**1]Write an ALP to implement bubble sort**

**INPUT –**

%macro print 2

    push eax

    push ebx

    push ecx

    mov eax, 4

    mov ebx, 1

    mov ecx, %1

    mov edx, %2

    int 0x80

    pop ecx

    pop ebx

    pop eax

%endmacro

%macro exit 0

    mov eax, 1

    xor ebx, ebx

    int 0x80

%endmacro

section .data

    prompt\_size db "Enter number of elements: ", 0

    prompt\_element db "Enter element: ", 0

    msg\_iteration db "Iteration ", 0

    msg\_array db "Array: ", 0

    msg\_complete db "Bubble sort executed", 10, 0

    msg\_sorted db "Sorted array: ", 0

    newline db 10, 0

    space db " ", 0

    minus db "-", 0

section .bss

    array resd 100

    size resd 1

    buffer resb 12

    negative resb 1

    swapped resb 1

section .text

global \_start

\_start:

    print prompt\_size, 25

    call read\_int

    mov [size], eax

    xor ebx, ebx

input\_loop:

    cmp ebx, [size]

    jge input\_done

    print prompt\_element, 15

    call read\_int

    mov [array + ebx\*4], eax

    inc ebx

    jmp input\_loop

input\_done:

    print newline, 1

    mov ecx, [size]

    dec ecx

    jle sort\_done

bubble\_sort:

    mov byte [swapped], 0

    xor ebx, ebx

inner\_loop:

    mov eax, [array + ebx\*4]

    cmp eax, [array + ebx\*4 + 4]

    jle no\_swap

    xchg eax, [array + ebx\*4 + 4]

    mov [array + ebx\*4], eax

    mov byte [swapped], 1

no\_swap:

    inc ebx

    cmp ebx, ecx

    jl inner\_loop

    push ecx

    print newline, 1

    print msg\_iteration, 10

    mov eax, [size]

    sub eax, ecx

    call print\_int

    print newline, 1

    print msg\_array, 7

    call print\_array

    print newline, 1

    pop ecx

    dec ecx

    cmp byte [swapped], 0

    jne bubble\_sort

sort\_done:

    print newline, 1

    print msg\_complete, 19

    print newline, 1

    print msg\_sorted, 14

    call print\_array

    print newline, 1

    exit

read\_int:

    push ebx

    push ecx

    push edx

    push esi

    mov eax, 3

    mov ebx, 0

    mov ecx, buffer

    mov edx, 12

    int 0x80

    mov byte [negative], 0

    mov esi, buffer

    xor eax, eax

    xor ebx, ebx

    mov bl, [esi]

    cmp bl, '-'

    jne convert\_start

    mov byte [negative], 1

    inc esi

convert\_start:

    movzx edx, byte [esi]

    inc esi

    cmp dl, 10

    je convert\_done

    cmp dl, '0'

    jb convert\_start

    cmp dl, '9'

    ja convert\_start

    sub dl, '0'

    imul eax, 10

    add eax, edx

    jmp convert\_start

convert\_done:

    cmp byte [negative], 1

    jne positive\_num

    neg eax

positive\_num:

    pop esi

    pop edx

    pop ecx

    pop ebx

    ret

print\_int:

    push eax

    push ebx

    push ecx

    push edx

    push esi

    push edi

    test eax, eax

    jns non\_negative

    push eax

    print minus, 1

    pop eax

    neg eax

non\_negative:

    mov ecx, 10

    mov edi, buffer

    add edi, 11

    mov byte [edi], 0

    mov esi, edi

    test eax, eax

    jnz digit\_loop

    dec edi

    mov byte [edi], '0'

    jmp print\_num

digit\_loop:

    dec edi

    xor edx, edx

    div ecx

    add dl, '0'

    mov [edi], dl

    test eax, eax

    jnz digit\_loop

print\_num:

    mov ecx, edi

    mov edx, esi

    sub edx, edi

    mov eax, 4

    mov ebx, 1

    int 0x80

print\_int\_done:

    pop edi

    pop esi

    pop edx

    pop ecx

    pop ebx

    pop eax

    ret

print\_array:

    push eax

    push ebx

    push ecx

    xor ebx, ebx

print\_loop:

    cmp ebx, [size]

    jge print\_done

    mov eax, [array + ebx\*4]

    call print\_int

    print space, 1

    inc ebx

    jmp print\_loop

print\_done:

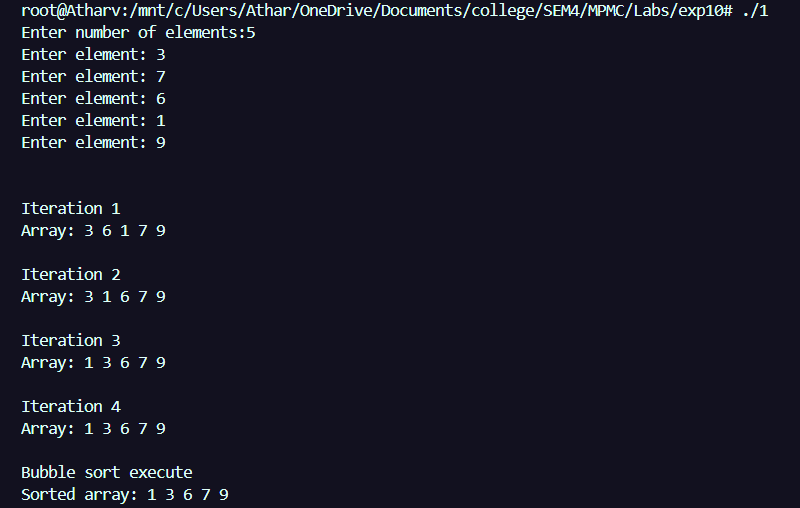
    pop ecx

    pop ebx

    pop eax

    ret

**OUTPUT –**

****

**2]Write an ALP to implement insertion sort**

**INPUT –**

%macro print 2

    push eax

    push ebx

    push ecx

    mov eax, 4

    mov ebx, 1

    mov ecx, %1

    mov edx, %2

    int 0x80

    pop ecx

    pop ebx

    pop eax

%endmacro

%macro exit 0

    mov eax, 1

    xor ebx, ebx

    int 0x80

%endmacro

section .data

    prompt\_size db "Enter number of elements: ", 0

    prompt\_element db "Enter element: ", 0

    msg\_iteration db "Iteration ", 0

    msg\_array db "Array: ", 0

    msg\_complete db "Insertion sort executed", 10, 0

    msg\_sorted db "Sorted array: ", 0

    newline db 10, 0

    space db " ", 0

    minus db "-", 0

section .bss

    array resd 100

    size resd 1

    buffer resb 12

    negative resb 1

    key resd 1

section .text

global \_start

\_start:

    print prompt\_size, 25

    call read\_int

    mov [size], eax

    xor ebx, ebx

input\_loop:

    cmp ebx, [size]

    jge input\_done

    print prompt\_element, 15

    call read\_int

    mov [array + ebx\*4], eax

    inc ebx

    jmp input\_loop

input\_done:

    print newline, 1

    mov ecx, 1

insertion\_sort:

    cmp ecx, [size]

    jge sort\_done

    mov eax, [array + ecx\*4]

    mov [key], eax

    mov ebx, ecx                *; j = i*

    dec ebx                     *; j = i-1*

    print newline, 1

    print msg\_iteration, 10

    mov eax, ecx

    call print\_int

    print newline, 1

    print msg\_array, 7

    call print\_array

    print newline, 1

while\_loop:

    cmp ebx, 0                  *; while j >= 0*

    jl end\_while

    mov eax, [array + ebx\*4]

    cmp eax, [key]              *; and array[j] > key*

    jle end\_while

    mov edx, [array + ebx\*4]    *; array[j+1] = array[j]*

    mov [array + ebx\*4 + 4], edx

    dec ebx                     *; j = j-1*

    jmp while\_loop

end\_while:

    mov eax, [key]              *; array[j+1] = key*

    mov [array + ebx\*4 + 4], eax

    inc ecx                     *; i++*

    jmp insertion\_sort

sort\_done:

    print newline, 1

    print msg\_complete, 23

    print newline, 1

    print msg\_sorted, 14

    call print\_array

    print newline, 1

    exit

read\_int:

    push ebx

    push ecx

    push edx

    push esi

    mov eax, 3

    mov ebx, 0

    mov ecx, buffer

    mov edx, 12

    int 0x80

    mov byte [negative], 0

    mov esi, buffer

    xor eax, eax

    xor ebx, ebx

    mov bl, [esi]

    cmp bl, '-'

    jne convert\_start

    mov byte [negative], 1

    inc esi

convert\_start:

    movzx edx, byte [esi]

    inc esi

    cmp dl, 10

    je convert\_done

    cmp dl, '0'

    jb convert\_start

    cmp dl, '9'

    ja convert\_start

    sub dl, '0'

    imul eax, 10

    add eax, edx

    jmp convert\_start

convert\_done:

    cmp byte [negative], 1

    jne positive\_num

    neg eax

positive\_num:

    pop esi

    pop edx

    pop ecx

    pop ebx

    ret

print\_int:

