Fun with Sorting

Provided by TryEngineering - www.tryengineering.org

Lesson Focus

The lesson focuses on introducing the fundamental problem of “sorting an array” to pre-university students.

Lesson Synopsis

Fun with Sorting introduces pre-university students to sorting, one of the most basic and fundamental problems in Computer Science. Rather than study from a textbook or listen to a lecture, students will discover the algorithms themselves in class, and come up with ways to sort the numbers themselves. Students are first introduced to smaller versions of the problem, which form the building blocks of the algorithms they themselves develop later. The problem is given the form of instructor-moderated in-class demonstrations and discussions, followed by group exercises and inter-group competitions.

Age Levels

10-16

Objectives

- Observe how there can be multiple ways of sorting numbers, and how some ways are more efficient than others.
- Observe how an algorithm is a “procedure” and should not be dependent on the input.
- Learn that an algorithm must be very explicit in its instructions.

Anticipated Learner Outcomes

After this lesson, students should have a basic understanding of:

- Iteration
- Sorting, and its building blocks
- Algorithm Complexity

Lesson Activities

Eight objects, each with a hidden number, will form a list that needs to be sorted in ascending order. The numbers will only be visible to one person called the “controller”. Everyone else will then give commands to the controller, and attempt to sort the list. However, the controller can only be asked certain specific questions, like comparing two objects by their numerical value, and may only obey certain instructions, like moving an object to a new position or swapping two objects.

Resources / Materials

- Teacher Resource Documents (attached)
- Student Worksheets (attached)
- Student Resource Sheets (attached)

Internet Connections

- Sorting Algorithm Animations (http://www.sorting-algorithms.com/)

Recommended Reading

- Art of Computer Programming, Volume 3 by Donald E. Knuth (ISBN: 0321751043)

Optional Writing Activity

Computers generally tend to spend about a full quarter of their processing power on sorting different data. As an example, a computer in a hospital may maintain a very large database of all patients who have ever been to the hospital for treatment in the past 5 years. Different people in the hospital might want different lists of patients. A person managing hospital finances might want a list of patients ordered by their hospital charges. A researcher might want a list ordered by the disease for which they were treated. An administrator might want a list ordered by the doctor who treated the patient. While generating these lists, the computer will have to sort the data afresh every time according to the need of the user. Can you think of any other scenario in which sorting is important? What advantages are there of maintaining sorted data over unsorted data? What are the possible disadvantages?

Fun with Sorting

For Teachers: Teacher Resources

◆ Lesson Goal

The lesson should be discovery-based, rather than lecture or textbook based. The students will come up with and discover the algorithms for sorting numbers themselves, rather than being taught conventionally.

◆ Lesson Objectives

- To introduce students to a fundamental Computer Science problem, and how it need not involve computers.
- To give students a feel of what "iteration" means.
- To make students realize how there may be multiple ways of sorting, and how some ways are better than others.
- Observe how an algorithm is a "procedure" and should not be dependent on the input, e.g. the sorting procedure should work for all lists of numbers.

◆ Materials

- Cardboard or Chart Paper
- 1 Large Marker
- Sticky Tape

◆ Procedure

The procedure is divided into three sections. The first section contains the general rules of the game that will be played. The second section contains the different variations of the game, and in what order should those variations be played. The third section contains some creative ways to encourage class participation, which is vital to the success of this lesson plan.

Section I – General Game Rules:

1. Choose 8 objects to represent the list to be sorted. You may pick anything to represent your list of numbers. Some ideas for the list are:
    a. Large playing plastic blocks
    b. Large Cardboard cones
2. Pick one person to be the "controller".
3. Random numbers should be written, or pasted, on the objects in such a way that they are hidden from the other students, and only the controller should be able to see them.
4. The students (except the controller) will now attempt to sort the list in ascending order. To do this, the students can do either of three things:
    a. Ask the controller a question. The only allowed question is: "Is this number larger than that number?" which allows students to compare any two numbers. The controller may only reply with a yes or a no.
    b. Ask the controller to swap any two objects. The controller will then swap the objects, taking care not to reveal the numbers on them.
    c. Ask the controller to move any object to a specific position, i.e. "Move this object to the third position" or "Move this object in between those two objects".

5. The controller will keep a count of the number of yes/no questions asked. The students should attempt to sort the numbers while keeping the question count to a minimum.
6. Once the students feel that the list is sorted, they will ask the controller to reveal the numbers. If the list is not sorted, then the controller shuffles up the list and starts over.

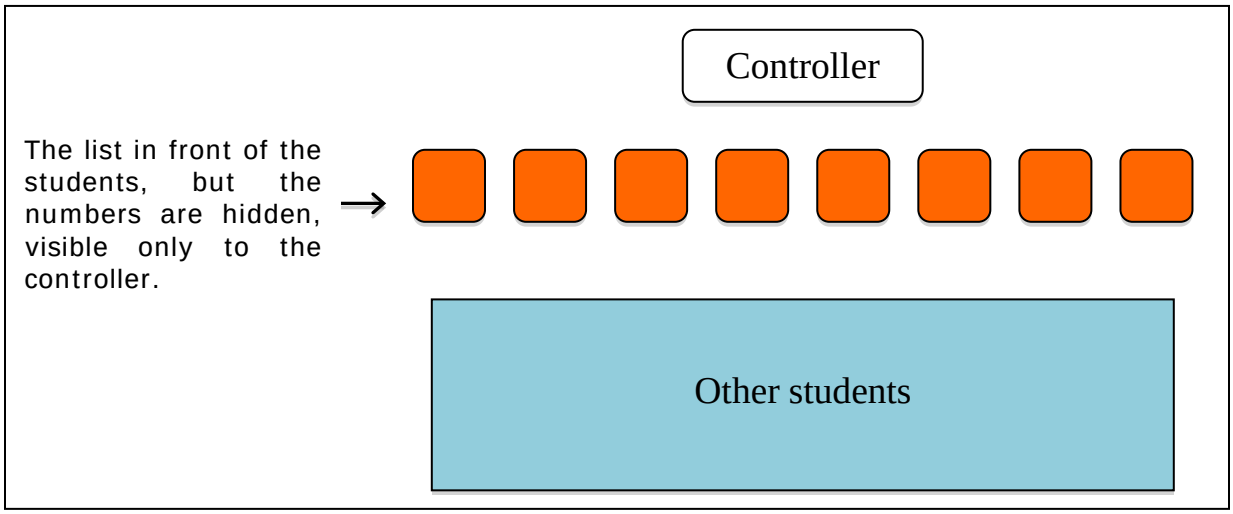


Fig. 1 – Game Setup

Section II – Variations:

The following are different variations and flavors of the sorting problem, and should be taught in the specified sequence. Each exercise should first be carried out as an in-class game in which the entire class participates, and then students should be split up into groups of 3, 4 or 5, in which they play the game among themselves (using small cards) and fill out their worksheets.

1. *Single Insertion.* Set up the initial list as follows: one of the numbers should be singled out, and the rest should be pre-arranged in an ascending order. The singled-out number should then be placed at one end of the sorted part of the list. Everyone now knows that the entire list is sorted, except for one number at the end, which is out of place.

Example:

The entire list is in ascending order, except for the last number, 4.

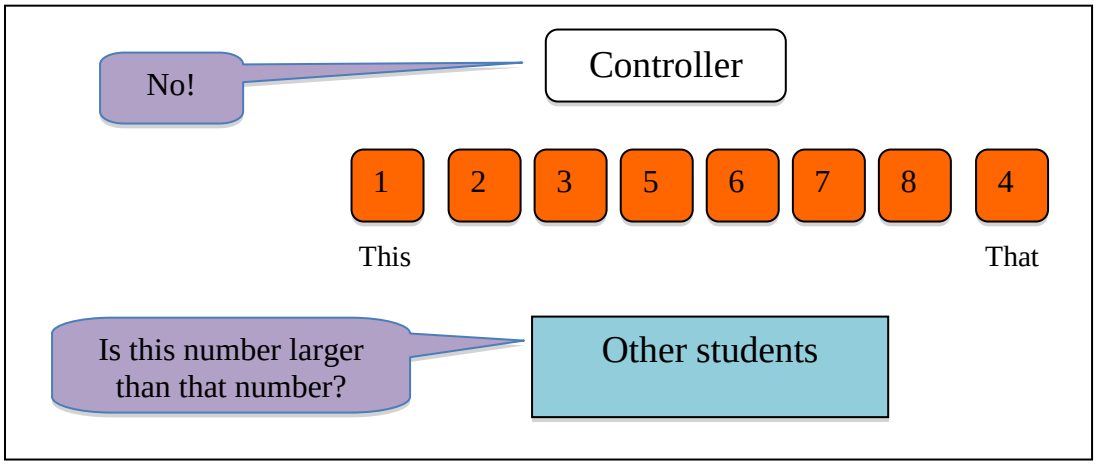
The students should then play by the rules mentioned in Section I, and attempt to completely sort the list. This is called Single Insertion, and helps as a basic building block for other sorting algorithms like Insertion Sort.

