CS 3353 Fall 2023

Program 1

Shell sort

Due: 9/22 (Fri) 11:59pm, with late pass deadline 9/25 (Mon) 11:59pm

The goal of this program is for you to implement shell sort and analyze its run time.

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

You are to implement the following function

- void ShellSort(vector<Stuff>& s, int code): Implement shellsort to sort the vector s in stuff ***IN DESCENDING ORDER.***
  - s: the vector to be sorted

- code: an integer denoting how the hlist array (as in the source code) is to be formed:
    - 0: hlist should be [1] (i.e. essentially insertion sort)
    - 1: hlist should be [k^2, $(k-1)^2$, …, 4, 1] where k is the maximum number such that k^2 is still (strictly) smaller than the number of objects to be sorted. (Notice that in this case 1 = 4*1 - 3)
    - 2: hlist should be [2^k, $2^{(k-1)}$, $2^{(k-2)}$, … 2, 1] where k is the maximum number such that 2^k is still (strictly) smaller than the number of objects to be sorted. (Notice that in this case $1 = 2^1 - 1$)
    - 3: hlist should be [2^k-1, $2^{(k-1)}$-1, $2^{(k-2)}$-1 … 3, 1] where k is the maximum number such that $2^k – 1$ is still (strictly) smaller than the number of objects to be sorted. (Notice that in this case $1 = 2^1 - 1$)
    - 4: hlist should be [$4^{k+1} + 3 * 2^k + 1$, … 8, 1], where k is the maximum number such that $4^{k+1} + 3 * 2^k + 1$ is still (strictly) smaller than the number of objects to be sorted. If there are 8 or fewer numbers in the array to be sorted, the list should be [1].
    - If the code is anything either than 0,1,2,3,4 you should print an error message and exit the program immediately.

Notice that you should implement ShellSort as an standalonet function, not a member of a separate class.

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

Part 2: Analysis

You are to analyze the various of option of shell sort by the following:

1. Generate 100 different sets of input of n=1000
2. For each case, apply ShellSort for each of the four cases (code = 0, 1, 2, 3, 4)
3. Record the number of comparisons for each case.
4. Calculate the average number of comparisons for each of the four cases. Also calculate the standard deviation.
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
7. Plot two graphs
    - For graph 1, the x-axis is n, and the y-axis is the average number of comparisons over the 100 sets. You should plot 4 lines, each line denotes the change of the number of comparisons for code = 0, 1, 2, 3 respectively
    - 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.
8. Use the information you collected and the graphs to compare the efficiency of shell sort 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):

- ShellSort.cpp : a file that contains your shellsort 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 ShellSort().
    - Also please make sure the function name and parameters match exactly (e.g. do not name your function shellSort(…)).
    - 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 contain 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.