Project 4: Shellsort Investigations

October 29, 2018

1 Description

Evaluate several increment sequences to be used in Shellsort. From the sequences required, the time required to sort the data set will be averaged over some number of runs. The number of runs will be a variable that a user can enter from the command line when starting the program.

2 Problem Statement

Write a driver to compare performances of increment sequences for Shellsort. Use the following increments:

The following three input sequences are to be used to perform the Shellsort comparisons. The increments you use will start with a larger increment, with the subsequent increments getting smaller. For example, although sequence 2 is shown as progressing in the sequence 1,8,23,77, 281, etc., the first increment you choose will be based on the number of elements in your data, N. If $N = 300$ items, start you increment at the first value below $N/2$, or here, since $N/2 = 150$, the sequence of values you should use are the values 77, 23,8,1.

1. `h[]` $= 2^i$: $\{1, 2, 4, 8, 16, \ldots\}$
2. `h[]` $= 4^i + 3 \cdot 2^{i-1} + 1$: $\{1, 8, 23, 77, 281, \ldots\}$
3. Successive terms of the form $2^p 3^q$: $\{1, 2, 3, 4, 6, 8, 9, 12, \ldots\}$
4. Successive terms of the form $2^i - 1$: $\{1, 3, 7, 15, 31, 63, \ldots\}$

Your program must take its input from the command line:

- the name of the file containing the data
- the number of trials to be performed with each sequence (so that an average time can be determined)

The runs are then performed and the average run times from each sequence along with the ratios (see below) are output to the console window.

Read in files of various random items and determine the time to complete the sort routine for each sequence. Your program should be able to sort Comparable items in general, e.g., Integers, Doubles, Strings, etc.

You are required to create a driver program that acts as a client which calls ShellSort.java:

```
// test client
    public static void main(String[] args) {
  // Get the file name of the data set, and the number of runs
  // for the data set from the command line


  // Read in the file of N random Doubles from the command line and
  // store in the array  data[]


  //Build increment sequence arrays for this data set
  // Determine the upper value of array based on h[k] <= 0.5*N
  //  h1[],  h2[], h3[], h4[]
  //  h1 is the increment sequence: 1,2,4,8,16,32,...2^i
  //  h2 is the increment sequence: 1,8,23,77,281,...4^i+3*2^{i-1} + 1
  //  h3 is the increment sequence: 1,2,3,4,6,8,9,12,...2^p*3^q
  //  h4 is the increment sequence: 1,3,7,15,31,63,...2^i - 1


  // sort the array of N items with each sequence j trials, timing each:
  ShellSort study = new ShellSort( data );
  start = System.nanoTime(); study.sortUsing( h1 );
  duration1 = System.nanoTime() - start;

  start = System.nanoTime(); study.sortUsing( h2 );
  duration2 = System.nanoTime() - start;

  start = System.nanoTime(); study.sortUsing( h3 );
  duration3 = System.nanoTime() - start;

  start = System.nanoTime(); study.sortUsing( h4 );
  duration4 = System.nanoTime()- start;

  // display results for an average generated from at least
}
```

Note:

- Develop a summary report
- Discuss results
- Be sure to NOT print out the entire data set (1 million items possible)

3 Example Code Layout

```
/*
 This class simulates the ShellSort algorithm.  After an object of
 type ShellSort is created, the sortUsing() method is called to sort
 the data of the ShellSort object using the ShellSort algorithm.
*/

public class ShellSort
{
  private Comparable[] data;

  public ShellSort( Comparable[] x )
  {
    data = new Comparable[x.length];
    for(int i=0; i < x.length; i++)
      this.data[i] = x[i];
  }

  public void sortUsing( int[] h)
  {
    //your code
  }

  private boolean less(Comparable v, Comparable w)
  {   return (v.compareTo(w) < 0); }

  private void exch(Comparable[] data, int i, int j)
  {   Comparable t = data[i]; data[i] = data[j]; data[j]=t; }

  /* . . .  */
}
```