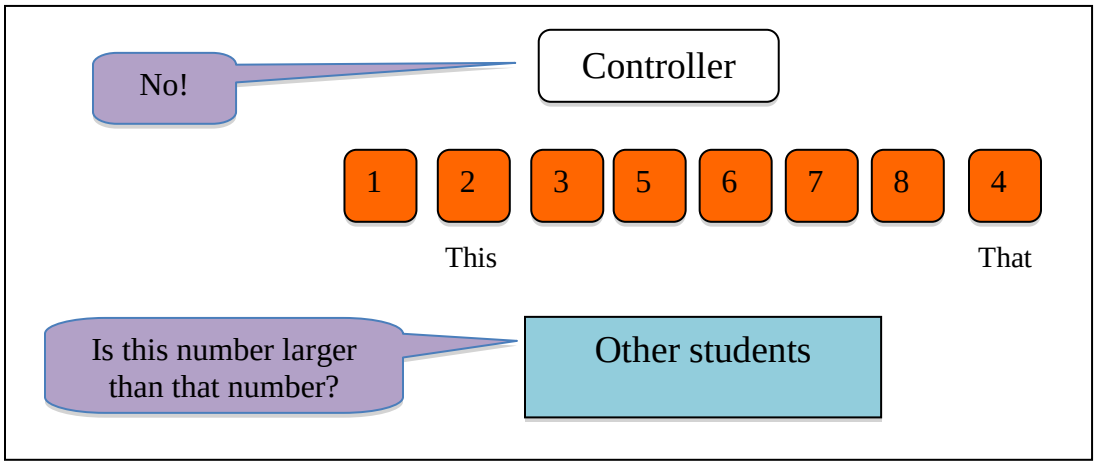
If one sees the numbers, it is quite obvious that we can simply place 4 between 3 and 5, and the list would be completely sorted. However, that is not the way computers work. Therefore, the students must start off with comparing numbers, and then ordering swaps and/or relocations.

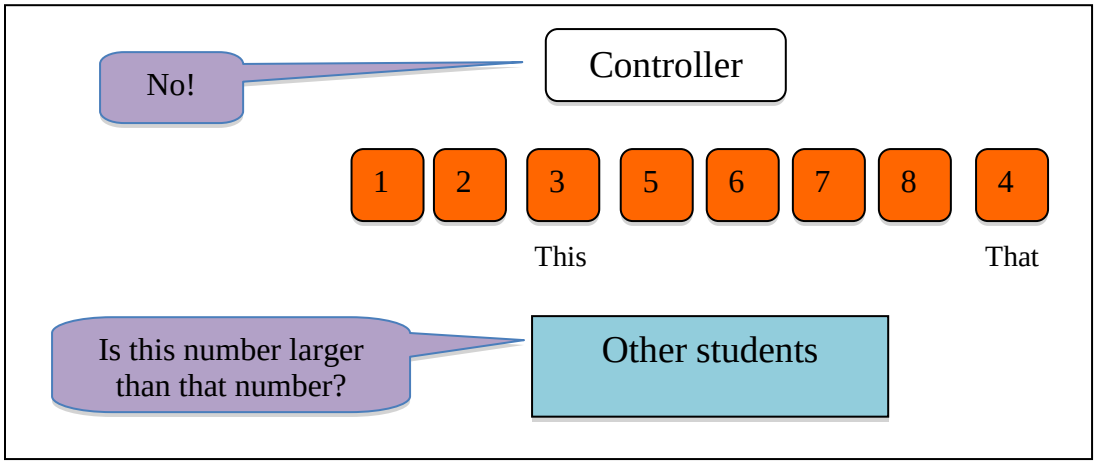
In this case, the students know that the entire list is in ascending order, except for the last number. This would lead to a realization within the students, rather than being told directly by the instructor, that they simply need to identify the position in which to insert the last unordered number.

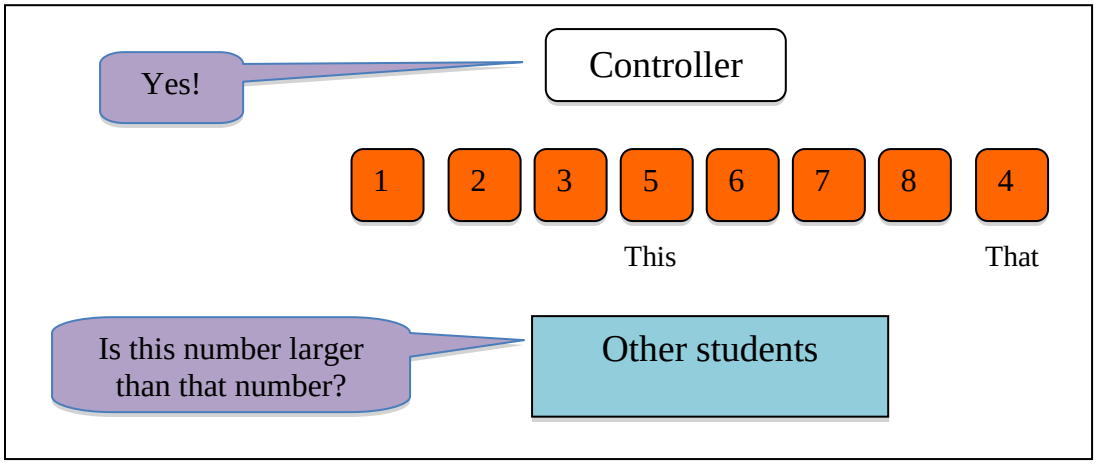
Solution:

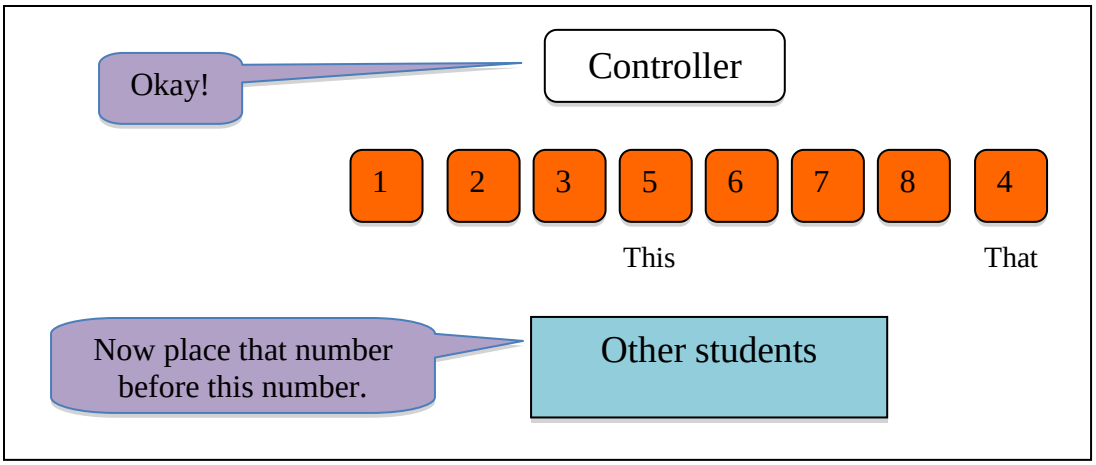
They can do this by asking comparing the last number to each of the numbers in the sorted list in order, going from left to right (smallest number first). Whenever they should find that the last number is smaller than the number that they are comparing it against, they should immediately use rule 4c mentioned in the game rules, and simply relocate the last number to the position behind the first number which is greater than it.

