Alphabetizing and Sorting

Activity #1 — Introduction: Sorting test papers
(Allocated time = 30 min)

A. The participants should be seated in groups of 5. Each participant is given a set of 20 quarter-sheets representing test papers. Ask the participants to put their packets of 20 test papers in alphabetical order.

Note that sorting tests or quizzes like this is a real-life activity that teachers are familiar with...and they will each likely have opinions about the best way to sort. This is good material to build on.

When they are done, ask the participants to share within their groups the methods they used to sort and the advantages or disadvantages of the methods. Also, encourage them to brainstorm to come up with the best sorting method they can find.

B. Regroup and poll the groups on the methods they discussed and found favorable. On the board or on the overhead, write down a description of their methods and elicit names for the different algorithms.

The algorithms that we will be doing formally with the participants include Straight Selection Sort, Insertion Sort, Bubble Sort and Merge Sort. If any of these methods come up, try to have the appropriate name naturally assigned to the method.

As the discussion proceeds, stress the qualities "speed" and "ease of implementation" just to make everyone aware that these are the characteristics that can make one method better than another.

C. Ask the participants to turn their test papers over (☺) and combine the piles into one large stack showing numbered papers. Ask them to consider the sorting algorithms that have been presented and to discuss which one or ones they think would be best to use to sort this large stack of 80 or 100 papers. Then ask them to try those methods.

Note that when the individual participant stacks are sorted by last name, then the numbers on the reverse sides are unsorted. So there is no need to have them shuffle the large stack that they form.

D. Regroup and poll the room on which methods they thought worked best. Try to have the participants verbalize why some of the earlier methods were no longer desirable, and if any new methods were mentioned, ask how they think those methods would perform with a smaller stack of papers to sort.

When all the tables have contributed, go over the list and indicate the methods that we will "officially" discuss during the rest of the workshop. These include, if they have

come up, straight selection sort, bubble sort and merge sort.

Note that when this workshop was given in May of 1997, several different groups discovered some sort of "bucket sort" in which ranges were pre-specified (such as A-H, I-R, S-Z), and test papers were placed in the appropriate buckets, to be sorted later. Variants included sorting the tests as they were placed into the buckets via insertion sort.

Activity #2 — Four Sorting Algorithms
(Allocated time = 40 min [10 min each])

A. Straight Selection Sort

Go over this method with them using TSP#1. For reference, this is the way a bridge hand is typically sorted.

Note: you may need to emphasize that all of our sorting will result in lists that go from smallest at the left to largest at the right. Another point that you can ask participants to deal with is what to do when you encounter two equal elements. It differs for the various algorithms. As you review this algorithm with them, try to get across the point that picking the smallest element is not a single step, but takes as many steps as there are elements remaining in the unsorted list. This will be important later when we compare the speed of various algorithms.

B. Insertion Sort.

This method is sort of the "dual" of Straight Selection. With this method, you're just taking the next element in the unsorted list, and then going through the sorted list to find the first element of the sorted list that is larger than your number, and inserting it into the sorted list immediately before that larger number. You can refer to TSP#2 for these directions and TSP#3 for an example.

What may not be obvious here is that it takes time to move the elements of the sorted list that come after the position of the new element, so that the new element can be inserted. If you mention this now, then it makes it easier to analyze later.

You may wish to show TSP#4 which shows a bucket sort, and TSP#5 which illustrates insertion sort within a bucket sort. We have prepared these slides because many teachers came up with the bucket sort method in May.

C. Bubble Sort

Have 12 volunteers come to the front of the room, handing them the placards as they come up. Ask each participant with a placard to use one of the markers you provide to write a randomly chosen whole number from 1 to 100 on the placard in neat, large digits, keeping their numbers secret for a moment. Then ask them to line themselves up at the front of the

room in some random order. This done, they should hold up their placards so that everyone can see their numbers. Then work through bubble sort, swapping the participants lined up at the front of the room as necessary.

You can put up TSPs#6 and 7 to illustrate this algorithm.

The algorithm is explained on TSP#6.

D. Merge Sort

Have 12 new volunteers line up at the front of the room with the placards (already numbered) randomly placed on them. Then perform a merge sort on them.

You can put up TSPs# 8 and 9 (illustrating the algorithm with an example).

The algorithm is explained on TSP#8.

A question that should occur to the participants is what algorithm to use in order to do the sorts on the half-lists. Of course, for merge sort, the idea is that these lists will also be sorted using merge sort. When the list gets down to having size 1 or 2, then you just sort it directly. You can have a discussion about how this would run on a very large example, for example, sorting a list of students in a school. You might divide the list in half until you get lists of size 5 or less, then use another algorithm to sort this short list, such as straight selection. It may even be the case that participants use this method to sort their tests already.

