SORTING ALGORITHMS LIBRARY (SAL)

Wahaj Javed Alam	**20K-0208**
Muhammad Hatif Mujahid	**20K-0218**
Agha Maarij Amir	**20K-0160**

Subject : Computer Organization and Assembly Language

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES - FAST

DECEMBER 2021

ABSTRACT:

In comparison to high-level languages, Assembly Language does not have a large number of libraries to assist with common tasks. This sorting algorithms library attempts to make programming easier for programmers by providing numerous sorting algorithms to assist in minimizing the time it takes to code and complete tasks quickly. As a result, the programmer can use the library to call the preset sorting algorithms and focus on the more important tasks.

INTRODUCTION:

The project was made in .masm architecture using the irvine32 library taught to us in our course this semester. Our project was to make a sorting library with the following sorting algorithms:

1. Heap sort
2. Merge sort
3. Insertion Sort
4. Selection sort
5. Gnome Sort
6. Shaker Sort
7. Bubble Sort
8. Cocktail Sort

PROBLEM DEFINITION:

Sorting is the most common issue that arises while dealing with data in any type of programming. Sorting is an important aspect of data management since it aids in a variety of applications, such as making searching easier. On the other hand, remembering all of the sorting algorithms is tedious.

METHODOLOGY/SOLUTION STATEMENT:

We emphasised how difficult it is to remember all of the sorting algorithms in the problem statement, and how the major goal of our project was to create a user-friendly approach to accomplish the sorting so that the user could go on to the other important tasks.

To address this issue, we created SAL, a sorting library consisting of all the above mentioned algorithms. The algorithms were designed with the help of Irvine32 library in visual studio 2019.

DETAILED DESIGN AND ARCHITECTURE:

The project is primarily made up of two levels. The user's input is taken on the project's front page. The user can enter any data type be it signed or unsigned, the sorting library will simply handle it.

The library consists mainly of two types of files, which the user must import into his working directory. The first is the **SourceCodes.asm** file, which contains all of the algorithms, and the second is **SAL.inc**, which contains all of the algorithms' headers.

In order to use the library, the user is required to include SAL.inc in their source code and simply call the function with the relevant register parameters as mentioned in the documentation provided below.

> **IMPORTANT NOTE:** If the user includes our library, he should not include **Irvine32.inc** in his source code since there will be a clash due to model redefinition because our library will automatically include Irvine for the user.

IMPLEMENTATION AND TESTING AND PROGRAMMING CODE:

After a brief trial and error period, we've opted to use register parameters to indicate the arguments that our algorithms will need to complete the task. The data will be sorted in no time if the appropriate arguments are provided and the relevant sort function is invoked. The algorithms are implemented in such a way that just after invoking the function, it will automatically decide the data type which is being passed and call the respective function accordingly. Regarding the implementation and testing purpose, the documentation for our code is provided below, along with pertinent examples for ease of comprehension.

Merge Sort:

Call args:	ESI contains the OFFSET of an array
	Eax contains TYPE of an array
	Ebx contains the starting point of array (0)
	Edx contains LengthOf array - 1
Return arg:	None

Example:

```
mov esi, OFFSET arr
mov eax, TYPE arr
mov ebx,0
mov edx, LENGTHOF arr
dec edx
call MergeSort
```

Heap Sort:

Call args: ESI contains the OFFSET of an array
Eax contains TYPE of an array
Edx contains LengthOf array

Return arg: None

Example:

```
mov esi, OFFSET arr
mov eax, TYPE arr
mov edx, LENGTHOF arr
call HeapSort
```

Bubble Sort:

Call args: ESI contains the OFFSET of an array
EBX contains the Length of an array
ECX contains Type of an array

Return arg: None

Example:

```
mov esi, OFFSET arr
mov ebx, LENGTHOF arr
mov ecx, TYPE arr
call BubbleSort
```

Selection Sort:

Call args: ESI contains the OFFSET of an array
Eax contains TYPE of an array
Ecx contains LengthOf array

Return arg: None

Example:

```
mov esi, OFFSET arr
mov eax, TYPE arr
mov ecx, LENGTHOF arr
call SelectionSort
```

Insertion Sort:

Call args: ESI contains the OFFSET of an array
Ebx contains TYPE of an array
Ecx contains LENGTH of an array

Return arg: None

Example:

```
mov esi, OFFSET arr
mov ebx, TYPE arr
Mov ecx, LENGTHOF arr
call Insertion_sort
```

Gnome Sort:

Call args: ESI contains the OFFSET of an array
Eax contains TYPE of an array
Ecx contains LENGTH of an array

Return arg: None

Example:

```
mov esi, OFFSET arr
mov ecx, LENGTHOF arr
Mov eax, TYPE arr
call Gnome_sort
```

Cocktail Sort:

Call args: ESI contains the OFFSET of an array
EBX contains Length of an array
ECX contains Type of array

Return arg: None

Example:

```
mov esi, OFFSET arr
mov ebx, LENGTHOF arr
mov ecx, TYPE arr
call CocktailSort
```

Shell Sort:

Call args: ESI contains the OFFSET of an array
EBX contains Length of an array
ECX contains Type of array

Return arg: None

Example:

```
mov esi, OFFSET arr
mov ebx, LENGTHOF arr
mov ecx, TYPE arr
call ShellSort
```

Programming Code:

The **SourceCodes.asm** contains the following source code:

```
TITLE SORTING ALGORITHMS LIBRARY
COMMENT !
	MADE BY:
		Wahaj Javed(20K-0208)
		Hatif Mujahid(20K-0218)
		Agha Maarij Amir(20K-0160)
!
Include SAL.inc
;==========================================================
;==========================================================
;====================SELECTION SORT ======================
;==========================================================
;==========================================================
```

```
.code
SelectionSort PROC
	cmp eax,4
	jz ISDWORD
	cmp eax,2
	jz ISWORD
	jmp ISBYTE
	ISDWORD:
		call DSelectionSort
		ret
	ISWORD:
		call WSelectionSort
		ret
	ISBYTE:
		call BSelectionSort
		ret
	ret
SelectionSort ENDP
;=========================DWORD=========================
;=======================================================
DSelectionSort PROC
	LOCAL i:DWORD,j:DWORD,min:DWORD,sizeOfArray:DWORD
	mov eax,0
	mov sizeOfArray,ecx
	mov i,eax
	mov j,eax
	mov min,eax
	mov ecx,sizeOfArray
	dec ecx
	outerLoop:
		mov ebx,i
		mov min,ebx
		push ecx
		mov edx,i
		mov j,edx
```

```
            innerLoop:
                inc j
                mov edx,j
                mov eax,[esi + edx * 4]
                mov edx,min
                mov ebx,[esi + edx * 4]
                cmp eax,ebx
                jl markNewMin
                jmp continueLoop
                markNewMin:
                    mov edx,j
                    mov min,edx
                continueLoop:
            loop innerLoop
            mov eax,i
            mov ebx,[esi + eax * 4]
            mov edx,min
            xchg ebx,[esi + edx * 4]
            mov [esi + eax * 4],ebx
            pop ecx
            inc i
        loop outerLoop
        ret
DSelectionSort ENDP
;=========================WORD=========================
;=============================================================

WSelectionSort PROC
        LOCAL i:DWORD,j:DWORD,min:DWORD,sizeOfArray:DWORD
        mov eax,0
        mov sizeOfArray,ecx
        mov i,eax
        mov j,eax
        mov min,eax
        mov ecx,sizeOfArray
        dec ecx
```

```
        outerLoop:
                mov ebx,i
                mov min,ebx
                push ecx
                mov edx,i
                mov j,edx
                innerLoop:
                        inc j
                        mov edx,j
                        mov ax,[esi + edx * 2]
                        mov edx,min
                        mov bx,[esi + edx * 2]
                        cmp ax,bx
                        jl markNewMin
                        jmp continueLoop
                        markNewMin:
                                mov edx,j
                                mov min,edx
                        continueLoop:
                loop innerLoop
                mov eax,i
                mov bx,[esi + eax * 2]
                mov edx,min
                xchg bx,[esi + edx * 2]
                mov [esi + eax * 2],bx
                pop ecx
                inc i
        loop outerLoop
        ret
WSelectionSort ENDP
;=========================BYTE=========================
;=====================================================

BSelectionSort PROC
        LOCAL i:DWORD,j:DWORD,min:DWORD,sizeOfArray:DWORD
        mov eax,0
```

```
        mov sizeOfArray,ecx
        mov i,eax
        mov j,eax
        mov min,eax
        mov ecx,sizeOfArray
        dec ecx
        outerLoop:
                mov ebx,i
                mov min,ebx
                push ecx
                mov edx,i
                mov j,edx
                innerLoop:
                        inc j
                        mov edx,j
                        mov al,[esi + edx]
                        mov edx,min
                        mov bl,[esi + edx]
                        cmp al,bl
                        jl markNewMin
                        jmp continueLoop
                        markNewMin:
                                mov edx,j
                                mov min,edx
                        continueLoop:
                loop innerLoop
                mov eax,i
                mov bl,[esi + eax]
                mov edx,min
                xchg bl,[esi + edx]
                mov [esi + eax],bl
                pop ecx
                inc i
        loop outerLoop
        ret
BSelectionSort ENDP
```

```
;==========================================================
;==========================================================
;=======================HEAP SORT =========================
;==========================================================
;==========================================================
HeapSort PROC
        push eax
        push edx
        mov eax,edx
        mov edx,0
        mov ebx,2
        div ebx
        cmp edx,0
        jnz isOdd
        jmp isEven
        isOdd:
                pop edx
                dec edx
                jmp nextComparison
        isEven:
                pop edx
        nextComparison:
                pop eax
        cmp eax,4
        jz ISDWORD
        cmp eax,2
        jz ISWORD
        jmp ISBYTE
        ISDWORD:

                call DHeapSort
                ret
        ISWORD:
                call WHeapSort
                ret
        ISBYTE:
```

```
                call BHeapSort
                ret
        ret
HeapSort ENDP

;=========================DWORD=========================
;=======================================================
DHeapSort PROC
        local i:DWORD,n:DWORD,temp:DWORD
        mov n,edx
        shr edx,1
        dec edx
        mov i,edx
        mov ecx,0
        buildHeap:
                mov ebx,i
                mov edx,n
                call DHeapify
                mov eax,0
                dec eax
                dec i
                cmp eax,i
                jz goOut
        loop buildHeap
        goOut:
        mov eax,n
        mov i,eax
        dec i
        mov ecx,i
        ExtractAndSort:
                mov eax,[esi]
                mov ebx,i
                xchg eax,[esi+ebx*4]
                mov [esi],eax
                mov ebx,0
                mov edx,i
```

```
                mov temp,ecx
                call DHeapify
                mov ecx,temp
                dec i
        loop ExtractAndSort
        ret
DHeapSort ENDP

; ESI = OFFSET of Array
; EDX = END
; EBX = START
DHeapify PROC
        LOCAL
largest:DWORD,left:DWORD,right:DWORD,n:DWORD,i:DWORD
        mov i,ebx
        mov n,edx
        mov largest,ebx
        shl ebx,1
        mov left,ebx
        mov right,ebx
        inc left
        add right,2
        mov eax,left
