Sorting Algorithms

Several algorithms exist for the sorting of data and we will begin by describing them and discussing their differences.

Insertion sort

Recall that the purpose of a sorting algorithm is to sort elements into an ordered list according to some definition of what an ordered list is.

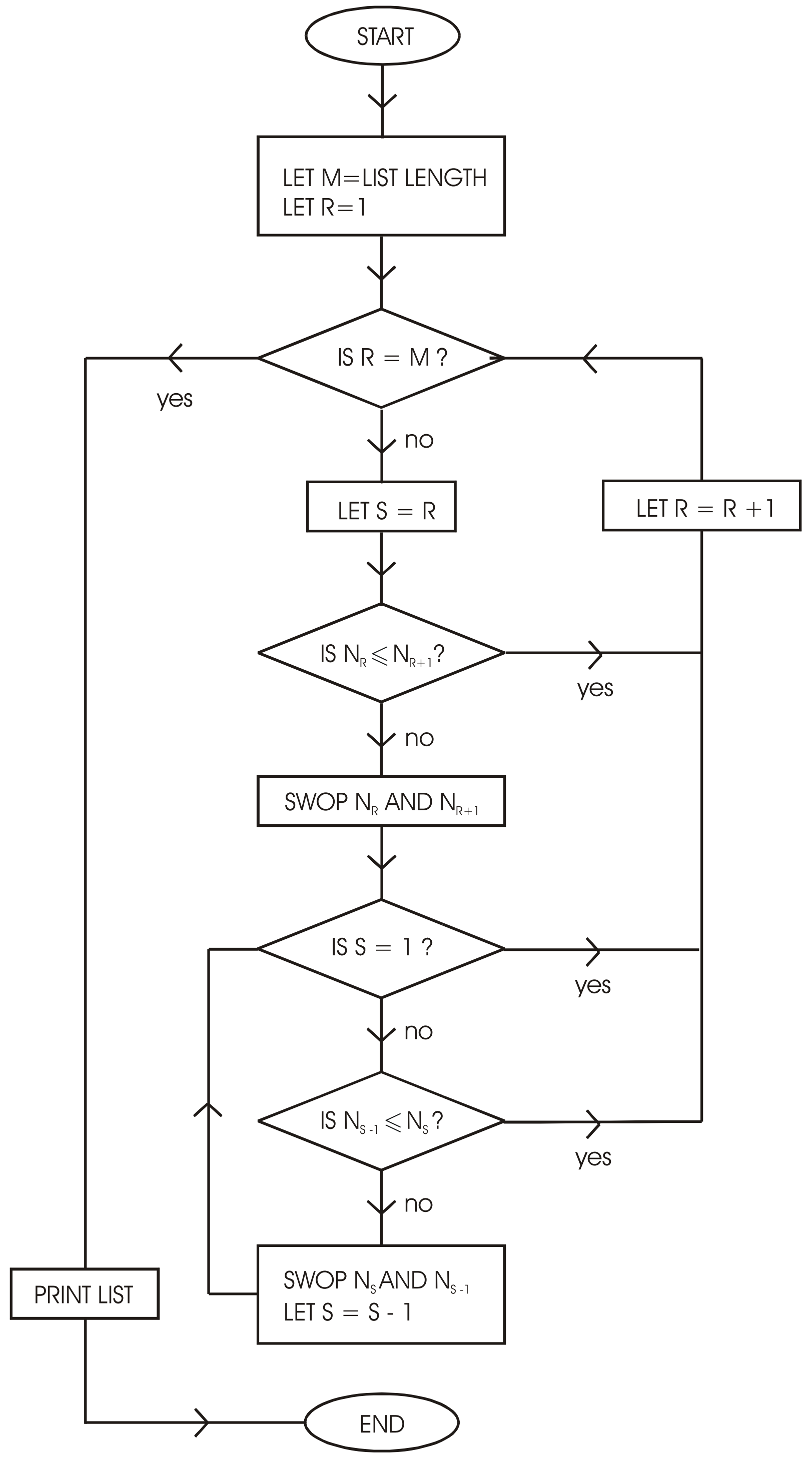
The insertion algorithm does this by comparing elements pair wise and putting them pair wise into the correct order.

