Experiments in Sorting

Senior Project Final Report

Department of Computer Science

Calvin University

Authors: *Joshua Wright, Bryce Allen, Bryan Fowler*

Date: 5/13/2020

Mentor: Dr. Adams

Honors Project: No

Project Purpose:

The purpose of this project was to identify best practices for teaching sorting algorithms to new CS students.

Research Question:

Does changing the sensory feedback accompanying sorting algorithms impact the students' understanding of those sorting algorithms?

Background:

During 2019-20, senior Nate Herder built audio-visualizations for several sorting algorithms. This project built on Nate's work. Our team worked with Professor Adams to devise and run experiments using 4 groups:

1. a control group that ran traditional sorting programs, with no audio or visual enhancements.
2. an experimental group that ran sorting programs with audio enhancement.
3. an experimental group that ran sorting programs with visual enhancement.
4. an experimental group that ran sorting programs with audio and visual enhancements.

A week after the experiment, the participants were assessed on their understanding of the sorting algorithms, to see if any significant differences exist between the groups. We hoped to run the experiment both semesters, in the hopes that sufficient data will be collected to provide interesting results. However, the first semester's experiment had to be cancelled because of a critical issue with the CS lab. Our team designed the experiment, ran it, created the assessment instrument, and evaluated the results.

Experimental Design:

We set up our experiment to try and keep the four group's experiences the same, except for the variable of what they see/hear when they run the programs. The experiments were conducted in the CS Maroon and Gold labs. If multiple groups had to go at the same time, we separated them into different labs and positioned them at machines in which they couldn't see the other groups through the windows. The format for a given group in the experiment was as follows (steps are moderated by the experiment leaders to prevent any participants from looking ahead to next steps):

1. Students log into their computers, plug in headphones if needed, open up the terminal, and navigate to our directory where the files are.
    a. **/home/cs/398/2020-21/sorting/**
    b. This is the directory where our files are if you want to run them. They are called bubble_sort, insertion_sort, merge_sort, and quick_sort. You can add on the flags “a” and “g” to include audio and/or graphics respectively.
2. The pseudo code for Bubble Sort was shown on the projector.
3. The instructor walked through the algorithm and explained how it works, similarly to how a professor would explain it to a student in a computer science course.
4. We then sent out further instructions through email (so they couldn’t work ahead). These instructions had a link to a website that explained the algorithm more in depth.
5. In the email, we also told them how to run the algorithm in the terminal. For example, it would be **$./bubble_sort -ag** if they were in the audio-graphical group.
6. The participants ran the programs, studied the times it took to complete the sorting, and listened/watched the algorithms work if they weren’t in the control group.
7. The process of interaction with the programs / algorithms was timed, (10 minutes) to prevent any bias. We didn’t want the participants to be given any extra time with a certain algorithm, or else this might skew the results.
8. We repeated these steps for the other three algorithms: Insertion Sort, Merge Sort, and Quick Sort.

Note: The slides for the presentation and the quiz we sent out to the participants are in our Google Drive folder which is linked in the references section later in the report. There is also information in the references section describing how to access/run our programs in the CS lab.

Here are four algorithms running:

Sorting Algorithms (control--no audio or video)
https://youtu.be/bJF4kfACSQU

Sorting Algorithms (experimental--audio and no graphics)
https://youtu.be/BlE2LP1qj5I

Sorting Algorithms (experimental--graphics and no audio)
https://youtu.be/fRdHUjd5dcQ

Sorting Algorithms (experimental--audio and graphics)
https://youtu.be/Iy8Gk27R-ww

Design Norms:

Ethical: Honest and transparent conduction of experiments; approval from Calvin's IRB.

Stewardship: efficiency of teaching sorting algorithms leads to efficiency of implementation of algorithms. This translates to a smaller environmental footprint when the students move into the industry.

Delightful Harmony / Aesthetics: The more enjoyable the stimuli (audio, graphics) the increased likelihood of information retention.

Problems Encountered/Solutions:

Small Group Size

For our Spring experiment we had 24 participants and the group sizes were as follows:

Control: 5
Audio: 6
Graphical: 6
Audio and Graphical: 7

These small group sizes proved troublesome when we tried to draw statistically significant conclusions from our results.

File System Issue

For our Fall experiment we were a few minutes from starting, when the file system in the CS labs had a critical issue which prevented students from logging in. So we were unable to conduct the experiment.

Guacamole Audio

Remote access services did not support the TSAL and TSGL libraries needed for the C++ programs to display graphics and play audio. This meant that to participate, the experimental subjects had to be in-person, not remote. This of course played a part in our small group size problem.

Results, Analysis, and Discussion:

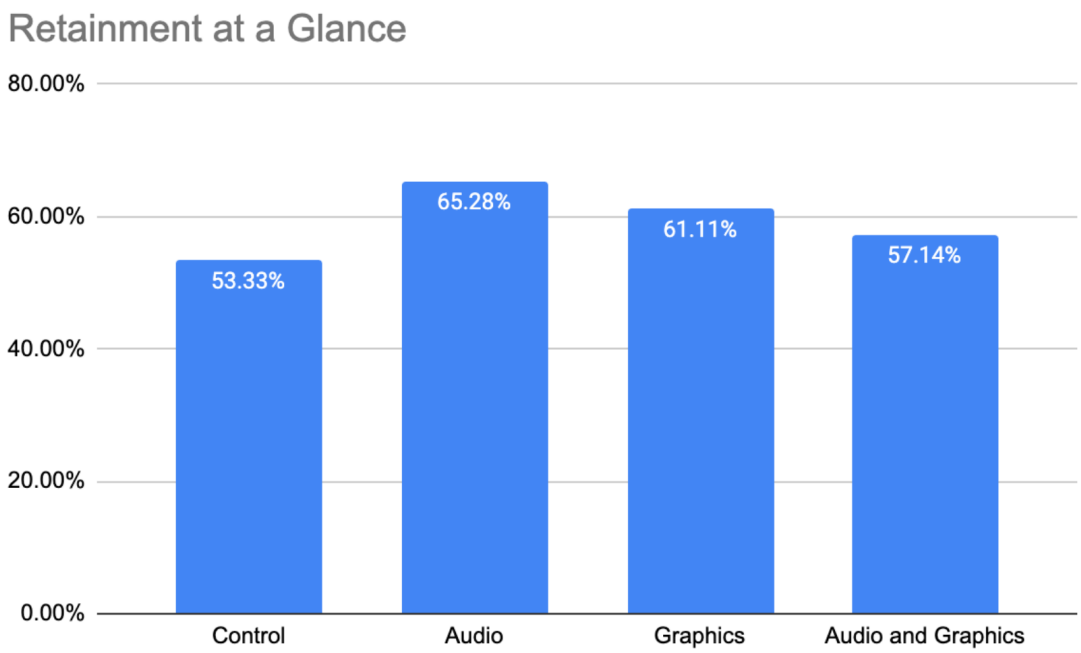
This section of the report lists the most interesting results of the experiment. The remaining results will be included in the References section, toward the end of the report.

In the context of our experiments, a p-value measures the likelihood that a

difference in the means of an experimental group and control group's quiz results may be attributed to anything more than random chance. So for our purposes, it helps measure the significance of our data. We decided that a significant threshold for our p-values would be anything less than 0.05.

The quiz had 12 questions relating to the material, and then four demographic questions. Each material-related question was worth one point. In the original quiz, there was one ‘matching’ question where the participants had to guess the time complexity of the algorithms. On the quiz this will appear as one question, but in reality we broke it down into four where guessing each algorithm’s time complexity correctly would be worth one point.

Graph 1, Retainment



P-values for Graph 1:

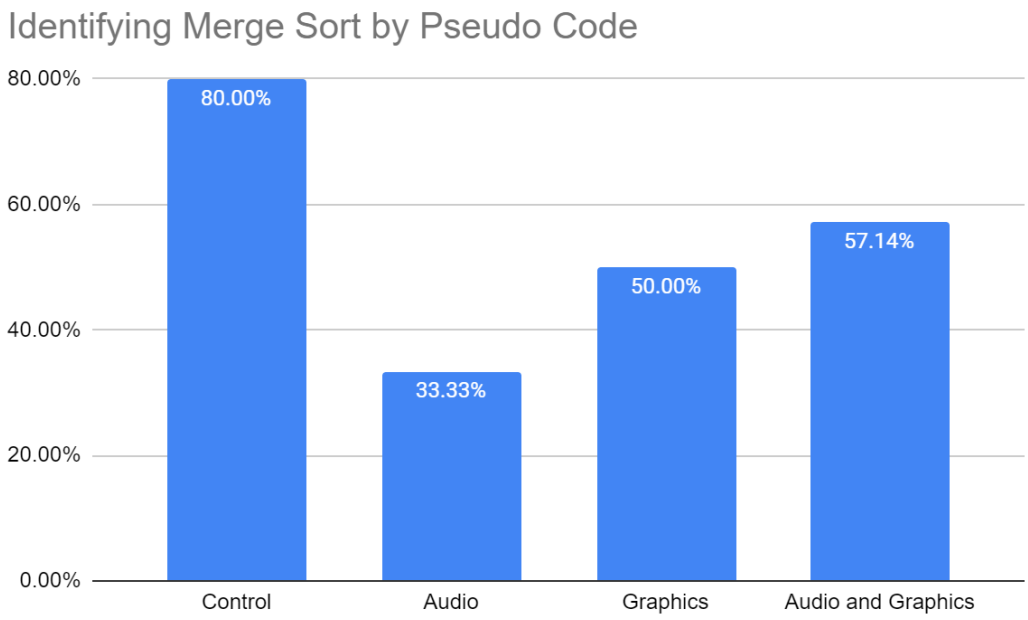
Audio: 0.4857654037
Graphics: 0.7856013487
Both: 0.5605067302

This graph shows the averaged quiz scores among the four groups. As you can see, the control group performed the worst while the audio only group performed the best.