        cmp eax,n
        jl checkSecondConditionForLargestLeft
        jmp checkSecondIfCondition
        checkSecondConditionForLargestLeft:
                mov ebx,left
                mov ecx,largest
                mov eax,[esi + ebx*4]
                cmp eax,[esi + ecx*4]
                jg largestLeftExists
        jmp checkSecondIfCondition
        largestLeftExists:
                mov largest,ebx
        checkSecondIfCondition:
```

```
		mov eax,right
		cmp eax,n
		jl checkSecondConditionForLargestRight
		jmp checkThirdCondition
		checkSecondConditionForLargestRight:
			mov ebx,right
			mov ecx,largest
			mov eax,[esi + ebx * 4]
			cmp eax,[esi + ecx * 4]
			jg largestRightExists
		jmp checkThirdCondition
		largestRightExists:
			mov largest,ebx
		checkThirdCondition:
			mov eax,i
			cmp eax,largest
			jnz RecurseHeapify
			jmp returnFromFunction
			RecurseHeapify:
				mov eax,i
				mov ebx,largest
				mov ecx,[esi + eax*4]
				xchg ecx,[esi + ebx*4]
				mov [esi + eax*4],ecx
				mov edx,n
				mov ebx,largest
				call DHeapify
		returnFromFunction:
	ret
DHeapify ENDP
;=========================WORD=========================
;==============================================================
WHeapSort PROC
	local i:DWORD,n:DWORD,temp:DWORD
	mov n,edx
	shr edx,1
```

```
        dec edx
        mov i,edx
        mov ecx,0
        buildHeap:
                mov ebx,i
                mov edx,n
                call WHeapify
                mov eax,0
                dec eax
                dec i
                cmp eax,i
                jz goOut
        loop buildHeap
        goOut:
        mov eax,n
        mov i,eax
        dec i
        mov ecx,i
        ExtractAndSort:
                mov ax,[esi]
                mov ebx,i
                xchg ax,[esi + ebx * 2]
                mov [esi],ax
                mov ebx,0
                mov edx,i
                mov temp,ecx
                call WHeapify
                mov ecx,temp
                dec i
        loop ExtractAndSort
        ret
WHeapSort ENDP
WHeapify PROC
        LOCAL
largest:DWORD,left:DWORD,right:DWORD,n:DWORD,i:DWORD
        mov i,ebx
```

```
mov n,edx
mov largest,ebx
shl ebx,1
mov left,ebx
mov right,ebx
inc left
add right,2
mov eax,left
cmp eax,n
jl checkSecondConditionForLargestLeft
jmp checkSecondIfCondition
checkSecondConditionForLargestLeft:
        mov ebx,left
        mov ecx,largest
        mov ax,[esi + ebx * 2]
        cmp ax,[esi + ecx * 2]
        jg largestLeftExists
jmp checkSecondIfCondition
largestLeftExists:
        mov largest,ebx
checkSecondIfCondition:
        mov eax,right
        cmp eax,n
        jl checkSecondConditionForLargestRight
        jmp checkThirdCondition
        checkSecondConditionForLargestRight:
                mov ebx,right
                mov ecx,largest
                mov ax,[esi + ebx * 2]
                cmp ax,[esi + ecx * 2]
                jg largestRightExists
        jmp checkThirdCondition
        largestRightExists:
                mov largest,ebx
        checkThirdCondition:
                mov eax,i
```

```
                cmp eax,largest
                jnz RecurseHeapify
                jmp returnFromFunction
                RecurseHeapify:
                        mov eax,i
                        mov ebx,largest
                        mov cx,[esi + eax * 2]
                        xchg cx,[esi + ebx * 2]
                        mov [esi + eax * 2],cx
                        mov edx,n
                        mov ebx,largest
                        call WHeapify
        returnFromFunction:
    ret
WHeapify ENDP
;=========================BYTE=========================
;=============================================================
BHeapify PROC
    LOCAL
largest:DWORD,left:DWORD,right:DWORD,n:DWORD,i:DWORD
    mov i,ebx
    mov n,edx
    mov largest,ebx
    shl ebx,1
    mov left,ebx
    mov right,ebx
    inc left
    add right,2
    mov eax,left
    cmp eax,n
    jl checkSecondConditionForLargestLeft
    jmp checkSecondIfCondition
    checkSecondConditionForLargestLeft:
        mov ebx,left
        mov ecx,largest
        mov al,[esi + ebx]
```

```
            cmp al,[esi + ecx]
            jg largestLeftExists
        jmp checkSecondIfCondition
        largestLeftExists:
            mov largest,ebx
        checkSecondIfCondition:
            mov eax,right
            cmp eax,n
            jl checkSecondConditionForLargestRight
            jmp checkThirdCondition
            checkSecondConditionForLargestRight:
                mov ebx,right
                mov ecx,largest
                mov al,[esi + ebx]
                cmp al,[esi + ecx]
                jg largestRightExists
            jmp checkThirdCondition
            largestRightExists:
                mov largest,ebx
            checkThirdCondition:
                mov eax,i
                cmp eax,largest
                jnz RecurseHeapify
                jmp returnFromFunction
                RecurseHeapify:
                    mov eax,i
                    mov ebx,largest
                    mov cl,[esi + eax]
                    xchg cl,[esi + ebx]
                    mov [esi + eax],cl
                    mov edx,n
                    mov ebx,largest
                    call BHeapify
            returnFromFunction:
        ret
BHeapify ENDP
```

```
BHeapSort PROC
        local i:DWORD,n:DWORD,temp:DWORD
        mov n,edx
        shr edx,1
        dec edx
        mov i,edx
        mov ecx,0
        buildHeap:
                mov ebx,i
                mov edx,n
                call BHeapify
                mov eax,0
                dec eax
                dec i
                cmp eax,i
                jz goOut
        loop buildHeap
        goOut:
        mov eax,n
        mov i,eax
        dec i
        mov ecx,i
        ExtractAndSort:
                mov al,[esi]
                mov ebx,i
                xchg al,[esi + ebx]
                mov [esi],al
                mov ebx,0
                mov edx,i
                mov temp,ecx
                call BHeapify
                mov ecx,temp
                dec i
        loop ExtractAndSort
        ret
BHeapSort ENDP
```

```
;==========================================================
;==========================================================
;=======================MERGE SORT=========================
;==========================================================
;==========================================================
MergeSort PROC
	LOCAL
mid:DWORD,lower:DWORD,higher:DWORD,typeArr:DWORD
	mov typeArr,eax
	mov lower,ebx
	mov higher,edx
	cmp ebx,edx
	jl recurse
