Sorting out sortall

- How to check if a vector is sorted:
  - using maps
  - using another sort function (certified sort)

- Why are smart pointers faster, what are they?
  - moving data is expensive, which $O(n^2)$ sort is good?
  - how to compare “smart” pointers? printing?

- what is a templated sort function?
- what is a function pointer?
- what is a typedef?
- what’s what?
Heaps and Huffman and ...

- Huffman coding is a greedy algorithm

  locally optimal steps turn into a globally optimal solution
  take nodes with minimal weights in Huffman tree,
  combine into a new node with weight = sum, repeat
  when one node is left in tree (with lots of descendents),
  this is the optimal Huffman tree

- (sidebar) Greedy doesn’t always work: making change:
  U.S. coins ok for fewest coinage change-making
  if coins are missing, greedy doesn’t work: make change for
  31 cents without using nickels
Creating Heaps

- Heap is an array-based implementation of a binary tree used for implementing priority queues, supports:
  insert, findmin, deletemin: complexities?

- Using array minimizes storage (no explicit pointers), faster too --- children are located by index/position in array

- Heap is a binary tree with shape property, value property
  shape: tree filled at all levels (except perhaps last) and filled left-to-right
  each node has value smaller than both children
Array-based heap

- store “node values” in array beginning at index 1
- for node with index \( k \)
  - left child: index \( 2 \times k \)
  - right child: index \( 2 \times k + 1 \)

- why is this conducive for maintaining heap shape?
- what about heap property?
- is the heap a search tree?
- where is minimal node?
- where are nodes added? deleted?
Adding values to heap

- To maintain heap shape, must add new value in left-to-right order of last level.
  - Could violate heap property.
  - Move value “up” if too small.

- Change places with parent if heap property violated.
  - Stop when parent is smaller.
  - Stop when root is reached.

- Pull parent down, swapping isn’t necessary (optimization).

![Diagram of heap transformation](image-url)
Adding values, details

void insert(int elt)
{
    // grow myList if needed
    myCount++;
    int loc = myCount;
    while (1 < loc &&
            elt < myList[loc/2])
    {
        myList[loc] = myList[loc/2];       // ???. move up
    }
    // add elt to tree
    
    Vector myList