An algorithm for an insertion sort is

```
STEP 0:    LET M = LIST LENGTH
           LET R = 1

STEP 1:    IF R = M THEN PRINT LIST & STOP

STEP 2:    LET S = R

STEP 3:    IF $N_R \leqslant N_{R+1}$ LET R = R + 1 AND GOTO STEP 1
           IF $N_R > N_{R+1}$ SWOP $N_R$ AND $N_{R+1}$

STEP 4:    IF S = 1 LET R = R + 1 AND GOTO STEP 1
           IF $N_{S-1} \leqslant N_S$ LET R = R + 1 AND GOTO STEP 1
           IF $N_{S-1} > N_S$ SWOP $N_S$ AND $N_{S-1}$
           LET S = S – 1 AND GOTO STEP 4
```

A flow chart for this algorithm is

Insertion sort algorithm



The algorithm performs the following process on the list of numbers

14, 29, 6, 51, 99, 37, 63, 37

STEP	N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	R	S	S - 1	M
	14	29	6	51	99	37	63	37				
0	14	29	6	51	99	37	63	37	1			8
1	14	29	6	51	99	37	63	37	1			8
2	14	29	6	51	99	37	63	37	1	1	0	8
3	14	29	6	51	99	37	63	37	2	1	0	8
1,2	14	29	6	51	99	37	63	37	2	2	1	8
3	14	6	29	51	99	37	63	37	2	2	1	8
4	6	14	29	51	99	37	63	37	2	1	0	8
4	6	14	29	51	99	37	63	37	3	1	0	8
1,2	6	14	29	51	99	37	63	37	3	3	2	8
3	6	14	29	51	99	37	63	37	4	3	2	8
1,2	6	14	29	51	99	37	63	37	4	4	3	8
3	6	14	29	51	99	37	63	37	5	4	3	8
1,2	6	14	29	51	99	37	63	37	5	5	4	8
3	6	14	29	51	37	99	63	37	5	5	4	8
4	6	14	29	37	51	99	63	37	5	4	3	8
4	6	14	29	37	51	99	63	37	6	4	3	8
1,2	6	14	29	37	51	99	63	37	6	6	5	8
3	6	14	29	37	51	63	99	37	6	6	5	8
4	6	14	29	37	51	63	99	37	7	6	5	8
1,2	6	14	29	37	51	63	99	37	7	7	6	8
3	6	14	29	37	51	63	37	99	7	7	6	8
4	6	14	29	37	51	37	63	99	7	6	5	8
4	6	14	29	37	37	51	63	99	7	5	4	8
4	6	14	29	37	37	51	63	99	8	5	4	8
1	6	14	29	37	37	51	63	99				

STOP

Bubble Sort

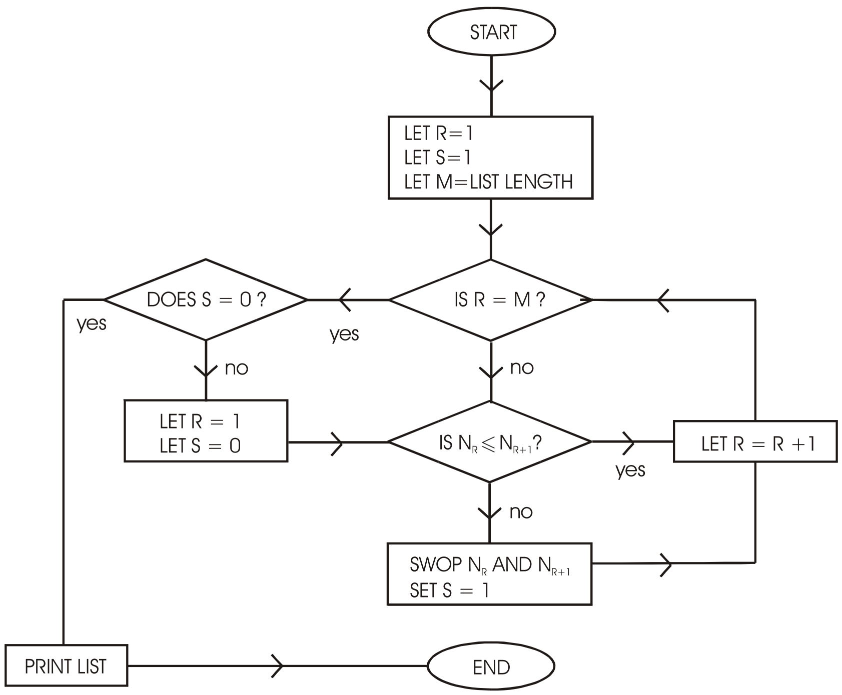
The idea of a bubble sort is that the solved item floats to the top of the list like a bubble in water. The algorithm passes through the list repeatedly comparing elements pair at a time and allowing the largest of these to "bubble" up to the top by swapping elements if the lower element is larger than the upper element. Each time a swap is made the whole process has to be repeated at least once, so a "flag" must be set to take account of this.