	jmp endProgram
	recurse:
		mov eax,lower
		add eax,higher
		shr eax,1
		mov mid,eax
		mov ebx,lower
		mov edx,mid
		mov eax,typeArr
		call MergeSort
		mov ebx,mid
		inc ebx
		mov edx,higher
		mov eax,typeArr
		call MergeSort
		mov ebx,lower
		mov edx,higher
		mov eax,typeArr
		cmp eax,4
		jz isDWORD
		cmp eax,2
```

```
                jz isWORD
                jmp isBYTE
                isDWORD:
                        mov eax,mid
                        call DMerge
                        jmp endProgram
                isWORD:
                        mov eax,mid
                        call WMerge
                        jmp endProgram
                isBYTE:
                        mov eax,mid
                        call BMerge
        endProgram:
                ret
MergeSort ENDP

;=========================DWORD=========================
;=======================================================
DMerge PROC
        LOCAL
i:DWORD,j:DWORD,k:DWORD,lower:DWORD,mid:DWORD,higher:DWO
RD,a[100]:SDWORD
        mov i,ebx
        mov k,ebx
        mov j,eax
        inc j
        mov mid,eax
        mov higher,edx
        mov lower,ebx
        mov ecx,0
        startLoop:
                mov edx,mid
                cmp i,edx
                jg outsideFirst
                mov edx,higher
```

```
        cmp j,edx
        jg outsideFirst
        mov ebx,i
        mov eax,[esi + ebx * 4]
        mov ebx,j
        cmp eax,[esi + ebx * 4]
        jl firstConditionTrue
        jmp secondConditionTrue
        firstConditionTrue:
                mov ebx,i
                mov edx,k
                mov eax,[esi + ebx * 4]
                mov [a + edx * 4],eax
                inc k
                inc i
                jmp continueLoop
        secondConditionTrue:
                mov ebx,j
                mov edx,k
                mov eax,[esi + ebx * 4]
                mov [a + edx * 4],eax
                inc k
                inc j
        continueLoop:
loop startLoop
outsideFirst:
mov ecx,0
secondLoop:
        mov eax,mid
        cmp i,eax
        jg outsideSecond
        mov ebx,i
        mov edx,k
        mov eax,[esi + ebx * 4]
        mov [a + edx * 4],eax
        inc k
```

```
            inc i
        loop secondLoop
        outsideSecond:
        thirdLoop:
            mov eax,higher
            cmp j,eax
            jg outsideThird
            mov ebx,j
            mov edx,k
            mov eax,[esi + ebx * 4]
            mov [a + edx * 4],eax
            inc k
            inc j
        loop thirdLoop
        outsideThird:
        mov eax,lower
        mov i,eax
        mov ecx,0
        finalLoop:
            mov eax,k
            cmp i,eax
            jge outsideFinal
            mov ebx,i
            mov eax,[a + ebx * 4]
            mov [esi + ebx * 4],eax
            inc i
        loop finalLoop
        outsideFinal:
        ret
DMerge ENDP
;=========================WORD=========================
;=============================================================
WMerge PROC
        LOCAL
i:DWORD,j:DWORD,k:DWORD,lower:DWORD,mid:DWORD,higher:DWO
RD,a[100]:SWORD
```

```
mov i,ebx
mov k,ebx
mov j,eax
inc j
mov mid,eax
mov higher,edx
mov lower,ebx
mov ecx,0
startLoop:
        mov edx,mid
        cmp i,edx
        jg outsideFirst
        mov edx,higher
        cmp j,edx
        jg outsideFirst
        mov ebx,i
        mov ax,[esi + ebx * 2]
        mov ebx,j
        cmp ax,[esi + ebx * 2]
        jl firstConditionTrue
        jmp secondConditionTrue
        firstConditionTrue:
                mov ebx,i
                mov edx,k
                mov ax,[esi + ebx * 2]
                mov [a + edx * 2],ax
                inc k
                inc i
                jmp continueLoop
        secondConditionTrue:
                mov ebx,j
                mov edx,k
                mov ax,[esi + ebx * 2]
                mov [a + edx * 2],ax
                inc k
                inc j
```

```
        continueLoop:
loop startLoop
outsideFirst:
mov ecx,0
secondLoop:
        mov eax,mid
        cmp i,eax
        jg outsideSecond
        mov ebx,i
        mov edx,k
        mov ax,[esi + ebx * 2]
        mov [a + edx * 2],ax
        inc k
        inc i
loop secondLoop
outsideSecond:
thirdLoop:
        mov eax,higher
        cmp j,eax
        jg outsideThird
        mov ebx,j
        mov edx,k
        mov ax,[esi + ebx * 2]
        mov [a + edx * 2],ax
        inc k
        inc j
loop thirdLoop
outsideThird:
mov eax,lower
mov i,eax
mov ecx,0
finalLoop:
        mov eax,k
        cmp i,eax
        jge outsideFinal
        mov ebx,i
```

```
                mov ax,[a + ebx * 2]
                mov [esi + ebx * 2],ax
                inc i
        loop finalLoop
        outsideFinal:
        ret
WMerge ENDP
;=========================BYTE=========================
;=======================================================
BMerge PROC
        LOCAL
i:DWORD,j:DWORD,k:DWORD,lower:DWORD,mid:DWORD,higher:DWO
RD,a[100]:SBYTE
        mov i,ebx
        mov k,ebx
        mov j,eax
        inc j
        mov mid,eax
        mov higher,edx
        mov lower,ebx
        mov ecx,0
        startLoop:
                mov edx,mid
                cmp i,edx
                jg outsideFirst
                mov edx,higher
                cmp j,edx
                jg outsideFirst
                mov ebx,i
                mov al,[esi + ebx]
                mov ebx,j
                cmp al,[esi + ebx]
                jl firstConditionTrue
                jmp secondConditionTrue
                firstConditionTrue:
                        mov ebx,i
```

```
                mov edx,k
                mov al,[esi + ebx]
                mov [a + edx],al
                inc k
                inc i
                jmp continueLoop
        secondConditionTrue:
                mov ebx,j
                mov edx,k
                mov al,[esi + ebx]
                mov [a + edx],al
                inc k
                inc j
        continueLoop:
loop startLoop
outsideFirst:
mov ecx,0
secondLoop:
        mov eax,mid
        cmp i,eax
        jg outsideSecond
        mov ebx,i
        mov edx,k
        mov al,[esi + ebx]
        mov [a + edx],al
        inc k
        inc i
loop secondLoop
outsideSecond:
thirdLoop:
        mov eax,higher
        cmp j,eax
        jg outsideThird
        mov ebx,j
        mov edx,k
        mov al,[esi + ebx]
```

```
                mov [a + edx],al
                inc k
                inc j
        loop thirdLoop
        outsideThird:
        mov eax,lower
        mov i,eax
        mov ecx,0
        finalLoop:
                mov eax,k
