Beyond Big O: Teaching Experimental Algorithmics*

Michael Shindler Michael T. Goodrich Ofek Gila
Michael Dillencourt
University of California, Irvine
{mikes, goodrich, ogila, mbdillen}@uci.edu

Abstract

We present a supplement to traditionally-taught topics with experimental explorations of algorithms.

1 Introduction

Algorithms courses are traditionally taught with an emphasis on general design techniques (like divide-and-conquer and dynamic programming) and the formal analysis of algorithms, e.g., see [8, 10, 14]. This is a valuable part of Computer Science education, which we are not proposing replacing. Instead, we are proposing here how one might supplement traditional algorithms instruction with projects or even an entire course in *experimental algorithmics*.[1]

Experimental algorithmics [18] studies the design, implementation, and experimental evaluation of algorithms and data structures. This topic has its own journal (JEA), which began in 1996 [19], its own U.S. conference (ALENEX), which began in 1999 [9], and its own European conference (SEA), which began in 2003 [11]. Experimental algorithmics can provide insights into the performance of algorithms that go beyond formal analysis, to address issues such

[1]See `https://www.ics.uci.edu/~goodrich/teach/cs165/` for a syllabus, schedule, lecture slides, readings, and detailed projects for a recent offering of our course.

as "real world" running times, the size of constant factors, and how well an algorithm achieves various optimization goals. Algorithm engineering and experimentation is also a useful skill for practicing computer scientists, as it can lead to better theoretical analysis and it can also demonstrate where such analysis is misleading, e.g., see [25]. Further, students using formal analysis alone may not fully appreciate how asymptotic running times manifest themselves in the real-world performance of algorithms. We describe in this paper methodologies for supplementing traditional algorithms instruction with modules or even an entire course in experimental algorithmics.

Our methodologies and modules for experimental algorithmics address the issues of discovering an algorithm's real-world running time and how well algorithms achieve optimization goals. Our course also covers the use of algorithms to test models of the real world; this last question is covered in the appendix.

A common and recurrent theme throughout all of our projects is the use of log-log plots as an analysis tool; hence, a side benefit of our projects is that students gain a deeper understanding and appreciation for log-log plots, both in general and in the experimental analysis of algorithms.

1.1 Related Work

Experimental algorithmics is a research field that is too deep to review here. In terms of general background, we refer the interested reader to the textbook by McGeoch [18] or the guide by Johnson [12], as well as past proceedings for ALENEX and SEA, or past issues of JEA. For a review of the related area of *algorithm engineering*, please see [24].

There has also been previous work on integrating experimental analysis. Ángel Velázquez-Iturbide and Debdi [1] describe a set of projects for experimenting with greedy algorithms, but they do not address their asymptotic analysis using best-fit functions. Berque *et al.* [5] describe a workbench, which they call KLYDE, for experimental algorithm analysis. Their system includes a tool for asymptotic-time analysis but does not include best-fit asymptotic functions or the analysis of other performance variables. Matocha [17] describes a set of projects that emphasize technical writing and the scientific method while also addressing experimental algorithm analysis but does not stress best-fit functions for analysis purposes. Sanders [23] describes his experiences in teaching empirical analysis of algorithms, focusing primarily on running time analysis, but he also does not include best-fit functions for asymptotic running times. In addition, there has also been previous work on introducing experimental algorithm analysis earlier in the Computer Science curriculum, such as for CS1 or CS2 courses. Such courses, however, either have a limited focus on performance (see [2]) or mainly analyze algorithms with different algorithmic complexity experimentally (see [13] and [26]). These courses don't empha-

size the real-world applications of algorithm design as much, e.g., they don't highlight the importance of constant factors, input distributions, heuristic optimization algorithms, etc.

1.2 Our Contributions

We present methodologies that supplement traditionally-taught algorithms topics with experimental explorations of algorithms. Such supplementation can either be as a part of a traditional algorithms course or as an additional project-oriented course. Our course required students to implement a variety of algorithms, run benchmarks, plot their results, compare alternative implementations, and draw conclusions. In particular, we provide projects that address the three distinct questions of algorithm analysis mentioned above.

We feel that the skills learned and practiced in these projects should appeal to a variety of Computer Science students with varying desired career paths.

In this paper, we provide two project types. We report on our experiences about how well students achieved the learning outcomes for the specific projects. These projects can be adjusted for the target audiences at other universities.