An algorithm for the bubble sort is

```
STEP 0:    LET R = 1
           LET S = 1
           LET M = LIST LENGTH

STEP 1:    IF R = M AND S = 0 THEN PRINT LIST AND STOP
           IF R = M AND S = 1 THEN LET R = 1 AND S = 0

STEP 2:    IF $N_R \leqslant N_{R+1}$ LET R = R + 1 AND GOTO STEP 1
           IF $N_R > N_{R+1}$ SWAP $N_R$ AND $N_{R+1}$
           LET S = 1, LET R = R + 1 AND GOTO STEP 1
```

A flow diagram for this algorithm is:

Bubble sort algorithm



The bubble sort algorithm performs the following operations on the list

14, 29, 6, 51, 99, 37, 63, 37

STEP	N1	N2	N3	N4	N5	N6	N7	N8	R	S	M
0	14	29	6	51	99	37	63	37	1	1	8
1	14	29	6	51	99	37	63	37	1	1	8
2	14	29	6	51	99	37	63	37	2	1	8
1,2	14	6	29	51	99	37	63	37	3	1	8
1,2	14	6	29	51	99	37	63	37	4	1	8
1,2	14	6	29	51	99	37	63	37	5	1	8
1,2	14	6	29	51	37	99	63	37	6	1	8
1,2	14	6	29	51	37	63	99	37	7	1	8
1,2	14	6	29	51	37	63	37	99	8	1	8
1	14	6	29	51	37	63	37	99	1	0	8
2	6	14	29	51	37	63	37	99	2	1	8
1,2	6	14	29	51	37	63	37	99	3	1	8
1,2	6	14	29	51	37	63	37	99	4	1	8
1,2	6	14	29	37	51	63	37	99	5	1	8
1,2	6	14	29	37	51	63	37	99	6	1	8
1,2	6	14	29	37	51	37	63	99	7	1	8
1,2	6	14	29	37	51	37	63	99	8	1	8
1	6	14	29	37	51	37	63	99	1	0	8
1,2	6	14	29	37	51	37	63	99	2	0	8
1,2	6	14	29	37	51	37	63	99	3	0	8
1,2	6	14	29	37	51	37	63	99	4	0	8
1,2	6	14	29	37	51	37	63	99	5	0	8
1,2	6	14	29	37	37	51	63	99	6	1	8
1,2	6	14	29	37	37	51	63	99	7	1	8
1,2	6	14	29	37	37	51	63	99	8	1	8
1	6	14	29	37	37	51	63	99	1	0	8
2	6	14	29	37	37	51	63	99	2	0	8
1,2	6	14	29	37	37	51	63	99	3	0	8
1,2	6	14	29	37	37	51	63	99	4	0	8
1,2	6	14	29	37	37	51	63	99	5	0	8
1,2	6	14	29	37	37	51	63	99	6	0	8
1,2	6	14	29	37	37	51	63	99	7	0	8
1,2	6	14	29	37	37	51	63	99	8	0	8
1	6	14	29	37	37	51	63	99	8	0	8

A question that arises at this point is which of these two sorting algorithms performs the task more quickly. On the face of it, the insertion algorithm is quicker than the bubble sort, because it uses fewer lines. However, that may only be a deceptive appearance.

One way of counting length is to count the number of operations performed during a procedure. We could count each comparison (is $N_R \leq N_{R+1}$), each setting of a register (let $R = R+1$), each process of stopping registers. However, it turns out that for a computer some procedures are more time consuming than others and here the swaps are crucial – hence the most efficient algorithm will be the one involving the fewest swops. On this basis the insertion algorithm involves 7 swaps on our data & the bubble sort 8.

However, the insertion and bubble sorts are regarded as inefficient, especially on longer lists. The reason why is that each swap results only in numbers being moved up or down one in the list. If the data is very unordered many swops will be required. In the algorithm devised by Donald Shell data elements are swapped over a greater distance initially and this greatly improves efficiency.

The algorithm works by dividing the original list into subsets drawn from elements from different ends of the data list and sorting these. This enables unsorted items to be moved over larger distances in the first instance. The early stages of the algorithm pre-sort the list so that when at the final stage an insertion (or bubble) sort is performed it is very much more efficient. We illustrate the idea firstly by shell sorting our sample data.