                cmp i,eax
                jge outsideFinal
                mov ebx,i
                mov al,[a + ebx]
                mov [esi + ebx],al
                inc i
        loop finalLoop
        outsideFinal:
        ret
BMerge ENDP
;==============================================================
;==============================================================
;=======================GNOME SORT =======================
;==============================================================
;==============================================================


Gnome_sort PROC
        cmp eax,4
        jz ISDWORD
        cmp eax,2
        jz ISWORD
        jmp ISBYTE
        ISDWORD:
                call DGnome_sort
                ret
```

```
        ISWORD:
                call WGnome_sort
                ret
        ISBYTE:
                call BGnome_sort
                ret
        ret
Gnome_sort ENDP

;=========================BYTE=========================
;=======================================================


BGnome_sort PROC USES eax ebx edx
   LOCAL index:BYTE, n:BYTE
   mov eax, 0
   mov index, al
   mov n, cl
   mov ecx, 0
   l1:
      push ecx
      mov ecx, 0
      mov al, n
      cmp index, al
      JGE l5
      mov eax, 0
      cmp index, al
      JNE l2
         mov bl, index
         inc bl
         mov index, bl
      l2:
      mov cl, index
      mov al, [esi+ecx]
      dec ecx
      cmp al, [esi+ecx]
```

```
        JL l3
            mov bl, index
            inc bl
            mov index, bl
            jmp l4
        l3:
            mov cl, index
            dec ecx
            mov dl, [esi+ecx]
            mov [esi+ecx], al
            inc ecx
            mov [esi+ecx], dl
            mov cl, index
            dec cl
            mov index, cl
        l4:
        pop ecx
    loop l1
    l5:
    ret
BGnome_sort ENDP

;=========================WORD=========================
;======================================================
WGnome_sort PROC USES eax ebx edx
    LOCAL index:DWORD, n:DWORD
    mov eax, 0
    mov index, eax
    mov n, ecx
    mov ecx, 0
    l1:
        push ecx
        mov ecx, 0
        mov eax, n
        cmp index, eax
        JGE l5
```

```
        mov eax, 0
        cmp index, eax
        JNE l2
            mov ebx, index
            inc ebx
            mov index, ebx
        l2:
        mov ecx, index
        mov ax, [esi+2*ecx]
        dec ecx
        cmp ax, [esi+2*ecx]
        JL l3
            mov ebx, index
            inc ebx
            mov index, ebx
            jmp l4
        l3:
            mov ecx, index
            dec ecx
            mov dx, [esi+2*ecx]
            mov [esi+2*ecx], ax
            inc ecx
            mov [esi+2*ecx], dx
            mov ecx, index
            dec ecx
            mov index, ecx
        l4:
        pop ecx
    loop l1
    l5:
          ret
WGnome_sort ENDP
;=========================DWORD==========================
;=========================================================
DGnome_sort PROC USES eax ebx edx
    LOCAL index:DWORD, n:DWORD
```

```
mov eax, 0
mov index, eax
mov n, ecx
mov ecx, 0
l1:
    push ecx
    mov eax, n
    cmp index, eax
    JGE l5
    mov eax, 0
    cmp index, eax
    JNE l2
        mov ebx, index
        inc ebx
        mov index, ebx
    l2:
    mov ecx, index
    mov eax, [esi+4*ecx]
    dec ecx
    cmp eax, [esi+4*ecx]
    JL l3
        mov ebx, index
        inc ebx
        mov index, ebx
        jmp l4
    l3:
        mov ecx, index
        dec ecx
        mov edx, [esi+4*ecx]
        mov [esi+4*ecx], eax
        inc ecx
        mov [esi+4*ecx], edx
        mov ecx, index
        dec ecx
        mov index, ecx
    l4:
```

```
        pop ecx
    loop l1
    l5:

    ret
DGnome_sort ENDP



;==========================================================
;==========================================================
;=====================INSERTION SORT =====================
;==========================================================
;==========================================================

Insertion_sort PROC
        cmp ebx, 4
        JE D
        cmp ebx, 2
        JE W
        cmp ebx, 1
        JE B
        jmp l1
        D:
        call DInsertion_sort
        jmp l1
        W:
        call WInsertion_sort
        jmp l1
        B:
        call BInsertion_sort
        l1:
        ret
Insertion_sort ENDP
```

```
;=========================BYTE==========================
;=======================================================
BInsertion_sort PROC USES eax ecx edx
	LOCAL x:BYTE, i:DWORD, j:DWORD, lengthArr:DWORD
	mov eax, 0
	mov ebx, 0
	mov edx, 0
	mov eax, 1
	mov i, eax
	mov eax, 0
	mov j, eax
	mov lengthArr,ecx
	mov ecx,lengthArr
	dec ecx
	D1:
		mov edx, i
		mov ax, [esi+edx]
		mov x, al
		push ecx
		mov ecx, 0
		mov eax, i
		mov j, eax
		dec j
		mov ebx, 0
		D2:
			mov ebx, 0
			mov edx, j
			cmp j, ebx
			JL D3
			mov bl, [esi+edx]
			cmp bl, x
			JLE D3
			inc edx
			mov [esi+edx], bl
			dec j
		loop D2
```

```
                D3:
                pop ecx
                mov al, x
                mov edx, j
                inc edx
                mov [esi+edx], al
                inc i
        loop D1
        return:
        ret
BInsertion_sort ENDP

;=========================WORD=========================
;=======================================================
WInsertion_sort PROC USES eax ecx edx
        LOCAL x:WORD, i:WORD, j:WORD, lengthArr:DWORD
        mov eax, 0
        mov ebx, 0
        mov edx, 0
        mov eax, 1
        mov i, ax
        mov eax, 0
        mov j, ax
        mov lengthArr,ecx
        mov ecx,lengthArr
        dec ecx
        D1:
                mov dx, i
                mov ax, [esi+edx*2]
                mov x, ax
                push ecx
                mov ecx, 0
                mov ax, i
                mov j, ax
                dec j
                mov ebx, 0
```

```
                D2:
                        mov ebx, 0
                        mov dx, j
                        cmp j, bx
                        JL D3
                        mov bx, [esi+edx*2]
                        cmp bx, x
                        JLE D3
                        inc edx
                        mov [esi+edx*2], bx
                        dec j
                loop D2
                D3:
                pop ecx
                mov ax, x
                mov dx, j
                inc dx
                mov [esi+edx*2], ax
                inc i
        loop D1
        return:
        ret
WInsertion_sort ENDP


;=========================DWORD==========================
