# Computer Science

## Bubble sort

# Lesson Objectives

Students will learn:

- Bubble sort algorithm
- How is a list sorted using bubble sort algorithm?
- Pseudocode for bubble sort algorithm

# KNOWING WHAT YOU KNOW

- Go to: https://joinmyquiz.com
- Write your name and grade level
- Join code: _____

# 1.

Content

# Searching and sorting algorithms

- Sorting algorithms arrange the data in particular order.
- Searching algorithms are used to search for data in a list.

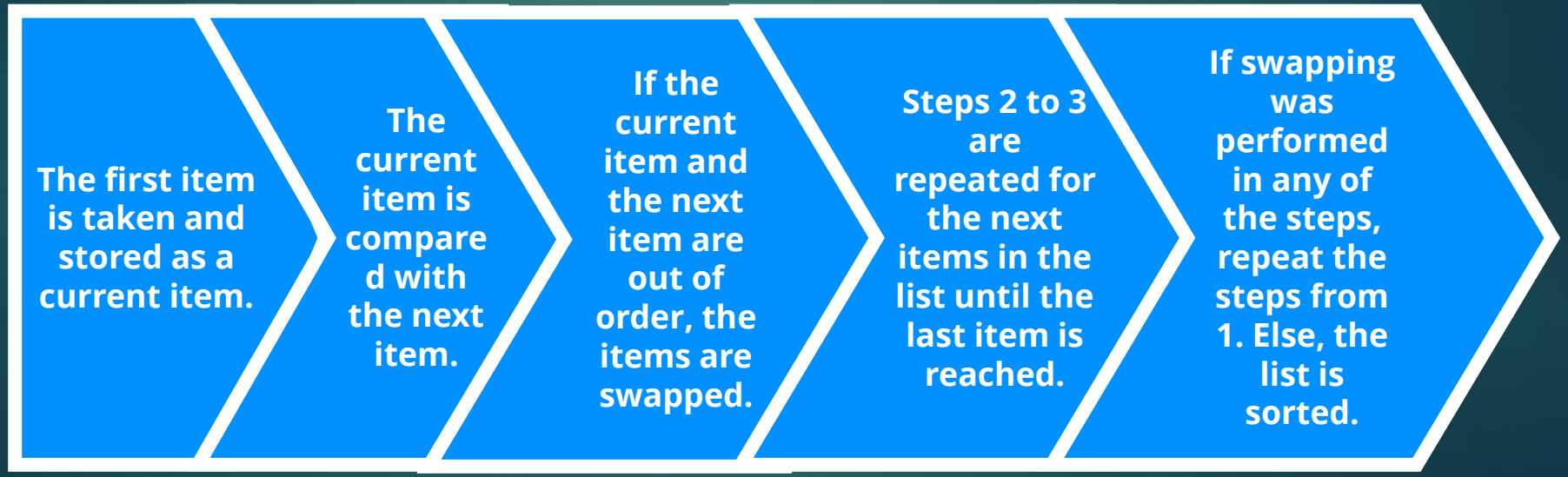
# List

- An example of list is:

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| **Item** | **C** | **A** | **D** | **F** | **E** | **B** |

# Bubble sort algorithm

An algorithm used to order a list in correct order.



# Bubble sort algorithm

- Bubble sort algorithm is inefficient.
- We will discuss insert sort and merge sort which are more efficient than bubble sort algorithm in the next lesson.

# Bubble sort: Example

- Let us understand bubble sort using a simple example of numbered list.
- Consider the list of number: 6, 5, 4, 3, 10

# Bubble sort: First pass

i. 6, 5, 4, 3, 10

6 >5 so the numbers are swapped.

ii. 5, 6, 4, 3, 10

6 >4 so the numbers are swapped.

iii. 5, 4, 6, 3, 10

6 >3 so the numbers are swapped.

iv. 5, 4, 3, 6, 10

6 < 10 so the list remains the same.

As this pass had swapping of numbers, this list enters in to a second loop.

# Bubble sort: Second pass

i. 5, 4, 3, 6, 10

5>4 so the numbers are swapped.

ii. 4, 5, 3, 6, 10

5>3 so the numbers are swapped.

iii. 4, 3, 5, 6, 10

5<6 so the list remains the same.

iv. 4, 3, 5, 6, 10

6<10 so the list remains the same.

As this pass had swapping of numbers, this list enters in to a third loop.

# Bubble sort: Third pass

i. 4, 3, 5, 6, 10

4>3 so the numbers are swapped.

ii. 3, 4, 5, 6, 10

4<5 so the list remains the same.

iii. 3, 4, 5, 6, 10

4<5 so the list remains the same.

iv. 3, 4, 5, 6, 10

6<10 so the list remains the same.

# Bubble sort: Fourth pass

- As this pass had swapping of numbers, this list enters into a fourth loop.
- In the fourth time there will be no swapping and, hence, the list is ordered and output is produced.

