

Activity 7

Lightest and heaviest—*Sorting algorithms*

Age group Early elementary and up.

Abilities assumed Using balance scales, ordering.

Time 10 to 40 minutes.

Size of group From individuals to the whole classroom.

Focus

Comparing.

Ordering.

Inequalities.

Summary

Computers are often used to put lists into some order, whether alphabetic, numeric, or by date. If you use the wrong method, it can take a long time to sort a large list into order, even on a fast computer. Fortunately several fast methods are known for sorting. In this activity children will encounter different methods for sorting, and see how a clever method can perform the task much more quickly than a simple one.

Technical terms

Sorting algorithms; insertion sort; selection sort; quicksort; recursion; divide and conquer; merging; mergesort; insertion sort; bubble sort.

Materials

Each child or group of children will need:

a set of about eight containers of the same size but different weights (e.g. milk cartons or film canisters filled with sand), and

balance scales.

What to do

In computer science, the term *sorting* usually refers to putting a list into alphabetical or numeric order. This is different from the common meaning in schools, where sorting involves placing objects into categories, or grouping identical objects together.

1. Discuss the computer science meaning of sorting, and see if the children can think of places where putting things into order is important (such as names in the telephone book, entries in a dictionary, the index of a book, the books on the shelves in a library, the letters in a postal worker's bag, a class roll, names in an address book, a list of files on a computer). Have the children think about the consequences if these things were not in order (usually the problem is that it takes a long time to locate an object in an unsorted list).

Point out that sorting lists helps us find things quickly, and also makes extreme values easy to see. If you sort the marks for a class test into order, the lowest and highest marks become obvious.

2. The children will be using balance scales with a set of about eight containers that are of different weights, but look identical. Having identical containers ensures that they must compare the weights using the scales instead of visually. If the weights can be compared simply by picking them up, assign a person for each balance scale to handle the weights under the direction of the children who are making the decisions. The only clues available to the children should arise from comparisons between pairs of weights on the scales.

It can be helpful for the teacher to keep track of the order of the weights by writing a code on each one that the children won't be able to interpret. For example, if you know the letters of the Greek alphabet you could label the containers from lightest to heaviest with α , β , etc.

Check that the children can use the balance scales to find the lightest and heaviest of two objects.

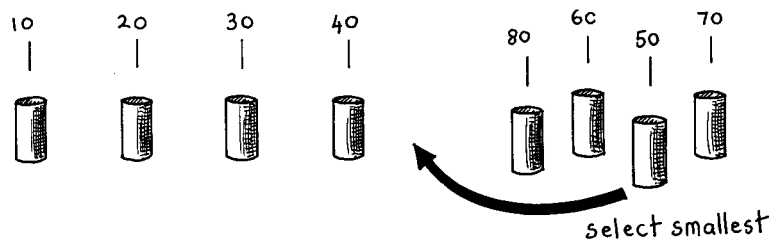


Figure 7.1: Sorting weights using the selection sort method

3. Have the children find the lightest of the eight or so weights they have been given, using only the balance scales. (The best way to do this is to go through each object in turn, keeping track of the lightest one so far. That is, compare two objects, and keep the lighter one. Now compare that with another, keeping the lighter from the comparison. Repeat until all the objects have been used.)
4. Discuss how you would check whether the weights are sorted into ascending order. (Check that the first object is lighter than the second, the second is lighter than the third, and so on. If each pair is in order then the whole list must be in order.)
5. Have the children sort three containers into ascending order of weight, using only comparisons on the balance scales. This can easily be done with three comparisons, and sometimes just two will suffice—if the children realize that the comparison operator is transitive (that is, if A is lighter than B and B is lighter than C, then A must be lighter than C).
6. Have the children sort all of the objects (about eight) into ascending order, using whatever strategy they wish. This might be very time consuming. When they think they have finished, check their ordering by comparing each adjacent pair of objects on the balance scales.
7. Have the children sort their weights into order using the following method, which is called *selection sort*. They should count how many comparisons they make to sort the objects.

This is how selection sort works. Find the lightest weight in the set using the method described above, and put it to one side. Next, find the lightest of the remaining weights, which belongs next in ascending order of weight, and remove it. Repeat this until all the weights have been removed. Figure 7.1 shows the fifth weight being selected and added to the end of the sorted list.