;========================================================
DInsertion_sort PROC USES eax ecx edx
        LOCAL x:DWORD, i:DWORD, j:DWORD,lengthArr:DWORD
        mov eax, 1
        mov i, eax
        mov eax, 0
        mov j, eax
        mov lengthArr,ecx
        mov ecx, lengthArr
        dec ecx
```

```
        D1:
                push ecx
                mov ecx, ebx
                mov edx, i
                mov eax, [esi+edx*4]
                mov x, eax
                push ecx
                mov ecx, 0
                mov eax, i
                mov j, eax
                dec j
                mov ebx, 0
                D2:
                        mov ebx, 0
                        mov edx, j
                        cmp j, ebx
                        JL D3
                        mov ebx, [esi+edx*TYPE esi]
                        cmp ebx, x
                        JLE D3
                        inc edx
                        mov [esi+edx*TYPE esi], ebx
                        dec j
                loop D2
                D3:
                pop ecx
                mov eax, x
                mov edx, j
                inc edx
                mov [esi+edx*TYPE esi], eax
                inc i
                pop ecx
        loop D1
        return:
        ret
DInsertion_sort ENDP
```

```
;==========================================================
;==========================================================
;=======================BUBBLE SORT =======================
;==========================================================
;==========================================================

BubbleSort PROC

cmp ecx,4
jz ISDWORD

cmp ecx,2
jz ISWORD

cmp ecx,1
jz ISBYTE

ret

ISDWORD:
call DBubbleSort
ret

ISWORD:
call WBubbleSort
ret

ISBYTE:
call BBubbleSort
ret

BubbleSort endp
```

```
;=========================DWORD==========================
;=========================================================
DBubbleSort PROC

local i:DWORD
local OFFSETHOLDER:DWORD
local len:DWORD
local temp:DWORD

mov OFFSETHOLDER,esi
mov edi,esi
mov i,0

mov len,ebx
mov ecx,ebx
dec ecx
mov ebx,0

OUTERLOOP :

        mov temp,ecx
        mov ecx,len
        dec ecx
        sub ecx,i

INNERLOOP:
        add edi,4
        mov eax,[edi]
        cmp [esi],eax
        jng NOSWAP

        mov ebx,[esi]
        mov [esi],eax
        mov [edi],ebx

        NOSWAP:
```

```
        add esi,4

loop INNERLOOP

        mov esi,OFFSETHOLDER
        mov edi,OFFSETHOLDER
        inc i
        mov ecx,temp

loop OUTERLOOP


ret

DBubbleSort endp
;=========================WORD=========================
;=====================================================
WBubbleSort PROC

local i:WORD
local OFFSETHOLDER:DWORD
local len:WORD
local temp:WORD

mov OFFSETHOLDER,esi
mov edi,esi
mov i,0

mov len,bx
mov ecx,ebx
dec ecx
mov ebx,0

OUTERLOOP :

        mov temp,cx
```

```
        mov cx,len
        dec cx
        sub cx,i

INNERLOOP:
        add edi,2
        mov ax,[edi]
        cmp [esi],ax
        jng NOSWAP

        mov bx,[esi]
        mov [esi],ax
        mov [edi],bx

        NOSWAP:
        add esi,2

loop INNERLOOP

        mov esi,OFFSETHOLDER
        mov edi,OFFSETHOLDER
        inc i
        mov cx,temp

loop OUTERLOOP


ret

WBubbleSort endp

;=========================BYTE=========================
;=============================================================
BBubbleSort PROC

local i:DWORD
```

```
local OFFSETHOLDER:DWORD
local len:DWORD
local temp:DWORD

mov OFFSETHOLDER,esi
mov edi,esi
mov i,0

mov ecx,0

mov len,ebx
mov ecx,ebx
dec ecx
mov ebx,0

OUTERLOOP :

        mov temp,ecx
        mov ecx,len
        dec ecx
        sub ecx,i

INNERLOOP:
        add edi,1
        mov al,[edi]
        cmp [esi],al
        jng NOSWAP

        mov bl,[esi]
        mov [esi],al
        mov [edi],bl

        NOSWAP:
        add esi,1

loop INNERLOOP
```

```
        mov esi,OFFSETHOLDER
        mov edi,OFFSETHOLDER
        inc i
        mov ecx,temp

loop OUTERLOOP


ret

BBubbleSort endp


;==============================================================
;==============================================================
;=====================COCKTAIL SORT =====================
;==============================================================
;==============================================================

CocktailSort PROC

cmp ecx,4
jz ISDWORD

cmp ecx,2
jz ISWORD

cmp ecx,1
jz ISBYTE

ret

ISDWORD:
call DCocktailSort
ret
```

```
ISWORD:
call WCocktailSort
ret

ISBYTE:
call BCocktailSort
ret


CocktailSort endp


;=========================WORD=========================
;=======================================================
WCocktailSort PROC

local start :WORD
local hold :SWORD
local i:WORD

mov eax,0
mov ecx,0
mov edx,0


mov cx,bx
dec cx


mov bx,1

WWHILELOOP:

        cmp bx,1
        jne OUTOFWWHILE
```

```
mov bx,0

mov ax,start



FORLOOPW1:

        cmp ax,cx
        jge OUTOFLOOPW1

        mov i,ax
        inc ax
        mov dx,[esi + eax * 2]
        mov ax,i
        cmp dx,[esi + eax * 2]
        jge NOWSWAP

        mov bx,[esi + eax * 2]
        mov [esi + eax * 2],dx
        inc ax
        mov [esi + eax * 2],bx
        mov bx,1

        NOWSWAP:
        mov ax,i
        inc ax

jmp FORLOOPW1


OUTOFLOOPW1:

        cmp bx,0
        je OUTOFWWHILE
```

```
        mov bx,0
        dec cx


        mov ax,cx
        dec ax
        mov hold,cx
        mov cx,ax



FORLOOPW2:

        cmp ax,start
        jnge OUTOFLOOPW2



        mov i,ax
        inc ax
        mov dx,[esi + eax * 2]
        mov ax,i
        cmp dx,[esi + eax * 2]

        jge NOSWAPPINGW


        mov bx,[esi + eax * 2]
        mov [esi + eax * 2],dx
        inc ax
        mov [esi + eax * 2],bx
        mov bx,1

NOSWAPPINGW:
```

```
            mov ax,i
                dec ax

        jmp FORLOOPW2

 OUTOFLOOPW2:

        mov cx,hold
        inc start
        mov ax,start

jmp WWHILELOOP

OUTOFWWHILE:


        ret

WCocktailSort endp

;=========================DWORD=========================
;=================================================================
DCocktailSort PROC

local start :DWORD
local hold :DWORD
local i:DWORD




mov ecx,ebx
dec ecx


mov ebx,1
```

```
WHILELOOP:

        cmp ebx,1
        jne OUTOFWHILE

        mov ebx,0

        mov eax,start



        FORLOOP1:

                cmp eax,ecx
                jge OUTOFLOOP1

                mov i,eax
                inc eax
                mov edx,[esi + eax * 4]
                mov eax,i
                cmp edx,[esi + eax * 4]
                jge NOSWAP

                mov ebx,[esi + eax * 4]