Note that at least one group discovered merge sort (with 5 piles instead of two) when we did this in May of '97.

When you have them perform this algorithm live, it is best not to do merge sort on the half-lists, at least not the first time through. Rather, do ad-hoc sort or insertion sort. When this was tried in May, the point got a little bit obfuscated when we did merge sort recursively.

Break (Alloted time = 10 min, announce 5 min)

Activity #3 — Analyzing Sorting Algorithms
(Allocated time = 45 min)

A. Turn the class's attention to the discussion of the efficiency of the various sorting algorithms. Ask them to discuss in their groups for a few minutes the issue of which algorithm is best. They should decide what "best" means. Then regroup and ask them for their ideas.

In what follows, the participants will compare the"average case" running times of the various algorithms. Of course, this will be done in an informal way. They will not discuss the "ease of implementation" in any formal way, so the instructor can deal with that in any way he wishes. It is indicated in these notes how each of the three algorithms performs on a list which is already sorted. If the instructor feels it won't be too confusing, and if time allows, he may discuss this with the participants.

B. Straight Selection Sort

Analyze, with the participants, the behavior of this algorithm.

As the participants have probably mentioned by now, there is not a big difference between the algorithms when you are sorting a small list. So ask them to imagine that they have to sort a list of 1000 items for our analyses of these algorithms.

The discussion will have to be fairly informal, so we will try to count the number of "steps" it takes to sort an "average" list using the different algorithms.

*For an average list, straight selection has to go through the entire remaining unsorted list each time you wish to find the smallest. So, that's 1000 steps for the first pass, 999 for the second pass, 998 for the third, etc... So the total number of passes is 1+2+3+...+1000, which they should recognize as the 1000th triangular number, equal to 1000*1001/2, or 500,500. That's a lot of steps.*

You can refer to TSP#10 for a summary of the analysis of straight selection.

The algorithm takes the same amount of time to run on an already-sorted list.

C. Insertion Sort

Analyze, with the participants, the behavior of this algorithm.

It's not worth it to dwell on the analysis of this algorithm. It turns out to be roughly the same as the algorithm above. Unfortunately, there is a little ambiguity as to whether the i'th step takes i or i+1 steps. So it's best to just say it's roughly the same, and move on.

D. Bubble Sort

Analyze, with the participants, the behavior of this algorithm.

Here, it takes 999 steps to make one pass through the list, but the number of passes depends greatly on how mixed-up the list is to begin with. For example, a sorted list requires just a single pass, but a reversed list requires 999 passes. A random list will tend to require close to N passes, in this case, about 960. Thus a bubble sort is good when your list is already nearly sorted. A summary can be found on TSP#11

E. Merge Sort

Analyze, with the participants, the behavior of this algorithm.

This sort, like the straight selection sort, doesn't take more or fewer steps if the data is more or less sorted. The analysis of the number of steps is a little bit involved, because it is a new concept...but should be graspable. An outline can be found on TSP#12 The idea is that, to determine the number of steps it takes to do mergesort on N elements, you split the list into two half-lists of N/2 elements each, sort them, merge the sorted half-lists, and you're done. Thus, the number of steps is equal to twice the number of steps required to sort each half-list (because there are two of them) plus the number of steps required to merge them (which is just N, because each element must be added to the merged list).

Thus, to analyze a list of size 1000, we recursively analyze lists half the size, as shown on TSP#14. Note that TSP#13 shows a table which can the instructor can start filling in from the bottom up. The completion of this task is shown on TSP#14. This turns out to be the fastest sort of all, in the average case. What is interesting is that bubble sort, which is the slowest in the average case, does the best, by FAR, if the list is nearly sorted to begin with.

F. Comparison and analysis

On TSP # 15 you can find a comparison of these four algorithms for 1000 items, and ont TSP#16 you can find a comparison of the four sorting algorithms for up to a million items to be sorted. It is clear that Merge wins.

You can follow this with a discussion of how in different situations, different sorts may be optimal.

Activity #4 — Sorting Networks
(Allocated time = 15 min)

A. Acting it out

Have a new group of 9 volunteers come to the front of the room and wear any 9 of the placards. Explain to the participants that they will be seeing a very different type of sorting algorithm now. Then proceed out to where the tarp with the sorting network has been prepared and instruct the participants about how to navigate the network.

It is good to establish right at the outset which way the larger and smaller numbers should proceed at each node. "Larger Left" is a rule which is easily memorized. The sorting network on the tarp is shown on TSP#17.

One of the points which you can make is that this sort of parallel-processing algorithm is very quick. The parallel processing comes in here because many decisions are being made independently at the same time by different "processors."

