Measuring sorting

The goal in this assignment is to measure in real time the efficiency of various sorting algorithms and to compare them.

You should write this code in a single file and print it out to turn in, along with the graphs and other data that you produce.

Step 1: random lists

Write a function that takes as input an integer n and returns a list of the integers 1 through n in random order. Make sure that it is truly random – all orders of the integers should be equally likely.

Step 2: implement sorts

You should implement bubble sort, insertion sort, merge sort, and quicksort. It is ok to use code that was given in lecture, though you should check it for bugs.

Step 3: timing sorts

You should time how long it takes each algorithm to sort lists of various sizes. You should create a graph that shows a line representing each sort's time as a function of the input length. Write a short note explaining how they compare at various input sizes and whether the lines you are getting are consistent with our theoretical expectations. You will find the "time" module useful for this. I recommend timing the algorithm sorting a number of lists and then averaging to find the time for a given list. That will give a more accurate estimate of the time it takes on average. Make sure you don't include generating the randomly-ordered lists when timing – you want to time only how long it takes to sort them. Choose enough different input sizes to get a good idea of the shape of the graph.

Step 4: changing base cases

Often it is found that for very small lists the algorithm with the fastest asymptotic run time is not the fastest. Create a hybrid sort algorithm that works like merge sort except that small lists are sorted according to insertion sort instead of recursing further. Does this run faster? If so, what list size is optimal for the switch to insertion sort? Give data to back up your answers.