An interesting observation on this data would be that while any combination of visual or auditory stimulation performed better than the control group, audio-only

was still best performance wise. However the p-values in this case do not show significant results.

Graph 2, Identifying Merge Sort



P-values for Graph 2:

Audio: 0.1478960787
Graphics: 0.3526756728
Both: 0.4543014344

This graph shows the average scores among the group on an individual question. The question had a block of pseudo code and the students had to choose out of a list of four algorithms, which one the pseudo code belonged to.

Note: A screenshot of this question from the quiz is shown below this discussion of the results.

The graph shows that the control group did the best on this question compared to the other three groups. This was not the result we were expecting and it is interesting to see. We think that it is because the control group had less to interact with when running the programs, so they got to know the pseudo code better than the groups who were focused more on listening and watching the algorithms work.

Note: the other three “identify this algorithm” graphs and corresponding p-values will be shown in Appendix A at the end of the report.

Here is an algorithm for a sorting procedure.

Let a be an array/list.

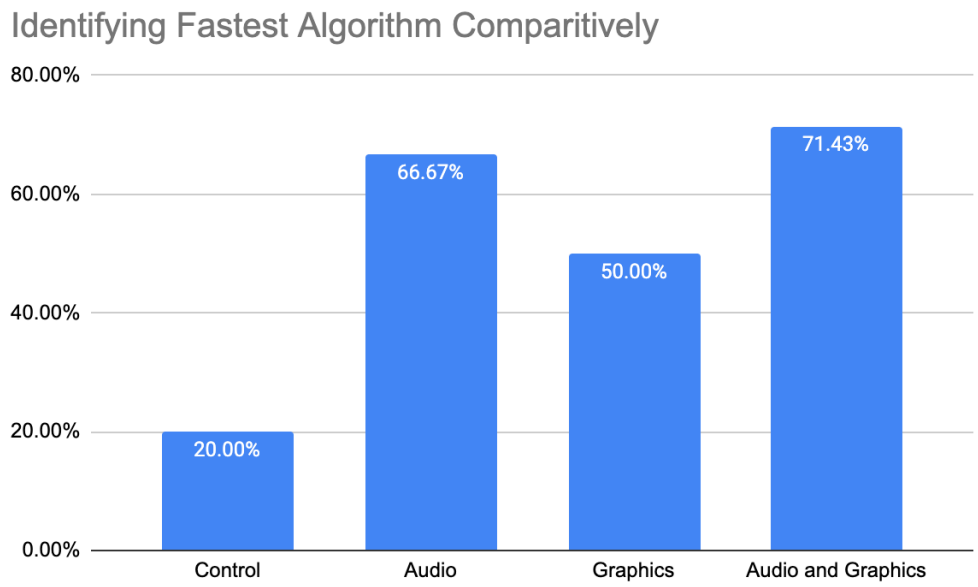
```
begin sort( a : array )
  n = a.size
  if ( n == 1 )
    return a
  mid = n/2
  a1 : array = a[0] ... a[mid]
  a2 : array = a[mid+1] ... a[n]
  a1 = sort( a1 )
  a2 = sort( a2 )
  return combine( a1, a2 )
end of sort

begin combine( a : array, b : array )
  c : array
  while ( a and b have elements )
    if ( a[0] > b[0] )
      itemToMove = b[0]
    else
      itemToMove = a[0]
    move itemToMove to the end of c
  if ( b is empty ) append rest of a to end of c
  if ( a is empty ) append rest of b to end of c
  return c
end of combine
```

Which sorting procedure is defined by the above algorithm? *

- ○ Quick Sort
- ○ Merge Sort
- ○ Insertion Sort
- ○ Bubble Sort

Graph 3, Identifying the Fastest Algorithm

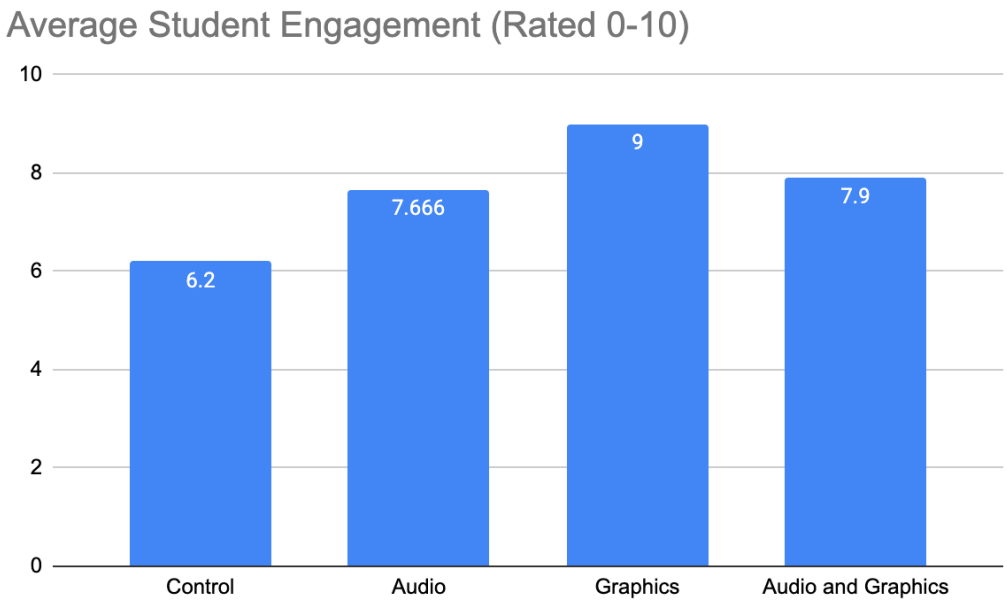


P-values for Graph 3:

Audio: 0.1478960787
Graphics: 0.3526756728
Both: 0.09243590837

This graph showcases the ability of the four groups to identify the fastest algorithm when comparing their capabilities. We purposefully left Quick Sort (the fastest algorithm of the four observed) out as a possible answer to this question because we wanted the students to recall what they observed in the lectures, and come up with the ‘next-best’ algorithm (in this case, Merge Sort). As you can see, the control group performed incredibly poorly compared to the experimental groups. We can infer then, from this data, that auditory or visual stimulation greatly assists students in identifying and ranking the speed of sorting algorithms. While the p-values still do not break our threshold of 0.05, they are still amongst some of the smallest p-values recorded.

Graph 4, Average Engagement



P-values for Graph 4:

Audio: 0.3024566523
Graphics: 0.02120600706
Both: 0.2053917574

This graph shows the average engagement score across the four groups. In the quiz, we asked the participants to rate the engagement of the experience on a scale of 0 to 10, with 10 being very engaging. This was our only piece of data that included a p-value below 0.05, and that was when comparing the control group's engagement against the graphics group's engagement. This graph tells us definitively that learning these algorithms with graphical enhancements is more engaging than running them without. Also, we can see that adding any sort of audio/visual feedback to the algorithms makes them more engaging to the person learning the material.

Conclusion of Results

While the data shows some promising trends, our p-values indicate that there is no significant data present which will allow us to answer our research question. In other words, more data is needed and additional analysis should be done in order to formally answer the question.

The one piece of significant data we do have, the subjective engagement ratings from the graphics group, allows us to formally conclude that some form of visual stimulation assists students in studying sorting algorithms.

Furthermore, we can affirm the integrity and dependability of our collected data. With additional executions of this experiment, the data can be supplemented and hopefully become more significant. Once the p-value thresholds have been met, the data will be considered reliable to make formal conclusions in response to our driving research question.

