# QUICKSORT AND IT'S PIVOTS

## CS 200 RECITATION 6

## A note about the Inductive Hypothesis

Last week I said that the inductive hypothesis was:

*If p(k) holds then p(k+1) also holds.*

This was not quite correct. The inductive hypothesis is actually:

*p(k) holds.*

Sorry for the error, haven't been able to let everyone know yet so I'm announcing it here. Sangmi made arrangements to keep this from costing points on the midterm.

## Today

We will instrument a quicksort algorithm so we can tell how often parts of it run. Then we will experiment with different Pivot selection techniques and see their effect on the run time.

## Given Files

- **MainClass.java** - will be used to test the quicksort. Generates arrays of ints, sorts them, checks the output, computes data about performance.
  - set the variable: **problem_style** to control what type of arrays will be used for testing. Valid settings are: SORTED or REVERSED or RANDOM
  - use **gen_array(int)** to generate an array for use in testing. Argument is desired size. Array will be sorted, reversed or random depending on the setting of problem_style
  - use **run_sort(int[])** to run the quicksort. It can thow an exception, so this method has a try... catch block to deal with that.
- **QSortAlgorithm.java** - the quicksort we looked at in class during exam review. Slightly tweaked (was orginally an applet) to allow easier setting of the pivot.
  - modify the **pivot_select** variable (line 79) to try different pivots.
    - note that this algorithm uses the variable **lo** as the lowest index and **hi** as the highest index. Set the pivot relative to one or both of these for best results
  - add counters (variables in Globals.java) where you feel the should go.
- **Globals.java** - has some global variables which will be used as counters when the quicksort is running.
  - use the variables in here to count things in the quicksort. These should be visible to any class in the same project
  - you may create additional counters if you wish.
  - **numRecursions** - counts the number of recursive calls the algorithm makes

- **numLeftLoops** - counts the number of times the left index is moved
- **numRightLoops** - counts the number of times the right index is moved
- **numLoops** - counts the number of outer loops
  - Remember that quicksorts have a left and a right index that they walk through the array, comparing values with the pivot and switching if necessary. The left and right loops counters count how many times these indicies increment. numLoops counts any other loops

# Tips

- The Java.util.Arrays class is very useful:
  - Arrays.sort(...) - sorts an array, in case you want to check correctness of our quicksort
  - Arrays.equals(...) - checks equality of two arrays.
  - Arrays.copyOf(...) - copys arrays.
  - Arrays.toString(...) - returns the passed array as a nicely formatted string (good for printing)

# Exercise

- Implement the **run_test(int)** method in MainClass.java. It should:
  - get an array of ints (sorted, reversed or random)
  - argument is the size of that array
  - run the quicksort on that array
  - compare the result with the output of Arrays.sort(), to see if our quicksort sorted correctly
  - return true if sorted correctly
- Implement the **run_tests()** method in MainClass.java. It should:
  - run many tests using run_test(int).
  - collect counts for each test (using the global variables)
  - print counts, and success or failure of each test (sorted correctly or not)
  - average those counts over all the tests
  - print averages of the counts
- You may add any methods you want to add to MainClass.java
- Be able to run tests of this size:
  - 100 arrays, 10,000 integers per array
- Experiment with the pivot location:
  - try leftmost index (lo) and rightmost index (hi) as pivots. Note the averages for sorted, reveresed and random arrays
  - try different pivots of you own design, look for a better pivot choice
    - does the choice of a sorted, reversed or random array affect which pivot is best?
  - try pivots which do not use lo or hi. maybe just pick a number. what effect does this have on the algorithm? What happens if you use 0 as a pivot? what about array.length - 1 ?

## Grading

- Sign the attendance sheet
- Checkin a text file with the results of your above experiments:
  - averages for 100 arrays of 10,000 integers each:
    - with pivot = lo, pivot = high, pivot of your own design.
      - for each pivot, include averages for sorted, reversed and random arrays
  - Here are checkin commands for each recit:
    - Monday: ~cs200/bin/checkin R6L01 R6L01.txt
    - Tuesday: ~cs200/bin/checkin R6L02 R6L02.txt
    - Wednesday: ~cs200/bin/checkin R6L03 R6L03.txt
    - Thursday: ~cs200/bin/checkin R6L04 R6L04.txt