# Bubble sort algorithm: Pseudocode

- We require a variable to know whether swapping has been performed.
- This is because the algorithm ends only when no swapping has been performed.
- So, a variable swapflag is used to determine whether swapping has been performed. Initially, it is set as true.

*swapflag=true*

# Bubble search algorithm: Pseudocode

- In a while loop, the value of swap flag is checked.
- Inside the loop, this value is initially set to be false.
- Using a for loop, all the elements are compared with its next element.
- The elements at position 0 and position 1 are checked and swapped if required.

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
    NEXT position
END WHILE
```

# Bubble search algorithm: Pseudocode

- If swapped, the swapflag is set as true.
- Now the elements at position 1 and position 2 are checked and swapped if required.
- This for loop continues until all the elements have been checked.
- The while loop ends only if the swapflag is false. It means that no swapping has taken place in the for loop.

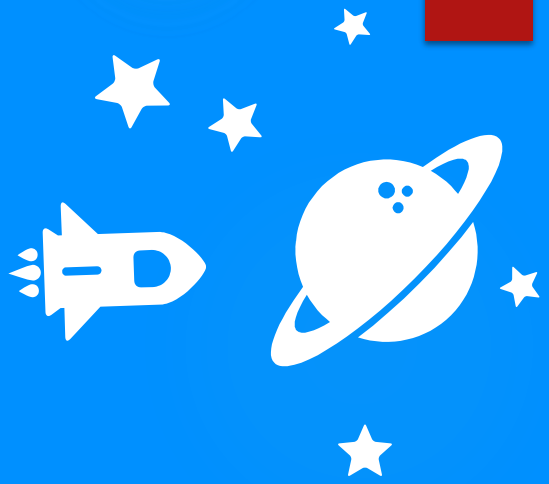
```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    NEXT position
    position = position +1
END WHILE
```

# Analysing pseudocode

- Let us analyse this pseudocode with an example. Let us consider a list:

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | C | A | D | F | E | B |

- Each for loop is denoted as a step.



# 1st For loop

i. swapflag= false

Comparing items at position 0 and 1.

Swapping is required. C and A are swapped.

Swapflag=true

Position=1

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | C | A | D | F | E | B |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
    NEXT position
END WHILE
```

ii. Comparing items at position 1 and 2.

Swapping is not required.

Position=2

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | D | F | E | B |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
    NEXT position
END WHILE
```

iii. Comparing items at position 2 and 3.

Swapping is not required.

Position=3

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | D | F | E | B |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
   NEXT position
END WHILE
```

iv. Comparing items at position 3 and 4.

Swapping is required. F and E are swapped

Swapflag=true

Position=4

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | D | F | E | B |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
    NEXT position
END WHILE
```

v. Comparing items at position 4 and 5.

Swapping is required. F and B are swapped

Swapflag=true

Position=5

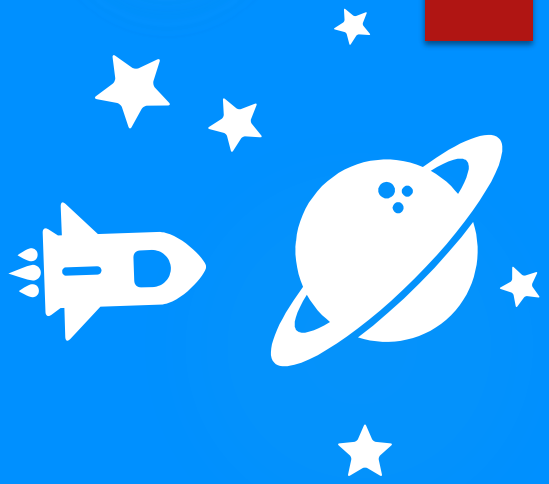
| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | D | E | F | B |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
    NEXT position
END WHILE
```

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | D | B | E | F |

Because the swapflag is set true in some of these steps. The for loop is executed once again.

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
   NEXT position
END WHILE
```



# 2nd For loop

i. swapflag= false

Comparing items at position 0 and 1.

Swapping is not required.

Position=1

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | D | B | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
   NEXT position
END WHILE
```

ii. Comparing items at position 1 and 2.

Swapping is not required.

Position=2

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | D | B | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
   NEXT position
END WHILE
```

iii. Comparing items at position 2 and 3.

Swapping is required. D and B are swapped.

Swapflag=true

Position=3

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | D | B | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
    NEXT position
END WHILE
```

iv. Comparing items at position 3 and 4.

Swapping is not required.

