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 –

```
root@Atharv:/mnt/c/Users/Athar/OneDrive/Documents/college/SEM4/MPMC/Labs/exp10# ./1
Enter number of elements:5
Enter element: 3
Enter element: 7
Enter element: 6
Enter element: 1
Enter element: 9


Iteration 1
Array: 3 6 1 7 9

Iteration 2
Array: 3 1 6 7 9

Iteration 3
Array: 1 3 6 7 9

Iteration 4
Array: 1 3 6 7 9

Bubble sort execute
Sorted array: 1 3 6 7 9
```

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 –

```
root@Atharv:/mnt/c/Users/Athar/OneDrive/Documents/college/SEM4/MPMC/Labs/exp10# ./2
Enter number of elements:6
Enter element: 12
Enter element: -5
Enter element: 3
Enter element: -12
Enter element: 1
Enter element: 4


Iteration 1
Array: 12 -5 3 -12 1 4

Iteration 2
Array: -5 12 3 -12 1 4

Iteration 3
Array: -5 3 12 -12 1 4

Iteration 4
Array: -12 -5 3 12 1 4

Iteration 5
Array: -12 -5 1 3 12 4

Insertion sort executed
Sorted array: -12 -5 1 3 4 12
```

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 –

```
root@Atharv:/mnt/c/Users/Athar/OneDrive/Documents/college/SEM4/MPMC/Labs/exp10# ./3
Enter number of elements:7
Enter element: 9
Enter element: 8
Enter element: 7
Enter element: 6
Enter element: 5
Enter element: 4
Enter element: 3

Iteration 1
Array:
9 8 7 6 5 4 3

Iteration 2
Array:
3 8 7 6 5 4 9

Iteration 3
Array:
3 4 7 6 5 8 9

Iteration 4
Array:
3 4 5 6 7 8 9

Iteration 5
Array:
3 4 5 6 7 8 9

Iteration 6
Array:
3 4 5 6 7 8 9

Iteration 7
Array:
3 4 5 6 7 8 9

Selection sort executed
Sorted array: 3 4 5 6 7 8 9
```

CONCLUSION – Sorting algorithms were successfully implemented using NASM.