    push eax

    push ebx

    push ecx

    push edx

    push esi

    push edi

    test eax, eax

    jns non\_negative

    push eax

    print minus, 1

    pop eax

    neg eax

non\_negative:

    mov ecx, 10

    mov edi, buffer

    add edi, 11

    mov byte [edi], 0

    mov esi, edi

    test eax, eax

    jnz digit\_loop

    dec edi

    mov byte [edi], '0'

    jmp print\_num

digit\_loop:

    dec edi

    xor edx, edx

    div ecx

    add dl, '0'

    mov [edi], dl

    test eax, eax

    jnz digit\_loop

print\_num:

    mov ecx, edi

    mov edx, esi

    sub edx, ecx

    mov eax, 4

    mov ebx, 1

    int 0x80

print\_int\_done:

    pop edi

    pop esi

    pop edx

    pop ecx

    pop ebx

    pop eax

    ret

print\_array:

    push eax

    push ebx

    push ecx

    xor ebx, ebx

print\_loop:

    cmp ebx, [size]

    jge print\_done

    mov eax, [array + ebx\*4]

    call print\_int

    print space, 1

    inc ebx

    jmp print\_loop

print\_done:

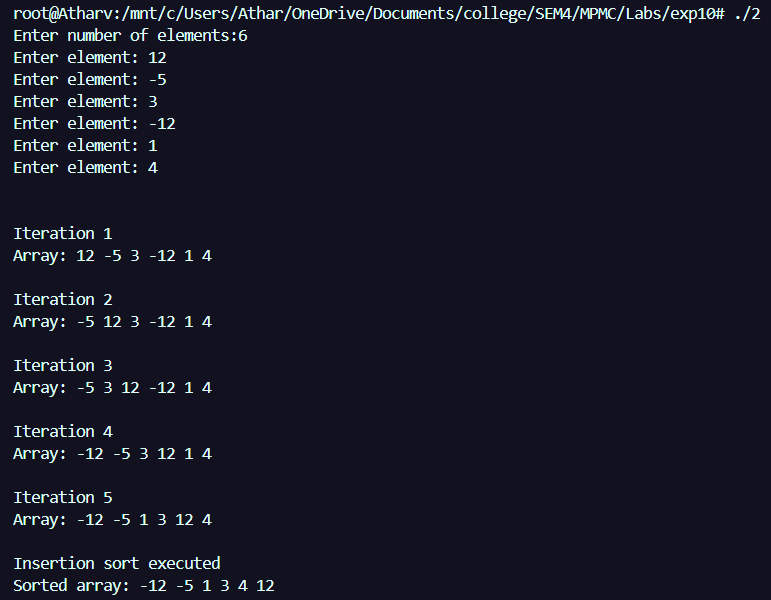
    pop ecx

    pop ebx

    pop eax

    ret

**OUTPUT –**

****

**3]Write an ALP to implement selection sort**

**INPUT –**

%macro print 2

    push eax

    push ebx

    push ecx

    mov eax, 4

    mov ebx, 1

    mov ecx, %1

    mov edx, %2

    int 0x80

    pop ecx

    pop ebx

    pop eax

%endmacro

%macro exit 0

    mov eax, 1

    xor ebx, ebx

    int 0x80

%endmacro

section .data

    prompt\_size db "Enter number of elements: ", 0

    prompt\_element db "Enter element: ", 0

    msg\_iteration db "Iteration ", 0

    msg\_array db "Array: ", 0

    msg\_complete db "Selection sort executed", 10, 0

    msg\_sorted db "Sorted array: ", 0

    newline db 10, 0

    space db " ", 0

    minus db "-", 0

section .bss

    array resd 100

    size resd 1

    buffer resb 12

    negative resb 1

    min\_idx resd 1

section .text

global \_start

\_start:

    print prompt\_size, 25

    call read\_int

    mov [size], eax

    xor ebx, ebx

input\_loop:

    cmp ebx, [size]

    jge input\_done

    print prompt\_element, 15

    call read\_int

    mov [array + ebx\*4], eax

    inc ebx

    jmp input\_loop

input\_done:

    print newline, 1

    mov ecx, 0

selection\_sort:

    cmp ecx, [size]

    jge sort\_done

    mov [min\_idx], ecx

    mov ebx, ecx

    inc ebx

    print msg\_iteration, 10

    mov eax, ecx

    inc eax

    call print\_int

    print newline, 1

    print msg\_array, 7

    print newline, 1

    call print\_array

    print newline, 1

    print newline, 1

inner\_loop:

    cmp ebx, [size]