                mov [esi + eax * 4],edx
                inc eax
                mov [esi + eax * 4],ebx
                mov ebx,1

                NOSWAP:
                mov eax,i
                inc eax

        jmp FORLOOP1
```

```
OUTOFLOOP1:

        cmp ebx,0
        je OUTOFWHILE

        mov ebx,0
        dec ecx


        mov eax,ecx
        dec eax
        mov hold,ecx
        mov ecx,eax



FORLOOP2:

        cmp eax,start
        jnge OUTOFLOOP2



        mov i,eax
        inc eax
        mov edx,[esi + eax * 4]
        mov eax,i
        cmp edx,[esi + eax * 4]

        jge NOSWAPPING


        mov ebx,[esi + eax * 4]
        mov [esi + eax * 4],edx
```

```
                inc eax
                mov [esi + eax * 4],ebx
                mov ebx,1

        NOSWAPPING:
           mov eax,i
                dec eax

        jmp FORLOOP2

 OUTOFLOOP2:

        mov ecx,hold
        inc start
        mov eax,start

jmp WHILELOOP

OUTOFWHILE:


        ret

DCocktailSort endp



;=========================BYTE=========================
;=====================================================
BCocktailSort PROC

local start :BYTE
local hold :SBYTE
local i:BYTE

mov eax,0
```

```
mov ecx,0
mov edx,0


mov ecx,ebx
dec cl


mov bl,1

BWHILELOOP:

        cmp bl,1
        jne OUTOFBWHILE

        mov bl,0

        mov al,start



        FORLOOPB1:

                cmp al,cl
                jge OUTOFLOOPB1

                mov i,al
                inc al
                mov dl,[esi + eax ]
                mov al,i
                cmp dl,[esi + eax ]

                jge NOBSWAP


                mov bl,[esi + eax]
```

```
        mov [esi + eax ],dl
        inc al
        mov [esi + eax ],bl
        mov bl,1

        NOBSWAP:
        mov al,i
        inc al

jmp FORLOOPB1


OUTOFLOOPB1:

        cmp bl,0
        je OUTOFBWHILE

        mov bl,0
        dec cl


        mov al,cl
        dec al
        mov hold,cl
        mov cl,al




FORLOOPB2:

        cmp al,start
        jnge OUTOFLOOPB2
```

```
                mov i,al
                inc al
                mov dl,[esi + eax]
                mov al,i
                cmp dl,[esi + eax]

                jge NOSWAPPINGB


                mov bl,[esi + eax ]
                mov [esi + eax],dl
                inc al
                mov [esi + eax ],bl
                mov bl,1

        NOSWAPPINGB:
           mov al,i
                dec al

        jmp FORLOOPB2

 OUTOFLOOPB2:

        mov cl,hold
        inc start
        mov al,start

jmp BWHILELOOP

OUTOFBWHILE:


        ret

BCocktailSort endp
```

```
;=============================================================
;=============================================================
;=======================SHELL SORT ======================
;=============================================================
;=============================================================

ShellSort PROC

enter 0,0
cmp ecx,4
jz ISDWORD

cmp ecx,2
jz ISWORD

cmp ecx,1
jz ISBYTE


ISDWORD:
call DShellSort
leave
ret

ISWORD:

call WShellSort
leave
ret

ISBYTE:
call BShellSort
leave
ret
```

```
ShellSort endp

;=========================DWORD=========================
;=======================================================
DShellSort PROC

local gap:DWORD
local len:DWORD
local temp:DWORD




mov edx,0
mov eax,ebx
mov len,ebx
mov ebx,2

div ebx
mov gap,eax
mov eax,0
mov ecx,0
mov edi,esi

GAPLOOP:

        mov eax,gap

        INNERLOOP:

                mov ebx,[esi + eax * 4]
                mov temp,ebx
                mov edx,eax

                FORLOOP3:
```

```
                cmp edx,gap
                jnae OUTOFFORLOOP3

                mov ebx,edx
                sub ebx,gap
                mov ebx,[esi + ebx * 4]
                cmp ebx,temp
                jl OUTOFFORLOOP3

                mov [edi + edx * 4],ebx
                sub edx,gap

            loop FORLOOP3

            OUTOFFORLOOP3:

                mov ebx,temp
                mov [esi + edx * 4],ebx
                inc eax
                cmp eax,len

        jl INNERLOOP

        mov edx,0
        mov eax,gap
        mov ebx,2
        div ebx
        mov gap,eax
        cmp gap,0

        je FINISH

jne GAPLOOP

FINISH:
```

```
ret
DShellSort endp


;=========================WORD=========================
;=======================================================
WShellSort PROC

local gap:WORD
local len:WORD
local temp:SWORD




mov ax,bx
mov dx,0
mov len,ax
mov ebx,0
mov edx,0
mov bx,2

div bx
mov gap,ax
mov eax,0
mov ecx,0

mov edi,esi

GAPLOOP2:

        mov ax, gap

        INNERLOOP2:
```

```
        mov bx,[esi + eax * 2]
        mov temp,bx
        mov dx,ax

        FORLOOP32:

                cmp dx, gap
                jnae OUTOFFORLOOP32

                mov bx,dx
                sub bx, gap
                mov bx,[esi + ebx * 2]
                cmp bx, temp
                jl OUTOFFORLOOP32

                mov [edi + edx * 2],bx
                sub dx, gap

        loop FORLOOP32

        OUTOFFORLOOP32:

                mov bx, temp
                mov [esi + edx * 2],bx
                inc ax
                cmp ax, len

jl INNERLOOP2

mov dx,0
mov ax, gap
mov bx,2
div bx
mov gap,ax
cmp gap,0
```

```
        je FINISH2

jne GAPLOOP2

FINISH2:


ret

WShellSort endp

;=========================BYTE=========================
;=====================================================
BShellSort PROC

local gap:DWORD
local len:DWORD
local temp:SBYTE




mov eax,ebx

mov edx,0
mov len,eax
mov ebx,0
mov ebx,2

div ebx
mov gap,eax
mov eax,0
mov ecx,0

mov edi,esi
```

```
GAPLOOP3:

        mov eax, gap

        INNERLOOP3:

                mov bl,[esi + eax ]
                mov  temp,bl
                mov dl,al

                FORLOOP33:

                        cmp dl,BYTE PTR gap
                        jnae OUTOFFORLOOP33

                        mov bl,dl
                        sub bl, BYTE PTR gap
                        mov bl,[esi + ebx]
                        cmp bl, temp
                        jl OUTOFFORLOOP33

                        mov [edi + edx ],bl
                        sub dl,BYTE PTR gap
                loop FORLOOP33
                OUTOFFORLOOP33:
                        mov bl, temp
                        mov [esi + edx],bl
                        inc al
                        cmp al,BYTE PTR len
        jl INNERLOOP3
        mov dl,0
        mov eax, gap
        mov ebx,2
        div ebx
        mov  gap,eax
        cmp gap,0
```

```
        je FINISH3
jne GAPLOOP3
FINISH3:
ret
BShellSort endp
END
;NICE
```

