Sorting Algorithms (and the UnorderedList)

Implement the following projects and collect any results in a Word (or other word-processing) doc to be printed out, stapled and upload to dropbox in by the due-date. Copy any questions, in **bold**, into your document so that I have context for your answers. Title and label axes of all graphs, if graphs are required.

Part I:
- BubbleSort
- SelectionSort
- InsertionSort

Implement the sorting algorithms above, and generate average timings (e.g. using time.time()) on random Lists of various sizes of *fraction* objects *as well as* same-size lists of integers. (**Hint**: should you want to print out list-contents to ensure your sort-algorithms are actually sorting, you may want to modify your Fraction class's __str__() method to return the str() of a floating point number rounded to about five decimal places, rather than the numerator**/**denominator format)

- Pass a *copy* of your random list to each sorting-function

  e.g.:
  ```
  L = GenRandomList()
  begTime = time()
  sortedL = bubbleSort(L[:])
  endTime = time()
  bubSum = bubSum + endTime-begTime

  begTime = time()
  sortedL = SelectionSort(L[:])
  :
  :
  etc....
  ```

- Why do we pass *L[:]* to each function, rather than simply passing L?
- Collect timings of ten evenly-spaced list-sizes (e.g. range(10000, 100001, 10000)), *using the same list of values for each sort-algorithm.* Compare same-size fraction-sort

Sorting Algorithms (and the UnorderedList)

Implement the following projects and collect any results in a Word (or other word-processing) doc to be printed out, stapled and upload to dropbox in by the due-date. Copy any questions, in **bold**, into your document so that I have context for your answers. Title and label axes of all graphs, if graphs are required.

Part I:
- BubbleSort
- SelectionSort
- InsertionSort

Implement the sorting algorithms above, and generate average timings (e.g. using time.time()) on random Lists of various sizes of *fraction* objects *as well as* same-size lists of integers. (**Hint**: should you want to print out list-contents to ensure your sort-algorithms are actually sorting, you may want to modify your Fraction class's __str__() method to return the str() of a floating point number rounded to about five decimal places, rather than the numerator**/**denominator format)

- Pass a *copy* of your random list to each sorting-function

  e.g.:
  ```
  L = GenRandomList()
  begTime = time()
  sortedL = bubbleSort(L[:])
  endTime = time()
  bubSum = bubSum + endTime-begTime

  begTime = time()
  sortedL = SelectionSort(L[:])
  :
  :
  etc....
  ```

- Why do we pass *L[:]* to each function, rather than simply passing L?
- Collect timings of ten evenly-spaced list-sizes (e.g. range(10000, 100001, 10000)), *using the same list of values for each sort-algorithm.* Compare same-size fraction-sort

times to integer-sort times. Represent the results of your timings in one or more Excel charts, to best illustrate any differences in each algorithm-class.

- Explain the results shown in your charts. What big-O running-times generate which shapes?

Part II:

Write me some *pseudocode* for one pass of the SelectionSort algorithm through random values stored in an Unordered-List, or Linked-List, of values (rather than a standard "Python" List).

- How would you implement a swap using a Linked-List?
- What is the minimum number of variables required?

Step me through an *illustrated* example (including all needed variables/arrows) of a Linked-List of size 5, where the positions of nodes 3 and 4 are swapped.
Show the pseudocode to accomplish this in your report.