Some interesting questions can arise when the participants get the idea of the network. These may include "will it work the other way?" or "what if larger goes right?" or "how many nodes are there?" The answers are "no", "it will still sort, but from larger to smaller" and "25." The question about it working the other way has become a topic of ongoing research in CS. Interestingly, research questions like these are posed by very

young children who are just naturally curious. As for the 25, noone knows if that is the smallest number of intermediate nodes that suffice to build a network to sort 9 items!

B. Analyze, with the participants, the speed of sorting networks.

It turns out that sorting networks operate in O(log n) time. This is shown numerically, without saying "log," on TSP#18, which is the same as TSP#16 but with Sorting Netowrks included. Thus, to sort a thousand item list, as we are doing in the other examples, you can expect it to be do-able in less than (and I'm mostly making this up) 50 steps, assuming time roughly 5log(n), base 2.

F. Show the part of the video "Sorting out Sorting" at the end which shows the entire first part of the video in fast-motion. You may wish to point out where the algorithms discussed appear in the analysis phase. These are bolded in the chart below. (Allocated time = 5 min)

Insertion Sort	**Bubble Sort**	**Straight Selection**
Binary Insertion Sort	Shaker Sort	Tree Selection Sort
Shell Sort	**QuickSort (a kind of recursive Bucket Sort)**	Heap Sort

Straight Selection Sort

1. Begin with an unsorted list, with a marker (★) to the left of the list.
2. Go left to right through the unsorted list (right of the marker), find the smallest element, and add it to the end of the sorted list, to the left of the marker.
3. Repeat step 2 until every element has been moved to the left of the marker. The result is your sorted list.

.

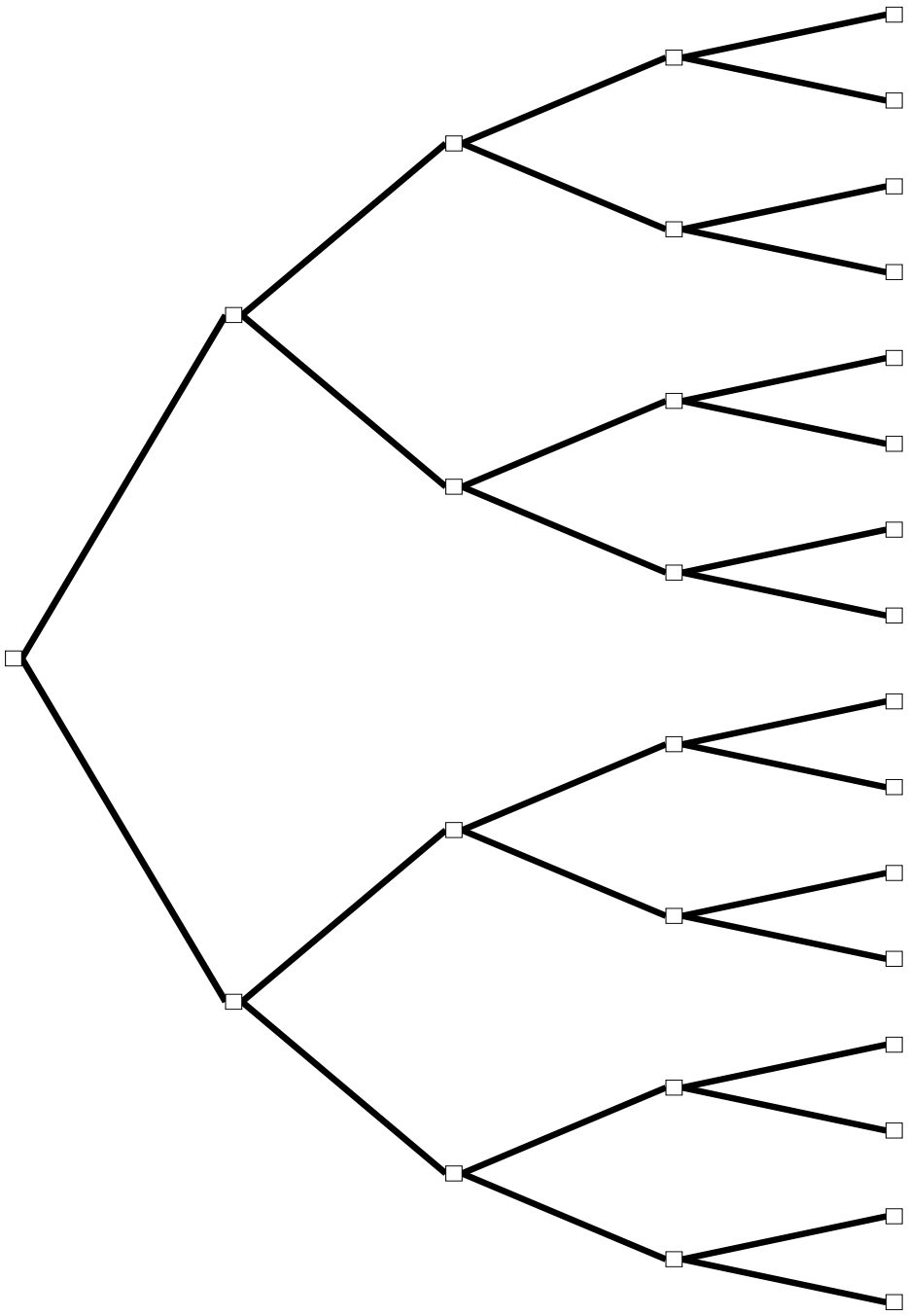
Unsorted List	Comments
★ 3 6 9 5 8	Start with list and marker
3 ★ 6 9 5 8	3=smallest, add to list on left
3 5 ★ 6 9 8	5=smallest, add to list on left
3 5 6 ★ 9 8	6=smallest, add to list on left
3 5 6 8 ★ 9	8=smallest, add to list on left
3 5 6 8 9 ★	9=smallest, add to list on left
3 5 6 8 9	Sorted List

