Compare our various sorting algorithms. Which is best?

Answer: Merge sort is $\Theta(n \log n)$, but requires a lot of extra space. Quicksort is $\Theta(n \log n)$ if the pivot is chosen carefully (e.g. using the selection algorithm), but choosing the pivot carefully seems to take a lot of time in practice, and not choosing the pivot carefully leads to $\Theta(n^2)$ performance. Heap sort is $\Theta(n \log n)$ and is easy to implement, so it seems best. (But in practice many people use quicksort with a somewhat-careful pivot-choosing algorithm.)

Compare the performance of `__setitem__` for our various implementations of `Dictionary`.

Answer: In our list-of-pairs implementation, `__setitem__` is $\Theta(n)$ because it has to scan through all of the pairs (in the worst case). In our `BinarySearchTreeNode` implementation, `__setitem__` is $\Theta(n)$, because the tree may be unbalanced or a bad hash may produce a long chain. In our `AVLTreeNode` implementation there is no problem of balance; `__setitem__` is $\Theta(\log n)$ with a perfect hash function, but still $\Theta(n)$ with a bad hash that produces a long chain. In our hash table implementation, `__setitem__` is $\Theta(n)$ because it may need to resize the hash table; on the other hand, the amortized cost of `__setitem__` is $\Theta(1)$.

The following graph shows seven computers in a network. Each network link is bidirectional and labeled with its latency (the time cost of making contact across that link). Execute Dijkstra's algorithm on the network completely, so that the shortest path from computer D to computer G is found. Your work must clearly show how the results-so-far and the queue evolve over time.

Answer: Here is the final results dictionary; you can see what happened when I updated the entries for E and G.

D, None, 0

A, D, 8

C, D, 7

E, ~~D, 16~~ C, 12

F, D, 2

G, ~~F, 22~~ E, 20

B, C, 4

Here is the priority queue with distances for priorities. When I dequeue, I just look through this list to find the lowest distance, and then I put an X next to that entry. Again you can see what happened when I updated the entries for E and G.

D, 0 X

A, 8 X

C, 7 X

E, ~~16~~ 12 X

F, 2 X

G, ~~22~~ 20 X

B, 11 X

From the results we immediately determine that the shortest path from D to G is the one of length 20 that arrives at G from E, at E from C, and at C from D.

Below is a list of numbers. Perform heap sort on this list. For grading purposes, on each intermediate step you must display the heap as a list.
$[17, 32, 21, 20, 15, 35, 12, 15]$

Answer: First we build the heap by percolating down elements 4, 3, 2, 1, and 0.

$[17, 32, 21, 20, 15, 35, 12, 15]$

$[17, 32, 35, 20, 15, 21, 12, 15]$

$[35, 32, 17, 20, 15, 21, 12, 15]$

$[35, 32, 21, 20, 15, 17, 12, 15]$

Now we start removing elements. We remove the first element, put the last element into its place, and then percolate down that element.

$[35, 32, 21, 20, 15, 17, 12, 15]$ (remove 35)

$[15, 32, 21, 20, 15, 17, 12]$

$[32, 15, 21, 20, 15, 17, 12]$

$[32, 20, 21, 15, 15, 17, 12]$ (remove 32)

$[12, 20, 21, 15, 15, 17]$

$[21, 20, 12, 15, 15, 17]$

$[21, 20, 17, 15, 15, 12]$ (remove 21)

$[12, 20, 17, 15, 15]$

$[20, 12, 17, 15, 15]$

$[20, 15, 17, 15, 12]$ (remove 20)

$[12, 15, 17, 15]$

$[17, 15, 12, 15]$ (remove 17)

$[15, 15, 12]$ (remove 15)

$[12, 15]$

$[15, 12]$ (remove 15)

$[12]$ (remove 12)

The list formed by the removed elements is $[35, 32, 21, 20, 17, 15, 15, 12]$.

Give an example of a problem that is better solved using depth-first search than using breadth-first search.

Answer: Topological sort is naturally done by depth-first search; it can be used to sequence a graph of interdependent tasks. Evaluation of parse trees of algebraic expressions is done depth-first. In our textbook, the knight's tour problem is solved depth-first.

Recall that a scene tree is a tree of drawable objects, such that each object is positioned relative to its parent. That is, a parent sets up its coordinate system before telling its children to draw; each child's drawing automatically happens in whatever coordinate system is current. Our main example was a dog regarded as a tree of body parts. In reality, scene trees are usually called *scene graphs*, because they are graphs not trees. Why? What extra capability do graphs offer, that might be useful for organizing drawable objects in a scene?

Answer: Graphs have the extra capability that there may be more than one path from one node to another. In our dog example, there are four identical paws. Rather than having four identical tree nodes to store the paw-drawing information, why not just have one, with all four lower legs pointing to it? A graph allows us to exploit redundancy in the scene data.