Possible Future Work:

We hope that in the future these experiments can be run again with more significant results. We have a few ideas of what future groups could do with this experiment:

1. Future projects could run our experiments on a wider range of students, instead of just 108 and 112.
2. They could also change the experiment to only have a control group and an audio-visual experimental group. This would help the small group size problem and could show whether or not these audio/visual enhancements help retention of the material.
3. The experiments could also be run again without any changes, which could allow for continuity of data from our experiments this year.

Acknowledgements:

- Professor Adams for all his help and work in this project
- The many authors/programmers of TSGL and TSAL over the years
- Chris Wierenga for keeping the lab running and giving us a place to run the experiments
- Nate Herder and Ian Adams for past work in this project's vein

References:

Our Project Website:
https://experiments-in-sorting.github.io/

Our Google Drive Folder:

https://drive.google.com/drive/folders/1fg5tKmzANrd0I84nz8CDz2v67GoZ5-NZ?usp=sharing

Programs in the CS Lab:

On a CS lab machine, navigate to **/home/cs/398/2020-21/sorting/** to see the four binaries of our project. Use the ls command to show the names of them, and run them by entering a command like this: ./bubble_sort
If you want to see audio and/or graphics, add on the flags “a” or “g” respectively.

Website used in experiments as supplemental reading material:

https://www.tutorialspoint.com/data_structures_algorithms/

Nate Herder and Ian Adams - 2020 Project Website:

https://cs-396-398-calvin-2020-ian-nate.github.io/website/

TSGL GitHub:

https://github.com/Calvin-CS/TSGL

TSAL GitHub:

https://github.com/Calvin-CS/TSAL

Calvin Computer Science Website:

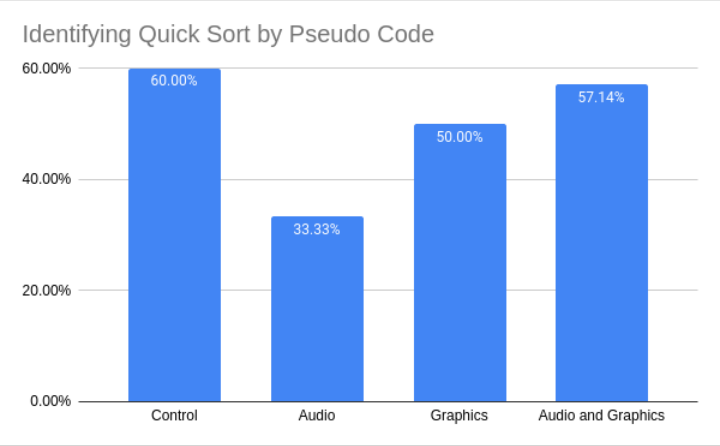
https://computing.calvin.edu/

The link to our dataset (additional graphs, quiz questions and data):

https://docs.google.com/spreadsheets/d/1ydo297RxdvL8sEPaSFJJLbz1LrRJ6FVqB8uRYOOk9Y4/edit?usp=sharing

Appendix A:

Show all four “identify this algorithm” graphs and their p-values here.



Quick Sort p-values:

Audio: 0.4279743738

Graphical: 0.7698749999

Both: 0.9297639744

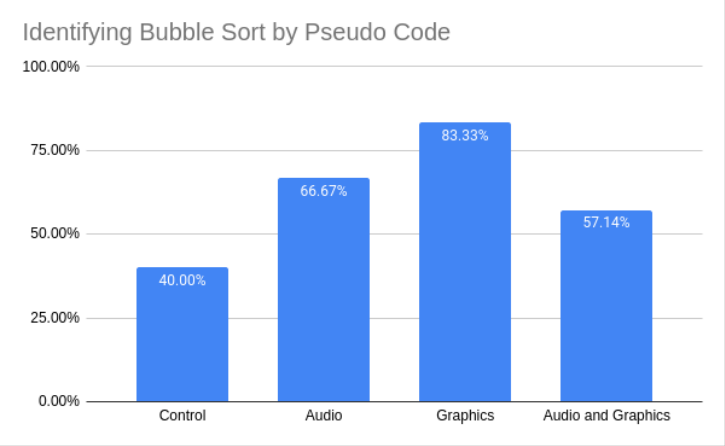
Provided Quick Sort pseudocode:

```
Let a be an array/list.

begin sort(left, right, a :
array)
   if right-left <= 0
      return
   else
      pivot = a[left]
      partition = partition(left,
right, pivot, a)
      sort(left, partition-1, a)
      sort(partition+1, right, a)
end sort

begin partition(left, right,
pivot, a : array)
   leftPointer = left
   rightPointer = right - 1
   while True do:
      while a[++leftPointer] <
pivot do //do-nothing
      while rightPointer > 0 &&
a[--rightPointer] > pivot do
//do-nothing
      if leftPointer >=
rightPointer
         break
      else
         swap a[leftPointer],
a[rightPointer]

   swap a[leftPointer], a[right]
   return leftPointer
end partition
```



Bubble Sort p-values:

Audio: 0.4279743738

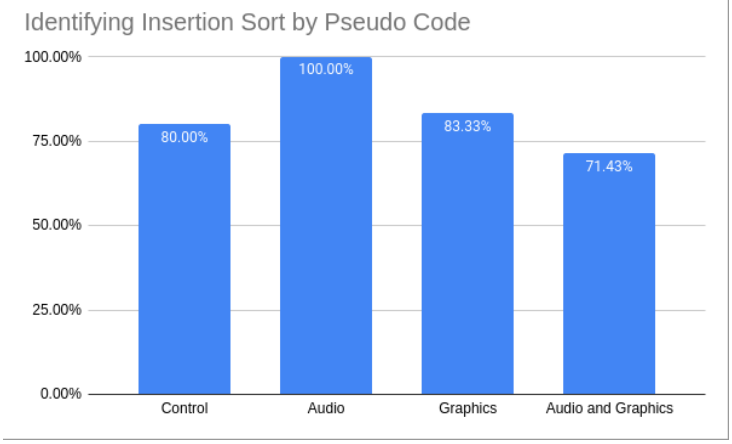
Graphical: 0.1664373461

Both: 0.5994703107

Provided Bubble Sort pseudocode:

```
Let a be an array/list.

begin sort(a : array)
   while anything changed:
      for all elements of a
            if a[i] > a[i+1]
               swap(a[i], a[i+1])
   return a
end sort
```



Insert Sort p-values:

Audio: 0.2966650369

Graphical: 0.9000259772

Both: 0.7628607573

Provided Insertion Sort pseudocode:

```
Let a be an array/list.

begin sort(a : array)
   int holePosition, value
   for i = 1 to length(a) inclusive do:
      value = a[i]
      holePosition = i
      while holePosition > 0 and a[holePosition-1] > value do:
            a[holePosition] = a[holePosition-1]
            holePosition = holePosition -1
      a[holePosition] = value
end sort
```

Appendix B: Quiz Questions

Here is an algorithm for a sorting procedure.

Let a be an array/list.

```
begin sort(left, right, a : array)
  if right-left <= 0
    return
  else
    pivot = a[left]
    partition = partition(left, right, pivot, a)
    sort(left, partition-1, a)
    sort(partition+1, right, a)
end sort

begin partition(left, right, pivot, a : array)
  leftPointer = left
  rightPointer = right - 1
  while True do:
    while a[++leftPointer] < pivot do //do-nothing
    while rightPointer > 0 && a[--rightPointer] > pivot do //do-nothing
    if leftPointer >= rightPointer
      break
    else
      swap a[leftPointer], a[rightPointer]

  swap a[leftPointer], a[right]
  return leftPointer
end partition
```

Which sorting procedure is defined by the above algorithm?

- ◯ Quick Sort
- ◯ Merge Sort
- ◯ Bubble Sort
- ◯ Insertion Sort

Here is an algorithm for a sorting procedure.

Let a be an array/list.

```
begin sort(a : array)
  while anything changed:
    for all elements of a
        if a[i] > a[i+1]
          swap(a[i], a[i+1])
  return a
end sort
```

Which sorting procedure is defined by the above algorithm?

- ◯ Merge Sort
- ◯ Quick Sort
- ◯ Insertion Sort
- ◯ Bubble Sort

Here is an algorithm for a sorting procedure.

Let a be an array/list.

```
begin sort(a : array)
  int holePosition, value
  for i = 1 to length(a) inclusive do:
    value = a[i]
    holePosition = i
    while holePosition > 0 and a[holePosition-1] > value do:
        a[holePosition] = a[holePosition-1]
        holePosition = holePosition -1
    a[holePosition] = value
end sort
```

Which sorting procedure is defined by the above algorithm?

- ◯ Bubble Sort
- ◯ Merge Sort
- ◯ Quick Sort
- ◯ Insertion Sort

Here is an algorithm for a sorting procedure.