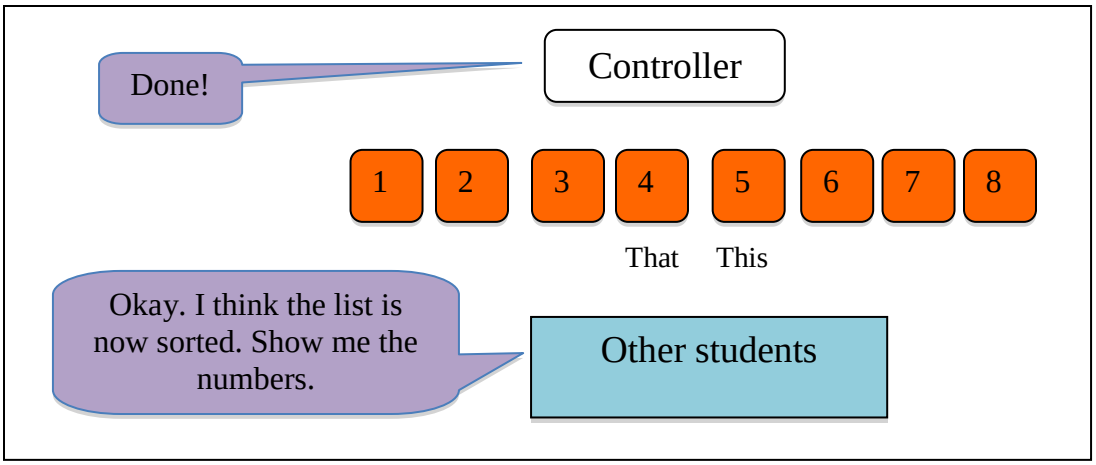



No!
Controller
1
2
3
5
6
7
8
4
This
That
Is this number larger than that number?
Other students


No!
Controller
1
2
3
5
6
7
8
4
This
That
Is this number larger than that number?
Other students


Yes!
Controller
1
2
3
5
6
7
8
4
This
That
Is this number larger than that number?
Other students





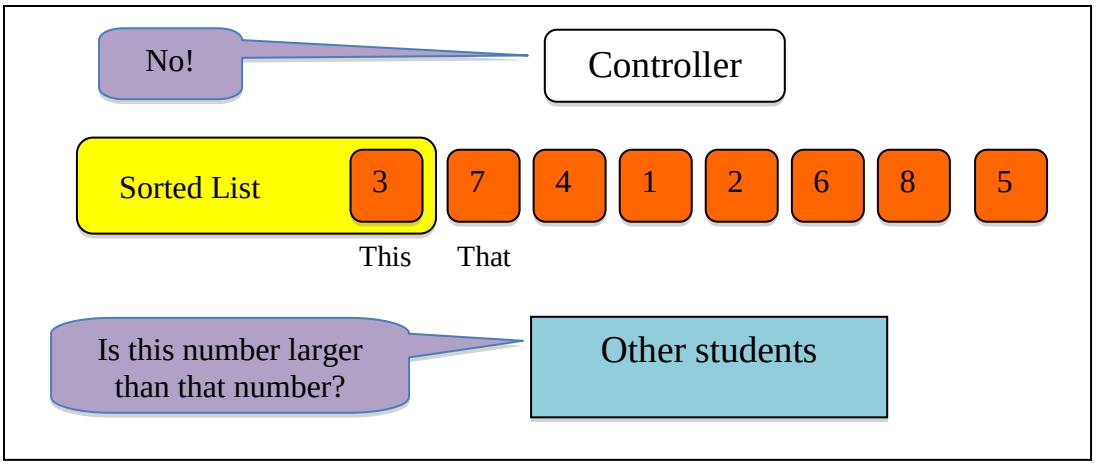
The above sequence of diagrams shows what should ideally happen in the class. Note that, throughout the procedure, the students are ignorant of the numbers on the list, and simply gives a set of instructions by comparing two numbers at every step.

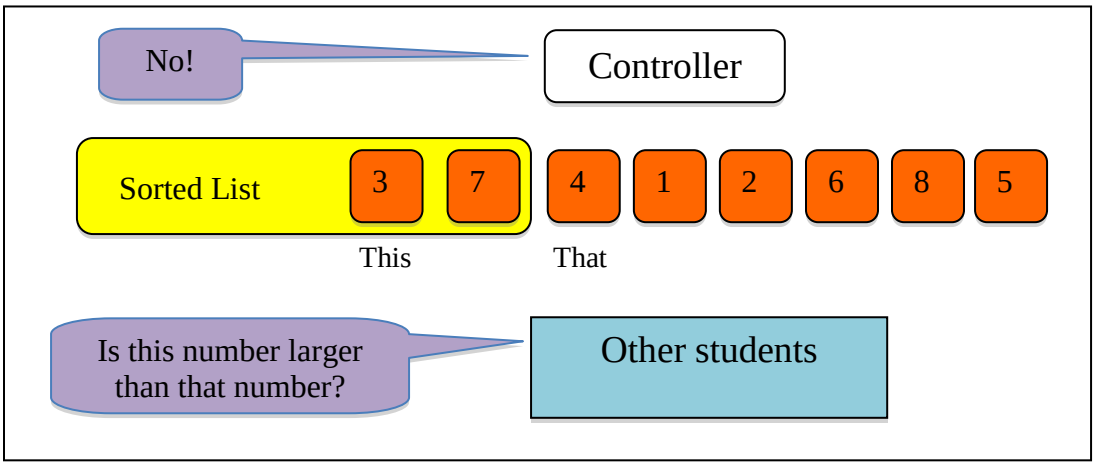
This is probably the first occasion that the students will be introduced to the concept of **iteration**. They need to iterate over all the elements on the left in order to find the correct placement for the out-of-order element. This is a very useful way of thinking, and should be constantly referred to while teaching the following algorithms.

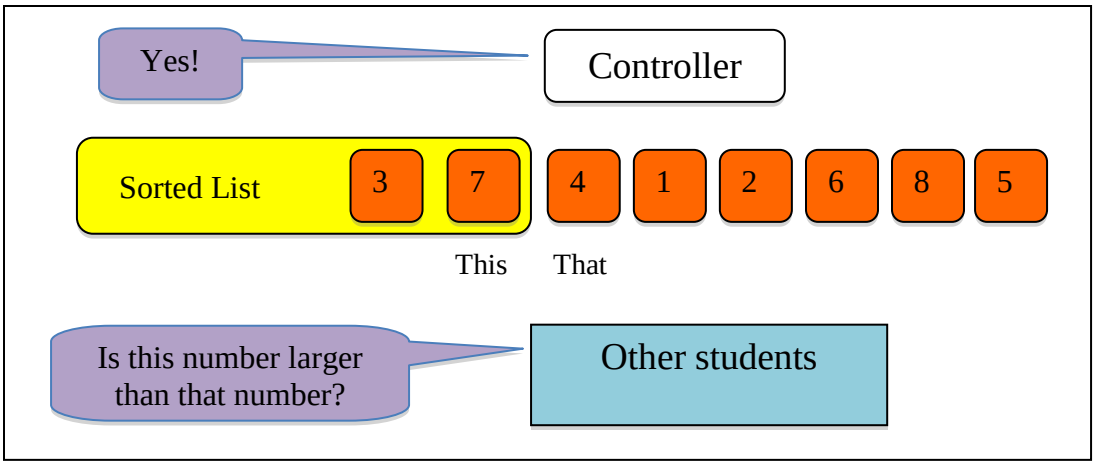
2. *Insertion Sort.* The list should now be arranged randomly. The students should now attempt to sort it using their prior knowledge of Single Insertion.

