THE BUBBLE SORT:

INTRODUCTION

-Suppose we have an array, **A**, with members in any mixed up order. The problem is to sort **A** so that members are in ascending order.

-When **A** is sorted, each member will be less than or equal to the next member: **A(1) <= A(2) and A(2) <= A(3)** and so on.

-In the **BUBBLE SORT** we compare adjacent members of the array. If the 1st member is greater than the 2nd member - that is, if 2 members are in the **wrong order** - we swap the two members.

-If the 1st member is less than or equal to the 2nd member - that is, if the 2 members are in the **right order** - we leave the two members alone.

-Every time we swap 2 members, the array gets a little closer to being sorted.

-If we continue swapping long enough eventually the whole array will be sorted.

EXAMPLE CASE

A				
1	2	3	4	5
3	7	1	9	6

INITIAL VALUES OF A

A				
1	2	3	4	5
3	7	1	9	6
3	1	7	6	9

VALUES OF A AFTER 1ST PASS

A				
1	2	3	4	5
3	7	1	9	6
3	1	7	6	9
1	3	6	7	9

VALUES OF A AFTER 2ND PASS

PROGRAM

The **BUBBLE SORT** involves repeated passes through the array, swapping members that are in the **wrong** order until the array is sorted.

a) Comparison and Swapping Steps

(1) The bubble sort requires that we compare adjacent members of **A**. That is, we want to compare:

A(1) WITH A(2)
A(2) WITH A(3)
A(3) WITH A(4)
. . . .
A(N-1) WITH A(N)

HOW CAN WE PERFORM ALL THESE COMPARISONS

-We **don't** want to write individual **IF** statements
-We only need on **IF** statement if we compare **A(k) with A(k+1)**
-We put this **IF** statement inside a **FOR/NEXT** loop in which k is the counter variable
-When k=1 we will compare A(1) WITH A(2)

- When k=2 we will compare A(2) WITH A(3) ...

Notice, however, that the highest value **k** should have is **n-1**
When k=n-1 => A(K+1)=A(N)
A(K) = A(N-1)
So when k=n-1 we will compare A(N-1) WITH A(N)

(2) If the condition **A(K) <= A(K+1)** is **TRUE**, we want to leave those two members alone, so we branch directly to the **NEXT** statement.

If on the other hand, the condition **FALSE**, we want to **SWAP A(K) AND A(K+1)**

HOW CAN WE SWAP A(K) AND A(K+1)

-First thought => A(K) = A(K+1) BUT THIS DOES NOT WORK
A(K+1) = A(K)

Trace:

A(2)	A(3)
7	1

Since A(2) is greater than A(3) we want to swap these two members.
Using our first thought:
A(2) = A(3)
A(3) = A(2)

A(2) = 1
A(3) = 1 THE SECOND STATEMENT ACCOMPLISHES NOTHING

Note:

-The 7 that was originally in A(2) has been lost
To save that 7 we must store it in a **TEMPORARY** area.
Let's call the temporary area **T**
The series of statements that will swap the values in **A(K)** and **A(k+1)** are:
T = A(K)
A(K) = A(K+1)
A(K+1) = T

(3) Comparison and Swapping Module

```
for k = 1 to n-1
    if a(k) > a(k+1) then
        t = a(k)
        a(k) = a(k+1)
        a(k+1) = t
    end if

next k
```

b) Use of a Flag

HOW DO WE KNOW WHEN TO STOP

-Suppose we are at the position right after next k in the program above.

-If all the pairs of members of **A** are in the right order, **the sort is finished**, and we can go on to the next step.

-But if even one pair of members of **A** is in the **wrong** order the sort is not finished, and we must loop back to the start of the for/next loop and execute another pass through.

HOW DO WE KNOW THAT ALL THE PAIRS OF MEMBERS ARE IN THE RIGHT ORDER

-If the condition **A(K) <= A(K+1)** were always true, all the pairs of members would be in the right order and the sort would be complete.

-But how do we know that on the previous pass the condition was always true.

-To answer this question we introduce a **FLAG**

THE FLAG

(1) Call the flag 'flag'
(2) Let it have the value **yes** if the sort is finished and **no** if it is not.
(3) We set flag="yes" just before the for/next loop
(4) We set it to **no** if a(k) > a(k+1) is **true** (the swapping module)
(5) We test flag (for yes/no) just after the next statement
(6) If flag="no" at this test position we know the sort is not yet finished. We branch back to 3.
(7) If flag = "yes" , go on to the next steps below.

c) The Bubble Sort

```
sub bubblesort

do
        flag = "yes"
        for k = 1 to n-1
                if a(k) > a(k+1) then
                        t = a(k)
                        a(k) = a(k+1)
                        a(k+1) = t
                        flag = "no"
                end if
        next k
loop until flag = "yes"

end sub
```

THE MODIFIED BUBBLE SORT

In the above bubble sort we compare all the adjacent members of array A on each pass. Some of these comparisons are unnecessary. Lets introduce a new variable H. Every time we swap A(k) and A(k+1), we set H equal to K. Since K starts from one and gets larger, H will be equal to the highest value of K for which A(k) and A(k+1) were swapped.

For example, suppose A has 20 members and on a particular pass members 7 and 8,10,11 and 13 and 14 were swapped. Then on this pass H will be set to 13. A little thought will show that all members of A greater than or equal to 13 are now sorted. (That is why we did not have to swap any of them). Therefore on the next pass through A we do not have to compare all the adjacent members of A. We can stop at the comparison between A(12) and A(13). The easiest way to do this is to allow the final value of the FOR statement to vary, instead of holding it constant at N-1. Let the final value of the FOR statement be the variable U. On each pass set U equal to H-1. (Of course on the first pass it is necessary that U equal N-1).

Furthermore we do not need a separate flag, we can use H as a flag. As we start each pass through A, we set H equal to 0. If at the end of the pass H is less than or equal to 1, the SORT IS FINISHED.

THE SHELL SORT - WALKTHROUGH

EVEN CASE

	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)
	2	6	4	3	5	1
GAP=3 MAX =3	2	5	1	3	6	4
GAP=1 MAX=5	2	1	5	3	6	4
	2	1	3	5	6	4
	2	1	3	5	4	6
	1	2	3	4	5	6

ODD CASE

	A(1)	A(2)	A(3)	A(4)	A(5)
	4	3	5	2	1
GAP=2 MAX =3	4	2	1	3	5
GAP=1 MAX=4	1	2	4	3	5
	1	2	3	4	5