    jge swap\_elements

    mov eax, [array + ebx\*4]

    mov edx, [min\_idx]

    mov edx, [array + edx\*4]

    cmp eax, edx

    jge skip\_update

    mov [min\_idx], ebx

skip\_update:

    inc ebx

    jmp inner\_loop

swap\_elements:

    mov eax, [array + ecx\*4]

    mov edx, [min\_idx]

    mov ebx, [array + edx\*4]

    mov [array + ecx\*4], ebx

    mov [array + edx\*4], eax

    inc ecx

    jmp selection\_sort

sort\_done:

    print msg\_complete, 23

    print newline, 1

    print msg\_sorted, 14

    call print\_array

    print newline, 1

    exit

read\_int:

    push ebx

    push ecx

    push edx

    push esi

    mov eax, 3

    mov ebx, 0

    mov ecx, buffer

    mov edx, 12

    int 0x80

    mov byte [negative], 0

    mov esi, buffer

    xor eax, eax

    xor ebx, ebx

    mov bl, [esi]

    cmp bl, '-'

    jne convert\_start

    mov byte [negative], 1

    inc esi

convert\_start:

    movzx edx, byte [esi]

    inc esi

    cmp dl, 10

    je convert\_done

    cmp dl, '0'

    jb convert\_start

    cmp dl, '9'

    ja convert\_start

    sub dl, '0'

    imul eax, 10

    add eax, edx

    jmp convert\_start

convert\_done:

    cmp byte [negative], 1

    jne positive\_num

    neg eax

positive\_num:

    pop esi

    pop edx

    pop ecx

    pop ebx

    ret

print\_int:

    push eax

    push ebx

    push ecx

    push edx

    push esi

    push edi

    test eax, eax

    jns non\_negative

    push eax

    print minus, 1

    pop eax

    neg eax

non\_negative:

    mov ecx, 10

    mov edi, buffer

    add edi, 11

    mov byte [edi], 0

    mov esi, edi

    test eax, eax

    jnz digit\_loop

    dec edi

    mov byte [edi], '0'

    jmp print\_num

digit\_loop:

    dec edi

    xor edx, edx

    div ecx

    add dl, '0'

    mov [edi], dl

    test eax, eax

    jnz digit\_loop

print\_num:

    mov ecx, edi

    mov edx, esi

    sub edx, ecx

    mov eax, 4

    mov ebx, 1

    int 0x80

print\_int\_done:

    pop edi

    pop esi

    pop edx

    pop ecx

    pop ebx

    pop eax

    ret

print\_array:

    push eax

    push ebx

    push ecx

    xor ebx, ebx

print\_loop:

    cmp ebx, [size]

    jge print\_done

    mov eax, [array + ebx\*4]

    call print\_int

    print space, 1

    inc ebx

    jmp print\_loop

print\_done:

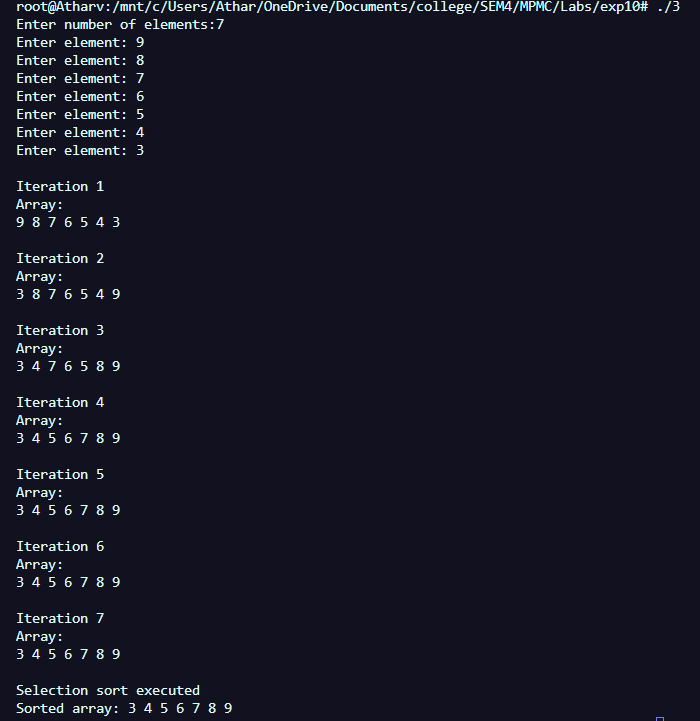
    pop ecx

    pop ebx

    pop eax

    ret

**OUTPUT –**

****

**CONCLUSION – Sorting algorithms were successfully implemented using NASM.**