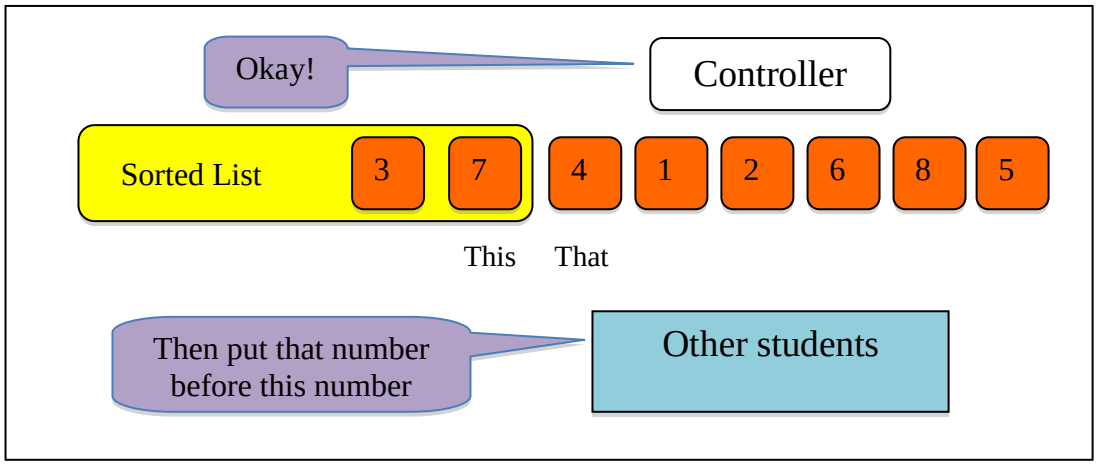
   The problem can be broken down into a series of Single Insertions. **A lone number by itself is a sorted list.** Hence we can say that the single number on the extreme left of the randomized list is a sorted list. Now, we consider this 1-element sorted list, and do an Insertion Sort treating the second element as being the out-of-place number in Single Insertion. Once done, we now have a 2-element sorted list. The procedure is then repeated treating the 3rd element as the out-of-place number, so that we get a 3-element sorted list. We keep on repeating this sequence of Single Insertions, until we finally get our entire sorted list.

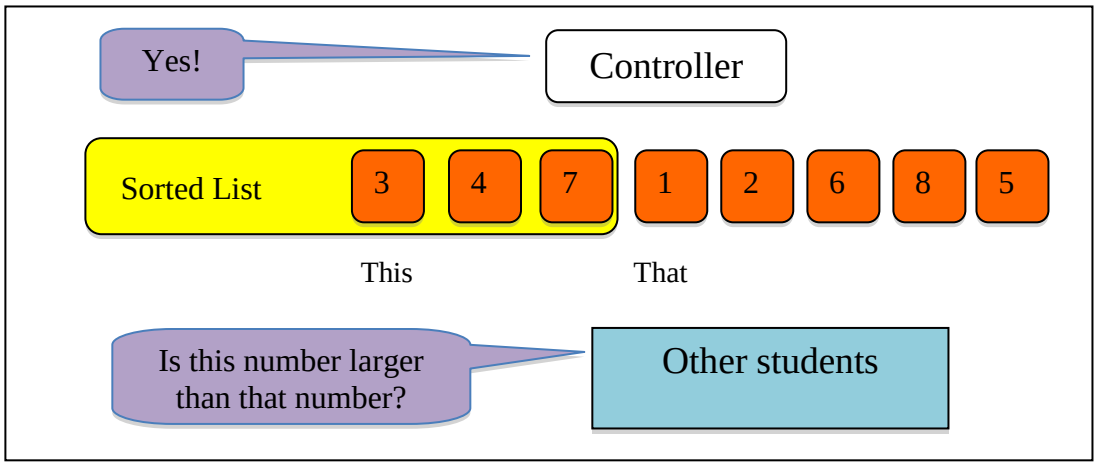
The following sequence of diagrams show what the students should ideally attempt. Notice how the yellow box, which denotes the sorted portion of the list, grows over time. Note that Single Insertion is carried out repeatedly using the yellow part of the list as the sorted list, and the next element on the right as the out-of-place number. The procedure will eventually sort the whole list!

