Sorting algorithms

# Bubble sort algorithm

This algorithm sorts an integer array into ascending order using a bubble sort method.

On each pass, the algorithm compares each pair of adjacent items in the array. If the pair is out of order, they are switched; otherwise they remain in the original order. So, at the end of the first pass, the largest element in the array will have bubbled to the last position in the array.

The next pass will work only with the remaining elements, and will move the next largest element to the second-last position in the array and so on.

In the algorithm:

ARRAY = array to be sorted

NUMBER\_OF\_ELEMENTS = number of elements in the array

ELEMENTS\_SWITCHED = flag to record if the elements have been switched in the current pass

TEMP = temporary area for holding an array element that is being switched

I = index for outer loop

J = index for inner loop.

Assume that the contents of ARRAY and NUMBER\_OF\_ELEMENTS have already been established.

1. Bubble\_sort\_algorithm
2. I = NUMBER\_OF\_ELEMENTS
3. ELEMENTS\_SWAPPED = true
4. loop while ELEMENTS\_SWAPPED
5. J = 0
6. ELEMENTS\_SWAPPED = false
7. loop while J <= I – 1
8. if ARRAY[J] > ARRAY[J + 1] then
9. TEMP = ARRAY[J]
10. ARRAY[J] = ARRAY[J + 1]
11. ARRAY[J + 1] = TEMP
12. ELEMENTS\_SWAPPED = true
13. end if
14. J = J + 1
15. end loop
16. I = I – 1
17. end loop
18. end

Information taken from

Robertson, Lesley Anne. *Simple Program Design: A Step-by-step Approach*. 5th ed. Southbank: Thomson, 2007. Print.

# Insertion sort algorithm

This algorithm sorts an integer array into ascending order using an insertion sort method.

“To perform an insertion sort, begin at the left-most element of the array and invoke Insert to insert each element encountered into its correct position. The ordered sequence into which the element is inserted is stored at the beginning of the array in the set of indices already examined.” (Wikipedia 2020)

In the algorithm:

ARRAY = array to be sorted

NUMBER\_OF\_ELEMENTS = number of elements in the array

TEMP = temporary area for holding an array element while correct position is being searched

I = current position of the element

J = index for inner loop.

Assume that the contents of ARRAY and NUMBER\_OF\_ELEMENTS have been established.

1. Insertion\_sort\_algorithm
2. I to 1
3. loop while I < NUMBER\_OF\_ELEMENTS
4. J = I
5. loop while J > 0 AND ARRAY[J-1] > ARRAY[J]
6. TEMP = ARRAY[J]
7. ARRAY[J] = ARRAY[J-1]
8. ARRAY[J-1] = TEMP
9. J = J - 1
10. end loop
11. I = I + 1
12. end loop
13. end

Wikipedia. “Insertion Sort.” *Wikipedia*, Wikimedia Foundation, 19 Feb. 2020, [en.wikipedia.org/wiki/Insertion\_sort#Algorithm](http://en.wikipedia.org/wiki/Insertion_sort#Algorithm).

# Selection sort algorithm

This algorithm sorts an integer array into ascending sequence using a selection sort method.

On the first pass the algorithm finds the smallest element in the array and moves it to the first position in the array by switching it with the element originally in that position. Each successive pass moves one more element into position. After the number of passes is one number less than the number of elements in the array, the array will be in order.

In the algorithm:

ARRAY = array being sorted

NUMBER\_OF\_ELEMENTS = number of elements in the array

SMALLEST\_ELEMENT = area for holding the smallest element found in that pass

CURRENT\_SMALLEST\_POSITION = the value of the current position in which to place the smallest element

I = index for outer loop

J = index for inner loop.

Assume that the contents of ARRAY and NUMBER\_OF\_ELEMENTS have been established.

1. Selection\_sort\_algorithm
2. CURRENT\_SMALLEST\_POSITION = 1
3. loop while CURRENT\_SMALLEST\_POSITION <= [NUMBER\_OF\_ELEMENTS – 1]
4. I = CURRENT\_SMALLEST\_POSITION
5. SMALLEST\_ELEMENT = ARRAY[I]
6. J = I + 1
7. loop while J <= NUMBER\_OF\_ELEMENTS
8. if ARRAY[J] < SMALLEST\_ELEMENT then
9. I = J
10. SMALLEST\_ELEMENT = ARRAY[J]
11. end if
12. J = J + 1
13. end loop
14. ARRAY[I] = ARRAY[CURRENT\_SMALLEST\_POSITION]
15. ARRAY[CURRENT\_SMALLEST\_POSITION] = SMALLEST\_ELEMENT
16. CURRENT\_SMALLEST\_POSITION = CURRENT\_SMALLEST\_POSITION + 1
17. end loop

18 end

Adapted from

Robertson, Lesley Anne. *Simple Program Design: A Step-by-step Approach*. 5th ed. Southbank: Thomson, 2007. Print.

# Alternative insertion sort algorithm

In the algorithm, the array is scanned until an out-of-order element is found. The scan is then temporarily halted while a backward scan is made to find the correct position to insert the out-of-order element. Elements bypassed during this backward scan are moved up one position to make room for the element being inserted. This method of sorting is more efficient than the bubble sort.

1. Insertion\_sort\_algorithm
2. set I to 1
3. loop while I < NUMBER\_OF\_ELEMENTS
4. TEMP = ARRAY[I]
5. J = I-1
6. loop while [J ≥ 0 AND ARRAY[J] > TEMP]
7. ARRAY[J + 1] = ARRAY[J]
8. J = J – 1
9. end loop
10. ARRAY[J + 1] = TEMP
11. I = I + 1
12. end loop
13. end

Adapted from

Robertson, Lesley Anne. *Simple Program Design: A Step-by-step Approach*. 5th ed. Southbank: Thomson, 2007. Print.