CS 3353 Spring 2024

Program 1

Quicksort

Due: 2/19 (Mon) 11:59pm, with late pass deadline 2/22 (Thu) 11:59pm

The goal of this program is for you to implement a couple of variation of QuickSort and to test its behavior.

Class Stuff

For this program, you are to sort the object of a class called Stuff. The class has the following definition:

- Members:
  - int a, b, c: integers
  - static int compareCount: return the number of times the compare operator (see below) is called.

  (Notice that all members (except compareCount) is private).

- Constructor
  - Stuff(bool israndom = true, int x = 0, int y = 0, int z = 0): If isRandom is false, the three integers that passed to a, b, and c. However, if isRandom is true, then x, y, z will be ignored, and random values will be generated for a, b and c.
  - Stuff(const Stuff&): copy constructor
- Overloaded operators:
  - bool operator<(const Stuff& s): return true if the object is less than the object s. Notice that I may change the definition of this method after you hand in the program, but your program should work as long as the method is correctly defined. (i.e. given two stuff object x and y, either (x < y) or (y < x) is true, but not both). Also, each time the operator is called, compareCount is incremented by 1.
  - friend ostream& operator<<(ostream& os, Stuff& s): output the content of a stuff object
  - friend istream& operator>>(istream& os, Stuff& s): read the value of a stuff object

You are given 2 files:

1. Stuff.h : contains the declaration of the SortClass class
2. Stuff.cpp : contains implementation of all the methods (and overload operators) of the SortClass class

Things to do

Part 1: Implementation of Quicksort

You are to implement the following functions

Part 1a: “Vanilla” QuickSort

You are to implement the following function

- void quickSort(vector<Stuff>& s): Implement quicksort to sort the vector s ***IN DESCENDING ORDER.***
    - s: the vector to be sorted

Notice that you should implement quicksort() as an standalone function, not a member of a separate class.

You should use the algorithm described in class, down to how the partition function is implemented. Please remember that in the slides your array starts at location 1, and if this code, the vector starts at location 0.

Part 1b: QuickSort with some choice of pivot

You are to implement the following function:

- void quickSort2(vector<Stuff>& s): Implement quicksort to sort the vector s ***IN DESCENDING ORDER.***
    - s: the vector to be sorted

The only difference from Part1b is that in the partition function, if the vector has at least 3 elements, you will look at the last 3 elements, and pick the middle one as the pivot. You need to modify the partition function to do that. If the vector has 2 elements, then just sort it by doing one comparison. If the vector has 0 or 1 elements, then do nothing.

Part 1c: stop the recursion early

You are to implement the following function:

- void quickSort3(vector<Stuff>& s, int k): Implement quicksort to sort the vector s ***IN DESCENDING ORDER.***
    - s: the vector to be sorted
    - k: if the vector has k or fewer elements, then stop the recursion and apply insertion sort to sort the vector. (You will need to implement insertion sort yourself).

Your partition function should be the same partition function as of part 1a.

Part 1d; stop the recursion early (part 2)

- void quickSort4(vector<Stuff>& s, float p): Implement quicksort to sort the vector s ***IN DESCENDING ORDER.***
    - s: the vector to be sorted
    - p: should be a floating point number between 0 and 1. if the number of elements in the vector is less than int(p * (number of elements of the original vector)), then stop the recursion and use insertion sort to sort the vector. If p is > 1, then set p to 1, if p is < 0, then set p to 0.

The easiest way to do this is to call quickSort3 with the right parameter.

I have written a driver program quickTest.cpp which you can use to test the correctness of the algorithm.

Part 2: Analysis

You are to analyze the various versions of quicksort by the following:

1. Generate 500 different sets of input of n=1000
2. For each case, apply quickSort(), quickSort2(), quickSort3() and quickSort4() to sort it. For quickSort3() consider cases where k is 10 and 100. For quickSort4() consider cases where p = 0.01, 0.1, 0.25.
3. Record the number of comparisons for each case.
4. Calculate the average number of comparisons for the 500 cases for each of the 7 cases. Also calculate the standard deviation. In addition record the maximum number of comparisons over all 500 sets
5. Repeat step 1-4 for n = 3000, 5000, 7000, 9000, 11000, 13000, 15000
6. Use a table to record the average/standard derivation of number of comparisons of each case for each different n.
7. Plot four graphs
    - For graph 1, the x-axis is n, and the y-axis is the average number of comparisons over the 500 sets. You should plot 7 lines, each line denotes the change of the number of comparisons for the seven cases (quicksort(), quickSort2(), 2 cases of quickSort3() and 3 cases of quickSort4())
    - Graph 2 is the same, with the only exception that the x-axis is log(n) and the y-axis is log(average number of comparisons) – you can use either base 2 or base 10.
    - Graph 3 and 4 corresponds to graph 1 and 2, but instead of using the average. Ise the ,axo,l,
8. Use the information you collected and the graphs to compare the efficiency of quicksort for the four cases – both which one run fastest, and which one is the most efficient asymptotically.

Notice that for your program to be able to generate random numbers, your main program should start by calling srand(time(0)) to seed the random number generator.

What to hand in

You should hand in the following 2 files (and 2 files ONLY):

- QuickSort.cpp : a file that contains your four versions of quickSort implementation. DO NOT contain any main() program. You can implement any extra functions necessary if you want, and include in that file. But be aware when I test the program the only function I will call is quickSort()., quickSort2(), quickSort3(), quickSort4().
    - Also please make sure the function name and parameters match exactly (e.g. do not name your function quickSort(…)).

    - You are welcomed to modify anything that I send you, but for grading purposes, I will use the original version of the files that I provide.
    - You do NOT need to send me the file that contains the main() function.
- Analysis.pdf : a pdf file that store your table and graphs, plus 1-2 paragraphs for your analysis.

You should have a zip file that contains both files and upload it to Canvas.