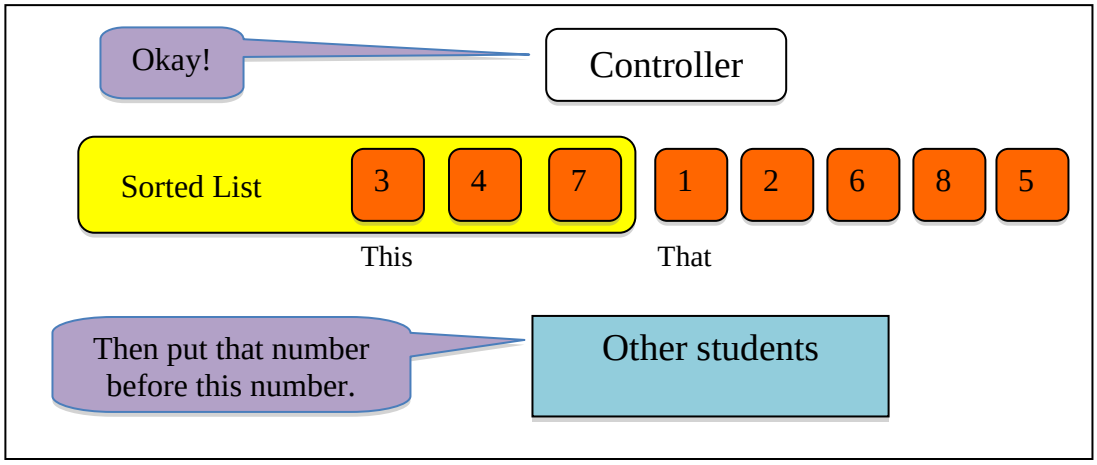


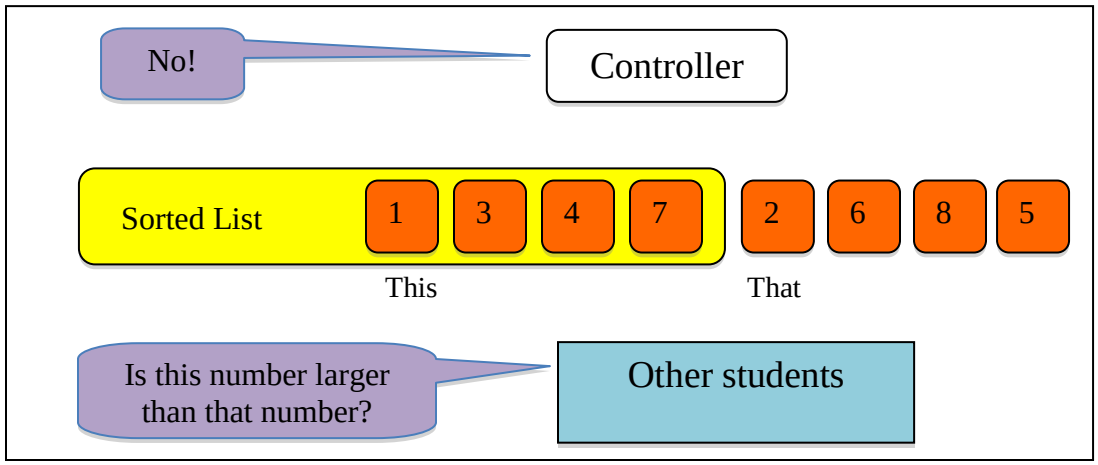


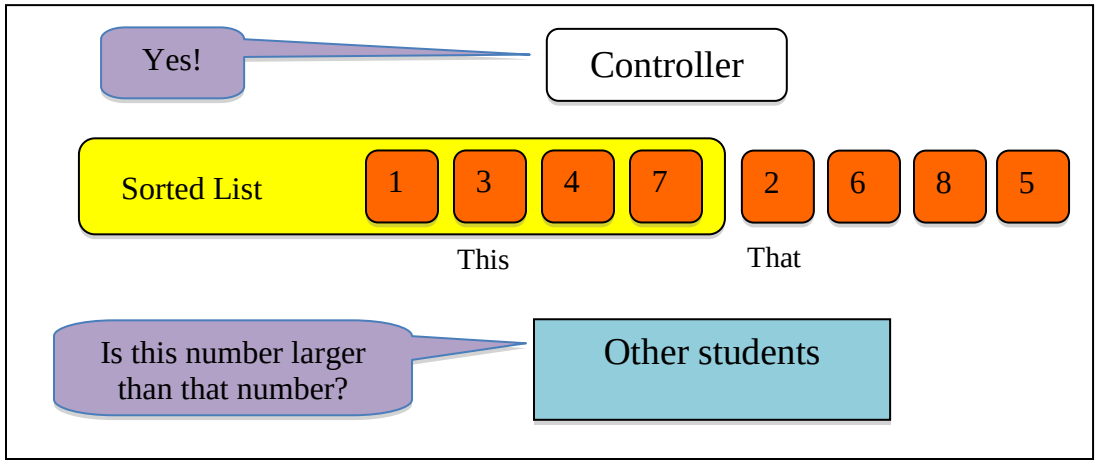


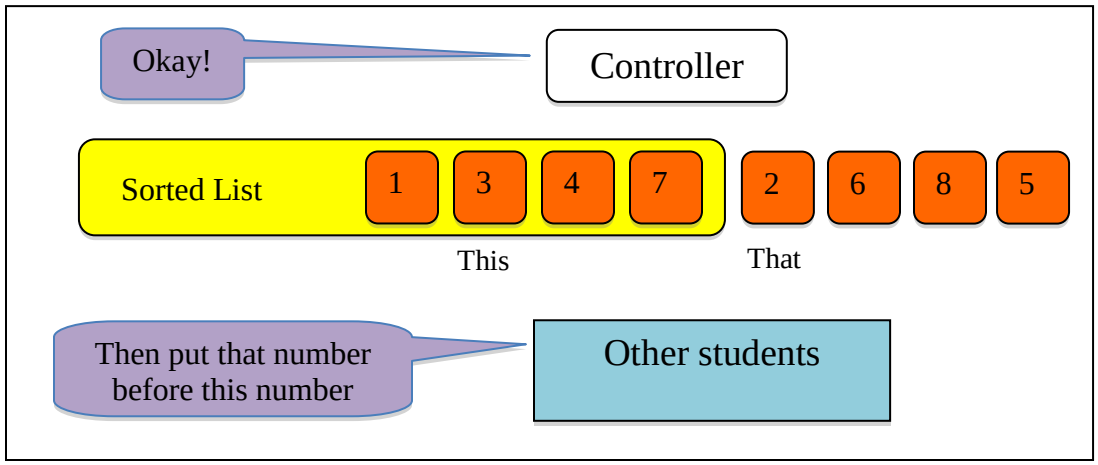

Okay!
Controller
Sorted List
3
7
4
1
2
6
8
5
This
That
Then put that number before this number
Other students

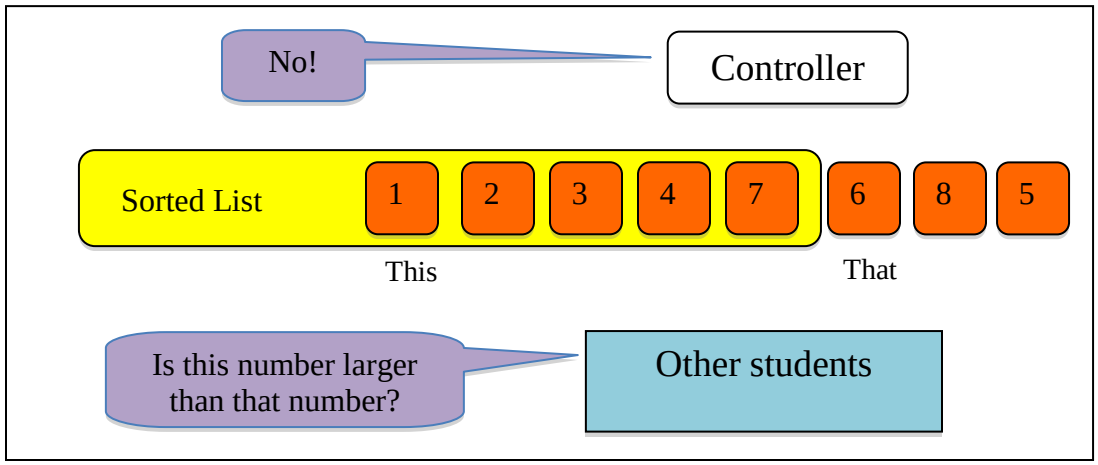

Yes!
Controller
Sorted List
3
4
7
1
2
6
8
5
This
That
Is this number larger than that number?
Other students


Okay!
Controller
Sorted List
3
4
7
1
2
6
8
5
This
That
Then put that number before this number.
Other students


No!
Controller
Sorted List
1
3
4
7
2
6
8
5
This
That
Is this number larger than that number?
Other students


Yes!
Controller
Sorted List
1
3
4
7
2
6
8
5
This
That
Is this number larger than that number?
Other students

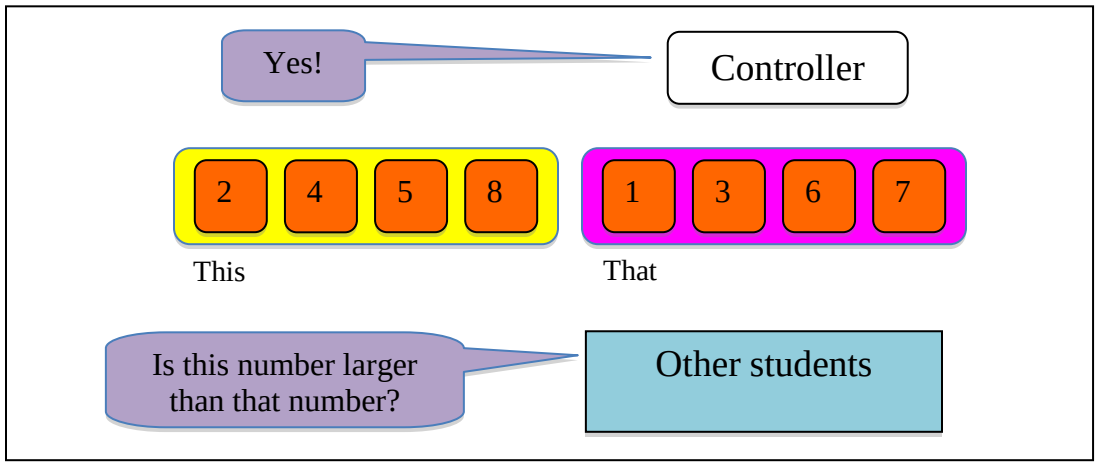

Okay!
Controller
Sorted List
1
3
4
7
2
6
8
5
This
That
Then put that number before this number
Other students

