Sorts

1) Selection
   a) Big O best case time - O(n^2)
   b) Big O ave case time - O(n^2)
   c) Big O worst case time - O(n^2)
   d) worst case data set - reverse order
   e) space requirements - one vector
   f) other characteristics or comments - two nested loops, terribly, slow sort, inefficient for n>100 data sets, many comparisons, could be many assignments if in reversed order, only one swap per pass of the inner loop, no early exits

2) Bubble
   a) Big O best case time - O(n)
   b) Big O ave case time - O(n^2)
   c) Big O worst case time - O(n^2)
   d) worst case data set - reverse order since a lot of values must "bubble down"
   e) space requirements - one vector
   f) other characteristics or comments – two nested loops (for nested in a while typically) with early exit in the outer loop, good when the data is mostly sorted to start with, could cause a lot of swaps

3) Insertion
   a) Big O best case time - O(n)
   b) Big O ave case time - O(n^2)
   c) Big O worst case time - O(n^2)
   d) worst case data set - reverse order
   e) space requirements - only one vector if you are efficient but commonly done with two vectors (1 unsorted, other sorted), two nested loops
   f) other characteristics or comments – good when the data is mostly sorted to start with, absolutely terrible in many cases, aka poker hand sort, early exit from inner loop

4) Shell
   a) Big O best case time - O(n(log n)^2)
   b) Big O ave case time - O(n(log n)^2)
   c) Big O worst case time - between O(n^1) and O(n^2) (but closer to O(n^2))
   d) worst case data set - reverse order
   e) space requirements - one vector
   f) other characteristics or comments – the "inner sort" of the partition elements which are separated by a gap can be performed with whatever sorting algorithm you wish (our author uses an insertion sort, aka a diminishing increment sort, best case is when successive gap sizes are relatively prime
5) Quick
   a) Big O best case time - O(n log n)
   b) Big O ave case time - O(n log n)
   c) Big O worst case time - O(n^2)
   d) worst case data set - when data is already ordered and when poor pivot values are chosen
   e) space requirements - Big O for space is O(n) in worst case, one vector in each recursive stack frame
   f) other characteristics or comments – may be worthwhile to randomize the data so that data is not close to sorted before using quick sort
6) Heap
   a) Big O best case time - O(n log n)
   b) Big O ave case time - O(n log n)
   c) Big O worst case time - O(n log n)

   d) worst case data set - more swaps if data is in reverse order, but no real worst case
   e) space requirements - one vector though in some implementations people use two vectors (one for the heap and one for the sorted values)
   f) other characteristics or comments - two phases to this algorithm: putting values into a heap (i.e. binary tree with the heap property) and then "picking" the sorted values out of the heap one-by-one
7) Merge
   a) Big O best case time - O(n log n)
   b) Big O ave case time - O(n log n)
   c) Big O worst case time - O(n log n)
   d) worst case data set - none
   e) space requirements - high, log n recursive stack frames each with a separate vector
   f) other characteristics or comments - good to use if data compared to the quick sort if the data is almost sorted
8) Radix
   a) Big O best case time - O(n)
   b) Big O ave case time - O(n)
   c) Big O worst case time - O(n)
   d) worst case data set - if data elements are large it will consume a lot of memory
   e) space requirements - excessive
   f) other characteristics or comments - not suitable for all types of data such as floating-point values, has a large constant of proportionality despite being O(n)