Lab 5

Due March 10

This lab is the first of two labs that will perform experimentation on various aspects of quicksort. In this first lab, you will look at the impact of pivot picking on the performance of quicksort.

Objectives:

- Modify an aspect of a standard algorithm to behave differently
- gather data through experimentation
- create graphs to gain better understanding of the data

Starter file:

- generatedata.cpp
- lab5.cpp
- lab5Tester.cpp
- lab5main.cpp

Setup

- Create a branch called lab5working
- In that branch, create a folder called lab5
- copy in the starter files from the course repo

Background

In this course we spend a lot of time to perform theoretical classification of algorithms into various run times. In practice, though algorithms with the same run time do not actually perform equally. In this lab we will look at how to set up experiments and gather empirical data, and analyze that data to form conclusions.

Before you begin, please watch this video (its not exactly what we will be looking at right now but its a really interesting talk, definitely worth watching):

- Andrei Alexandrescu, CppCon 2019, "Speed is Found in the Minds of People"

Inside the starter files...

generatedata.cpp

In generatedata.cpp , there are a number of generate...() functions. These functions generate data in various "shapes". The shape of the data (especially for quicksort) can actually have a huge impact on performance.

The following describes what each function generates:

function name	what it generates
generateRandom	n unique numbers in random order
generateSorted	0, 1, 2, ... , n-3, n-2, n-1
generateReverse	n-1, n-2, n-3, ... , 2, 1, 0
generateOrganPipe	0, 1, 2,... , n/2, n/2, ..., 2, 1, 0
generateRotated	1, 2, ..., n-2, n-1, 0

Lab5.cpp

you will find the quickSort*() function. The quick sort functions are named after the 4 methods of pivot picking we want to investigate.

Each quicksort function checks to ensure that it has not yet reached the threshold for insertion sort. If it hasn't, it performs partitions the current part of the array being sorted and then quicksorts each partition.

The part of function you must alter is in the partition() function.

The first thing the partition function does, is pick a pivot. This pivot picking is what you will be coding.

lab5Tester.cpp

This is a simple program that will ensure that your sorted array is behaving as expected (you didn't create a bug by putting in your pivot picking code). compile your lab5.cpp with this tester to see if you introduced a bug. Note this tester doesn't guarantee that you picked the correct pivot according to description, only that quicksort works. You must ensure that you do the correct implementation for pivot picking

A simple first experiment:

Use threshold of 16 (this is the threshold used by gnu) for when you stop quick sort partitioning and switch to insertionSort()

In this first experiment we will perform pivot picking in different ways:

- choose middle value of partition as pivot
- choose last value of partition as pivot
- choose random() element of partition as pivot
- choose median of 3 as pivot - picks the median(first,middle,last) of partition as the pivot, use the median of the 3 values as pivot. Note that in this pivot picking algorithm, you place smallest of the 3 values at front, biggest in middle and median at end (it needs to be out of the way for partitioning

No matter what pivot you pick and where it was originally, be sure to move it to the back of the partition or it will get in the way of the partitioning alg.

Experiment and gather results

For each of the 4 pivot picking methods above:

- modify the corresponding quicksort/partition functions to pick the pivot as designated.
- For each data shape, perform quicksort() on an array of 1 million elements***. Do this 5 times. Record all 5 results into a table, calculate and store the average of these 5 results.

NOTE: for some data shapes/pivot picking combo, 1 million may be too much data (ie it takes way too long.) If you find that you waited for 5 seconds (for a single sort call) and it still hasn't finished(or it crashes because recursion caused a stack overflow).

Only rule is that you can't compare apples to oranges, so if for a particular data shape, a pivot picking algorithm requires you to use a smaller set of data, then you need to use that same data size for the other 3 pivot picking algorithms also.

HINT: you can use the main as is, one call at a time or you can change the main to actually output the results in a list or even a CSV file ... takes a bit more coding but it might actually make your life easier in the long run

Generate graphical data:

Take the data you generated and create a vertical bar graph (bars go up and down) using a spreadsheet program such as google spreadsheet or excel. The bar graph should be organized as follows:

- the bar graph should have 20 bars in total
- organize the data into 5 groups. Each group representing a data shape
- within each group, graph the average time for each pivot picking algorithm
- provide clear labelling, titles and a legend

Write a discussion about your data

Write a one paragraph discussion about the results of your experiment. In your discussion be sure to discuss:

- whether any pivot picking algorithm stand out (either good or bad) in particular.
- if you had to choose a pivot picking method, which would you choose and why.

Submitting your lab

In order for your lab to be considered complete you must submit:

- A test verification
- Your discussion paragraph.
- A wiki page containing the graphs. The graphs should be embeded (ie you shouldn't have to click anything to see the graph, it should be right there). This can be accomplished by either exporting the graph as an image and then linking the image into your wiki or if you are using google docs, you can get a link to the

graph by publishing it (use the ellipse in top right of graph, get a link (doesn't need to be iframe, just an url will do) then simply use that in a markdown image tag on github

Submit a test verification

To submit your program please run the following command in the same directory as your lab5 source on matrix:

```
~catherine.leung/submit dsa555-L5
```

This step only verifies that you have compiled and tested your program on matrix.

Submitting your code on github

- ensure that you have added and committed all your code to lab5working
- push your code into github lab5working (git push origin lab5working)
- On github, perform the pull request to merge lab5working into master
- on matrix, checkout your master branch (git checkout master)
- on matrix, pull your repo (git pull origin)
- **DO NOT ZIP YOUR FILES PLEASE!**
- **DO NOT PUT YOUR WORK INTO MORE SUBFOLDERS. lab5.cpp should be in lab5 folder.. that's it**

All branches should be pushed to your private DSA555 github repos (These were provided by completing lab 0). **DO NOT** put it in your own public repo as **that would violate academic policy (cheating)**