* Priortity Queues
  + To find left child
    - 2i + 1
  + To find right child
    - 2i + 2
  + To find the parent it’s the inverse
    - (I-1)/2
  + Instead of container we will use vector and <class T>
  + Push heap (O(logn)
    - Template <class Iterator>
    - Void Push\_heap(iterator start, iterator stop)
      * Unsigned int position = (stop – start) -1;
      * Unsigned int parent = (position –1)/2;
      * While (position > 0 && start[position] > start[parent])
        + Swap(start[position], start[parent]);
        + Position = parent;
        + Parent (position –1)/2
  + If you have a heap of N elements the height is (floor) log(base2)n.
  + Floor is the rounded down number, ceiling you round up.
  + Pop Heap (O(logn))
    - Template <class iterator>
    - Void pop\_heap(iterator start, iterator stop)
      * Unsigned int lastPosition = (stop – start) -1
      * Swap (start[0], start[lastPosition]);
      * Adjust\_heap(start, lastPosition, 0);
  + Adjust heap
    - Template <class iterator>
    - Void Adjust\_heap(iterator start, unsigned heapSize, unsigned position)
      * While(poition < heapSize)
        + Unsigned int childpos = position \* 2 + 1;
        + If (childpos < heapsize)

If (childpos + 1 < heapSize) &&(start[childpos +1] > start[childpos])

Childpos++;

If(start[position] > start[childpos])

Return;

Else

Swap(start[position], start[childpos]);

* + - * + Position = childpos;
  + Heap Sort
    - Template <class Iterator>
    - Void make\_heap(iterator start, iterator stop)
      * Unsigned int heapSize = stop – start;
      * For (int I = (heapSize/2) - 1; I >= 0; I--;)
        + Adjust\_heap(start, heapSize, I);
    - Template <class iterator>
    - Void heap\_sort(iterator start, iterator stop)
      * Make\_heap(start, stop)
      * Sort\_heap(start, stop);
    - Template <class iterator>
    - Void sort\_heap(Iterator start, iterator stop)
      * Unsigned int lastPosition = stop – start – 1;
      * While (lastPosition > 0)
        + Swap(start[0], start[lastPosition]);
        + Adjust\_heap(start, lastPosition, 0);
        + LastPosition--;
* Map or Dictionary or Associative memory
  + Operator [] - Indexed data structure
  + Efficient
    - All operations are in o(logn)
  + Implementation:built on sets/multisets
  + Collection of key\_value pairs
  + Map - > unique keys
  + Multi map -> duplicate keys
  + Ex:
    - Include <map>
    - Map<string, int> m; (strings are keys, ints are values)
    - M["me"] = 5;
    - M{"us"] = 2;
    - M["you"] = 4;
    - M["zoo"] = 3;
  + Void readItems(map<string. Int> & m)
    - String word;
    - Int value;
    - While(cin >> word >> value)
      * M[word] += value;