1]Write an ALP to implement linear search

INPUT –

```
%macro print 2
    push eax
    push ebx
    mov eax, 4
    mov ebx, 1
    mov ecx, %1
    mov edx, %2
    int 0x80
    pop ebx
    pop eax
%endmacro
%macro exit 0
    mov eax, 1
    xor ebx, ebx
    int 0x80
%endmacro
%macro read_stdin 2
    mov eax, 3
    mov ebx, 0
    mov ecx, %1
    mov edx, %2
    int 0x80
%endmacro
section .data
    prompt_size db "Enter the number of elements: ", 0
    prompt_element db "Enter element: ", 0
    prompt_target db "Enter the target number to search: ", 0
    msg_found db "Element found at index: ", 0
    msg_not_found db "Element not found", 0
    msg_iteration db "Iteration ", 0
    msg_checking db ", Checking index: ", 0
    msg_value db ", Value: ", 0
    newline db 10, 0
section .bss
    array resd 100
    size resd 1
    target resd 1
    buffer resb 10
    current resd 1
section .text
global _start
_start:
    print prompt_size, 30
    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 prompt_target, 34
    call read_int
```

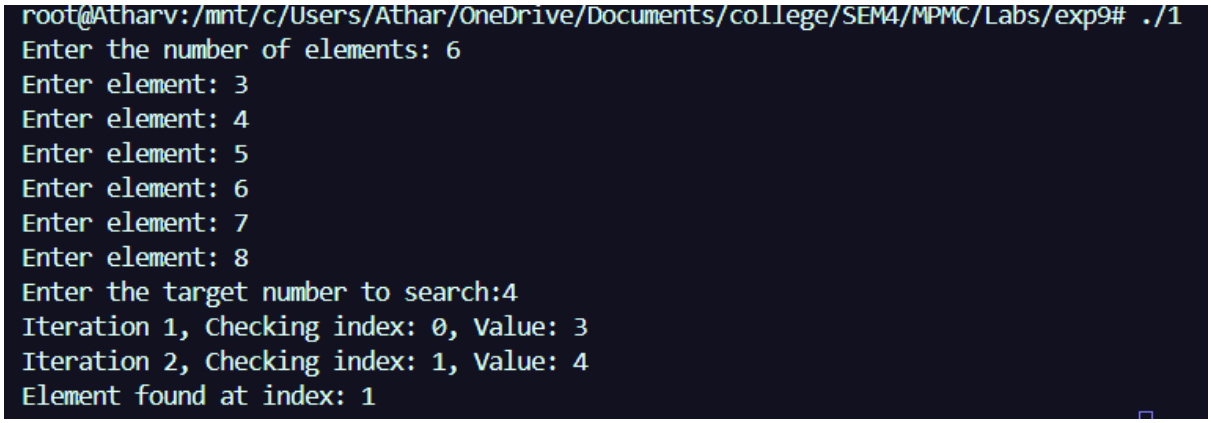
```
    mov [target], eax
    xor ebx, ebx
search_loop:
    cmp ebx, [size]
    jge not_found
    print msg_iteration, 10
    mov eax, ebx
    inc eax
    call print_int
    print msg_checking, 18
    mov eax, ebx
    call print_int
    print msg_value, 9
    mov eax, [array + ebx*4]
    mov [current], eax
    call print_int
    print newline, 1
    mov eax, [current]
    cmp eax, [target]
    je found
    inc ebx
    jmp search_loop
found:
    print msg_found, 24
    mov eax, ebx
    call print_int
    print newline, 1
    jmp exit_program
not_found:
    print msg_not_found, 17
    print newline, 1
exit_program:
    exit
```

```
read_int:
    push ebx
    push ecx
    push edx
    push esi
    read_stdin buffer, 10
    dec eax
    mov ecx, eax
    mov esi, buffer
    xor eax, eax
convert_loop:
    test ecx, ecx
    jz convert_done
    movzx edx, byte [esi]
    inc esi
    cmp dl, 10
    je convert_skip
    sub dl, '0'
    cmp dl, 9
    ja convert_skip
    imul eax, 10
    add eax, edx
convert_skip:
    dec ecx
    jmp convert_loop
convert_done:
    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
    jnz non_zero
    mov byte [buffer], '0'
    mov byte [buffer+1], 0
    print buffer, 1
    jmp print_int_done
non_zero:
    mov ecx, 10
    mov edi, buffer
    add edi, 9
    mov byte [edi], 0
    mov esi, edi
digit_loop:
    dec edi
    xor edx, edx
    div ecx
    add dl, '0'
    mov [edi], dl
    test eax, eax
    jnz digit_loop
    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

OUTPUT –



2]Write an ALP to implement binary search

INPUT –