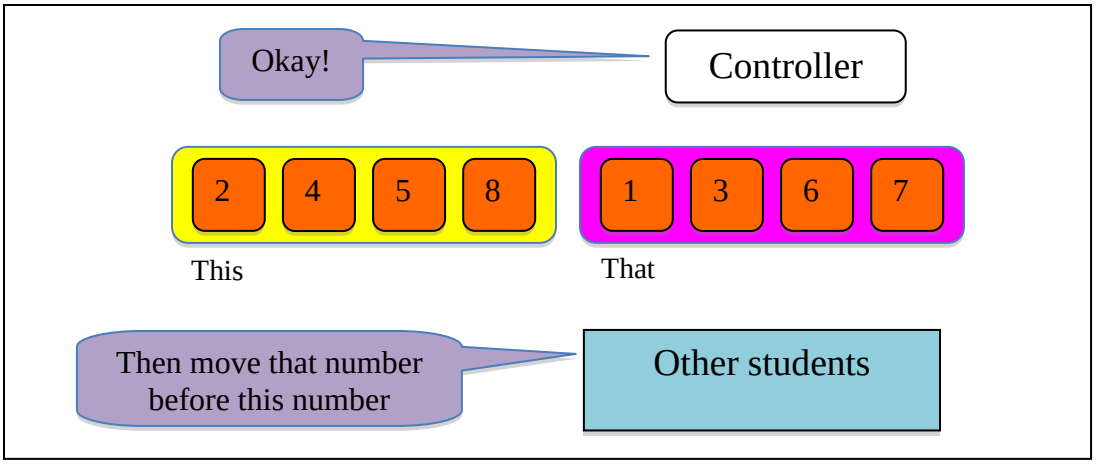


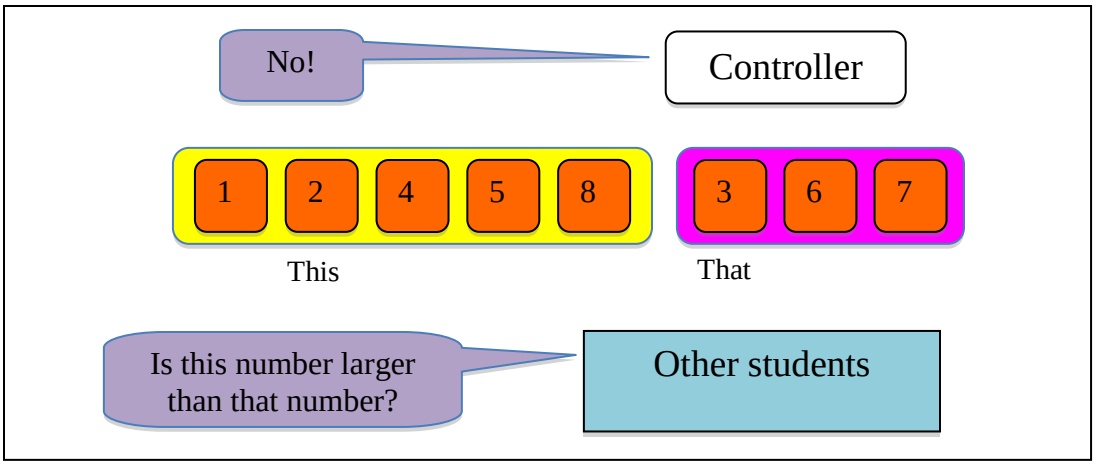
The instructor should refrain from simply giving out the solution to the students. Rather, the instructor should encourage students to think out loud, and ask the students questions so that they may themselves "discover" the algorithm in class.

3. *Two-List Merge.* Set up the initial list as follows: Divide the 8 objects into 2 lists of 4 objects each, and the two lists should be independently sorted. The lists should then be placed side by side, and the students should now attempt to “merge” the two sorted lists into one sorted list by following the rules of the game. This operation, Two-List Merge, is a useful starting point and building block for a more sophisticated algorithm: Merge Sort.

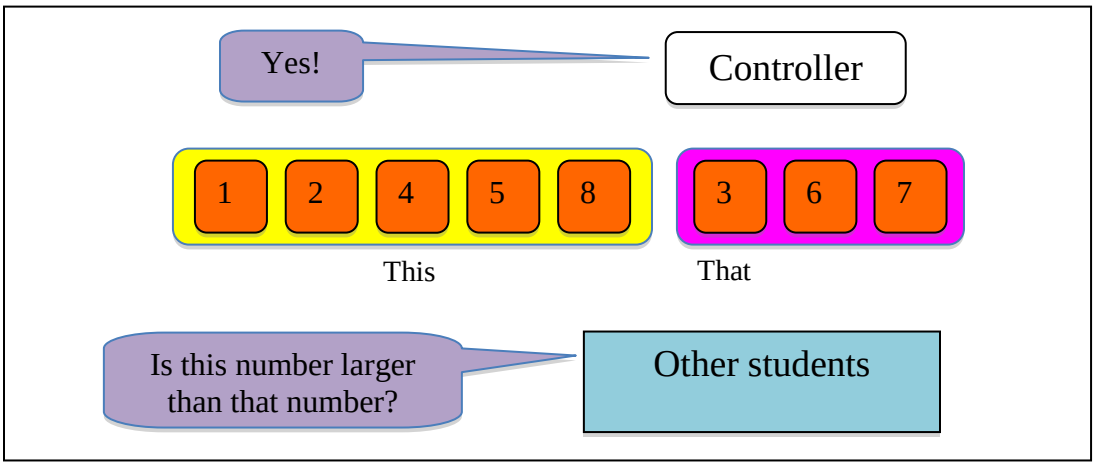
   This also requires students to recall and reuse their knowledge of Single Insertion. The list on the left is sorted, so you can treat the 5th number as the out-of-place number and sort it within the left list. However, there is one bit of cleverness that the students must discover for themselves. They must take advantage of the fact that the list on the right is also sorted, so the next number to be inserted is always greater than the last number inserted. Therefore, when they want to merge the second element in the right list into the left list, they really do not need to re-run Single Insertion from the extreme left, but simply start from the point at which they inserted the last element.

