Jack Byrne
I pledge my honor that I have abided by the Stevens Honor System.

Code and final output are displayed at the bottom of the document.

The selection sort produces a sorted array by repeatedly finding the minimum value and swapping it to left, in order.

My implementation of the algorithm in ARMv8 utilizes an outer loop which keeps track of the bounds of the array and an inner loop which looks for the minimum value each iteration. The top of my code initializes several variables including: the length of the array, the address of the array, and several counter variables.

```
        mov x2, #0 //for loop. i = 0


outer_loop:
        mov x5, x2 //minimum index = i
        mov x10, x1 //addr of x1 in x10
```

```
        mov x6, x2 //for loop. j=i
        add x6, x6, #1 //add 1 to j
        mov x9, x1 //addr of x1 in x9
```

The outer loop initializes the counter variables as well as the address of the array in multiple variables. This is so the code can access the array at different locations at the same time.

```
//swap elem at arr[min] and arr[i]
ldr x13, [x10] //arr[min]
ldr x14, [x1] //arr[i]
str x13, [x1]
str x14, [x10]


cmp x2, x3 //is i less than size
add x1, x1, #8 //increment offset
add x2, x2, #1 //i++
b.le outer_loop //loop exit
```

The bottom of the outer loop swaps the two targeted values and then increments the variable as well as the offset. It performs a check to see if the counter variable is less than or equal to the length of the array.

```
inner_loop:

        ldr x12, [x10] // arr[min]
        ldr x11, [x9] // arr[j]
        cmp x11, x12 //if arr[j] < arr[min]: min = j
        b.ge jump
        mov x5, x6
-------------------------------------------------------
        mov x10, x9
jump:
        add x6, x6, #1 //j++
        add x9, x9, #8 //offset 8
        cmp x6, x3 //is j less than size
        b.le inner_loop
```

The inner loop starts by loading in the first available value in the array as the minimum and also the value at the point in the array following the second counter variable. If the value at the counter variable is less than the current store minimum, then it becomes the new minimum and its address is stored with it. Whether the condition is met or not, the code then increments the counter, adds offset to the address designated by the counter, and checks if the counter has reached the end of the array.

```
end:
        sub sp, sp, #16
        str x30, [sp]
        bl printarr
        ldr x30, [sp]
        add sp, sp, #16

        b exit
```

At the end of the selection sort, the printarr function is called. The stack is subtracted from to make room for the variables stored in the function.

```
.func printarr
printarr:

        mov x2, #0 //for loop. i = 0
        ldr x5, =arr //loads address of array into reg 5
        ldr x4, [x5] //loads value of element at x5 in array into reg 4
ploop:

        stp x4, x5, [sp, #-16]!
        stp x2, x30, [sp, #-16]!

        mov x1, x4 //put int to print in reg 1
        ldr x0, =out_string //print. loads print template in reg 0
        bl printf

        ldp x2, x30, [sp], #16
        ldp x4, x5, [sp], #16
        ldr x3, size

        add x5, x5, #8 //add address offset
        ldr x4, [x5] //load next elem

        add x2, x2, #1 //i++
        cmp x2, x3 //is i less than size
        b.lt ploop //loop exit
        br x30 //go back
.endfunc
```

The print function prints the sorted array line by line and then returns. It accomplishes this with a for loop initialized at the top. Before each print, the 4 registers used for keeping track of the array are saved in the stack and then returned after the print. This is because the print function changes the values of the other registers. At the end, the counter is incremented, the address is offset, and the loop is checked to see if it has or has not reached the end. If it has, the function returns.