The original list is	14	29	6	51	99	37	63	37
In the first pass, four pairs of elements	14				99			
are formed & sorted		29				37		
			6				63	
				51				37
The list becomes	14	29	6	37	99	37	63	51
In the second pass, two sets of four	14		6		99		63	
elements are formed and sorted		29		37		37		51
The list becomes	6	29	14	37	63	37	99	51
In the last pass the whole list is sorted	6	14	29	37	37	51	63	99

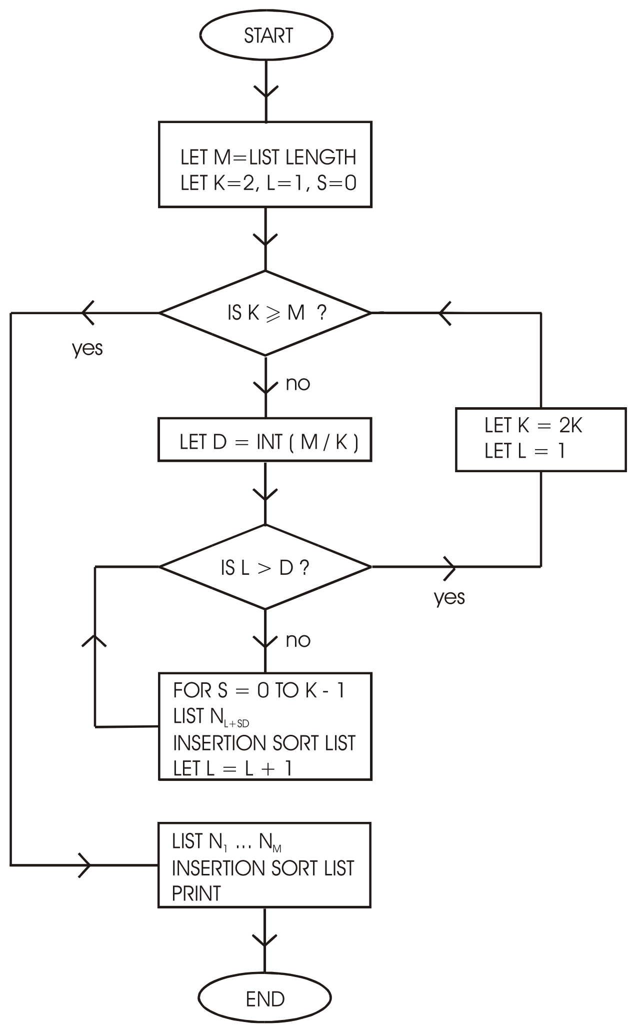
This, in fact, only involves five swaps, so whilst it looks more complicated it is more efficient. Its efficiency is more readily demonstrated on longer lists. We should also illustrate its use on lists with odd lengths and lengths not equal to a power of 2.

Firstly, an algorithm for the shell sort is

```
STEP 0:     LET M = LIST LENGTH
            LET K = 2
            LET L = 1
            LET S = 0

STEP 1:     IF $K \geqslant M$ LIST $N_1 \ldots N_M$
            INSERTION SORT, PRINT AND END

STEP 2:     LET D = INT (M / K)
STEP 3:     IF L > D LET K = 2K, LET L = 1 AND GOTO STEP 1

STEP 4:     FOR S = 0 TO K – 1
            LIST $N_{L+SD}$
            INSERTION SORT LIST
            LET L = L + 1 AND GOTO STEP 3
```

Flow chart for shell sort:

Shell sort algorithm



This shell sort algorithm would perform the following process on the list

14, 29, 6, 83, 51, 99, 37, 63, 27, 2, 9

Pass	N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	N_9	N_{10}	N_{11}	K	L	D
1	14	29	6	83	51	99	37	63	37	2	9	2		6

	14						37						1	
		29						63					2	
			6						37				3	
				83						2			4	
					51						9		5	
						99							6	
2	14	29	6	2	9	99	37	63	37	83	51	4		3
	14			2			37			83			1	
		29			9			63			51		2	
			6			99			37				3	
3	2	9	6	14	29	37	37	51	99	83	63	8		2
	2		6		29		37		99		63			
		9		14		37		51		83				
4	2	9	6	14	29	37	37	51	63	83	99	16		
	2	6	9	14	29	37	37	51	63	83	99			

An even more efficient method of sorting longer lists is the quick sort.