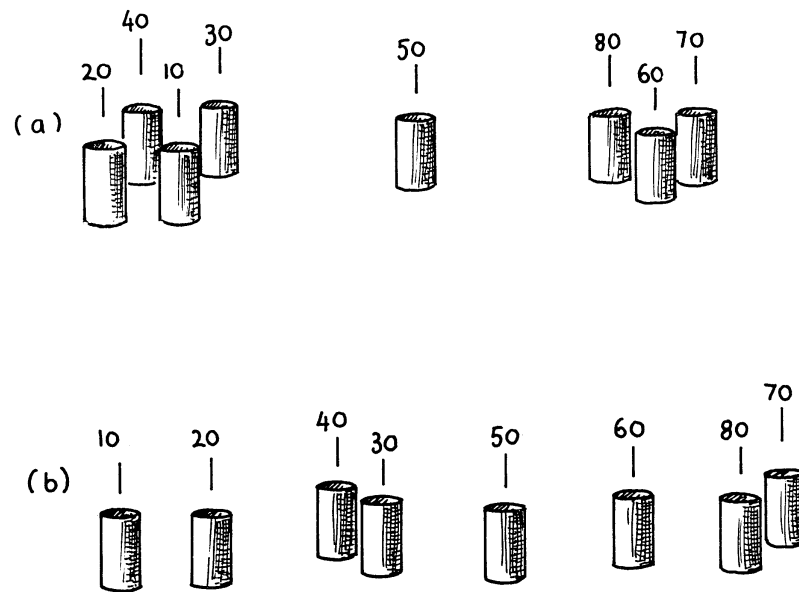


Figure 7.2: Sorting weights using the quicksort method

8. If the children have sufficient mathematical background, they should be able to calculate how many comparisons are required to sort a group of objects using this method. (To find the minimum of n objects requires $n - 1$ comparisons. For example, to find the minimum of two objects requires only one comparison, and to find the minimum of five objects requires four comparisons. To sort eight objects using selection sort, there will be seven comparisons to find the smallest one, six to find the next smallest, five for the next, and so on, giving a total of $7 + 6 + 5 + 4 + 3 + 2 + 1 = 28$ comparisons.)
9. Have the children try out the following method of sorting, which is called *quicksort*. As its name might imply, quicksort is a lot faster than selection sort, particularly for larger lists. In fact, it is one of the best methods known. Have the children count how many comparisons they need to make when they sort their objects using quicksort.

This is how quicksort works. Choose one of the objects at random, and place it on one side of the balance scales. Now compare each of the remaining objects with it, and put them in one of two groups, those that are lighter, and those that are heavier. Put the lighter group on the left, the chosen object in the middle, and the heavier ones on the right (it is possible that there may be no objects in one of the groups). Now repeat this procedure on each of the groups—that is, use the balance to divide the groups into subgroups. Keep repeating on the remaining groups until no group has more than one object in it. Once all the groups have been divided down to single objects, the objects will be in ascending order.

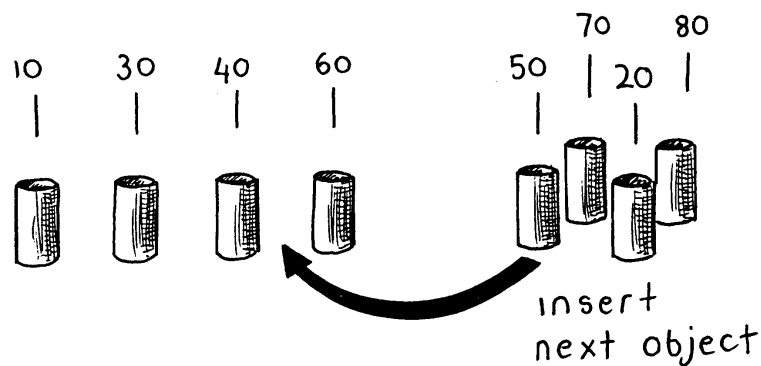


Figure 7.3: Sorting weights using the insertion sort method

Figure 7.2 shows how this process might go. In Figure 7.2a the 50 gram weight has been chosen at random, and the balance was used to group the lighter weights to the left and the heavier ones to the right. In Figure 7.2b the lighter group has been split using the 20 gram weight, and the heavier one was split using the 60 gram weight. In both cases the split is uneven, and in the second case one of the two subgroups has no objects in it! Once the two groups of two objects are split up—which is trivial because they are so small—the sorting will be complete.

In practice the way that the objects are split up will depend on which objects are chosen at random. For example, if the 80 gram weight were chosen as the first one to divide the groups in Figure 7.2, the groups would be very unbalanced. Nevertheless, the list will always end up in increasing order. Unbalanced splits simply increase the number of comparisons required to complete the sorting.