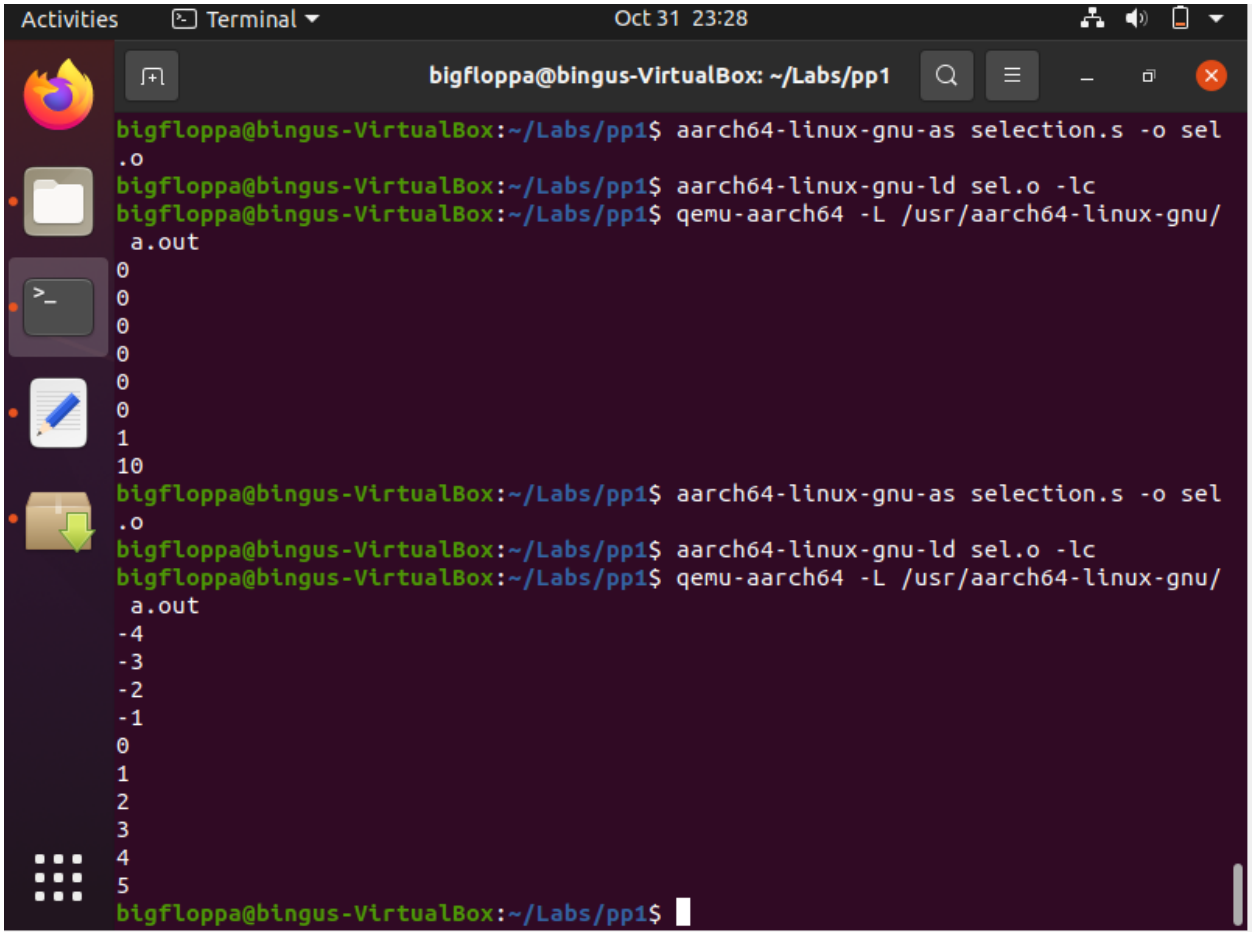
```
exit:
        mov x0, #0
        mov w8, #93
        svc #0
```