Let a be an array/list.

```
begin sort( a : array )
  n = a.size
  if ( n == 1 )
    return a
  mid = n/2
  a1 : array = a[0] ... a[mid]
  a2 : array = a[mid+1] ... a[n]
  a1 = sort( a1 )
  a2 = sort( a2 )
  return combine( a1, a2 )
end of sort

begin combine( a : array, b : array )
  c : array
  while ( a and b have elements )
    if ( a[0] > b[0] )
      itemToMove = b[0]
    else
      itemToMove = a[0]
    move itemToMove to the end of c
  if ( b is empty ) append rest of a to end of c
  if ( a is empty ) append rest of b to end of c
  return c
end of combine
```

Which sorting procedure is defined by the above algorithm?

- ◯ Quick Sort
- ◯ Merge Sort
- ◯ Insertion Sort
- ◯ Bubble Sort

Which of the following algorithms is fastest?

- ◯ Bubble Sort
- ◯ Merge Sort
- ◯ Insertion Sort
- ◯ They're about the same.

Which of the following algorithms is slowest?

- ◯ Insertion Sort
- ◯ Quick Sort
- ◯ Merge Sort
- ◯ They're about the same.

Which algorithm is the fastest of the four you observed?

- ◯ Bubble sort
- ◯ Insertion sort
- ◯ Quick sort
- ◯ Merge sort

Which algorithm is the slowest of the four you observed?

- ◯ Insertion sort
- ◯ Merge sort
- ◯ Bubble sort
- ◯ Quick sort

What is the time complexity of each algorithm?

	O(1)	O(n)	O(n*log(n))	O(n^2)
Bubble sort	◯	◯	◯	◯
Insertion sort	◯	◯	◯	◯
Merge sort	◯	◯	◯	◯
Quick sort	◯	◯	◯	◯

How engaging did you find the sorting lessons?

	1	2	3	4	5	6	7	8	9	10	
Boring, uninteresting	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	I loved it!

What is your first and last name? (used only for awarding extra credit)

Your answer

As what gender do you identify?

- ◯ Male
- ◯ Female
- ◯ Other
- ◯ Prefer not to say

In what CS class are you currently enrolled?

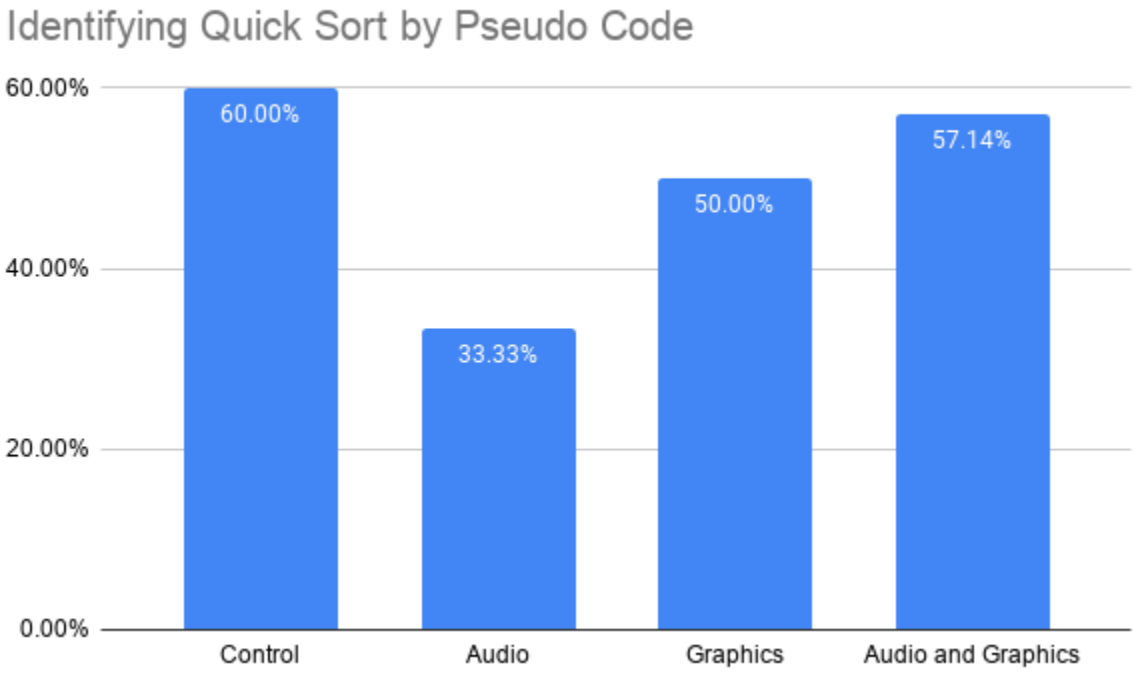
- ◯ CS 108
- ◯ CS 112

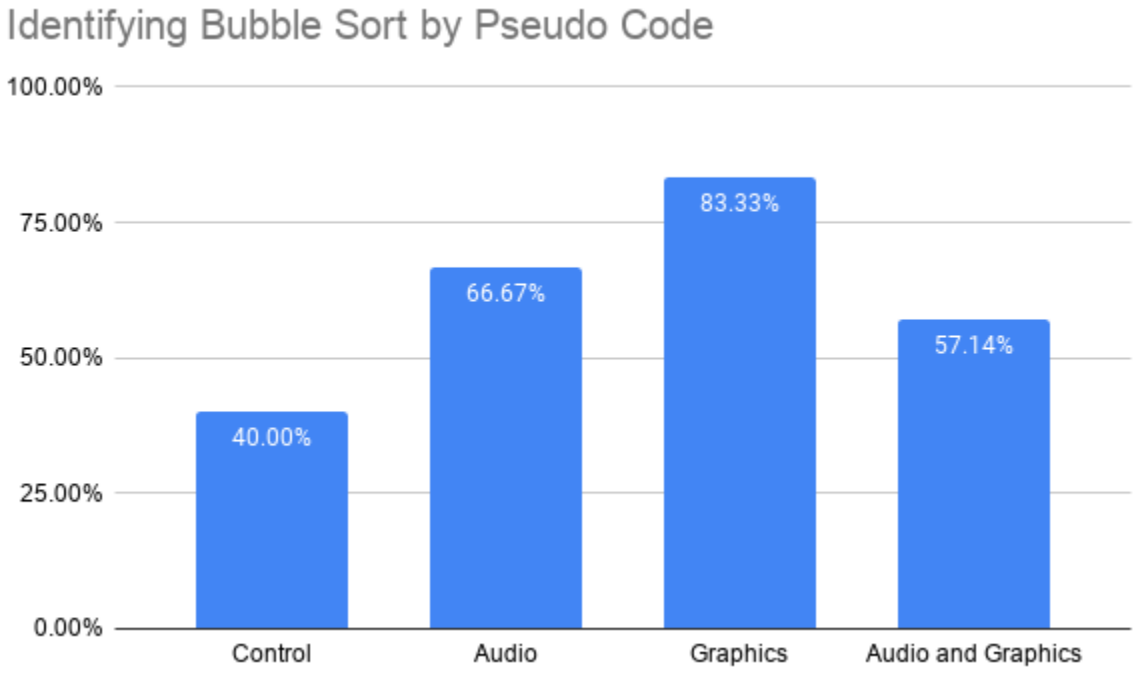
What is your major?

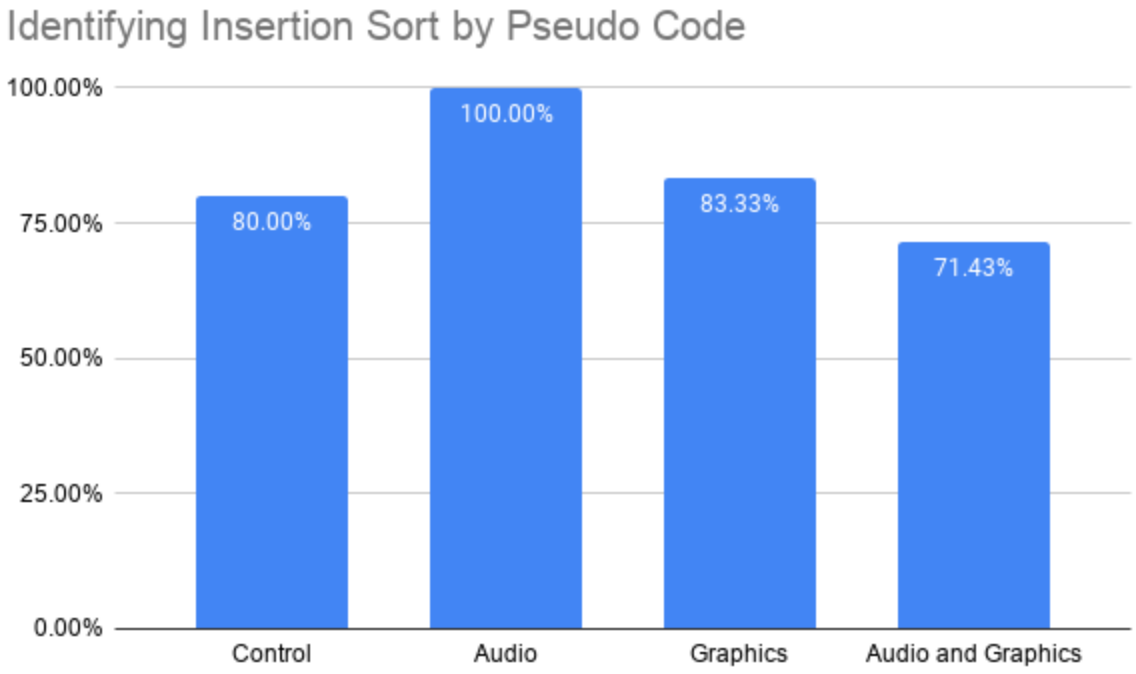
- ◯ Computer Science
- ◯ Data Science
- ◯ Engineering
- ◯ Other:

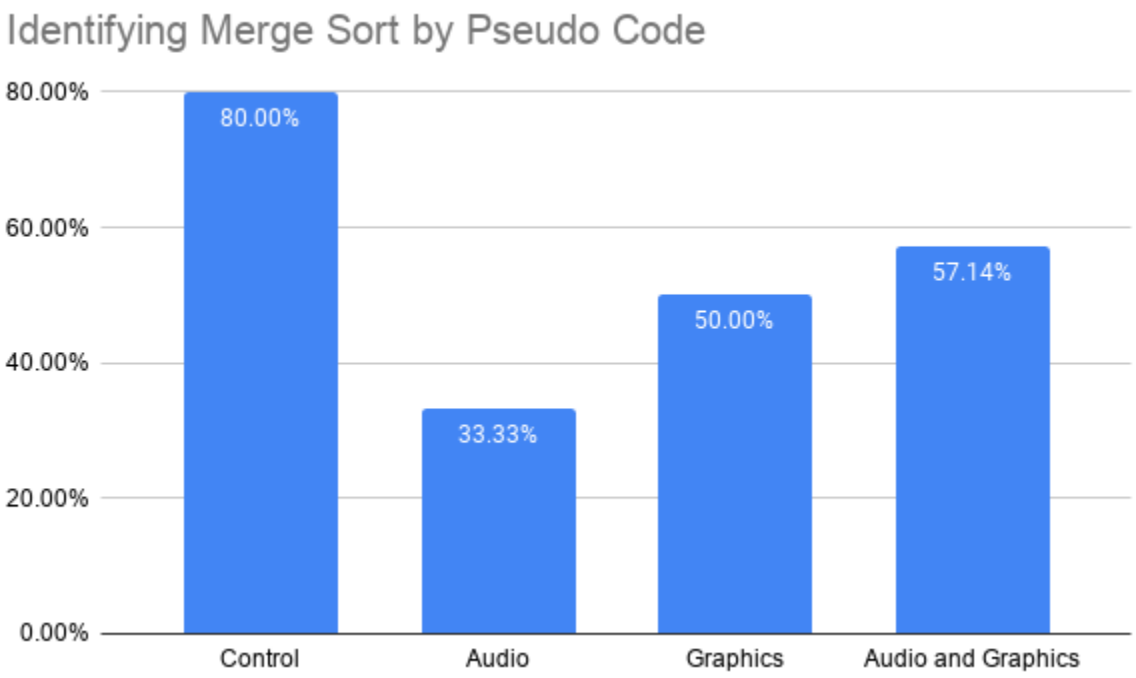
Appendix C: Charts

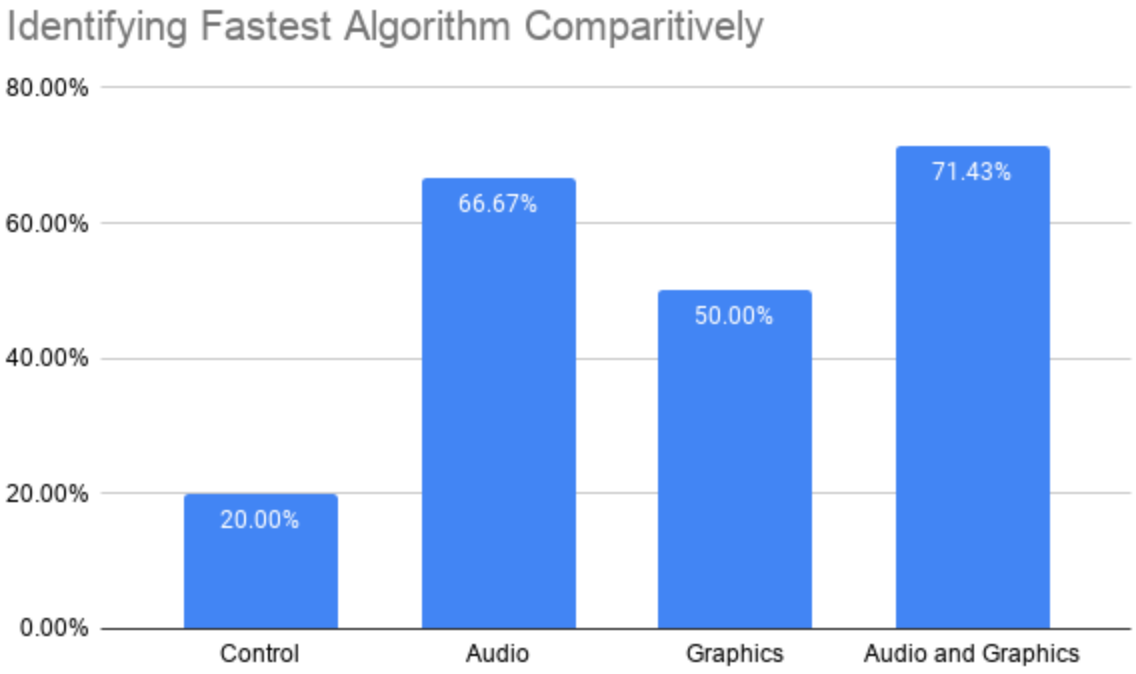
Here are the charts showcasing all of our recorded data. The data recorded in the charts correspond to the questions listed in the previous appendix (first chart corresponds to first question, third chart to third question, etc.)