Secondly, the **SAL.inc** contains the following headers:

```
Include Irvine32.inc

SelectionSort PROTO

HeapSort PROTO

MergeSort PROTO

Gnome_sort PROTO

Insertion_sort PROTO

BubbleSort PROTO

CocktailSort PROTO

ShellSort PROTO
```

RESULTS SOFTWARE SIMULATION AND DISCUSSION:

Since the sorting is independent of the type provided, we are providing random data types for each type of sorting algorithm. The following code is written in a test file which has our library included.

Testing Merge Sort:

Test.asm

```
Include Macros.inc
Include SAL.inc
.data
arr SBYTE 5,-4,3,-2,1,4,20
.code
MAIN PROC
mov eax,TYPE arr
mov esi,OFFSET arr
mov ebx,0
mov edx,LENGTHOF arr
dec edx
call MergeSort
```

Microsoft Visual Studio Debug Console

```
Merge Sort Results:
-4-2+1+3+4+5+20
E:\SortingAlgosLib\Debug\SortingAlgosLib.exe
To automatically close the console when debug
le when debugging stops.
Press any key to close this window . . .
```

Testing Heap Sort:

Test.asm

```
Include Macros.inc
Include SAL.inc
.data
arr SDWORD 5,-4,3,-2,1,4,20,69,420
.code
MAIN PROC
mov eax,TYPE arr
mov esi,OFFSET arr
mov edx,LENGTHOF arr
call HeapSort
```

Microsoft Visual Studio Debug Console

```
Heap Sort Results:
-4-2+1+3+4+5+20+69+420
E:\SortingAlgosLib\Debug\SortingAlgosLib.exe
To automatically close the console when debug
le when debugging stops.
Press any key to close this window . . .
```

Testing Bubble Sort:

Test.asm

```
Include Macros.inc
Include SAL.inc
.data
arr SWORD 5,-4,3,-2,1,4,20,69,420
.code
MAIN PROC
mov ecx,TYPE arr
mov esi,OFFSET arr
mov ebx,LENGTHOF arr
call BubbleSort
```

Microsoft Visual Studio Debug Console

```
Bubble Sort Results:
-4-2+1+3+4+5+20+69+420
E:\SortingAlgosLib\Debug\SortingAlgosLib.exe
To automatically close the console when debug
le when debugging stops.
Press any key to close this window . . .
```

Testing Selection Sort:

Test.asm

```
Include Macros.inc
Include SAL.inc
.data
arr SBYTE 5,-4,3,-2,1,4,20,69,42
.code
MAIN PROC
mov eax,TYPE arr
mov esi,OFFSET arr
mov ecx,LENGTHOF arr
call SelectionSort
```

Microsoft Visual Studio Debug Console

```
Selection Sort Results:
-4-2+1+3+4+5+20+42+69
E:\SortingAlgosLib\Debug\SortingAlgosLib.exe
To automatically close the console when debug
le when debugging stops.
Press any key to close this window . . .
```

Testing Insertion Sort:

Test.asm

```
Include Macros.inc
Include SAL.inc
.data
arr SWORD 5,-4,3,-2,1,4,20,69,42
.code
MAIN PROC
mov ebx,TYPE arr
mov esi,OFFSET arr
mov ecx,LENGTHOF arr
call Insertion_sort
```

Microsoft Visual Studio Debug Console

```
Insertion Sort Results:
-4-2+1+3+4+5+20+42+69
E:\SortingAlgosLib\Debug\SortingAlgosLib.exe
To automatically close the console when debug
le when debugging stops.
Press any key to close this window . . .
```

Testing Gnome Sort:

Test.asm

```
Include Macros.inc
Include SAL.inc
.data
arr SDWORD 5,-4,3,-2,1,4,20,69,42
.code
MAIN PROC
mov eax,TYPE arr
mov esi,OFFSET arr
mov ecx,LENGTHOF arr
call Gnome_sort
```

Microsoft Visual Studio Debug Console

```
Gnome Sort Results:
-4-2+1+3+4+5+20+42+69
E:\SortingAlgosLib\Debug\SortingAlgosLib.exe
To automatically close the console when debug
le when debugging stops.
Press any key to close this window . . .
```

Testing Cocktail Sort:

Test.asm

```
Include Macros.inc
Include SAL.inc
.data
arr SBYTE 5,-4,3,-2,1,4,20,69,42
.code
MAIN PROC
mov ecx,TYPE arr
mov esi,OFFSET arr
mov ebx,LENGTHOF arr
call CocktailSort
```

Microsoft Visual Studio Debug Console

```
Cocktail Sort Results:
-4-2+1+3+4+5+20+42+69
E:\SortingAlgosLib\Debug\SortingAlgosLib.exe
To automatically close the console when debug
le when debugging stops.
Press any key to close this window . . .
```

Testing Shell Sort:

Test.asm

```
Include Macros.inc
Include SAL.inc
.data
arr SWORD 5,-4,3,-2,1,4,20,69,42
.code
MAIN PROC
mov ecx,TYPE arr
mov esi,OFFSET arr
mov ebx,LENGTHOF arr
call ShellSort
```

Microsoft Visual Studio Debug Console

```
Shell Sort Results:
-4-2+1+3+4+5+20+42+69
E:\SortingAlgosLib\Debug\SortingAlgosLib.exe
To automatically close the console when debug
le when debugging stops.
Press any key to close this window . . .
```

CONCLUSION, COST AND FUTURE WORK:

Because the project is software-based, no money was spent on it. On the other hand, we feel that this library can be valuable when writing in assembly language, either in terms of saving time or making it easier to remember the various sorting methods. The most significant aspect of this project is that it handles all data kinds generically, so the user doesn't have to worry about different data types. If you're working in assembly and cooperating with hardware in the future, you may utilize our library to quickly use the function to complete most of the repetitive tasks.

REFERENCES:

Merge Sort Algorithm in Data Structures

Selection Sort

HeapSort

Bubble Sort

Gnome Sort

Cocktail Sort

Insertion Sort

ShellSort

Books:

Assembly Language for x86 processors - Kip R. Irvine