Position=4

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | B | D | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
   NEXT position
END WHILE
```

v. Comparing items at position 4 and 5.

Swapping is not required.

Position=5

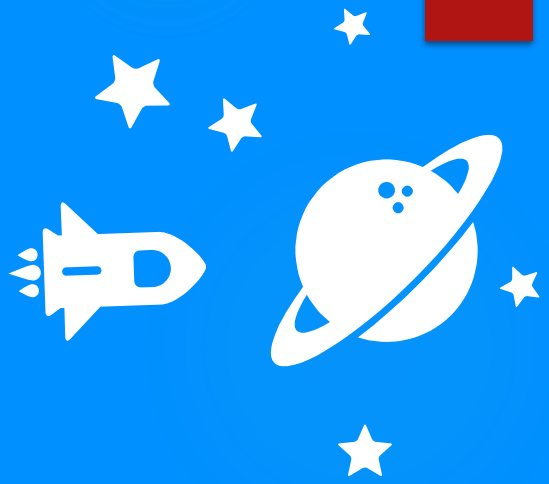
| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | B | D | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
   NEXT position
END WHILE
```

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | B | D | E | F |

Again, in this set of steps, swapflag is set true in step 3. These set of sets are again repeated in for loop.

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
   NEXT position
END WHILE
```



# 3rd For loop

i. swapflag=false

Comparing items at position 0 and 1.

Swapping is not required.

Position=1

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | B | D | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
    NEXT position
END WHILE
```

ii. Comparing items at position 1 and 2.

Swapping is required. C and B are swapped.

Swapflag=true

Position=2

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | B | D | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
    NEXT position
END WHILE
```

iii. Comparing items at position 2 and 3.

Swapping is not required.

Position=3

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | B | D | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
   NEXT position
END WHILE
```

iv. Comparing items at position 3 and 4.

Swapping is not required.

Position=4

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | B | D | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
   NEXT position
END WHILE
```

v. Comparing items at position 4 and 5.

Swapping is not required.

Position=5

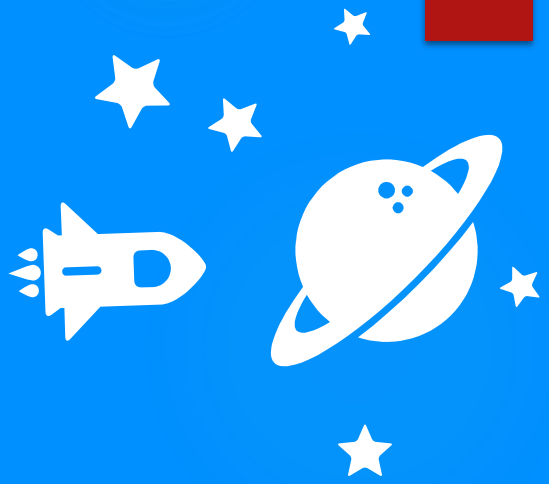
| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | B | D | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
   NEXT position
END WHILE
```

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | B | D | E | F |

- Again, in this set of steps, swapflag is set true in step 2.
- Even though the list is now sorted, these set of sets are again repeated in for loop.

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
    NEXT position
END WHILE
```



# 4th For loop

i. swapflag=false

Comparing items at position 0 and 1.

Swapping is not required.

Position=1

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | B | D | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
    NEXT position
END WHILE
```

ii. Comparing items at position 1 and 2.

Swapping is not required.

Position=2

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | B | D | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
    NEXT position
END WHILE
```

iii. Comparing items at position 2 and 3.

Swapping is not required.

Position=3

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | B | D | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
   NEXT position
END WHILE
```

iv. Comparing items at position 3 and 4.

Swapping is not required.

Position=4

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | B | D | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
    NEXT position
END WHILE
```

v. Comparing items at position 4 and 5.

Swapping is not required.

Position=5

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | B | D | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
   NEXT position
END WHILE
```

- In this final set of for loop, the swapflag has not been set true.
- Hence, the while loop ends and the output of this pseudocode is the sorted list.
- It is important to remember that characters can be compared using their ASCII codes.

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Item | A | C | B | D | E | F |

```
swapflag=true
WHILE swapflag==true
    swapflag= false
    position = 0
    FOR position = 0 to length_of_list-2
        compare (current_item, next_item)
        IF (current_item > next_item)
            swap (current_item, next_item)
            swapflag=true
        ENDIF
    position = position +1
    NEXT position
END WHILE
```

# 2.

Activity

# Activity-1

Duration: 15 minutes

1. Analyse in detail how the following list of numbers are sorted using bubble sort algorithm.

| | | | | | |
|---|---|---|---|---|---|
| 4 | 9 | 8 | 1 | 6 | 5 |

# KNOWING WHAT YOU LEARNED

- Go to: https://joinmyquiz.com
- Write your name and grade level
- Join code: _____

# 3.

End of topic questions

# End of topic questions

1. What is a bubble sort algorithm used for?
2. How does a bubble sort algorithm work?
3. For the given list, how does the bubble sort algorithm work? Explain briefly.

| | | | | | |
|---|---|---|---|---|---|
| F | R | G | E | A | U |

4. What is the purpose of the last pass in the bubble sort algorithm?

# End of topic questions

5. How many maximum comparisons and swaps take place to sort a list of 6 numbers using bubble sort algorithm? Show your working.

# CREDIT

**Teach Computer Science**