Tree Sort

1. Begin with an unsorted list.

2. Build a binary tree in front of the list, adding enough "blank spaces" to your list so that you have a power of 2.

3. Each number (student) tries to move to the next node, toward the top, subject to the following rules:
    a. If the node above them is occupied, then they have to wait
    b. If the node is unoccupied, and you are not competing (with a sister node) for that node, then you can move up
    c. If the node is unoccupied, and you are competing with a sister node, then the smaller number gets to move up
4. When a number reaches the top, it "plucks" itself off and moves to the end of the sorted list (the first number to reach the top begins the sorted list.)
5. When all numbers have moved, then the sorting is done!



A tree diagram with 16 nodes on the end. Note that if you have to sort some number of objects which is not 2, 4, 8, 16, 32, etc..., then you just use the next largest size tree, and leave as many nodes blank as you need to. Also, if you have to leave some nodes blank, it doesn't matter where you leave the blank spaces, the tree will still sort the students!

Insertion Sort

1. Begin with an unsorted list, with a marker (★) to the left of the list.

2. Pick the next element from the unsorted list (right of the marker) and call it "It."

3. Go left to right through the sorted list (left of the marker) until you find an element greater than It, or until you reach the end.

4. Insert It into the sorted list, and, if It is not being inserted at the end, move the rest of the list to the right to make room for It.