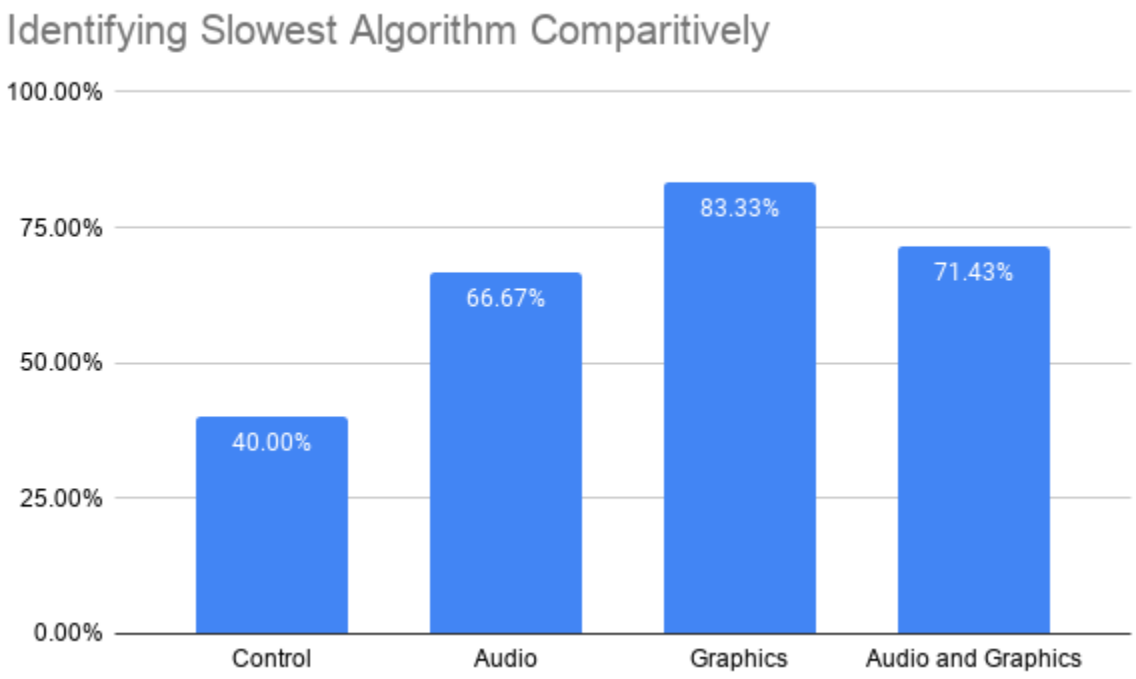




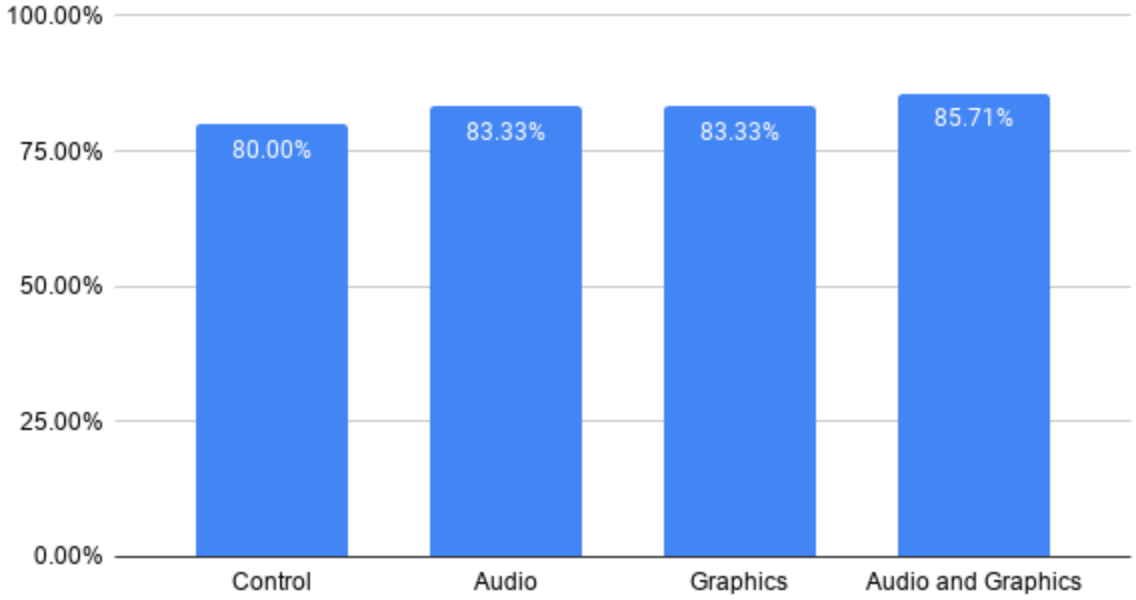

Identifying Insertion Sort by Pseudo Code
100.00%
75.00%
50.00%
25.00%
0.00%
80.00%
100.00%
83.33%
71.43%
Control
Audio
Graphics
Audio and Graphics


Identifying Merge Sort by Pseudo Code
80.00%
60.00%
40.00%
20.00%
0.00%
80.00%
33.33%
50.00%
57.14%
Control
Audio
Graphics
Audio and Graphics

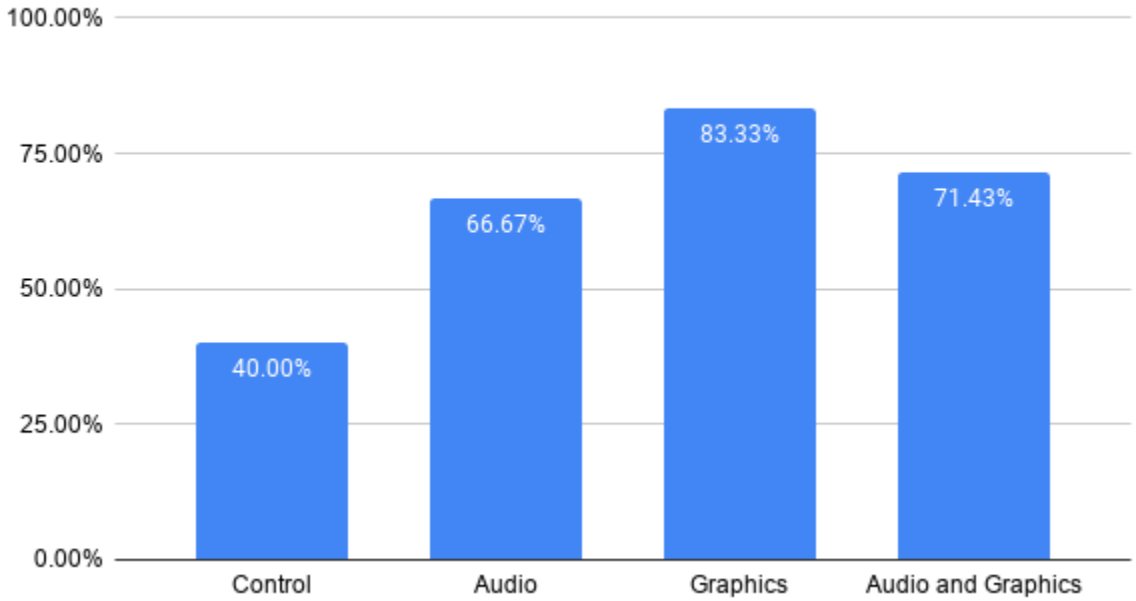

Identifying Fastest Algorithm Comparitively
80.00%
60.00%
40.00%
20.00%
0.00%
20.00%
66.67%
50.00%
71.43%
Control
Audio
Graphics
Audio and Graphics

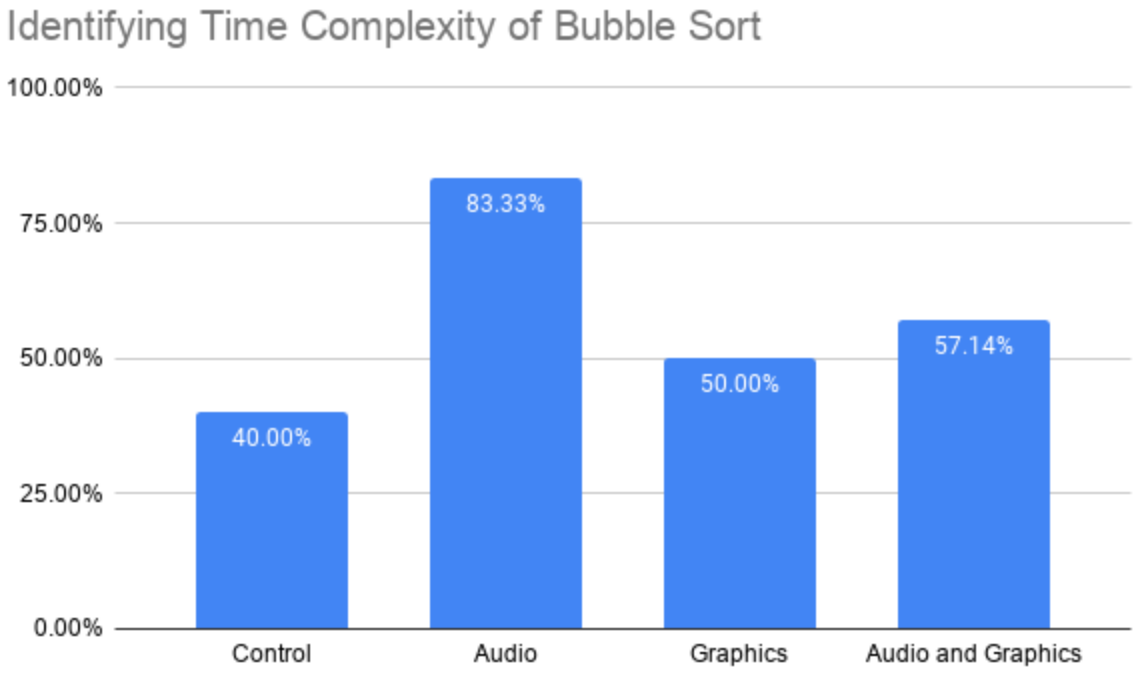

Identifying Slowest Algorithm Comparitively
100.00%
75.00%
50.00%
25.00%
0.00%
40.00%
66.67%
83.33%
71.43%
Control
Audio
Graphics
Audio and Graphics