Lastly, the exit code ends the program and resets the 0 register.
Thank you.

```
.text
.global _start
.extern printf
.equ ELEM, 10 //array size

_start:
        .global main
main:
        ldr x0, =stack
        mov sp, x0
        ldr x1, =arr //loads address of array into reg 1
        ldr x0, [x1] //loads value of element 0 in array into reg 0
        ldr x3, size //loads size into reg 3
        mov x5, #0 //initialize reg 5
        mov x6, #0 //init reg 6
        mov x7, #0 //init reg 7
        mov x8, #1 //init reg 8
        mov x9, #0 //counter for j
        mov x10, #0 //counter for min
        mov x11, #0 // var for elem of arr[j]
        mov x12, #0 // var for elem of arr[min]
        mov x13, #0 //temp var for swap
        mov x14, #0 //temp var for swap



        mov x2, #0 //for loop. i = 0


outer_loop:
        mov x5, x2 //minimum index = i
        mov x10, x1 //addr of x1 in x10

        mov x6, x2 //for loop. j=i
        add x6, x6, #1 //add 1 to j
        mov x9, x1 //addr of x1 in x9
inner_loop:

        ldr x12, [x10] // arr[min]
        ldr x11, [x9] // arr[j]
        cmp x11, x12 //if arr[j] < arr[min]: min = j
        b.ge jump
        mov x5, x6
        mov x10, x9
jump:
        add x6, x6, #1 //j++
        add x9, x9, #8 //offset 8
        cmp x6, x3 //is j less than size
        b.le inner_loop
```

```
        //swap elem at arr[min] and arr[i]
        ldr x13, [x10] //arr[min]
        ldr x14, [x1] //arr[i]
        str x13, [x1]
        str x14, [x10]


        cmp x2, x3 //is i less than size
        add x1, x1, #8 //increment offset
        add x2, x2, #1 //i++
        b.le outer_loop //loop exit


end:
        sub sp, sp, #16
        str x30, [sp]
        bl printarr
        ldr x30, [sp]
        add sp, sp, #16

        b exit


.func printarr
printarr:

        mov x2, #0 //for loop. i = 0
        ldr x5, =arr //loads address of array into reg 5
        ldr x4, [x5] //loads value of element at x5 in array into reg 4
ploop:

        stp x4, x5, [sp, #-16]!
        stp x2, x30, [sp, #-16]!

        mov x1, x4 //put int to print in reg 1
        ldr x0, =out_string //print. loads print template in reg 0
        bl printf

        ldp x2, x30, [sp], #16
        ldp x4, x5, [sp], #16
        ldr x3, size

        add x5, x5, #8 //add address offset
        ldr x4, [x5] //load next elem

        add x2, x2, #1 //i++
        cmp x2, x3 //is i less than size
        b.lt ploop //loop exit
        br x30 //go back
.endfunc
```

```
exit:
        mov x0, #0
        mov w8, #93
        svc #0

.data
size:
        .dword 10
out_string:
        .ascii "%d\n\0"
arr:
        .dword 5, 4, 3, 2, 1, 0, -1, -2, -3, -4
.bss
        .align 8
out:
        .space 8
        .align 16
        .space 4096
stack:
        .space 16
.end
```


Activities
Terminal
Oct 31 23:28
bigfloppa@bingus-VirtualBox: ~/Labs/pp1
bigfloppa@bingus-VirtualBox:~/Labs/pp1$ aarch64-linux-gnu-as selection.s -o sel
.o
bigfloppa@bingus-VirtualBox:~/Labs/pp1$ aarch64-linux-gnu-ld sel.o -lc
bigfloppa@bingus-VirtualBox:~/Labs/pp1$ qemu-aarch64 -L /usr/aarch64-linux-gnu/
a.out
0
0
0
0
0
0
1
10
bigfloppa@bingus-VirtualBox:~/Labs/pp1$ aarch64-linux-gnu-as selection.s -o sel
.o
bigfloppa@bingus-VirtualBox:~/Labs/pp1$ aarch64-linux-gnu-ld sel.o -lc
bigfloppa@bingus-VirtualBox:~/Labs/pp1$ qemu-aarch64 -L /usr/aarch64-linux-gnu/
a.out
-4
-3
-2
-1
0
1
2
3
4
5
bigfloppa@bingus-VirtualBox:~/Labs/pp1$