

1 Objectives

- Practice with recursion
- Look at program efficiency
- Improve a program's efficiency
- Learn to plot graphs for a given set of data using MATLAB

In this lab you will learn about program efficiency by analyzing the efficiency of quicksort. You will also learn a technique that improves the efficiency of quicksort by providing better pivots in the case of certain inputs that cause the unmodified quicksort to take a long time (i.e., $O(n^2)$ or quadratic time) to run.

2 Efficiency of Quicksort

2.1 Problem Description

In this lab you will analyze the efficiency of quicksort. You will test the number of key comparisons that quicksort uses for random data and for data that is already sorted. (Recall that key comparisons are comparisons between the values being sorted. Also recall that quicksort has its worst-case performance on inputs that are already sorted in ascending or descending order.) Then, you will modify quicksort so that its worst-case behavior on sorted lists is dramatically improved. After you have completed test runs of the unmodified and modified versions of quicksort and collected data on their performance when sorting random data and data that is already sorted, you will graph your data and compare it with our predicted behavior of quicksort.

2.2 Details

Create a lab07 project for today lab and then import the two files `MyGraph.m` and `Quicksort.java` from

```
~csci204/2009-fall/student/labs/lab07
```

Add code to the `main()` method in `Quicksort` so that it uses a loop in which it generates arrays of increasing size from 8, 16, 32, ..., 16384 (i.e., $2^3, 2^4, \dots, 2^{14}$). That is, the array size will double after each iteration. You should then fill the generated array with random values using the `randInt()` method. Once you've got an initialized array, you should call the `quicksort()` method to sort the array.

Try to run your code to make sure that you don't have any unexpected runtime errors.

You will probably need to increase the size of Java's stack space for this lab. Select **Run** → **Run Configurations** Make sure Quicksort is highlighted on the left. Click on the (x)= **Arguments** tab and type the following into the VM arguments window:

```
-Xss100000k
```

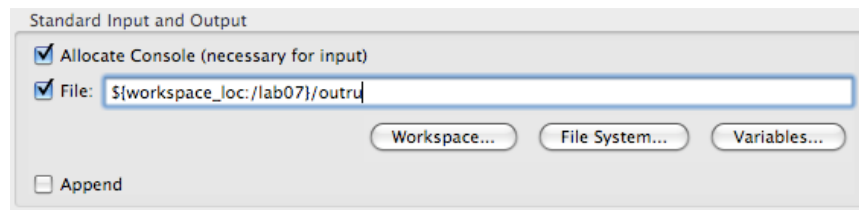
Click **Apply** and close the Config window.

2.3 Counting Key Comparisons

A key comparison occurs whenever two items in the array are compared. Revise the `partition()` method so that it will count the *number of key comparisons* that were required for a sort. Be sure to count only key comparisons.

Reset the count to zero before calling `quicksort()` in `main()`. Then increment it each time you do a key comparison in `partition()`.

After each iteration in `main()`, output a line containing the size of the array and the number of key comparisons. Print *only* these two numbers separated by a space character on each line using `System.out.println()`. Don't label your output or you will need to edit it by hand before graphing your data. Save your output in a file called `outru` by redirecting the output to this file. Do this by selecting **Run** → **Run Configurations**. Make sure Quicksort is highlighted on the left. Click on the **Common** tab, check the **File:** box. Enter the file name as shown below.



(The last thing in the File: line in the picture is the cursor, not a letter). Press the **Apply** button and then **Run**. A file named `outru` should appear in your project folder.

Make sure you name the file as specified because the plotting program in MATLAB will look for these names automatically.

Now change `main()` method so that the array you are sorting is already sorted. Comment out the line using random numbers, you will need it again later. The easiest way to get a sorted array is to assign position i of the array the value i . Run

the changed program and save its output in a file called `outsu` under the **Run** → **Run Configurations** menu as you did before. A file named `outsu` should appear in your project folder.

Look at `outru` and `outsu`. Compare the number of comparisons when the array is already sorted with the number of comparisons when it contains random data. Quicksort's worst case behavior occurs when the array is already sorted.

Note: `outru` stands for output, random numbers, unmodified quicksort. `outsu` stands for output, sorted numbers, unmodified quicksort.

2.4 Modify Quicksort

As you saw in the previous section, quicksort's behavior is fairly poor when the array is already sorted. There's an easy fix to this problem that also has a very high probability of working on other bad inputs.

The reason that quicksort performs so poorly is that we always choose the first element in the array as the partition element (i.e., the *pivot*). When the array is already sorted, the result is a bad split of the array. The fix to the problem is to use *median-of-three* partitioning.

You will implement median-of-three partitioning by examining the values in the left, middle and right positions of the array. (What is the index of the middle position?) Swap the median of these values into the first/pivot position and then proceed as usual. You can determine which of these three values has the median value by using just three `if` statements and the `swapElements()` method. (**Hint:** First, put the largest element in the last place (requires at most two comparisons). Then, move the median value into the pivot slot (one comparison). Try this out on paper to convince yourself it works. It should use exactly three simple `if` statements and three calls to `swap`.) Count these three comparisons in your key comparison count. Using sorted input, run your program and save the output in a file called `outsm`. Apply your change and run quicksort. A file named `outsm` should appear in your project folder.

Use this modified version of quicksort with arrays that begin with random values (instead of with sorted values). Save output from this run in a file called `outrm`. A file named `outrm` should appear in your project folder.

2.5 Plot Your Results

Now you will use the MATLAB commands in `MyGraph.m` to produce the plot. Open a UNIX terminal window and change to the directory for today's lab and copy `MyGraph.m` into the lab folder. Start MATLAB at the UNIX prompt by typing `matlab`. Open the command file `MyGraph.m` by typing

```
edit MyGraph.m
```

at the MATLAB prompt. Change the `title` command so that it inserts *your* name and this lab number. Save your work and close the editor. Then type `MyGraph` at the MATLAB command prompt and it will execute the script for you.

If MATLAB doesn't do a good job of placing the legend on the graph, you can move it with the mouse.

When the graph is nicely formatted, select **File** → **Export Setup...** from the plot window. In the resulting dialog, click on the **Export...** button. In the next dialog select **Portable Document Format** as the file format and name the file `graphlab07.pdf`. Finally, click on the **Save** button.

3 Upon Completion

Now make Eclipse aware of your graph. An easy way to do this is to import the pdf file into your `src` directory in the `lab07` project. To view your pdf, open a terminal window and type

```
acroread graphlab07.pdf &
```

Finally, make sure that all of your files, including your graph have been committed to your svn repository.