10. Compare the number of comparisons that were made for selection sort and quicksort. If different groups have performed the experiment, have them share their results with the whole class. (Using selection sort on 8 items will always take 28 comparisons. In the example in Figure 7.2, quicksort used only 14 comparisons to sort the 8 items. Sometimes it will take more, depending on which weights are chosen as the splitting point. In the worst case, if the randomly chosen weight always happens to be the lightest or heaviest, it will take the same number of comparisons as selection sort, but this is unlikely. It will never be worse than selection sort, and usually a lot better.)

Variations and extensions

Many different methods for sorting have been invented. More advanced classes may wish to try sorting weights using other methods. In addition to selection sort and quicksort, the following methods are common.

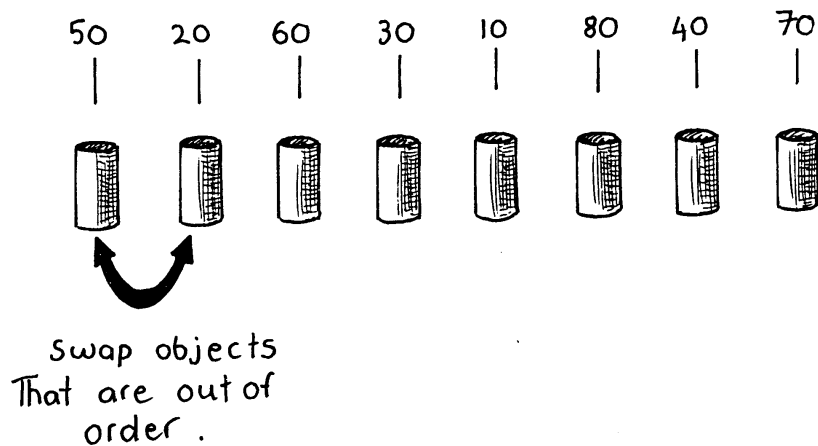


Figure 7.4: Sorting weights using the bubble sort method

Insertion sort works by removing each object from an unsorted group and inserting it into its correct position in a growing list (see Figure 7.3). With each insertion the group of unsorted objects shrinks and the sorted list grows, until eventually the whole list is sorted. This method is often used by card players to sort a hand into order.

Bubble sort involves repeatedly going through the list, swapping any adjacent objects that are in the wrong order (Figure 7.4). The list is sorted when no swaps occur after a pass through the list. This method is not very efficient, but some people find it easier to understand than the others.

Mergesort is one of the fastest known sorting methods, even though it might sound like hard work. The objects are distributed randomly into two groups of equal size (or nearly equal size if there is an odd number of objects). Each of these half-size groups is then sorted into order (we won't worry yet about how this is done), and then the two groups are merged together. Merging two sorted lists together is easy: you repeatedly remove the smaller of the two objects at the front of the two lists. In Figure 7.5 the 40 and 60 gram weights are at the front of the two lists respectively, so the 40 gram one is removed and added to the end of the final sorted list. This leaves the 50 gram weight at the front of the first list, which will be compared with the 60 gram weight and removed next.

There remains the question of how to sort the two half-sized lists. Simple—they are sorted using mergesort! This means they will each be divided into quarter-sized lists, sorted, and merged back to give a sorted half-sized list. Each time a list is divided into two, it is sorted using mergesort, unless it only contains one item. In that case it is already sorted, and so nothing needs to be done. Eventually, all the lists will be divided down to individual items, and so there is no danger of repeatedly having to perform mergesort *ad infinitum*.

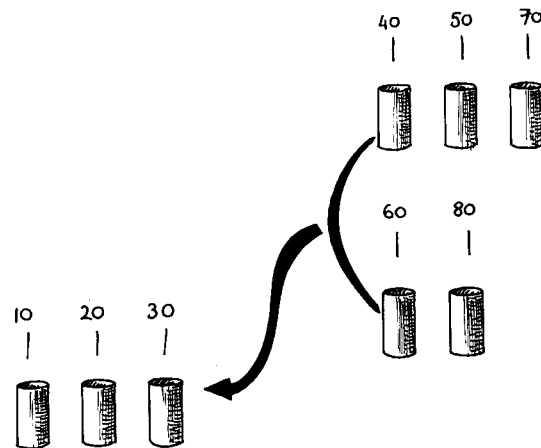


Figure 7.5: Sorting weights using the mergesort method

The relative efficiency of each sorting method becomes more obvious when a larger number of objects (two dozen or more) are being sorted. Write random numbers on about 50 cards, and have children try to sort them into increasing order using selection sort and quicksort. Selection sort will require 1225 comparisons of the values, whereas quicksort will require only about 271 on average. This is quite a saving!