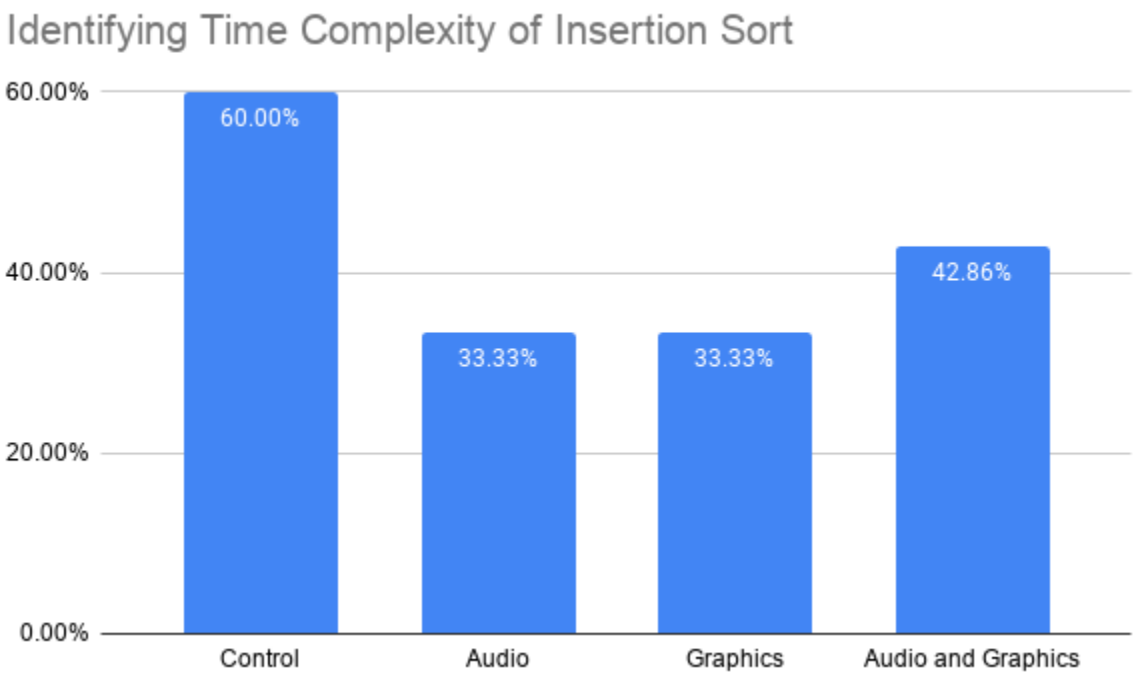
Identifying Fastest Algorithm of the Four Observed

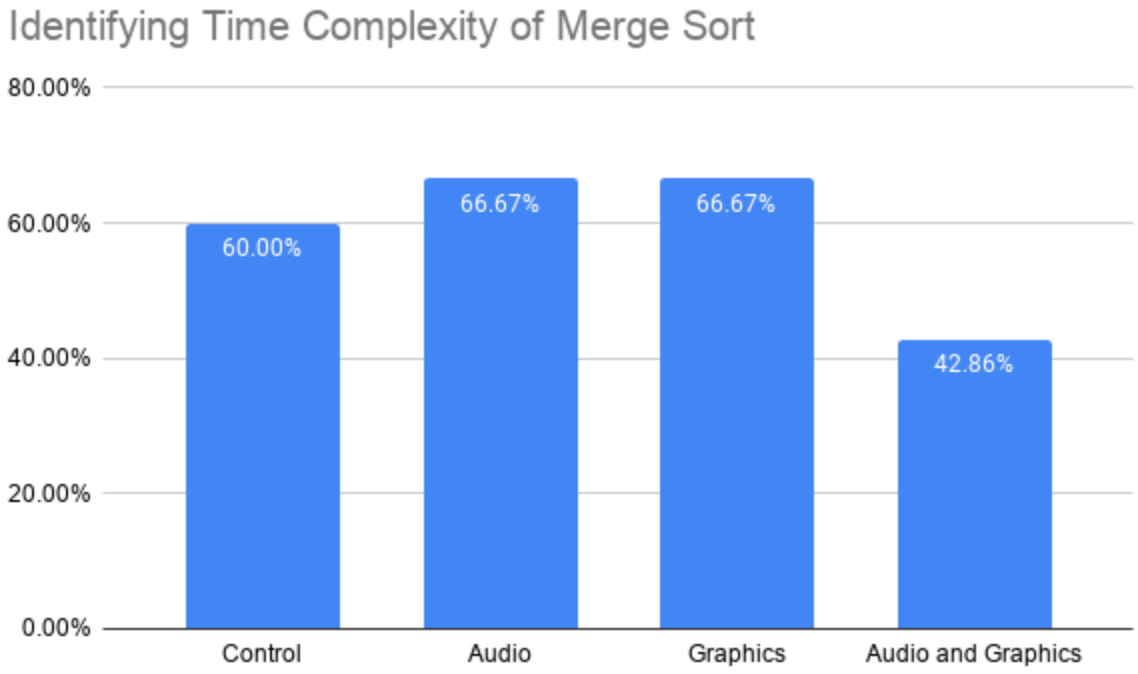


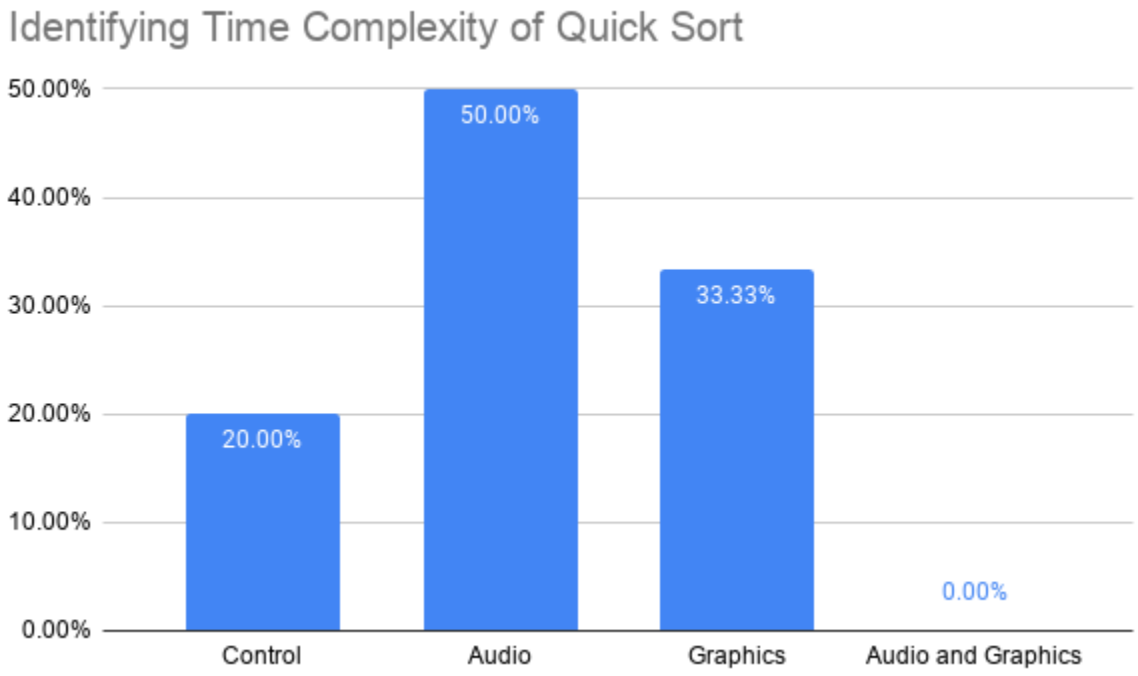
Identifying Slowest Algorithm Comparitively

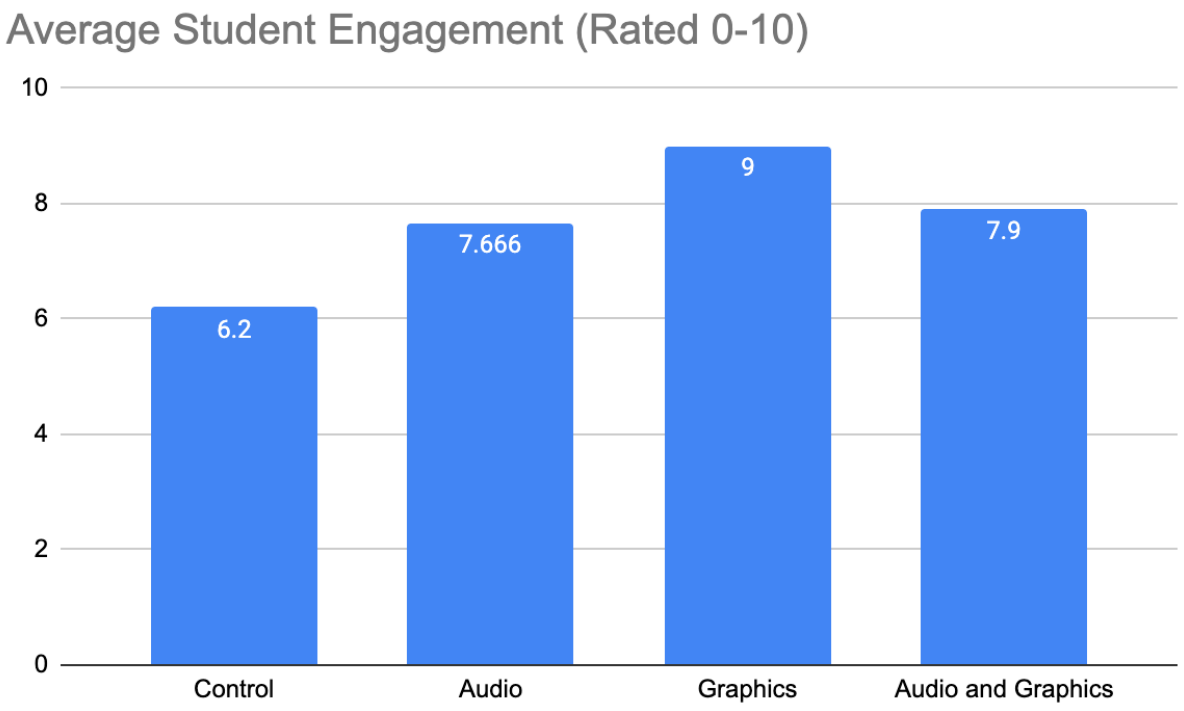



Identifying Time Complexity of Bubble Sort
100.00%
75.00%
50.00%
25.00%
0.00%
40.00%
83.33%
50.00%
57.14%
Control
Audio
Graphics
Audio and Graphics

