NAME: SADAR SARMAD.
ROLL_NUMBER: 22P-9009.
SECTION: BS_AI_4A.

NOTE: THERE ARE TWO CODES SO SCROLL DOWN FOR FURTHER CODES.

CODE:

```
; bubble sort algorithm as a subroutine
[org 0x0100]
jmp start
data: dw 60, 55, 45, 50, 40, 35, 25, 30, 10, 0
swap: db 0
bubblesort: dec cx ; last element not compared
shl cx, 1 ; turn into byte count
mainloop: mov si, 0 ; initialize array index to zero
mov byte [swap], 0 ; reset swap flag to no swaps
innerloop: mov ax, [bx+si] ; load number in ax
cmp ax, [bx+si+2] ; compare with next number
jbe noswap ; no swap if already in order
mov dx, [bx+si+2] ; load second element in dx
mov [bx+si], dx ; store first number in second
mov [bx+si+2], ax ; store second number in first
mov byte [swap], 1 ; flag that a swap has been done
noswap: add si, 2 ; advance si to next index
cmp si, cx ; are we at last index
jne innerloop ; if not compare next two
cmp byte [swap], 1 ; check if a swap has been done
je mainloop ; if yes make another pass
ret ; go back to where we came from
start: mov bx, data ; send start of array in bx
mov cx, 10 ; send count of elements in cx
call bubblesort ; call our subroutine
mov ax, 0x4c00 ; terminate program
int 0x21
```

EXPLANATION:

- **Data Section**

1. data: holds the array to be sorted.
2. swap: is a flag used to determine if any swaps occurred during a pass.

- **Bubble Sort Subroutine**

1. The subroutine bubblesort sorts the array using the bubble sort algorithm.
2. outer_loop: decrements the element count cx and converts it to a byte count by shifting left (shl cx, 1).
3. mainloop: initializes the array index si to zero and resets the swap flag.
4. innerloop: compares each pair of adjacent elements and swaps them if they are out of order.
5. The loop continues until no swaps are needed, indicated by the swap flag.

- **Start Section**

1. start: initializes the array pointer bx to the start of the data array and sets the element count cx.
2. Calls the bubblesort subroutine to sort the array.
3. Terminates the program with a DOS interrupt int 0x21.

- **Notes:**

1. The subroutine bubblesort uses bx to point to the array and cx to determine the number of elements.
2. The program uses si to iterate through the array indices.
3. Each element is 2 bytes (a word), so the array is handled accordingly.
4. The loop ensures that the largest unsorted element is moved to its correct position with each pass through the array. The process repeats until no swaps are necessary.

● CODE NO 2:

```
; bubble sort subroutine called twice
[org 0x0100]
jmp start
data: dw 60, 55, 45, 50, 40, 35, 25, 30, 10, 0
data2: dw 328, 329, 898, 8923, 8293, 2345, 10, 877, 355, 98
dw 888, 533, 2000, 1020, 30, 200, 761, 167, 90, 5
```

```
swap: db 0
bubblesort: dec cx ; last element not compared
shl cx, 1 ; turn into byte count
mainloop: mov si, 0 ; initialize array index to zero
mov byte [swap], 0 ; reset swap flag to no swaps
innerloop: mov ax, [bx+si] ; load number in ax
cmp ax, [bx+si+2] ; compare with next number
jbe noswap ; no swap if already in order
mov dx, [bx+si+2] ; load second element in dx
mov [bx+si], dx ; store first number in second
mov [bx+si+2], ax ; store second number in first
mov byte [swap], 1 ; flag that a swap has been done
noswap: add si, 2 ; advance si to next index CS401@vu.edu.pk
cmp si, cx ; are we at last index
jne innerloop ; if not compare next two
cmp byte [swap], 1 ; check if a swap has been done
je mainloop ; if yes make another pass
ret ; go back to where we came from
start: mov bx, data ; send start of array in bx
mov cx, 10 ; send count of elements in cx
call bubblesort ; call our subroutine
mov bx, data2 ; send start of array in bx
mov cx, 20 ; send count of elements in cx
call bubblesort ; call our subroutine again
mov ax, 0x4c00 ; terminate program
int 0x21
```

● EXPLANATION:

1. Initialization and Jump to Start:

The program begins by setting the origin of the code to a specific memory address (common for DOS programs) and then immediately jumps to the start label to begin execution.

2. Data Declarations:

Two arrays (data and data2) are declared. The first array (data) contains 10 elements, and the second array (data2) contains 20 elements. These arrays will be sorted by the bubble sort subroutine.

A swap variable is also declared and initialized to 0. This variable will be used as a flag to indicate if any swaps occurred during a pass through the array.

3. **Bubble Sort Subroutine:**

The subroutine begins by decrementing the element count (cx) because the last element does not need to be compared.

The element count is then converted into a byte count by shifting it left, as each element is 2 bytes (a word).

The main sorting loop (mainloop) initializes the array index (si) to zero and resets the swap flag to indicate no swaps have occurred yet.

The inner loop (innerloop) iterates through the array, comparing each pair of adjacent elements. If an element is out of order, it swaps the two elements and sets the swap flag.

After completing one pass through the array, the subroutine checks if any swaps occurred. If swaps were made, it repeats the process; otherwise, it ends.

4. **Start Section:**

The start section initializes the pointer (bx) to the beginning of the first array (data) and sets the element count (cx) to 10.

It calls the bubble sort subroutine to sort the first array.

It then reinitializes the pointer (bx) to the beginning of the second array (data2) and sets the element count (cx) to 20.

It calls the bubble sort subroutine again to sort the second array.

Finally, the program terminates by invoking a DOS interrupt.

5. **Summary**

The program sorts two arrays using a bubble sort algorithm implemented as a subroutine.

It first sorts a smaller array (`data` with 10 elements) and then sorts a larger array (`data2` with 20 elements).

The bubble sort subroutine works by repeatedly passing through the array, swapping out-of-order elements, and continuing this process until no more swaps are needed, indicating the array is sorted.

The program terminates after sorting both arrays.