Sorting algorithms

Bubble sort algorithm

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

The algorithm compares each pair of adjacent items in the array on each pass. They are switched if the pair is out of order; 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 = 1
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
```

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           add 1 to CURRENT_SMALLEST_POSITION
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.
The algorithm scans the array until an out-of-order element is found. The scan is then temporarily halted while a backward scan finds 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 inserted element. This method of sorting is more efficient than the bubble sort.

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           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
```

Alternative insertion sort algorithm

```
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.