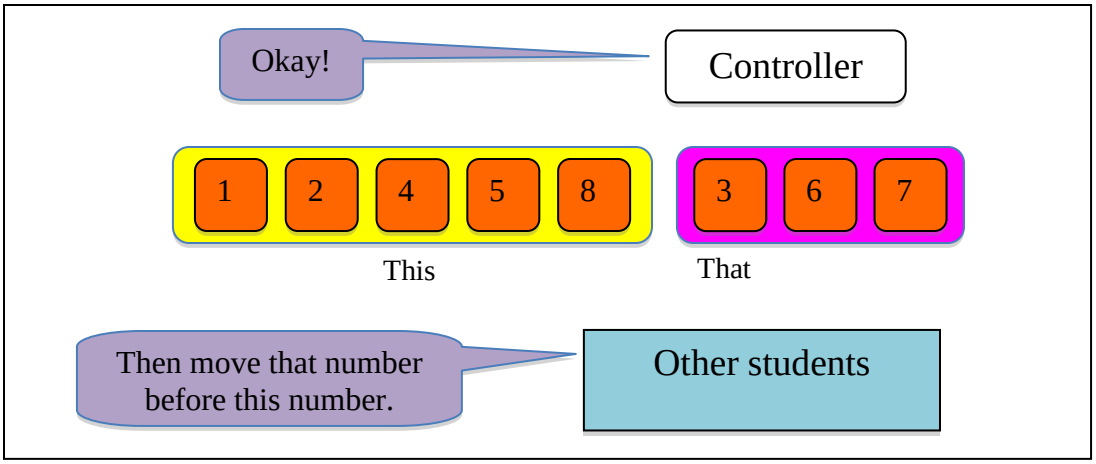


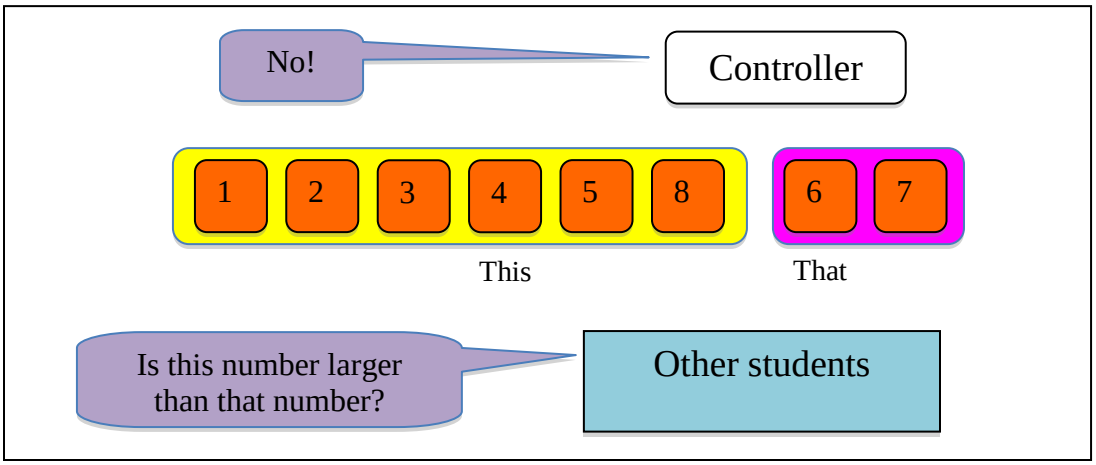


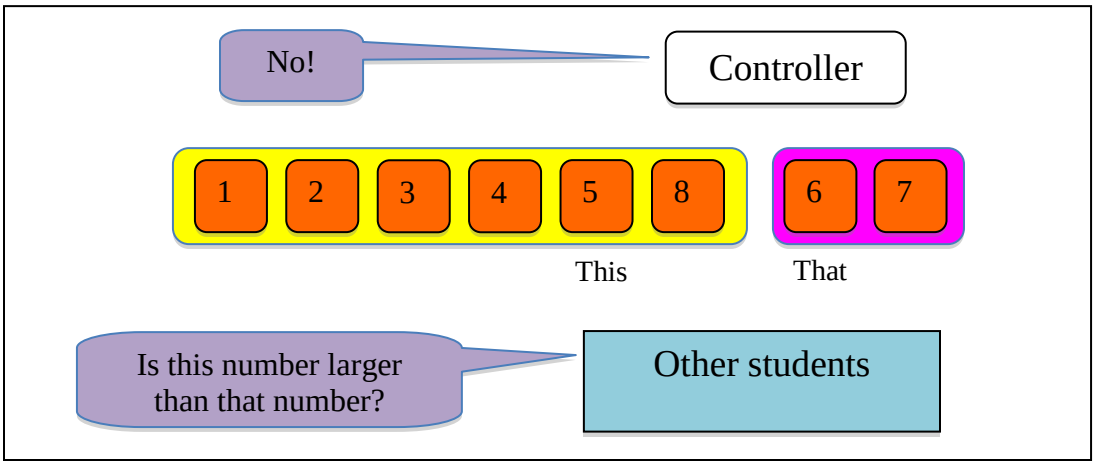


Note that in the step above, we did not start the questions all over from the start of the yellow list, which is what we did in Insertion Sort. This is because we know that the next number to be inserted in the yellow list is already larger than the number inserted last, since the pink list was already sorted.




Okay!
Controller
1
2
4
5
8
3
6
7
This
That
Then move that number before this number.
Other students


No!
Controller
1
2
3
4
5
8
6
7
This
That
Is this number larger than that number?
Other students


No!
Controller
1
2
3
4
5
8
6
7
This
That
Is this number larger than that number?
Other students

4. *Merge Sort.* The list should now be arranged randomly. The students should now attempt to sort the list using their previous knowledge of Two-List Merge. The main idea behind Merge Sort is the principle of Divide and Conquer. It is tricky to teach it to pre-university students, but it is one of the fundamental concepts behind every modern-day sophisticated sorting algorithm, and so deserves a place in this lesson plan.

   The students will need to consider each number as an individual sorted list of size 1. From there, students will attempt to perform the known Two-List merge on the lists. However, at first sight, what students will attempt will be not very different from Insertion Sort. They will first merge two numbers and then a third and then a fourth, and so on. This is essentially Insertion Sort. A small bit of guidance from the instructor might be called for. The trick to Merge Sort is the following:

   - From the randomized list, treat every number as a sorted list of size 1.
   - Then, form pairs of adjacent lists and merge them, so that we now have 4 sorted lists, each of size 2.
   - Then, form list pairs again and merge, to get 2 lists, each of size 4.
   - One final merge, and we get a complete sorted list of size 8.

   This approach ensures that iterations over each individual number are minimized, so the number of yes/no questions asked is also minimized. It is this distinction that should be appreciated by the students, and how a small piece of cleverness can drastically change the performance of an algorithm or procedure.

   The instructor should NOT give away the solution to this problem straight away. This is a problem that, if solved by the students themselves, opens up a whole new world of possibilities and ways of thinking about a problem, and most importantly, the benefits of Divide and Conquer, which is not just an important principle in Computer Science and Algorithm Design, but is applicable in many areas in the natural sciences (Physics, Chemistry and Biology) as well.

Section III – Tips and Troubleshooting

Throughout the exercises, students should be given every opportunity to come up with the answers themselves. The purpose of the game is for students to “play around” with the problem and see for themselves what works and what doesn’t. This form of discovery-based learning is extremely effective at developing students’ high-level cognitive reasoning and problem-solving skills. Discussions among students should also be encouraged

Using the blackboard (or whiteboard) to draw pictures and diagrams can be highly effective in communicating this particular topic to the students. The graphics provided in the teacher resources can be used as a guide on how to draw the different steps on the board. Students can also be encouraged to step up to the board, and draw diagrams depicting their proposed algorithms and solutions.

If students are having difficulty getting started, here are some tips on how to get students on the right track:

- Encourage students to discuss the task at hand. More often than not, students will inadvertently come up with solutions themselves, when they "think out loud" and discuss the problem and/or its solution with their peers.
- Jump-start the discussion by asking students about their viewpoints, and encouraging students with differing viewpoints to engage in active discussions.
- Offer a potential solution and ask students what they think would happen if it were to be tried. Such a question will most likely trigger a new thought process in the students' minds, and help them see where they were getting off track. The question itself might refer to what the class was already discussing, or it could be an entirely new, but not necessarily correct thought. The aim is not to provide the correct answer, but encourage students to evaluate possible solutions and think and reason for themselves.

Utilize the inter-team competitions effectively. Try and draw a correlation between the number of questions asked versus the time taken to sort the list, and whether there was a correlation between winning and asking less questions.

Time Needed

- Single Insertion and Insertion Sort should be held together in a 90-minute session.
  - 10 minutes - Introduction
  - 20 minutes - Single Insertion Demo and Discussion
  - 20 minutes - Insertion Sort Demo and Discussion
  - 40 minutes - Group exercises, worksheets and inter-group sorting races
- Two-List Merge and Merge Sort should be held together in a 90-minute session.
  - 10 minutes - Insertion Sort Recap and Review
  - 20 minutes - Two-List Merge Demo and Discussion
  - 20 minutes - Merge Sort Demo and Discussion
  - 40 minutes - Group exercises, worksheets and inter-group sorting races

Fun with Sorting

Student Resource: Game Rules

1. There is a list of 8 objects in front of you. Each object has a number on it. Your job is to sort the entire list in ascending order without knowing what those numbers actually are.
2. There is a controller who is the only person who can see what the numbers are, and who can actually move the numbered objects around. It is your job to give the controller very specific instructions as to how to do it.
3. You can only ask the controller certain types of questions, or give certain types of commands. There are three things you can do:
    a. Compare two numbers. Ask the controller, “Is this number larger than that number?” and very directly point out the two numbers you want to compare. The controller is only allowed to answer with a “yes” or “no”.
    b. Tell the controller to swap two objects. The controller will then swap the objects with each other. The controller must be careful not to reveal the hidden numbers in the process.
    c. Tell the controller to directly move an object to a specified position. For example, “Put this object after that object”. The controller will then follow your command and move the object to the specified position.
4. If you think the numbers are sorted, tell the controller to reveal all the numbers. If the list is sorted, congratulations! If not, the controller will mix them up, and you can start over.
5. The object of the game is to sort the list with the minimum number of yes/no questions asked.

◆ Single Insertion

In this first exercise, the list is sorted already, except for one single number at the very end. You need to come up with a procedure to insert the out-of-place element into the pre-sorted list.

It's really not that hard a problem if you think about it. Remember to follow the rules of the game very strictly, and SPEAK UP if you're unsure about what to do. Discuss with your classmates. Ask your instructor. Remember, this is a think-out-loud session. You're playing a game. You don't play games sitting down quietly. Your instructor will help you out with this if you are unsure.

Be sure to get understand this problem thoroughly. This is a fundamental "building block", so to say, that will be vital in the following exercises, so be sure to get it right.

◆ Insertion Sort

This problem gets trickier. You now have a list that is completely random, and you need to sort it.

Stumped? Perhaps the following points will help an idea click:

- A single number by itself is a sorted list. Think about it! It's a list, that has only one number in it, and so, by itself, it qualifies as being a "sorted list".
- You can do Single Insertion on a list of ANY size (the "size" of a list is how many numbers it has) and ONE out-of-place element, so that you have a complete sorted list.
- Observe what happened to the size of the sorted list when you did Single Insertion. How many numbers were sorted before Single Insertion, and how many numbers were sorted after you carried out that procedure.