2 Project 1: Running Times

In the first type of project, students are asked to implement several sorting algorithms, of which students are expected to have seen only one or two prior to doing this project. For example, Bubble and Insertion sort can be chosen as examples students have seen, and parameterizations of Spin-the-bottle sort and Shellsort can be used as ones the students may not have seen.

All of these algorithms are relatively easy to program—the goal here is not to test students' ability to implement complicated algorithms. Instead, we wish to teach techniques of experimental algorithmics. Students are asked to experiment with each algorithm using ever increasing input sizes and various arrangements. Students are then asked to plot their results on a log-log scale in order to find expected running times and find a best-fit line equation for their data. An example set of charts that were produced empirically for this project are shown in Figure 1. This set of experiments shows an example of a student who empirically determined expected running times close to $O(n^2)$ for bubble sort, insertion sort, and spin-the-bottle sort using uniform random inputs, and close to $O(n^{3/2})$ and $O(n^{6/5})$ for two versions of Shellsort. It also shows that for almost-sorted inputs, insertion sort runs in almost linear time—much faster than its worst-case bound (consistent with its $\Theta(n+I)$ time complexity, where I is the number of inversions)—but bubble sort and spin-the-bottle perform consistent with their respective bounds. Therefore, a student with such data

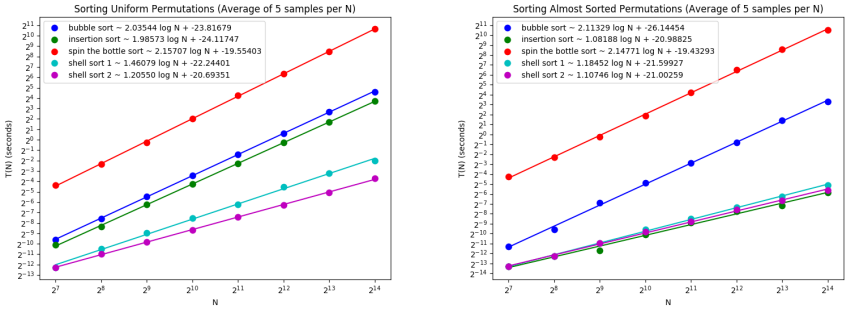

Figure 1: Project 1 charts.

should also learn a lesson as to how input distributions can impact real-world running times.

Further, we feel that this project serves as a good introduction to experimental algorithmics. The students likely know the "correct answer" to at least two of the algorithms (Bubble Sort and Insertion Sort), and this can serve as a "sanity check" for their early work.

Alternative choices and options for this type of project include the use of other sorting algorithms or other input distributions. One could also use a different problem, such as selection, connected components in a graph, minimum spanning trees, or big-integer multiplication. The only requirement is that there be multiple algorithms with varying running times for solving the same problem.

3 Project 2: Optimization

The second type of project deals with empirically testing how well algorithms achieve an optimization goal. From a pedagogical point of view, there are a number of challenges to overcome for this project. An ideal project addresses a non-trivial (NP-hard) optimization problem for which multiple heuristic algorithms exist and are easy to program. Furthermore, the relative quality of solutions must be efficiently computable.

One example is to have students test heuristic algorithms for bin packing, where we are given a set of items with (normalized) sizes between 0 and 1 and asked to pack them into as few bins of size 1 as possible without overflowing any bin [7]. Rather than have the student empirically compare algorithms based on how close they get to the optimal number of bins (which is computationally difficult to determine), we instead recommend using a *waste* parameter, which

is the number of bins used by an algorithm minus the total size (i.e., the sum) of all the items. We had students implement the first-fit and best-fit heuristics, both in an online setting and in the setting in which we can first sort elements in decreasing order of size. Students implemented each and measured the waste produced by each on randomly generated item lists of various sizes.

An outcome chart for a set of such experiments is the following:

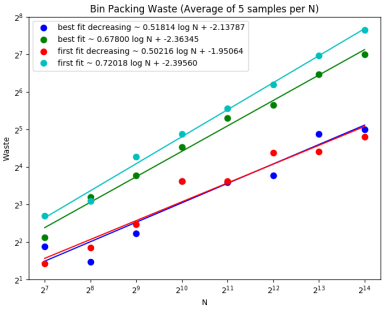

Figure 2: Bin packing waste plots.