Older children may enjoy a short cut for adding up the number of comparisons that selection sort makes. We pointed out above that n objects will take $1 + 2 + 3 + 4 \cdots + n - 1$ comparisons to sort. Adding up these numbers is made easy by regrouping them. For example, to add up the numbers $1 + 2 + 3 + \cdots + 20$, regroup them as

$$\begin{aligned} & (1 + 20) + (2 + 19) + (3 + 18) + (4 + 17) + (5 + 16) + \\ & (6 + 15) + (7 + 14) + (8 + 13) + (9 + 12) + (10 + 11) \\ = & 21 \times 10 \\ = & 210. \end{aligned}$$

In general, the sum $1 + 2 + 3 + 4 \cdots + n - 1 = n(n - 1)/2$.

Another approach to sorting is described in Activity 8 on sorting networks.

What's it all about?

We are used to working with ordered lists. Telephone directories, dictionaries and book indexes all use alphabetical order, and life would be far more difficult if they didn't. Information is much easier to find in a sorted list. Also, if a list of numbers (such as a list of expenses) is sorted into order, the extreme cases become apparent because they are at the beginning and end of the list. Duplicates are also easy to find, because they end up adjacent.

Sorting is an important task that computers perform frequently. In the 1970s it was estimated that most computers spent a quarter of their time doing sorting—and for some it was over half.¹ A good deal of computer output is sorted: lists of names and address need to be in alphabetical order, and files displayed on the screen are often shown alphabetically, or in order of when they were created or last modified. Unsorted output can be confusing because people expect the order to be of some significance—it looks more professional if a list of names is printed in alphabetical order, even when it is not strictly necessary.

Using a balance scale for sorting weights is an accurate model of what happens on computers, because the basic operation in sorting is to compare two values to see which is the larger. Most sorting programs are based on this operation, and their efficiency is measured in terms of how many comparisons they require to sort a list, because that is a good indicator of how long they will take regardless of the computer they run on.

Researchers have worked for decades on the sorting problem, and literally dozens—if not hundreds—of methods have been invented. Those described above are the most common, and many others are related to them. They can be divided into two groups, those that are relatively slow (insertion sort, selection sort, and bubble sort), and those that are fast (quicksort and mergesort). The slower ones are only useful for special situations, and are just too slow if there is a large amount of data to be sorted. Quicksort and mergesort are standard methods on many computers.

Both quicksort and mergesort introduce an important and powerful concept in computer science: recursion. This is where a method (algorithm) is described in terms of itself. Both methods work by dividing a list into parts, and then performing the same kind of sort on each of the parts. This approach is called *divide-and-conquer*. The list is divided repeatedly until it is small enough to conquer. For both methods, the lists are divided until they contain only one item. It is trivial to sort one item into order!

Merging two sorted lists, which is the basis of mergesort, is an efficient method for the more general problem of matching items from two groups. For example, a postal worker takes a sorted pile of envelopes and “merges” it with a sorted street of mailboxes—the next letter to be delivered will always be at the top of the pile. This technique is also used for the controversial practice of computer matching, where two agencies (such as social welfare and the tax department) look for clients who are enrolled in both to detect fraud—for example, if someone is declaring an income to the tax department, they probably should not be collecting an unemployment benefit. Lists from the agencies are matched by sorting each into order, and then matching duplicates in the same way that a postal worker matches envelopes to mailboxes.

Further reading

Sorting is a standard topic in computer science courses, and is discussed in many introductory texts, such as Kruse’s *Data Structures and Program Design*. Harel touches on sorting several times in *Algorithmics*. The classic (albeit slightly dated) text on sorting is Knuth’s *The Art*

¹Nowadays most computer time is spent idling or running those programs that draw changing patterns or pictures on the screen (“screen savers”), because greatly reduced prices have removed the necessity to have a machine performing useful work all the time.

ACTIVITY 7. LIGHTEST AND HEAVIEST—*SORTING ALGORITHMS*

of Computer Programming, Volume 3: Sorting and Searching, published in 1973. Dewdney discusses mergesort, and an interesting method called heapsort, in the *Turing Omnibus*.