Lab 6

Due March 20 - draft, specifics will change soon.

This lab is the second of two labs that will perform experimentation on various aspects of quicksort. In this second lab, you will look at measuring improvements on an algorithm.

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
- lab6.cpp
- lab6tester.cpp
- lab6main.cpp

Setup

- Create a branch called lab6working
- In that branch, create a folder called lab6
- 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, if you haven't done this yet, please watch (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

lab6.cpp

you will find the quickSort() function. In this lab there are two versions of quicksort.

- quickSortStandard()
- quickSortModified()

quickSortStandard() is the standard quicksort function, you won't be modifying this function other than to put in your pivot picking method. It will be used to generate times that you can use to compare against your other data.

quickSortModified() is the function that you will modify (more on this later) to see how performance is affected.

lab6tester.cpp

This is a simple program that will ensure that your sorted array is behaving as expected (you didn't create a bug when altering the code). compile your lab6.cpp with this tester to ensure your code works after you alter it.

The experiment:

Our experiment has to do with the value of threshold.

To begin, we will establish a base line set of timings. Using an array 1 million elements, run quicksortStandard() with your chosen pivot picking algorithm and use the following threshold values: 16, 32, 64, 128, 256, 512, 1024, and 2048, 4096. Perform this experiment 5 times, use the average of the times when plotting results.

In Andrei Alexandrescu, CppCon 2019, "Speed is Found in the Minds of People", Andrei Alexandrescu discusses 2 alterations to the insertionsort portion of quicksort. One that performed worse (perform binary search for location then insert) and one that performed better (perform makeheap() on partition then insertionsort). Choose 1 of these alterations and modify your insertionsort accordingly. Test to see if you see similar changes in performance. Please note that if you wish to alter it with some of the other suggestions in the video, you are free to do so, please make sure to test using the tester.

Using the the altered form of your insertionsort, (but keeping the same pivot picking algorithm and array size), get the times for sorting a million items with threshold values: 16, 32, 64, 128, 256, 512, 1024, and 2048, 4096. Perform this experiment 5 times, use the average of the times when plotting results.

Experiment and gather results

Choose ONE pivot picking method that worked well for all data shapes (based on observations from lab5). If you did not do lab 5, use the random pivot picking method.

- Using the original algorithm, gather timing with threshold set at: 16, 32, 64, 128, 256, 512, 1024, and 2048, 4096.
- Choose ONE of the alterations from the video and implement it. Gather timing with threshold set at: 16, 32, 64, 128, 256, 512, 1024, and 2048, 4096
- 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.

Generate graphical data:

Take the data you generated and create a line graph showing the performance of the two algorithms as threshold increases. Thus, the value of threshold is on the x axis, the time on the y axis.

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:

- which alteration you did for the insertion sort?

- was the performance better or worse and by how much?
- should the threshold be different depending on how insertion sort is done

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 lab1 source on matrix:

```
~catherine.leung/submit dsa555-L7
```

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 lab6working
- push your code into github lab6working (git push origin lab6working)
- On github, perform the pull request to merge lab6working 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. lab6.cpp should be in lab6 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)**