Bubble Sort

Papers Dock

COMPUTER SCIENCE 9618 PAPER 2

Bubble Sort

It's an algorithm to arrange an array in either ascending or descending order

Swapping Values

Incorrect

```
array = [ 9, 0 ]
array[1] = array[2]
array[2] = array[1]
```

Correct

```
array = [ 9, 0 ]
temp = array[1]
array[1] = array[2]
array[2] = temp
```

There are two loops in a Bubble Sort Algorithm.
One inner loop means that one value is at the correct position and is responsible for swapping of each element

```
array = [ 10,   5,   6,   3,   7,  8]
```

After one correct positioning of an element there will be another outer loop which basically determines the number of elements in an array

Difference Between Efficient and Inefficient code

Inefficient code	Efficient code
Uses both For Loops and performs extra unwanted loops	Outer loop is conditional loop with flag looping and inner loop is for loop

Question

There is an array with the name studentID and it contains 10 elements

Write a Bubble Sort PseudoCode to sort the studentID in ascending order

```
Boundary ← 5
REPEAT
    NoSwaps ← TRUE
    FOR J ← 1 TO Boundary
        IF StudentID[J] > StudentID[J + 1] THEN
            Temp ← StudentID[J]
            StudentID[J] ← StudentID[J + 1]
            StudentID[J + 1] ← Temp
            NoSwaps ← FALSE
        ENDIF
    NEXT J
    Boundary ← Boundary - 1
UNTIL NoSwaps = TRUE
```

Practice Question

5 A global 2D array `Result` of type `INTEGER` is used to store a list of exam candidate numbers together with their marks. The array contains 2000 elements, organised as 1000 rows and 2 columns.

Column 1 contains the candidate number and column 2 contains the mark for the corresponding candidate. All elements contain valid exam result data.

A procedure `Sort()` is needed to sort `Result` into ascending order of mark using an efficient bubble sort algorithm.

Write pseudocode for the procedure `Sort()`.

```
PROCEDURE Sort()
  DECLARE Temp : INTEGER
  DECLARE NoSwaps : BOOLEAN
  DECLARE Boundary, Row, Col : INTEGER

  Boundary ← 999
  REPEAT
     NoSwaps ← TRUE
     FOR Row ← 1 TO Boundary
        IF Result[Row, 2] > Result[Row + 1, 2] THEN
           FOR Col ← 1 TO 2
              Temp ← Result [Row, Col]
              Result [Row, Col] ← Result [Row + 1, Col]
              Result [Row + 1, Col] ← Temp
           NEXT Col
           NoSwaps ← FALSE
        ENDIF
     NEXT J
     Boundary ← Boundary - 1
  UNTIL NoSwaps = TRUE

ENDPROCEDURE
```