Computer Organization  
Lab Assignment

# Assignment 9 Quick Sort using EMU8086

Name: Beeta Samad  
Roll Number: 181210016

1. **Write‌ ‌an assembly program to implement Quick Sort. ‌**

org 100h

.data

    initial\_statement db "The array before sorting: $"

    final\_statement db "The array after sorting: $"

    arr db 7, 6, 3, 1, 4    *;The array to be sorted*

    arr\_length db 5   *;The number of elements in the array*

    p DB ?

    q DB ?

    i DB ?

    l DB 0

    h DB 5

*;macro to print a string*

PRINT MACRO string

    mov dx, offset string   *;storing the offset of the string in dx*

    mov ah, 09h *;interrup method to print a string*

    int 21h *;INTERRUPT*

PRINT ENDM

.code

    main PROC

        PRINT initial\_statement *;printing the initial statement : "the array before sorting: "*

        CALL PRINT\_ARRAY *;printing the elements of the array (before sorting)*

        CALL QuickSort

        end\_quicksort:  *;when the quicksort function has ended.*

            PRINT final\_statement

            CALL PRINT\_ARRAY    *;print the final sorted array*

        RET

    main ENDP

*;PROCEDURE to print the array elements:*

    PRINT\_ARRAY PROC

        mov cl, arr\_length

        print\_loop:

           mov bl, arr\_length  *;store the array length in bl*

           sub bx, cx  *;subtract the counter pointer from the bx register (array\_length - cx)*

           mov ah, 02h *;the interrup method to print a digit*

           mov dl, arr[bx]

           add dx, 30h *;adding 30h for the ASCII conversion*

           int 21h *;INTERRUPT*

        loop print\_loop

        RET

    PRINT\_ARRAY ENDP

*;quicksort algo:*

*;quickSort(arr[], low, high)*

*;{*

*;    if (low < high)*

*;    {*

*;        pivot = partition(arr, low, high);*

*;        quickSort(arr, low, pivot - 1);*

*;        quickSort(arr, pivot + 1, high);*

*;    }*

*;}*

    QuickSort PROC

        mov al, l

        cmp al, h   *;if l=>h then end the function*

            jge end\_quicksort

        mov al, l

        PUSH ax

        mov al, h

        PUSH ax

        CALL partition

        mov q, ax               *; store result in q*

*; pushing values to stack to keep the values stored when doing the recursive calls*

        inc ax                  *; do q+1*

        push ax                 *; push the value of q+1*

        push r                  *; push the value of r*

*; setting the parameters for the first call, arr, p and q-1*

        mov ax, q               *; get value of q*

        mov r, ax               *; set second parameter to q*

        dec r                   *; do q - 1*

        CALL QuickSort              *; p is already p here, and r is now set to q-1*

*;get previous values that were pushed, arr, q+1, r*

        pop r                   *; pop to r (we last pushed to r)*

        pop p                   *; pop to p = q+1*

        CALL QuickSort              *; p is q+1 here, and r is r*

        ret

    QuickSort ENDP

*;partition algo:*

*;partition (arr[], low, high)*

*;{*

*;    pivot = arr[high];*

*;    i = (low - 1) // Index of smaller element*

*;    for (j = low; j <= high- 1; j++)*

*;    {*

*;        if (arr[j] < pivot)*

*;        {*

*;            i++;    // increment index of smaller element*

*;            swap arr[i] and arr[j]*

*;        }*

*;    }*

*;    swap arr[i + 1] and arr[high])*

*;    return (i + 1)*

*;}*

    partition PROC

        mov si, OFFSET arr      *; load address of array*

    mov ax, r               *; get r*

*; since every int is 2 bytes, we need to move index\*2 times from start of array*

    SHL ax, 1               *; shift left will multiply ax by 2*

    add si, ax              *; add the result to start of array, we are at A[r] now*

*; copy A[r] to x*

    mov ax, [si]

    mov x, ax               *; x = A[r]*

*; copy p - 1 to i*

    mov ax, p

    mov i, ax               *; i = p*

    dec i                   *; i = i - 1*

*; copy P to j*

    mov ax, p

    mov j, ax               *; j = p*

    for\_loop:                       *; for j=p to r-1*

        mov si, OFFSET arr      *; get start of array*

        mov ax, j               *; move current value of j to ax*

        SHL ax, 1               *; again int is 2 bytes, move index\*2*

        add si, ax

        mov ax, [si]            *; move A[j] to ax*

*; if A[j] <= x*

        cmp ax, x

        JG bigger\_number               *; if x > A[j] no need to swap*

        inc i                   *; otherwise do i = i+1*

*; swap A[i] and A[j]*

        mov di, OFFSET arr      *; get start of array again*

*; values is at index\*2*

        mov cx, i

        SHL cx, 1

        add di, cx

        mov cx, [di]            *; do temp = A[i]*

        mov [di], ax            *; do A[i] = A[j]*

        mov [si], cx            *; do A[j] = temp*

    bigger\_number:

        inc j                   *; do j = j+1 for for loop*

*; check for for loop condition, if j < r loop again*

        mov ax, r

        cmp j, ax

        JL for\_loop

*; swap A[i+1] with A[r]*

        inc i                   *; do i=i+1*

        mov si, OFFSET arr      *; get start of array*

*; get A[i+1]*

        mov ax, i

        SHL ax, 1

        add si, ax

        mov ax, [si]            *; ax = A[i+1]*

*; get A[r]*

        mov di, OFFSET arr

        mov cx, r

        SHL cx, 1

        add di, cx

        mov cx, [di]            *; cx = A[r]*

*; swap A[i+1] and A[r]*

        mov [di], ax            *; A[r] = ax*

        mov [si], cx            *; A[i+1] = cx*

        mov ax, i               *; i is already i+1, set ax to return value*

        ret                     *; and return*

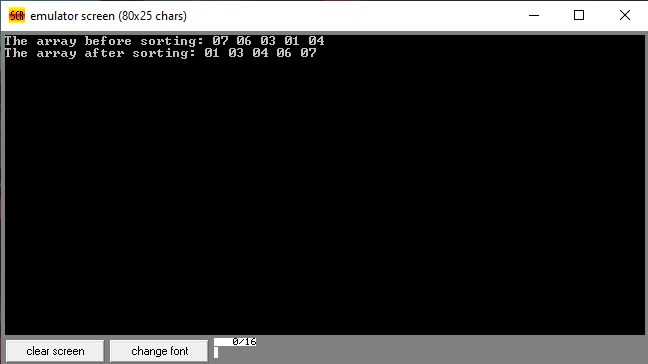
    ret

    partition ENDP

    end:

ret

**Output:**

****