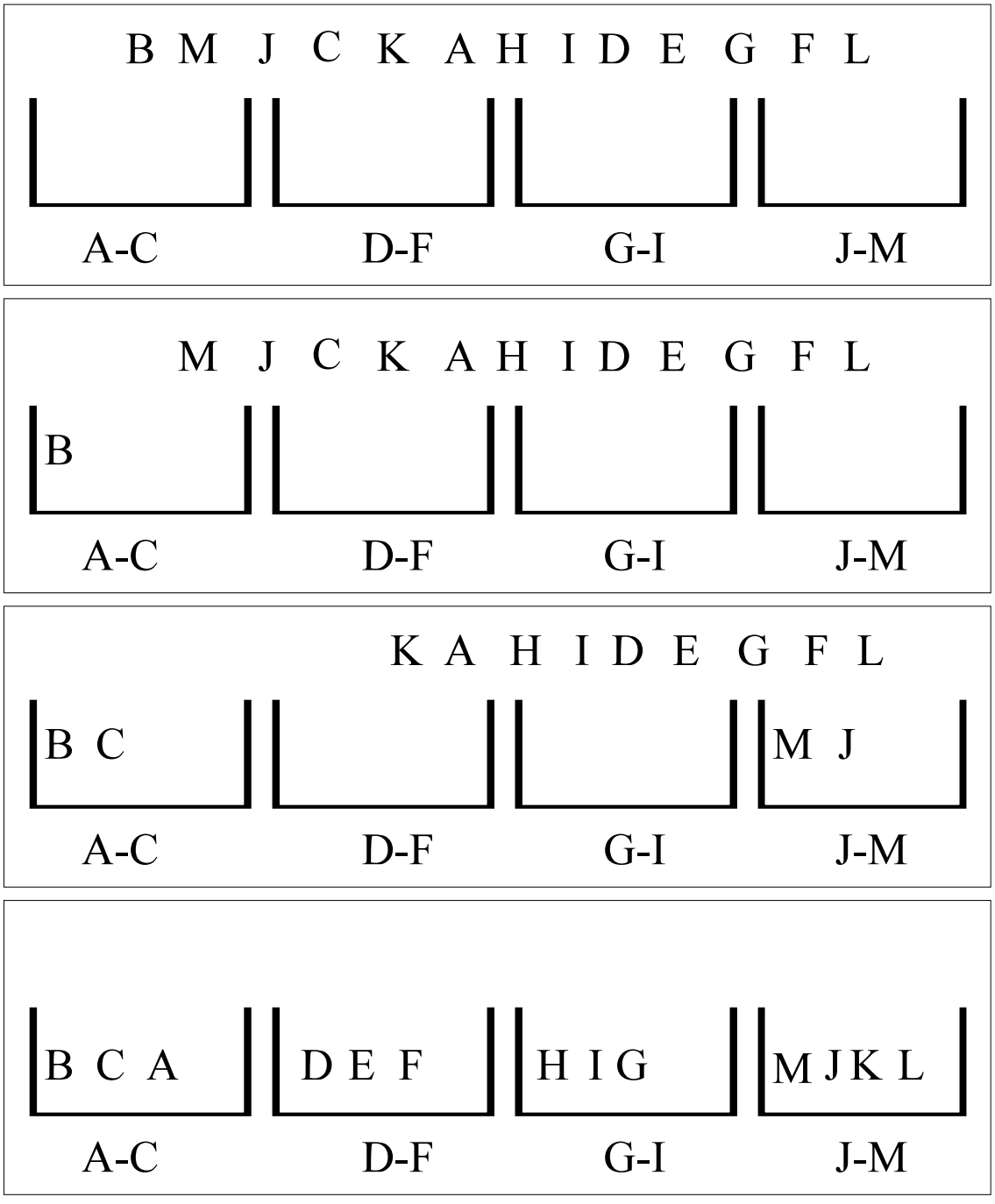
5. Repeat steps 2-4 until every element has been moved to the left of the marker. The result is your sorted list.

Insertion Sort Example

Unsorted List	Comments
★ 3 6 9 5 8	Start with unsorted list
3 ★ 6 9 5 8	Move first key to the left, and insert
3 **6** ★ 9 5 8	Move second key to the left, and insert
3 6 **9** ★ 5 8	Move third key to the left, and insert
3 **5** 6 9 ★ 8	Move fourth key to the left, and insert
3 5 6 **8** 9★	Move fifth key to the left, and insert
3 5 6 8 9	Sorted list