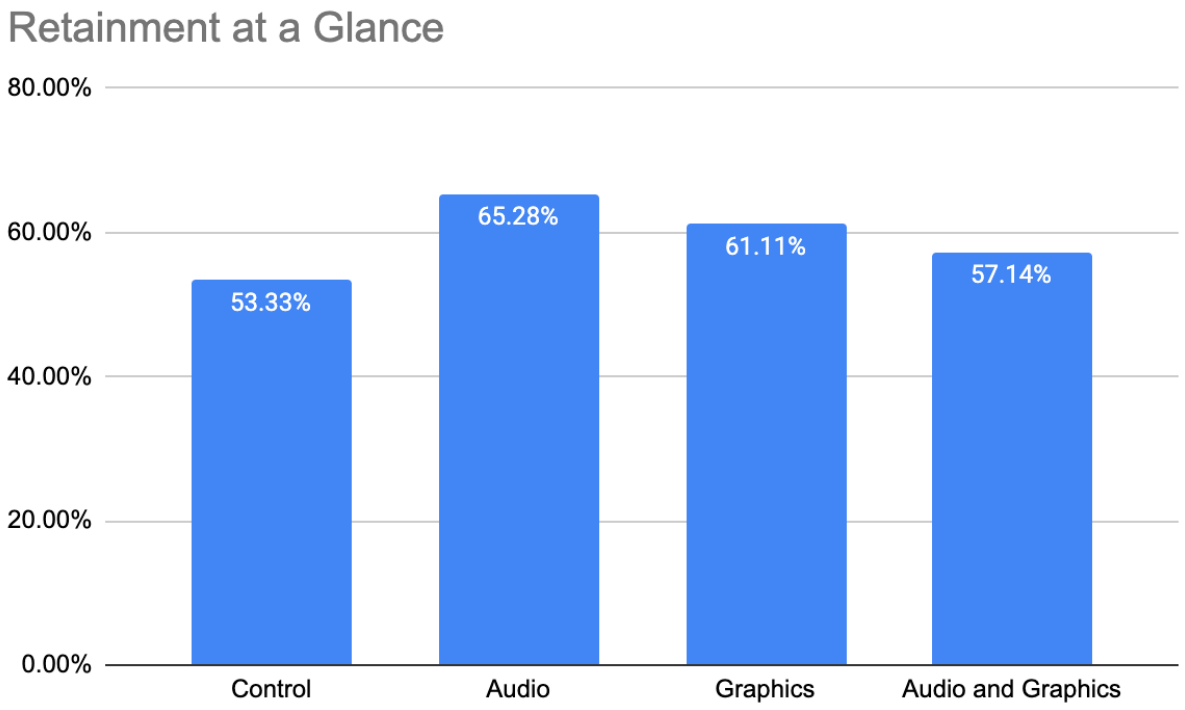

Identifying Time Complexity of Insertion Sort
60.00%
40.00%
20.00%
0.00%
60.00%
33.33%
33.33%
42.86%
Control
Audio
Graphics
Audio and Graphics


Identifying Time Complexity of Merge Sort
80.00%
60.00%
40.00%
20.00%
0.00%
60.00%
66.67%
66.67%
42.86%
Control
Audio
Graphics
Audio and Graphics


Identifying Time Complexity of Quick Sort
50.00%
40.00%
30.00%
20.00%
10.00%
0.00%
20.00%
50.00%
33.33%
0.00%
Control
Audio
Graphics
Audio and Graphics



This last chart once again shows results overall (also in our formal results section of this report):



Appendix D: T-tests and p-values

group	Which sorting procedure is defined by the above algorithm? A:quick	Which sorting procedure is defined by the above algorithm? A:bubble	Which sorting procedure is defined by the above algorithm? A:insertion	Which sorting procedure is defined by the above algorithm? A: merge	Which of the following algorithms is fastest? A: Merge	Which of the following algorithms is slowest? A: insertion	Which algorithm is the fastest of the four you observed? Quick	Which algorithm is the slowest of the four you observed? A: bubble	What is the time complexity of each algorithm? [Bubble sort] A: on2	What is the time complexity of each algorithm? [Insertion sort] A: on2	What is the time complexity of each algorithm? [Merge sort] onlogn	What is the time complexity of each algorithm? [Quick sort] A: onlogn	How engaging did you find the sorting lessons?
audio	0	0	1	0	1	1	1	1	0	0	1	0	10
audio	1	1	1	1	1	1	1	1	1	1	1	1	4
audio	0	1	1	0	1	1	1	1	1	0	0	1	10
audio	0	0	1	0	0	0	1	1	1	0	1	0	8
audio	1	1	1	1	0	0	0	1	1	0	0	0	6
audio	0	1	1	0	1	1	1	1	1	1	1	1	8
P-val(2,2)	0.427974 3738	0.427 97437 38	0.296 66503 69	0.147 89607 87	0.147 89607 87	0.427 97437 38	0.900 02597 72	0.103 88813 11	0.166 43734 61	0.427 97437 38	0.840 22766 69	0.352 67567 28	0.302 45665 23
P-val(2,3)	0.432039 7877	0.432 03978 77	0.373 90096 63	0.142 87300 84	0.142 87300 84	0.432 03978 77	0.901 17249 28	0.177 80780 84	0.185 20589 66	0.432 03978 77	0.841 46816 32	0.343 43639 61	0.296 59203 48
a + g	0	0	0	1	1	1	1	1	0	0	0	0	7
a + g	1	1	1	1	1	1	1	1	1	1	1	0	7
a + g	1	1	1	1	1	1	1	1	1	0	0	0	9
a + g	1	0	0	0	1	0	0	0	0	0	1	0	8
a + g	0	0	1	0	0	1	1	1	0	1	0	0	10
a + g	0	1	1	0	1	1	1	1	1	0	0	0	4
a + g	1	1	1	1	0	0	1	0	1	1	1	0	10
P-val(2,2)	0.929763 9744	0.599 47031 07	0.762 86075 73	0.454 30143 44	0.092 43590 837	0.319 78714 56	0.815 38070 82	0.711 39279 83	0.599 47031 07	0.599 47031 07	0.599 47031 07	0.254 98114 53	0.205 39175 74
P-val(2,3)	0.930353 2597	0.602 91147 64	0.759 71254 76	0.440 75983 13	0.090 42882 527	0.335 03460 13	0.822 15710 1	0.718 92666 34	0.602 91147 64	0.602 91147 64	0.602 91147 64	0.373 90096 63	0.206 36283 81
graphical	1	0	1	0	0	1	1	0	0	0	1	0	10

graphical	1	1	1	1	1	1	1	1	1	1	1	0	9
graphical	0	1	1	0	1	1	1	1	1	0	0	1	10
graphical	0	1	0	0	0	1	0	0	0	1	1	1	10
graphical	0	1	1	1	0	1	1	1	1	0	0	0	7
graphical	1	1	1	1	1	0	1	1	0	0	1	0	8
P-val(2, 2)	0.769874999	0.16643734 61	0.900025977 2	0.352675672 8	0.352675672 8	0.16643734 61	0.900025977 2	0.840227666 9	0.769874999	0.427974373 8	0.840227666 9	0.661780330 5	0.021206007 06
P-val(2, 3)	0.770152 5965	0.185205896 6	0.901172492 8	0.343436396 1	0.343436396 1	0.185205896 6	0.901172492 8	0.841468163 2	0.770152596 5	0.432039787 7	0.841468163 2	0.657271592 6	0.035080514 09
control	1	1	1	1	0	0	1	0	1	1	1	0	6
control	1	1	1	1	0	0	1	1	1	1	1	0	8
control	1	0	1	1	0	1	1	1	0	0	1	1	8
control	0	0	0	0	0	0	0	1	0	1	0	0	3
control	0	0	1	1	1	1	1	0	0	0	0	0	6
--	--	--	--	--	--	--	--	--	--	--	--	--	--