```
SECTION .data
   msg1 db 'Enter the number of elements in array:
'
   msg1len equ $-msg1

   msg2 db 'Enter the elements: '
   msg2len equ $-msg2

   msg3 db 'Enter the number to be searched: '
   msg3len equ $-msg3

   msg4 db 'Number found at Index '
   msg4len equ $-msg4

   msg5 db 'Number not found in array'
   msg5len equ $-msg5

   msgIter db 'Iteration ', 0
   msgLow db 'Low: ', 0
   msgMid db 'Mid: ', 0
   msgHigh db 'High: ', 0

   newline db 10
   n1 equ $-newline

%macro writeSys 2
   mov eax, 4
   mov ebx, 1
   mov ecx, %1
   mov edx, %2
   int 80h
%endmacro

%macro readSys 2
   mov eax, 3
   mov ebx, 2
   mov ecx, %1
   mov edx, %2
   int 80h
   mov eax, 3
   mov ebx, 2
   mov ecx, trash
   mov edx, 1
   int 80h
%endmacro

SECTION .bss
   num resb 1
   arr resb 10
   s resb 1
   i resb 1
   low resb 1
   mid resb 1
   high resb 1
   trash resb 1
   charBuffer resb 1
   iterations resb 1

SECTION .text
```

```
    GLOBAL _start

_start:
    writeSys msg1, msg1len
    readSys num, 1
    sub byte [num], '0'

    writeSys msg2, msg2len
    mov eax, arr
    movzx edx, byte [num]
    call input
    writeSys newline, n1

    writeSys msg3, msg3len
    readSys s, 1
    sub byte [s], '0'       ; Convert ASCII to number
    movzx edi, byte [s]
    mov eax, arr
    movzx edx, byte [num]
    call binarySearch

    mov eax, 1
    xor ebx, ebx
    int 80h

input:
    mov ecx, 0
iLoop:
    cmp ecx, edx
    jge inputDone
    mov esi, eax
    add esi, ecx
    pushad
    readSys esi, 1
    popad
    inc ecx
    jmp iLoop
inputDone:
    ret

binarySearch:
    mov byte [iterations], 0
    mov byte [low], 0
    mov [high], dl

bsLoop:
    ; Exit if low > high
    movzx ecx, byte [low]
    movzx edx, byte [high]
    cmp ecx, edx
    jg notFound

    ; Calculate mid = (low + high) / 2
    pushad
    mov al, [low]
    add al, [high]
    cbw
    mov bl, 2
    div bl
    mov [mid], al
    popad

    ; Print Iteration, Low, Mid, High
    inc byte [iterations]
    pushad
        writeSys msgIter, 9
```

```
    mov al, [iterations]
    add al, '0'
    mov [charBuffer], al
    writeSys charBuffer, 1
    writeSys newline, n1

    writeSys msgLow, 5
    mov al, [low]
    add al, '0'
    mov [charBuffer], al
    writeSys charBuffer, 1
    writeSys newline, n1

    writeSys msgMid, 5
    mov al, [mid]
    add al, '0'
    mov [charBuffer], al
    writeSys charBuffer, 1
    writeSys newline, n1

    writeSys msgHigh, 6
    mov al, [high]
    add al, '0'
    mov [charBuffer], al
    writeSys charBuffer, 1
    writeSys newline, n1

    writeSys newline, n1
popad

; Load mid value and compare
movzx edx, byte [mid]
    movzx esi, byte [eax + edx]
    sub esi, '0'        ; Convert ASCII to number
    cmp edi, esi
    je found
    jl goLeft

    ; Go right
    mov bl, [mid]
    inc bl
    mov [low], bl
    jmp bsLoop

goLeft:
    mov bl, [mid]
    dec bl
    mov [high], bl
    jmp bsLoop

found:
    add edx, '0'
    mov [i], dl
    pushad
    writeSys msg4, msg4len
    writeSys i, 1
    writeSys newline, n1
    popad
    ret

notFound:
    writeSys msg5, msg5len
    writeSys newline, n1
    ret
```

OUTPUT –

```
root@Atharv:/mnt/c/Users/Athar/OneDrive/Documents/college/SEM4/MPMC/Labs/exp9# ./2
Enter the number of elements in array: 6
Enter the elements: 3
4
6
7
8
9

Enter the number to be searched: 3
Iteration1
Low: 0
Mid: 3
High: 6

Iteration2
Low: 0
Mid: 1
High: 2

Iteration3
Low: 0
Mid: 0
High: 0

Number found at Index 0
```

CONCLUSION – Linear search and binary search were successfully implemented using NASM.