# Benchmarking Sorting Algorithms

Summary: In this homework, you will be implementing code to generate test data and benchmark sorting.

## 1 Background

In this assignment you will practice predicting and verifying the impact of data "nature" on the run-time of sorting algorithms. As we have seen with algorithms like insertion sort, the runtime (even worse case like Big-Oh) can be impacted by the nature of the input.

To do this, we will create three different types of input data, that may give different results when sorted. Two sorting algorithms will then be benchmarked on these three types of data. Each algorithm will be run twice, for different dataset sizes, in order to get times that we can use to apply the doubling formula. (see slide 23: Modeling Small Datasets) in the Analysis of Algorithms slide deck for details on the doubling formula.) The doubling formula is $lg\frac{T(2N)}{T(N)} = b$. If we compute the formula, then we will be able to figure out the algorithm's Big-Oh for a particular type of input data, since they will be $O(n^b)$. $b$ is simply the power.

This document is separated into four sections: Background, Requirements, Testing, and Submission. You have almost finished reading the Background section already. In Requirements, we will discuss what is expected of you in this homework. In Testing, we suggest some basic tests you use to start to verify your implementation. Lastly, Submission discusses how your source code should be submitted on Canvas.

## 2 Requirements [36 points]

For this assignment you will be writing code to generate test data and benchmark sorting algorithms on it (edited from Sedgewick and Wayne: 2.1.36). Already provided for you is BaseNonUniform.java which contains the existing sorting algorithms. This class extends an interface called SER222_02_01_HW02_Submission which defines all of the methods that you need to create. You may create additional private helper methods. First, write a series of methods to generate test data that is "non-uniform":

- Implement the method *public Integer[] generateTestDataBinary(int size).* Half the data is 0s, half 1s. For example, an input of length 8 might look like [0, 0, 0, 0, 1, 1, 1, 1]. See the interface. [4 points]
- Implement the method *public Integer[] generateTestDataHalfs(int size).* Half the data is 0s, half the remainder is 1s, half the reminder is 2s, half the reminder is 3s, and so forth. For example, an input of length 8 might look like [0, 0, 0, 0, 1, 1, 2, 3]. See the interface. [9 points]
- Implement the method *public Integer[] generateTestDataHalfRandom(int size).* Half the data is 0s, half random int values (use nextInt() from Java's Random package). For example, an input of length 8 might look like [0, 0, 0, 0, 54119567, 4968, -650736346, 138617093]. See the interface. [4 points]

Each of these three techniques should be implemented as a method that takes an int representing the size of a dataset, and returns an Integer array containing that number of elements generated with the corresponding rule. Do not randomize (shuffle) the contents of the generated arrays.

Using the three methods you implemented, develop and test hypotheses about the effect of input on the performance of insertion sort and shellsort. The code for these methods is already included in the base file for this assignment.

- In a separate document (to be submitted as a PDF), write up your hypotheses (3 per algorithm): describe what you think the running time will look like ($O(n)$? $O(n^2)$? $O(n^3)$?) on each data set, and explain briefly why you think that. As long as your ideas make sense, **and you do the analysis prior to benchmarking**, you will receive full credit on the hypotheses. There will be a separate submission link on Canvas. [5 points]

For each of the sorting algorithms, your program should run them on the three types of test data. Test them with datasets size of 4096 and 4096*2. (If your system is so fast that you don't get good results, you may increase the dataset size.) Time each of the tests with the Stopwatch class discussed in class. The program needs to compute the result of the doubling formula on the run times from the 2048 and 4096 result pairs to get the power ("b") for that algorithm on that type of input, and then display it. Six different values should be shown if you have properly implemented all of the benchmarks.

- Implement the method *public double computeDoublingFormula(double t1, double t2).* See the interface for more information. [12 points]
- Implement the method *public double benchmarkInsertionSort(Integer[] small, Integer[] large).* See the interface for more information. [4 points]
- Implement the method *public double benchmarkShellsort(Integer[] small, Integer[] large).* See the interface for more information. [4 points]
- Implement the method *public void runBenchmarks(int size).* See the interface for more information. Hint: should call the two benchmark methods above. The output should look like below. [4 points]

| | Insertion | Shellsort |
|---|---|---|
| Bin | #.### | #.### |
| Half | #.### | #.### |
| RanInt | #.### | #.### |

# 3 Testing

The main functionality to test is the methods that generate test data. You will want to run them multiple times, on different sizes, and display their output. Check that the output matches the patterns required above. There isn't much else to test for this homework. The algorithms you are benchmarking have already been tested for correctness. Optionally, you may want to try giving an input that is pure random numbers to each of the algorithms and checking if your doubling formula code gives the algorithms expected Big-Oh.

# 4 Submission

The submission for this assignment has one part: a source code submission. The file should be attached to the homework submission link on Canvas.

**Writeup:** For this assignment, no separate write up is required.

**Source Code:** Please name your main class as "LastNameNonUniform.java" (e.g. "AcunaNonUniform.java"). The class must be in the default package. For grading purposes, the source file you submit will be added to a project containing two files: SER222_02_01_HW02_Submission.java, and Stopwatch.java. Do not submit these files, only your LastNameNonUniform.java.