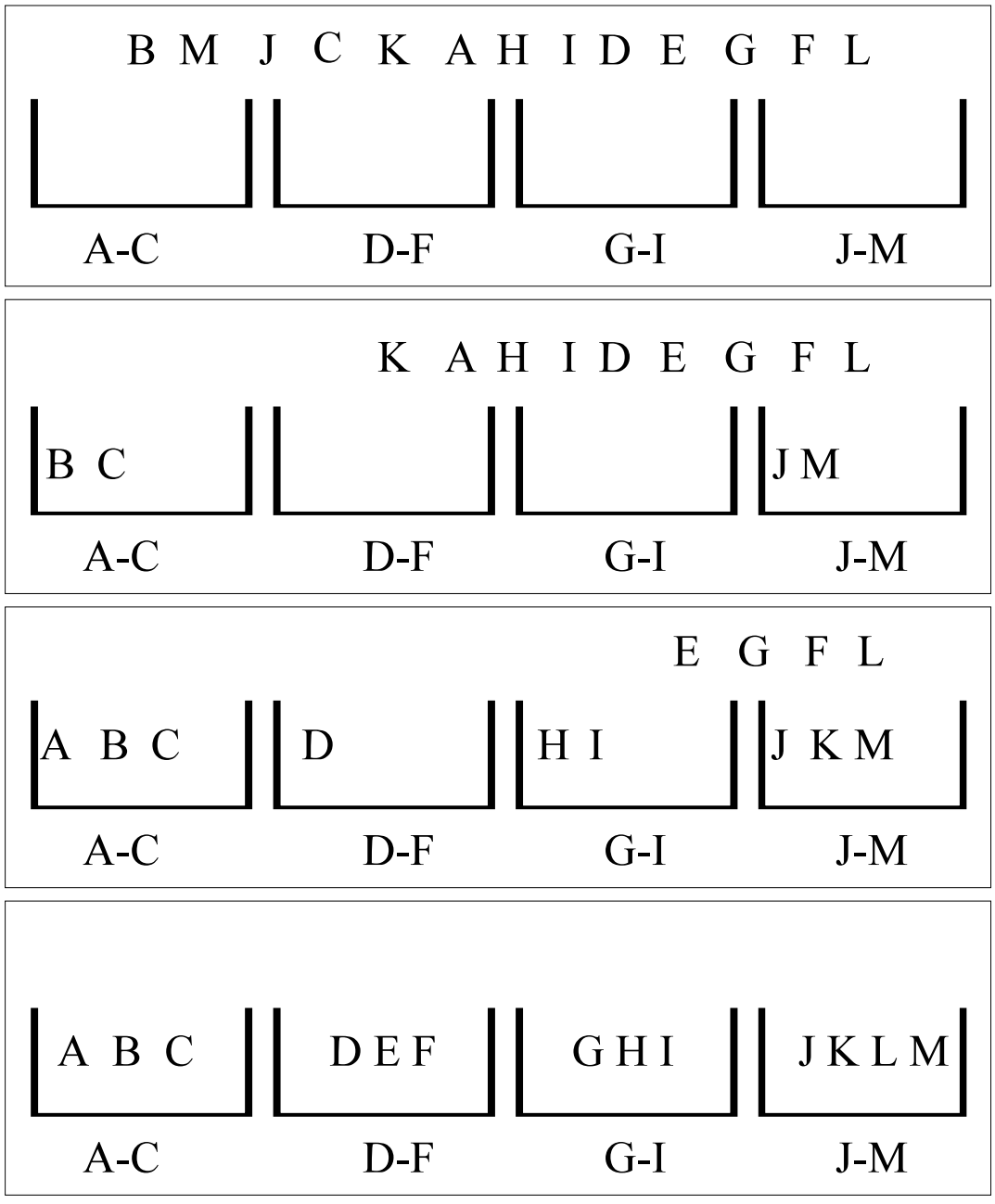
Bucket Sort

The next step would be to sort within the buckets, resulting in a sorted list.



Bucket Sort mixed with Insertion Sort

Here it took a little longer to place into the buckets, because we sorted as we went along.



But in the end, we have no further sorting to do.

Bubble Sort

1. Begin with an unsorted list.
2. Go left to right through the unsorted list, swapping adjacent pairs of numbers when you see a bigger number come before a smaller number.
3. Repeat step 2 until you have a pass with no swaps. You now have a sorted list.

Unsorted List	Comments
3 6 9 5 8	Start with List
3 6 9 5 8	Okay
3 **6 9** 5 8	Okay
3 6 **9 5** 8	Swap
3 6 5 **9 8**	Swap
3 6 5 **8 9**	This ends pass 1

Unsorted List	Comments
3 6 5 8 9	Okay
3 **6 5** 8 9	Swap
3 5 **6 8** 9	Okay
3 5 6 **8 9**	Okay
3 5 6 8 9	This ends pass 2
3 5 6 8 9	Okay
3 **5 6** 8 9	Okay
3 5 **6 8** 9	Okay
3 5 6 **8 9**	Okay
3 5 6 8 9	This ends Pass 3
3 5 6 8 9	No swaps, so we're done.