Thus, this data confirmed that best fit decreasing and first fit decreasing both empirically achieve an an asymptotic waste that is approximately $O(n^{1/2})$, with their unsorted versions empirically achieving asymptotic waste bounds that are approximately $O(n^{2/3})$.

Alternatives for this type of project include changing the input distribution or exploring a different optimization problem.

A third project, focusing on modeling real-world phenomena, has been omitted for space. Details for that will be made available in an online appendix.

4 Observations on Student Performance

In this section, we report on our observations of how well students performed the above tasks in a recent offering of our course incorporating the above projects. The course consisted primarily of seniors. We required expositions of the algorithms studied, plots of performance, and conclusions comparing the relative performances. Central to this plotting requirement was the use of log-log plots. The report was graded on having log-log plots with appropriate regression analysis, clarity and conciseness in describing algorithms, gathering an appropriate amount of data, and clear prose analysis of results.

Our desired learning objectives for the students, by project, along with its success rate in our pilot offering, is as follows. The data follows the entire

enrollment of the class of 103 students. Each student was given the option to opt out of being included in our study, and none elected to do so.

1. **Students should be able to use meaningful, random sample inputs and interpret running time data on this data by noticing which types of input produce better running times for different algorithms.** The comparison of different sorting algorithms' running times tested on almost-sorted and on uniformly-permuted random inputs should get students thinking about this consideration. Based on their reports, over 80% of the students achieved this learning outcome.

   Students should be able to use log-log plots and a best-fit line to analyze running-time growth. Based on their reports, over 90% of the students in the course met this objective.

2. **Students should understand how algorithms can be used to optimize objective functions.** Because of a prerequisite for our course, students should have had experience with NP-hard optimization problems prior to our course. To assess this learning outcome, we checked to see whether students were correctly describing the measurement of waste by their algorithms. Based on their reports, over 85% of students achieved this learning outcome.

   Students should be able to compare heuristic algorithms based on how well they optimize an objective function. We measured this by determining whether students discussed the relative performance of the different bin-packing heuristics, including whether they commented on how the best-fit and first-fit heuristics performed better on pre-sorted inputs than on unsorted inputs. Based on their reports, 77% of the students in the class achieve this learning outcome.

5 Conclusion

Based on our experiences, we believe that our projects, or similar projects, are useful supplements to the topics taught in a traditional algorithms course. Further, our project-based course appears to be a useful addition to any Computer Science curriculum that includes project-based courses as requirements for graduation (as is done at our institution) or as electives.

References

[1] J. Ángel Velázquez-Iturbide and O. Debdi. "Experimentation with optimization problems in algorithm courses". In: *2011 IEEE EUROCON - International Conference on Computer as a Tool*. Apr. 2011, pp. 1–4. DOI: 10.1109/EUROCON.2011.5929294.

[2] Doug Baldwin and Johannes A. G. M. Koomen. "Using Scientific Experiments in Early Computer Science Laboratories". In: *SIGCSE Bull.* 24.1 (Mar. 1992), pp. 102–106. ISSN: 0097-8418. DOI: 10.1145/135250.134532. URL: http://doi.acm.org/10.1145/135250.134532.

[3] Albert-László Barabási and Réka Albert. "Emergence of Scaling in Random Networks". In: *Science* 286.5439 (1999), pp. 509–512. ISSN: 0036-8075. DOI: 10.1126/science.286.5439.509. eprint: https://science.sciencemag.org/content/286/5439/509.full.pdf. URL: https://science.sciencemag.org/content/286/5439/509.

[4] Vladimir Batagelj and Ulrik Brandes. "Efficient generation of large random networks". In: *Physical Review E* 71.3 (2005), p. 036113.

[5] Dave Berque et al. "The KLYDE Workbench for Studying Experimental Algorithm Analysis". In: *Proceedings of the Twenty-fifth SIGCSE Symposium on Computer Science Education*. SIGCSE '94. Phoenix, Arizona, USA: ACM, 1994, pp. 83–87. ISBN: 0-89791-646-8. DOI: 10.1145/191029.191065. URL: http://doi.acm.org/10.1145/191029.191065.

[6] Marco Bressan et al. "Counting Graphlets: Space vs Time". In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. WSDM '17. Cambridge, United Kingdom: ACM, 2017, pp. 557–566. ISBN: 978-1-4503-4675-7. DOI: 10.1145/3018661.3018732. URL: http://doi.acm.org/10.1145/3018661.3018732.