◆ Two-List Merge

Okay, so now you're given two separately sorted lists, and your job is to "merge" the two lists into one complete sorted list. Once again, your knowledge of Single Insertion will prove key to this task, and you will, along the way, discover a neat little trick, and how you can make use of the fact that both lists are pre-sorted.

Points to consider:

- A single number is a sorted list on its own (of size 1). Therefore, Single Insertion is also a kind of Two-List Merge, in which one list had size 7 and the other list had size 1.
- The aim is to sort asking the least amount of yes/no questions.
- Insertion Sort works for ANY list, so it will work here as well. However, you would not be making use of the fact that the two lists are pre-sorted for you already.
- You'll probably be surprised at how you can cut down on your questions if you get clever!

Think carefully about this problem. You need to do something like Insertion Sort, but you can be more efficient because the elements you are inserting are not random. They are nicely ordered. So use that fact to decide at each step which numbers you need to compare, and, more importantly, which numbers you DON'T need to compare. This is very important as the aim of the game is to use the smallest number of yes/no questions.

◆ Merge Sort

This is a very important application of sorting. Modern computers sort large amounts of data using the procedure you are going to devise next. It's called Merge Sort, and you will create it yourself!

The list given to you is random. You know Insertion Sort will do the trick, and so you're tempted to try that. However, there is another approach which looks completely non-intuitive, weird and gibberish, but will end up working out much better than Insertion Sort. Consider the following points:

- You've already seen how you can merge two sorted lists into one sorted list.
- Each number by itself is a sorted list of size 1.
- Try merging large lists together. See if it helps, and what happens if you do. In Insertion Sort, all you were doing was doing Two-List merge in which one of the lists was always of size 1. Try changing that, and see how things shape up.

Fun with Sorting

Student Worksheet:
Session 1 – Insertion Sort

This is a group exercise. Before starting, please form a group of 3-5 students.

◆ Single Insertion

Draw up 8 cards, each with a different number. Have one student in your group be the controller, and the others should play along. Only the controller is allowed to see the numbers.

The controller should set up the cards for Single Insertion, and the others should play to sort the list. Do this at least 5 times, and fill up the table below.

Round	Number of yes/no questions asked	What was the out-of-place number?
1		
2		
3		
4		
5		

Based on the information you entered above, answer the following questions:

1. What is the average number of questions asked in one round?

   _____

2. Assume that you were to fill up the table above for 5000 Single Insertions. What do you think would be the largest possible value in the second column (The number of yes/no questions asked)?

   _____

3. Assume that you were working with a list of 10 numbers instead of 8. What would be your answer to Question 2 in that case?

   _____

4. Can you now relate the size of the list to the number of questions asked in the worst case? Give a very brief answer about how you think the size of the list and the questions asked in the worst case (Remember, the fewer the number of questions, the better. So when we say “worst case”, we mean the largest possible number of questions asked)

_____

_____

_____

_____

◆ Insertion Sort

Now, with those same cards, play the Insertion Sort game. Remember to start off with a completely random list.

Do Insertion Sort at least 5 times, and fill out the table below. The controller should keep a record of the third column before starting the sorting, and reveal it after the list has been sorted.

Round	Number of yes/no questions asked	What was the starting list?
1		
2		
3		
4		
5		

Based on the information you entered above, answer the following questions:

1. What is the average number of questions asked in one round?

_____

2. You must have realized that Insertion Sort is just a series of Single Insertions. Give a very brief answer about how you would relate the number of Single Insertions with the size of the list to be sorted.

_____

_____

_____

_____

3. Remember your answer to Question 2 from the Single Insertion exercise. Use that answer to calculate the number of questions you would need to ask in Insertion Sort in the “worst case”. Show your steps.

Final Answer: _____

4. Go head-to-head with another group in a sorting match to see who can sort a randomized list faster. There will be one controller from each team, in the interest of fairness. The game rules should be strictly followed, but you may use any procedure you want.

   Number of questions you asked: _____

   Number of questions your opponent asked: _____

   Did you win? _____

Fun with Sorting

Student Worksheet:
Session 2 – Merge Sort

This is a group exercise. Before starting, please form a group of 3-5 students.

◆ Two-List Merge

Draw up 8 cards, each with a different number. Have one student in your group be the controller, and the others should play along. Only the controller is allowed to see the numbers.

The controller should set up the cards for Two-list Merge, and the others should play to sort the list. Do this at least 5 times, and fill up the table below.

Round	Number of yes/no questions asked	What was the starting list?
1		
2		
3		
4		
5		

Based on the information you entered above, answer the following questions:

1. What is the average number of questions asked in one round?

   _____

2. Assume that you were to fill up the table above for 5000 Two-list Merges. What do you think would be the largest possible value in the second column (The number of yes/no questions asked)?

   _____

3. Assume that you were working with a list of 10 numbers instead of 8. What would be your answer to Question 2 in that case?

   _____

4. Now go back and have a look at the answers you gave for the exercise on Single Insertion. What similarities do you see? What differences? How do the numbers vary? Do they vary a little, or do they vary a lot? Write down your thoughts on this.

_____

_____

_____

_____

_____

_____

_____

_____

_____

◆ Merge Sort

Now, with those same cards, play the Merge Sort game. Remember to start off with a completely random list.

Do Merge Sort at least 5 times, and fill out the table below. The controller should keep a record of the third column before starting the sorting, and reveal it after the list has been sorted.

Round	Number of yes/no questions asked	What was the starting list?
1		
2		
3		
4		
5		

Based on the information you entered above, answer the following questions:

1. What is the average number of questions asked in one round?

   _____

2. You must have realized that Merge Sort is just a series of Two-list Merges. How many Two-list Merges did you do to Merge Sort the list of 8 numbers?

   _____

3. Remember your answer to Question 2 from the Two-list Merge exercise. Use that answer to calculate the number of questions you would need to ask in Merge Sort in the “worst case”. Show your steps.

   Final Answer: _____

4. Go head-to-head with another group in a sorting match to see who can sort a randomized list faster. There will be one controller from each team, in the interest of fairness. The game rules should be strictly followed, but you may use any procedure you want.

   Number of questions you asked: _____

   Number of questions your opponent asked: _____

   Did you win? _____

Fun with Sorting

Teacher Resource: Alignment to Curriculum Frameworks

Note: Lesson plans in this series are aligned to one or more of the following sets of standards:

- U.S. Science Education Standards (http://www.nap.edu/catalog.php?record_id=4962)
- U.S. Next Generation Science Standards (http://www.nextgenscience.org/)
- International Technology Education Association's Standards for Technological Literacy (http://www.iteea.org/TAA/PDFs/xstnd.pdf)
- U.S. National Council of Teachers of Mathematics' Principles and Standards for School Mathematics (http://www.nctm.org/standards/content.aspx?id=16909)
- U.S. Common Core State Standards for Mathematics (http://www.corestandards.org/Math)
- Computer Science Teachers Association K-12 Computer Science Standards (http://csta.acm.org/Curriculum/sub/K12Standards.html)

◆ Principles and Standards for School Mathematics

As a result of activities, all students should develop

Problem Solving Standard

- ✦ Apply and adapt a variety of appropriate strategies to solve problems.
- ✦ Solve problems that arise in mathematics and in other contexts.

◆CSTA K-12 Computer Science Standards Grades 3-6 (ages 8-11)

5.1 Level 1: Computer Science and Me (L1)

- ✦ Computational Thinking: (CT)

  2. Develop a simple understanding of an algorithm (e.g., search, sequence of events, or sorting) using computer-free exercises.

  4. Describe how a simulation can be used to solve a problem.

◆CSTA K-12 Computer Science Standards Grades 6-9 (ages 11-14)

5. 2 Level 2: Computer Science and Community (L2)

- ✦ Computational Thinking: (CT)

  3. Define an algorithm as a sequence of instructions that can be processed by a computer.

  4. Evaluate ways that different algorithms may be used to solve the same problem.

  5. Act out searching and sorting algorithms.

  8. Use visual representations of problem states, structures, and data (e.g., graphs, charts, network diagrams, flowcharts).

◆ Standards for Technological Literacy – All Ages

The Nature of Technology

- ✦ Standard 2: Students will develop an understanding of the core concepts of technology.