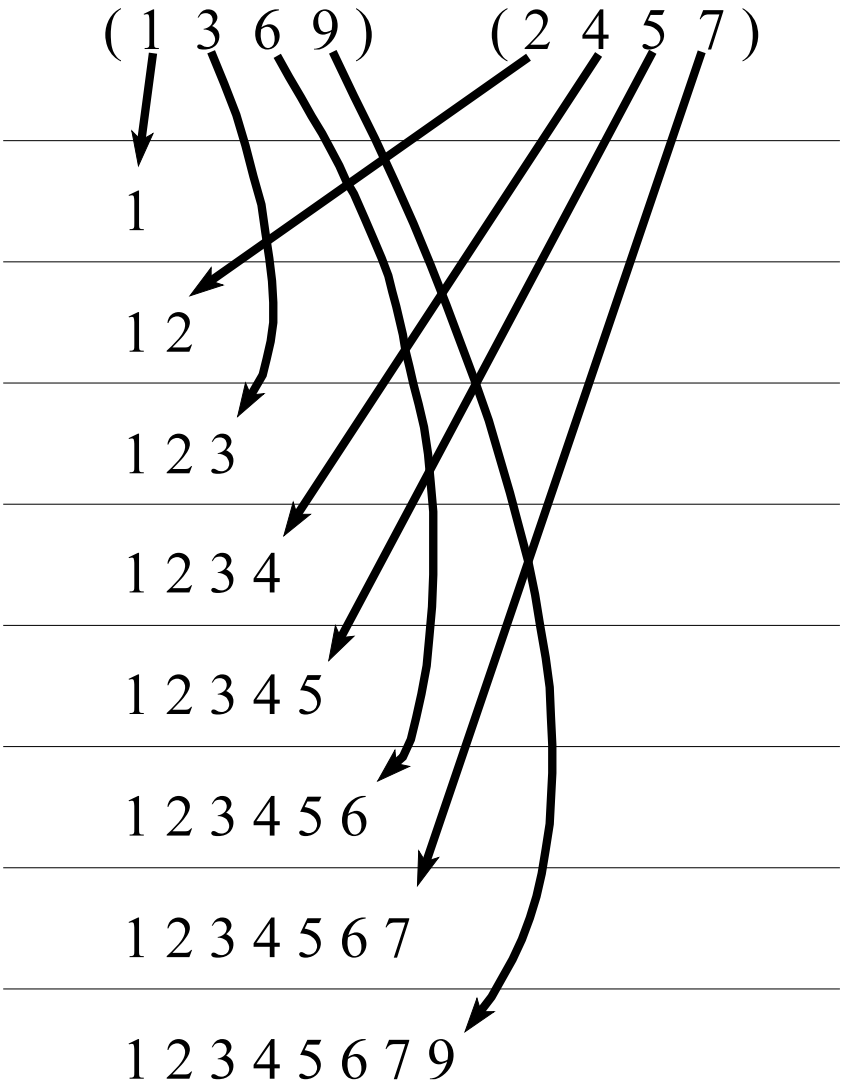
Merge Sort

1. Begin with an unsorted list
2. Divide the unsorted list into two roughly equal lists
3. Sort the two half-lists (*)
4. Merge the sorted sub-lists as follows:
   - Look at the leftmost (smallest) element on each list
   - Put the smaller of the two at the end of the merged list we are creating
   - Repeat until one half-list is empty, then put the remainder of the other list at the end of our sorted list

(*) It is customary to use Merge Sort to sort each of the smaller lists. This *recursive* nature makes Merge Sort very easy to program on a computer.

A Merge Sort Example

1. Unsorted List: 9 3 1 6 5 4 7 2
2. Divide: 9 3 1 6 5 4 7 2
3. Sort: 1 3 6 9 2 4 5 7
4. Merge:



Analysis of Straight Selection Sort

First pass, find smallest: 1000 steps
Second pass, find smallest: 999 steps
Third pass, find smallest: 998 steps
Etc...

Total number of steps:

$1000 + 999 + 998 + \cdots + 2 + 1 = 500{,}500$

(Note, this is the 1000^{th} triangular number)

Straight Selection Sort takes
500,500 *steps to sort 1000 items*

The number of steps required to sort *n* items by straight selection is $n(n+1)/2$.

Analysis of Bubble Sort

First pass: 999 steps
Second pass: 999 steps
Every pass: 999 steps

The total number of steps is 999 times the number of passes:
The number of passes depends on how badly out of order the initial list is:
Average list: About 960 passes

> *Bubble sort takes an average of* ***959,040*** *steps to sort 1000 items*

Sorted list: Just one pass
Reversed list: 999 passes

> *Bubble sort can take more or fewer steps on special lists, though:*
> *Sorted list* ***999*** *steps*
> *Reversed List* ***998,001*** *steps*

Analysis of Merge Sort

To do merge sort on 1000 objects, we split the list in half, do merge sort on each of the half-lists, and merge them.

So, to find the number of steps for 1000 objects, we find the number of steps for 500 objects (for a half-list), double it (there are 2 half-lists), and add 1000 (the number of steps it takes to merge the lists).

But to find the number of steps for 500 objects, we find the number of steps for 250 objects (for a half-list), double it (there are 2 half-lists), and add 500 (the number of steps it takes to merge the lists).

But to find the number of steps for 250, we need the number of steps for 125, and 63, and 32 and 16 and 8 and 4 and 2 and 1.

So, let's make a chart!

Analysis for Merge Sort

Number of elements	Expression for number of steps	Number of steps
1,000		
500		
250		
125		
63		
32		
16		
8		
4		
2		
1		

Analysis for Merge Sort

Number of elements	Expression for number of steps	Number of steps
1,000	2×5,076+1,000	11,152
500	2×2,288+500	5,076
250	2×1,019+250	2,288
125	2×447+125	1,019
63	2×192+63	447
32	2×80+32	192
16	2×32+16	80
8	2×12+8	32
4	2×4+4	12
2	2×1+2	4
1	1	1

Merge Sort takes
11,152 steps to sort 1000 items

Comparison of our algorithms
(in the average case)

Sorting Algorithm	Number of steps for 1000 items
Straight Selection	500,500
Insertion Sort	500,500
Bubble Sort	959,040
Merge Sort	11,152