[7] E. G. Coffman Jr., M. R. Garey, and D. S. Johnson. "Approximation Algorithms for NP-hard Problems". In: ed. by Dorit S. Hochbaum. Boston, MA, USA: PWS Publishing Co., 1997. Chap. Approximation Algorithms for Bin Packing: A Survey, pp. 46–93. ISBN: 0-534-94968-1. URL: http://dl.acm.org/citation.cfm?id=241938.241940.

[8] Thomas H. Cormen et al. *Introduction to Algorithms*. MIT press, 2009.

[9] Michael T. Goodrich and Catherine C. McGeoch, eds. *Algorithm Engineering and Experimentation (ALENEX)*. Vol. 1619. Lecture Notes in Computer Science. Springer, 1999. ISBN: 3-540-66227-8. DOI: 10.1007/3-540-48518-X. URL: https://doi.org/10.1007/3-540-48518-X.

[10] Michael T. Goodrich and Roberto Tamassia. *Algorithm Design and Applications*. Wiley Publishing, 2014.

[11] Klaus Jansen et al., eds. *Experimental and Efficient Algorithms, Second International Workshop, WEA 2003, Ascona, Switzerland, May 26-28, 2003, Proceedings*. Vol. 2647. Lecture Notes in Computer Science. Springer, 2003. ISBN: 3-540-40205-5. DOI: 10.1007/3-540-44867-5. URL: https://doi.org/10.1007/3-540-44867-5.

[12] David S. Johnson. "A Theoretician's Guide to the Experimental Analysis of Algorithms". In: *Data Structures, Near neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges* 59 (2002), pp. 215–250.

[13] Jason King. "Combining Theory and Practice in Data Structures Algorithms Course Projects: An Experience Report". In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 959–965. ISBN: 9781450380621. URL: https://doi.org/10.1145/3408877.3432476.

[14] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Pearson, 2006.

[15] Donald Ervin Knuth. *The Stanford GraphBase: a Platform for Combinatorial Computing*. ACM Press New York, 1993.

[16] Jure Leskovec and Rok Sosič. "SNAP: A General-Purpose Network Analysis and Graph-Mining Library". In: *ACM Trans. Intell. Syst. Technol.* 8.1 (July 2016), 1:1–1:20. ISSN: 2157-6904. DOI: 10.1145/2898361. URL: http://doi.acm.org/10.1145/2898361.

[17] J. Matocha. "Laboratory experiments in an algorithms course: technical writing and the scientific method". In: *32nd Annual Frontiers in Education*. Vol. 1. Nov. 2002, T1G–T1G. DOI: 10.1109/FIE.2002.1157917.

[18] Catherine C McGeoch. *A Guide to Experimental Algorithmics*. Cambridge University Press, 2012.

[19] In: *J. Exp. Algorithmics* 1 (1996). Ed. by Bernard M. E. Moret. ISSN: 1084-6654.

[20] Mark EJ Newman. "Power Laws, Pareto Distributions and Zipf's Law". In: *Contemporary Physics* 46.5 (2005), pp. 323–351.

[21] Mark Ortmann and Ulrik Brandes. "Triangle Listing Algorithms: Back from the Diversion". In: *Algorithm Engineering and Experiments (ALENEX)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2014, pp. 1–8. URL: http://dl.acm.org/citation.cfm?id=2790174.2790175.

[22] A. Rényi and P. Erdős. "On Random Graphs". In: *Publ. Math* 6 (1959), pp. 290–297.

[23] Ian Sanders. “Teaching Empirical Analysis of Algorithms”. In: *SIGCSE Bull.* 34.1 (Feb. 2002), pp. 321–325. ISSN: 0097-8418. DOI: 10.1145/563517.563468. URL: http://doi.acm.org/10.1145/563517.563468.

[24] Peter Sanders. “Algorithm engineering–an attempt at a definition using sorting as an example”. In: *2010 Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM. 2010, pp. 55–61.

[25] Richard T. Snodgrass. “On Experimental Algorithmics: An Interview with Catherine McGeoch and Bernard Moret”. In: *Ubiquity* 2011.August (Aug. 2011), 1:1–1:14. ISSN: 1530-2180. DOI: 10.1145/2015996.2015997. URL: http://doi.acm.org/10.1145/2015996.2015997.