4 Running Trials

```
ShellSort study = new ShellSort(data);

while( j < numberOfTrials)
{
  start = System.nanoTime();
  study.sortUsing(h1);
  duration1 += (System.nanoTime() - start);

  start = System.nanoTime();
  study.sortUsing(h2);
  duration2 += (System.nanoTime() - start);

  start = System.nanoTime();
  study.sortUsing(h3);
  duration3 += (System.nanoTime() - start);

  start = System.nanoTime();
  study.sortUsing(h4);
  duration4 += (System.nanoTime() - start);

  study = new ShellSort(data);
  j++;
}
```

5 Input/Output

Command Line Inputs:
Input 1: name of file containing the data to be sorted
Input 2: number times each file will be sorted in order to determine the average

If the name of the file containing the `main()` method is in `Demo.java`, it will be compiled using `javac Demo.java`, producing `Demo.class`. Demo.class will be run in the following manner to compile the list of 10^3 decimal fractions found in 1000input.txt:

```
C:\java.exe RunShells  1000input.txt  5
```

Console Output:

```
The number of trials is 5

seq 1:
 256 128 64 32 16 8 4 2 1

seq 2:
 281 77 23 8 1

seq 3:
 486 432 384 324 288 256 243 216 192 162 144 128 108 96 81 72 64 54 48 36 32 27 24 18 16 12 9 8 6 4 3 2 1

seq 4:
 255 127 63 31 15 7 3 1

 Comparisons:

Average comparisons for Sequence 1 is 23474
Average comparisons for Sequence 2 is 14533
Average comparisons for Sequence 3 is 32590
Average comparisons for Sequence 4 is 13748
-----------------------------------------
Average Time:

Average Time using Sequence h[i] = 2^i for 5 trials is             2.880055 ms
Average Time using Sequence h[i] = 4^i - 3*2^i + 1 for 5 trials is  0.700092 ms
Average Time using Sequence h[i] = 2^p*3^q for 5 trials is          2.638991 ms
Average Time using Sequence h[i] = 2^i - 1 for 5 trials is          1.792922 ms
-----------------------------------------
Ratios:

T1/T2 =      4.114 ms
T1/T3 =      1.091 ms
T1/T4 =      1.606 ms
T2/T3 =      0.265 ms
T2/T4 =      0.390 ms
T3/T4 =      1.472 ms
```

6 Rubric

```
// brad p  | 03434 | bp7675318@swccd.edu
// angleina | 2323 | aj7674542@swccd.edu

stu:

Proj 4  Shellsort:
===============

 1. read in a file of random numbers of **any** size or type from command line
 2. sort using Shellsort with 3 different increment sequences, various data sets
 3. count and report the average runtime for multiple trials, and ratios per assignment
 4. comments, white space,indentation, general structure,modularity (selection of method



 Programs that don't compile or run (or run in an infinite loop) will get 0 points.

Coding conventions:
http://www.oracle.com/technetwork/java/codeconv-138413.html

  points for above:
 -----------------
1. read from the command line:     x/1 pts
2. sort w/ various inputs:         x/5 pts
3. count and report avg runtime:   x/3 pts
4. modularity, comments:           x/1 pts



   total: /10


Input 1, avg from 2 runs follows:

Number of trials:  2

seq 1:
 256  128  64  32  16  8  4  2  1

seq 2:
 281  77  23  8  1
```

```
seq 3:
 486  432  384  324  288  256  243  216  192  162  144  128  108  96  81  72  64  54  48

seq 4:
 255  127  63  31  15  7  3  1

Comparisons:

Average Comparisons for Sequence 1 is 8489
Average Comparisons for Sequence 2 is 4610
Average Comparisons for Sequence 3 is 29024
Average Comparisons for Sequence 4 is 7498
-----------------------------------------------------------------
Average Time:

Average Time Using Sequence h[1] = 25.2456815 ms
Average Time Using Sequence h[2] = 4.97113 ms
Average Time Using Sequence h[3] = 13.0859135 ms
Average Time Using Sequence h[4] = 3.833739 ms
-----------------------------------------------------------------
Ratios

T1/T2 =      5.078459324137571
T1/T3 =      1.9292257663173458
T1/T4 =      6.585133077656042
T2/T3 =      0.37988406388289203
T2/T4 =      1.296679299242854
T3/T4 =      3.4133553431780306
  ----------------

Instructor Comments:
====================

Program console output appended below:
======================================
```