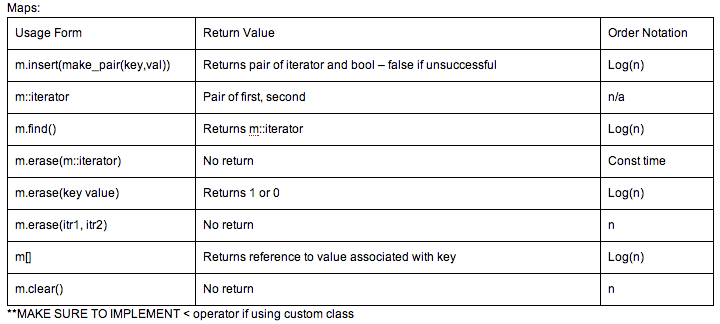
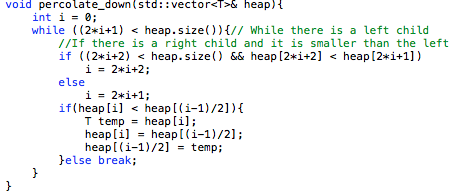
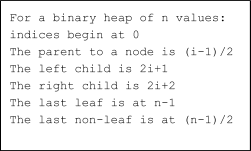
ALWAYS PASS BY CONST REFERENCE A LIST, STRING, 









Lists: (No [] opeators, navigate with iters)

|  |  |  |
| --- | --- | --- |
| Same l.begin, l.end, etc | Returns appropriate iterator | O(1) |
| l.push\_front(val), l.push\_back(val) | Insert element at beginning/end of list, none. | O(1) |
| l.pop\_front(), l.pop\_back() | Remove the first/last element in list, none. | O(1) |
| l.insert(iter, val) | Insert val before iter, returns iter to new item. | O(1) |
| l.erase(iter) || l.erase(iter\_begin, iter\_end) | Erases iters, returns item after last erased | O(1) |
| l.unique() | Removes duplicates from list, returns none. | O(n) |
| l.reverse() | Reverses elements in list, returns none. | O(n) |

|  |  |
| --- | --- |
| **Queue**  Implemented as hobbled linked list, where you can only pop from the front and push to the back. Member functions in STL are q.front(), q.back(), q.push(value), and q.pop(). pop and push have no return values.  **Priority Queue**  Implemented as complete min heaps.  Remember heaps are actually stored in arrays.  A complete heap has no “holes” in it.  All vector slots are filled except for the last leaves.  Every node has the max number of children possible, except the bottom layer. Leaf nodes are full and fill left to right in the heap.  Min value is the root, max is going to be a leaf node. | **Binary** Heap  A binary tree that:  Is complete; that is, all levels of the tree, except possibly the last one (deepest) are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right.  All nodes are *either* [greater than or equal to] *or* [less than or equal to] each of its children, according to a comparison [predicate](http://en.wikipedia.org/wiki/Predicate_(mathematical_logic)) defined for the heap.  Heaps with a mathematical "greater than or equal to" (≥) comparison predicate are called *max-heaps*; those with a mathematical "less than or equal to" (≤) comparison predicate are called *min-heaps*.  Min-heaps = Priority Queue |
| INSERT  To add an element to a heap we must perform an *up-heap* operation (also known as *bubble-up*, *percolate-up*, *shift-up*, *trickle up*, *heapify-up*, or *cascade-up*), by following this algorithm:   1. Add the element to the bottom level of the heap. 2. Compare the added element with its parent; if they are correctly ordered, stop. 3. If not, swap the element with its parent and return to the previous step. | DELETE  The procedure for deleting the root from the heap (effectively extracting the maximum element in a max-heap or the minimum element in a min-heap) and restoring the properties is called *down-heap* (also known as *bubble-down*, *percolate-down*, *shift-down*, *trickle down*, *heapify-down*, *cascade-down* and *extract-min/max*).   1. Replace the root of the heap with the last element on the last level. 2. Compare the new root with its children; if they are in the correct order, stop. 3. If not, swap the element with one of its children and return to the previous step. (Swap with its smaller child in a min-heap and its larger child in a max-heap.) |