[26] Jason Strahler et al. “Real-World Assignments at Scale to Reinforce the Importance of Algorithms and Complexity”. In: *CCSC NE* (). URL: https://par.nsf.gov/biblio/10158622.

A Project 3: Modeling the Real World

The third type of project uses algorithms to empirically measure how well various models of real-world phenomena match parameters found in actual data sets. The goal here is to further utilize performance plots as comparison tools, but now the algorithms are part of the data-gathering process. Ideally, the algorithms chosen for this type of project build upon algorithms that were taught in a prerequisite traditional algorithms course.

In our case, we chose *network science* as the application domain, that is, the study of the actual networks that arise from the Internet, social networks, computer networks, biological networks, etc. This is a rich and growing field of study, for which Computer Science has much to offer. Thus, focusing on this topic for our third type of project has the added benefit of providing useful domain knowledge for students in addition to further developing their skills in experimental algorithmics. In more detail, for this project, we focus on asking the students to design and implement algorithms that can compute the following statistics:

- **Diameter**: the length of a longest shortest path between two nodes in the graph. Students may choose to either compute this exactly or use a heuristic algorithm based on repeated breadth-first searches starting from random vertices or vertices far from the starting point of a previous breadth-first search.

- **Clustering-coefficient**: the ratio of three times the number of triangles over the number of paths of length 2 in a graph. It is used by social scientists to determine the degree to which a social network is separated into tightly-knit groups. To compute it, the students need to count the number of triangles in a graph, which is an interesting problem of growing interest in its own right, e.g., see [21].

- **Degree distribution**: for each possible degree in a graph, the number of vertices in the graph with that degree. This parameter is known to exhibit a power law [20] in many real-world networks,[2] so the goal of this component of the project is for the students to see if the graph in question has a degree distribution that exhibits a power law, by using the now-familiar log-log plot.

For reference graphs to use, one possibility is to use random graphs, which is the choice we took in our first implementation:

[2]Recall that data exhibits a power law if its frequency distribution can be characterized by a function $P(x) = cx^{-\alpha}$, for constants $c, \alpha > 0$. Typically, $2 < \alpha < 3$. See also, e.g., `https://en.wikipedia.org/wiki/Power_law`.

1. **Erdős-Rényi [22] random graphs**: these graphs, $G(n, p)$, are defined in terms of n vertices and the parameter, p, such that each pair of vertices in the graph is independently and uniformly chosen with probability p at random to form an edge. So as to avoid making the graph too dense, we recommend choosing p to be small, e.g., $p = (2 \ln n)/n$.

2. **Preferential-attachment [3] random graphs**: these graphs are defined by starting with two vertices connected by an edge and adding vertices to that at each step we add one new vertex v with a constant, d, number of edges back to previous vertices so that the probability a previously added vertex u receives a new edge from v is proportional to the (current) degree of u.

Using these definitions requires $\Theta(n^2)$ time to compute a given instance, even for sparse graphs. Fortunately, Batagelj and Brandes [4] provide simple linear-time algorithms for generating such graphs. Thus, in the final project, students generate random graphs for various numbers, n, of vertices. They then compute the diameter, clustering coefficient, and degree distribution for each graph. Students then plot average diameters and clustering coefficients for these graphs as a function of n, and students also plot the degree distribution for a specific graph instance to determine if it obeys a power law. In the former case, the growth rates tend to be so small that students are asked to plot the results using a lin-log scale, but they should still use a log-log scale for degree-distribution plots. The expected results are that Erdős-Rényi random graphs should not display a power law for their degree distributions, whereas preferential-attachment random graphs should.

An example set of charts produced from this project is as follows:

Thus, this latter chart empirically confirms a Erdős-Rényi random graph without a power law for its degree distribution and a preferential-attachment random graph with one.

Alternatives for this type of project include the following:

- Instead of using random graphs, use real-world graphs, such as found at the Stanford GraphBase [15] or SNAP [16].[3]

- Use other types of network science statistics, such as centrality measures, number of paths of length k, H-index, or numbers of small subgraphs of certain types, i.e., "graphlets" [6].

- Instead of problems in network science, choose a different application domain, such as machine learning, machine vision, or natural language processing.