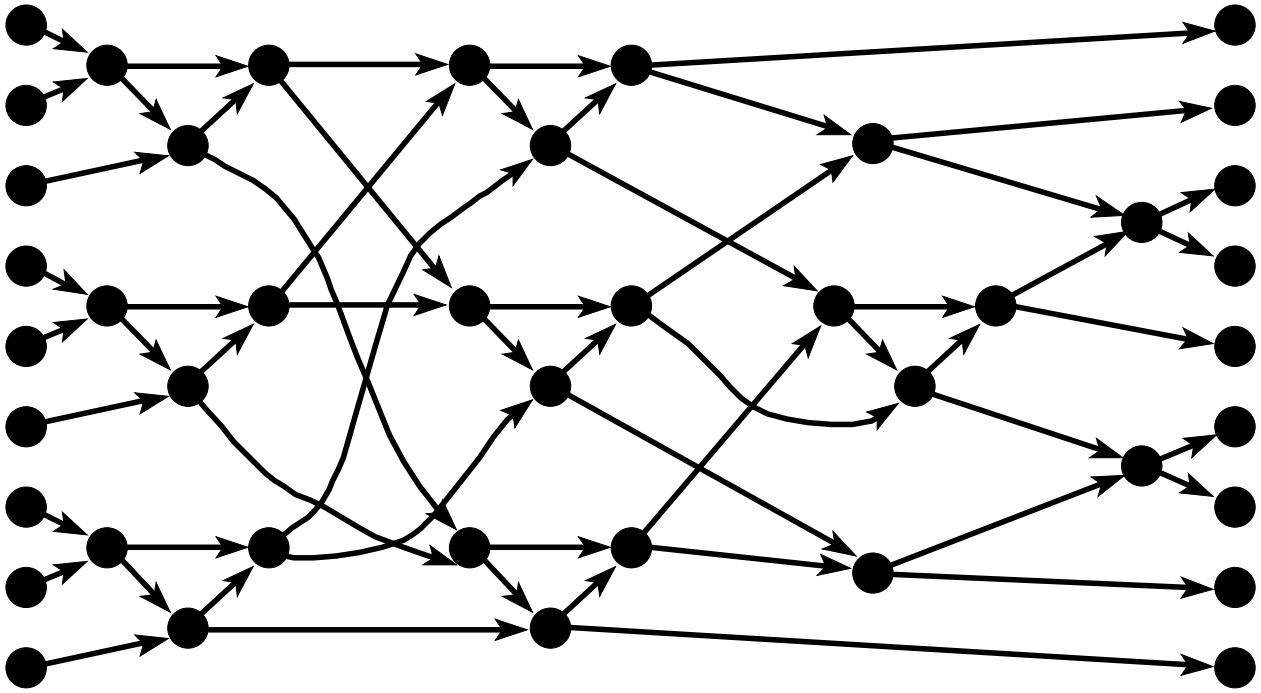
Comparison of our algorithms
(in the average case)

Number of items	Selection/ insertion	Bubble Sort	Merge Sort
10	55	81	37
50	1,275	2,401	316
100	5,050	9,801	743
500	125,250	249,001	5,016
1,000	500,500	998,001	11,152
5,000	12,502,500	24,990,001	68,750
10,000	50,005,000	99,980,001	148,690
50,000	1,250,025,000	2,499,900,001	873,359
100,000	5,000,050,000	9,999,800,001	1,858,619
500,000	125,000,250,000	249,999,000,001	10,592,213
1,000,000	500,000,500,000	999,998,000,001	22,303,425

Merge tends to be best.

Analysis of Sorting Networks

For the network shown above, any sort can be accomplished in eight steps.



If we had a sorting network which would sort 1000 objects, we would see that you don't need more than 50 steps!

Let us compare the efficiency of our sorts, taking into account sorting networks:

Comparison of our algorithms
(in the average case)

Number of items	Selection/ insertion	Bubble Sort	Merge Sort	Sorting Network
10	55	81	37	8
50	1,275	2,401	316	23
100	5,050	9,801	743	29
500	125,250	249,001	5,016	44
1,000	500,500	998,001	11,152	50
5,000	12,502,500	24,990,001	68,750	65
10,000	50,005,000	99,980,001	148,690	71
50,000	1,250,025,000	2,499,900,001	873,359	86
100,000	5,000,050,000	9,999,800,001	1,858,619	92
500,000	125,000,250,000	249,999,000,001	10,592,213	107
1,000,000	500,000,500,000	999,998,000,001	22,303,425	113

This gives an idea of how well parallel computation can improve the speed of certain tasks.