More sorting algorithms
Bubble sort (for int's)

\[ x = n - 1; \text{ sorted} = \text{false}; \]

\[
\text{while}(!\text{sorted})\{
\quad \text{sorted} = \text{true};
\quad x = x - 1;
\quad \text{for}(\text{int} \ y = 0; y < x; y++)
\quad \quad \text{if}(\text{array}[y] > \text{array}[y+1]) \{
\quad \quad \quad \text{swap}(\text{array}[y], \text{array}[y+1])
\quad \quad \quad \text{sorted} = \text{false};
\quad \quad \}
\}
\]
Example

20 15 17 14 6 9 4 1 8
15 17 14 6 9 4 1 8 20
15 14 6 9 4 1 8 17 20
14 6 9 4 1 8 15 17 20
14 6 9 4 1 8 14 15 17 20
6 9 4 1 8 14 15 17 20
6 4 1 8 9 14 15 17 20
4 1 6 8 9 14 15 17 20
1 4 6 8 9 14 15 17 20
1 4 6 8 9 14 15 17 20
Cost analysis

- **Worst case**
  - n times through the main loop
  - \( n + (n-1) + (n-2) + \ldots = O(n^2) \) comparisons and swaps

- **Best case**
  - the array is already sorted
  - \( O(n) \) comparisons and swaps
Heap sort

- A heap is an almost complete binary tree:
- The tree is completely filled on all levels except possibly lowest, where it is filled in left to right order.
- The height of a binary tree with n nodes in it is $O(\log_2(n))$. 
Example

The heap with 9 nodes
The heap property

- When storing comparable's in a heap, ensure that:
  - the value of the element at a node is greater than, or equal to, the value of the elements at its two children
  - the value at a node is less than, or equal to, the value at its parent.
Building a heap

Add node in the next available space in the almost complete binary tree and "bubble" that node up to restore the heap property for the tree.

Since this only involves operations from a leaf node along the path to the root node, there are at most $\log_2(n)$ operations required.
Example

Insert the values 9, 14, 10, 6, 17
Example

Insert the values 9, 14, 10, 6, 17
Example

Insert the values 9, 14, 10, 6, 17
Removing the largest element

- remove the root
- place the smallest element at the root and bubble it down:
  - if not a heap, swap it with the largest of its children
  - continue until you get a heap
Cost

for a heap with n elements:
the height of the tree is $O(\log(n))$
insert and remove move an element on a path from the root to a leaf
so both take $O(\log(n))$ operations
Data representation

The most common implementation uses an array of ...

For a heap with n elements, indices range from 0 to n-1

Easy access to parent and children:

left child(i) = 2i+1
right child(i) = 2i+2
parent(i) = (i-1)/2
Heap sort

Simple algorithm:
  create an empty heap
  insert all elements
Repeat
  remove the largest
  until the heap is empty

Cost for n elements: $O(n \log(n))$