[3]See also `https://snap.stanford.edu/data/`.

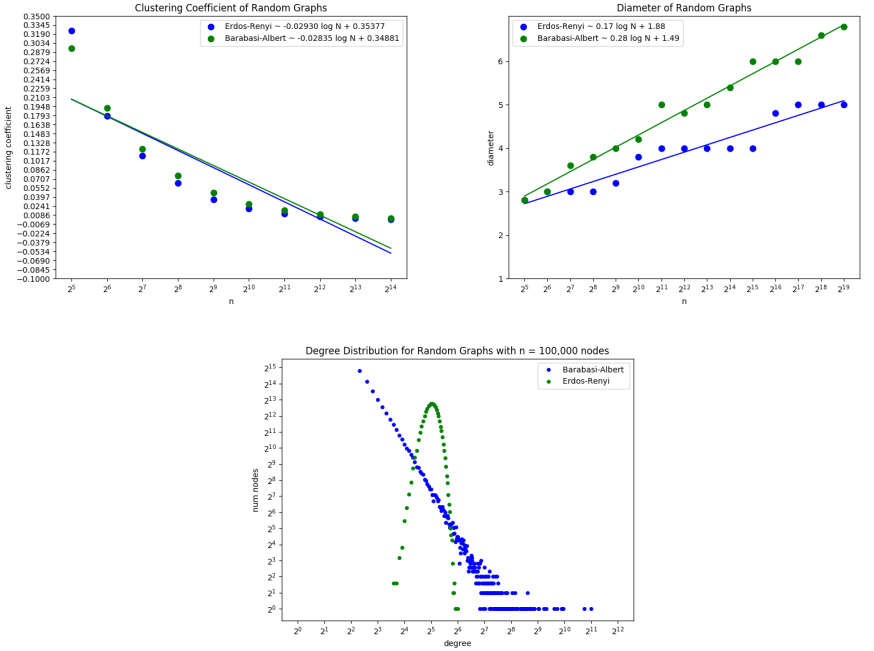

Figure 3: Project 3 charts.

A.1 Student Performance

3. (a) **Students should understand and be able to use mixed linear-log plots.** This learning outcome is a component of the third project when measuring average diameter of graphs, since, by the *small worlds* phenomenon, real-world graphs and popular random graphs tend to have small diameters. While students were told to plot this on a lin-log scale, completion of the analysis requires them to understand what the line fitting they will be doing means, as opposed to merely finding a new way to graph the output of their programs. This learning outcome also is designed to reinforce the related learning outcome for log-log plots. Based on our reading of student reports, over 90% of students achieved this learning outcome.

   (b) **Students should be able to determine whether the degree distributions of sample of graphs of increasing size exhibit a power-law distribution.** Students were taught the general concept of power-law distributions [20] and asked to determine for their random graphs, whether the distribution of degrees exhibited a power-law distribution. Based on our reading of the reports, over 80% of the students were able to successfully do this.

A.1.1 Sample Student Submissions

In Project 1, sorting algorithms, one of the questions asked of the students is to describe which sorting algorithms were most sensitive to input size and distribution, and which were least sensitive. Below are two student responses taken from their reports. In the first response, the student displays an accurate interpretation of the data. Conversely, in the second response, the student displays a clear lack of understanding of the sorting algorithm performance.

1. First student response:

   Insertion sort performs consistently worse on reverse input by a constant factor, and insertion sort performs significantly faster on almost-sorted input (the slope is much less).
   Merge sort has the same performance on all input permutations.
   All four shell sort versions have worse performance on uniform random input, and the same performance for reverse and almost-sorted input. The only exception is shell sort 3 having the same performance for all three input permutation types.
   Hybrid sort 1's asymptotic performance is most similar to insertion sort

with reverse input being slightly slower than uniform input and almost-sorted input being significantly faster. This is because a large part of hybrid sort 1 is insertion sort.
Hybrid sort 2's performance on different input permutation types are balanced. Hybrid sort 3 has equal performance in almost-sorted and reversed input, and uniform input is slightly worse. This is similar to merge sort because hybrid sort 3 uses mostly merge sort.

2. Second student response:

   Least sensitive to input size - Insertion sort
   Most sensitive to input size - Hybrid sort
   Least sensitive to distribution - Shell Sort (does not care at all)
   Most sensitive to distribution - Merge Sort (it's perfect)