CSci 3501 Lab 3

See canvas for due date

60 points between Lab 2 and Lab 3

Work in pairs

- All lab submissions should be done by canvas. Please be sure to include your group members.
- When working on the lab, please comment your work so that it is clear what the contributions from each person are.
- At the end of the lab each group should submit the results of their in-class work. Please indicate if this is your final submission. Don't forget to answer all the questions below.
- If your submission at the end of the lab time was not final, please submit a final copy before the due time.

Grading

1. Implementation of quicksort: 15 pts
2. Correct setup (using Comparable, using the standard Java sorting, etc): 10 pts
3. Test data and setup: 10 pts
4. Randomized quicksort: 6 pts
5. Median-of-three pivot: 6 pts
6. Switching to insertion sort: 6 pts
7. Observations and conclusions: 7 pts

Overview and goals

In this lab we will continue studying efficiency of quicksort
(see http://en.wikipedia.org/wiki/Quicksort). The goal is to develop and study approaches to make the quicksort split data as evenly as possible. The approaches include a randomized pivot selection, a median-of-three pivot selection, and use of insertion sort when the array is nearly sorted. You will continue experimenting with quicksort on different types of data (completely random, ordered, partially ordered) and compare it to the pre-defined sort (Timsort) in the number of comparisons. The goal is to learn practical approaches to efficient algorithm implementation.

Note that other ways of speeding up quicksort may reduce the program's running time by cutting down on recursive calls or by providing more efficient memory usage. However, they do not reduce the number of element comparisons, and thus will not be included in this lab.

Tasks

Use your implementation of quicksort and the testing code from the previous lab. As in the last lab, you will be using arrays of 10,000 elements filled as follows:

1. Generated at random
2. Sorted in increasing order (if this creates a stack overflow, use 1,000 elements)
3. 10 sorted sequences of 1,000 elements each
4. 100 sorted sequences of 100 elements each

Run all your tests (see below) 5 times on each of these sets.

You need to implement the modifications of quicksort listed below. Please write a new copy of quicksort for each of the three modifications. Then compare the results to the original quicksort and to the standard sorting (use the same data for all three sorting algorithms). Make sure to test (for each modification!) that the resulting array is sorted. Record the results (the number of comparisons).

- **Randomized quicksort:** choose the pivot at random at every step of the algorithm. For simplicity just exchange the randomly chosen pivot with the element in the position r before doing anything else, this way you don't have to rewrite your algorithm (see p. 179 for more details).
- **The median-of-three pivot selection:** to select a pivot, pick three subarray elements at random, then choose their median as the pivot (see exercise 7-5 p. 188). Use `compareTo` for comparison of the three elements since their comparison contributes to the total cost. Note that this approach becomes less efficient as the array size decreases. Use a threshold value k to switch to the usual pivot choice when the portion of the array passed to quicksort is less than k. Try different values of k and choose an optimal one (approximately).
- **Switching to insertion sort at the end:** when the array is nearly sorted, stop the quicksort without finishing the sorting, and then use insertion sort on the entire array to finish the process. As in the previous problem, choose a threshold subarray size to leave the array for the insertion sort. You will need to implement insertion sort (p. 18). Make sure to include the comparisons made by the insertion sort into the total.

In case you are getting a Stack Overflow exception: numbers in increasing order may generate a stack overflow because there are 10,000 calls (that's the worse case of quicksort when there is one call per element) which is more than the default stack size. You need to increase the stack size in the JVM (Java Virtual Machine) which you can do by setting a JVM flag in Eclipse:
`http://stackoverflow.com/questions/15313393/how-to-increase-heap-size-in-eclipse`,
or in Intellij:
`https://www.jetbrains.com/help/idea/increasing-memory-heap.html`