Eric Song
CS 382 Project 1: Selection Sort

I pledge my honor that I have abided by the Stevens Honor system.

Attached on the final pages of this document is my actual code. (actual file in zip)

Part 1: Commenting the registers.

```
/*
X20 base address of array
X21 size
X22 i
X23 min_j
X24 j
X25 value of array[j]
X26 value of array[min_j]
X27 length-1
X28 i+1
X29 print index
*/
```

Because this project involves so many different registers, I decided to comment out the saved callee registers, along with their associated variables, in order to help me throughout the code. I used the selection_sort we learned from CS385, which was also commented out.

```
void selection_sort(int array[], const int length) {
   for(int i = 0; i < length - 1; i++) {
       int min_j = i;
       for(int j = i + 1; j < length; j++) {
           if(array[j] < array[min_j]) {
               min_j = j;
           }
       }
       if(min_j != i) {
           swap(array, i, min_j);
       }
   }
}
```

Part 2: Selection Sort
Part 2a: Outer Loop

With selection sort, the outer loop is the main loop that runs the entire function. This outer loop was broken down into different parts, with labels at each part.

```
   ADR X20, arr //loads base address of array
   LDR X21, size
   MOV X27, X21
   SUB X27, X27, 1 // length - 1
   // for(int i = 0; i < length - 1; i++)
   MOV X22, 0
```

These first few lines loaded the base address of the array, the value of the size of the array, and began the initiation of the OUTER for loop.

```
OUTER:
   MOV X23, X22 //int min_j = i;

   MOV X28, X22 // x28 = i
   ADD X28, X28, 1 // x28 = i+1
   MOV X24, X28
```

The OUTER for loop began with the initialization of the INNER for loop, as well as the first line where min_j was set equal to i.

Part 2b: Inner Loop

```
INNER:
   LSL X1, X24, 3 // this offset of j * 8
   ADD X9, X20, X1 // base + offset of array
   LDUR X10, [X9, 0] // value of A[j] inside X10
   LSL X2, X23, 3 // this offset of min_j * 8
   ADD X11, X20, X2 // base + offset of array
   LDUR X12, [X11, 0] // value of A[min_j] inside X12

   CMP X12, X10
   B.GT IFMINJGREATERTHANJ // if X12 > X10
```

The inner for loop, with the INNER label, calculated the offsets of A[j] and A[min_j] respectively, utilizing left shifts, as well as loading to get the value of these array indexes. A[j] and A[min_j] was then compared, and if A[min_j] were to be greater than array[j], then it would branch to IFMINJGREATERTHANJ (if min_j greater than j).

Part 2c: Branch if greater than

```
  B.GT IFMINJGREATERTHANJ // if X12 > X10
   ADD X24, X24, 1
   CMP X24, X21
   B.NE INNER
   B AFTERINNER

IFMINJGREATERTHANJ:
   MOV X23, X24
   ADD X24, X24, 1
   CMP X24, X21
   B.NE INNER
   B AFTERINNER
```

Again, if array[min_j] were to be greater, then it would branch to the label, where min_j was effectively set equal to i. The compare statement compared j to length (in the for loop, j < length was the condition). If this condition was not met, the program would branch back to INNER, otherwise, it would branch straight to AFTERINNER (the part of code after the inner loop [where the swapping takes place])

Part 3: AFTERINNER

```
AFTERINNER:
   CMP X23, X22
   B.NE pleaseSwap
   B DONE

pleaseSwap:
   BL SWAP
DONE:
   ADD X22, X22, 1
   CMP X22, X27 //compares i and (length-1)
   B.NE OUTER // if i != length-1
   B printarr

SWAP:
```

What is effectively happening here is min_j is being compared to i. If these are not equal (if(min_j != i)), then swapping would have to take place. If they are equal, no swapping is done, so we branch automatically to DONE, which brings us back to the to comparison of i to see if we need to branch back to OUTER to run the outer for loop again.

Part 4: SWAP

This was the most tricky part, in my opinion, so just like before, I commented out all the registers I would use beforehand in order to help me with my code better.

```
SWAP: // swap(array, i, min_j); swap array[i] with array[min_j]
/*
X13 = i * 8
X14 = X20 + X13 ==> base + offset
X15 load from X14 ==> array[i]
X16 = min_j * 8
X17 = X20 + X16
X18 load from X17 ==> array[min_j]
X3 = X15
X15 = X18
X18 = X3
STUR X15, [X14, 0]
STUR X18, [X17, 0]
```

These are the registers and instructions I planned to use. The following code is the implementation (LITERALLY LINE BY LINE) of the comment. I think this was such a useful step. I had everything drawn out (i guess typed out), so there was no way I could get lost.

```
*/
   SUB SP, SP, 8
   STUR LR, [SP, 0]

   LSL X13, X22, 3
   ADD X14, X20, X13
   LDUR X15, [X14, 0]

   LSL X16, X23, 3
   ADD X17, X20, X16
   LDUR X18, [X17, 0]
   MOV X3, X15
   MOV X15, X18
   MOV X18, X3

   STUR X15, [X14, 0]
   STUR X18, [X17, 0]

   LDUR LR, [SP, 0] //loading return address to start
   ADD SP, SP, 8 //adding it back (pop) == deallocating
   BR LR // or X30
```

Now at this point, I know that my array has been sorted. The outer for loop has been finished. The function selection_sort has effectively been completed. I make sure to load the values of A[i] and A[min_j], I use offsets and base address and left shifts to make this happen. I use X3 as a temporary register to hold A[i] while swapping. I finally store these values back into the proper place in the address of the array. After that, I pop the SP and BR LR back to the original place in code where I called SWAP.

Part 5: PRINT ARRAY

```
printarr:
   //ideally print array
   ADR X0, str
   LDUR X1, [X20, 0]
   BL printf
   SUB X21, X21, 1
   ADD X20, X20, 8
   CBNZ X21, printarr
   B EXIT
```

This just loops and prints the array index, using a counter X20. I make sure to use X21 and X20 so that the printf does not change the values.

Part 6: special case

If the array were to be empty, I just want to print out the empty array.

```
_start:
   LDR X21, size
   CBZ X21, EXIT
   B SelectionSort
   // exit call
EXIT:
   MOV X0, 0
   MOV W8, 93
   SVC 0
```

I just branch to exit of the size of the array is 0.

Code:

```
/*
void selection_sort(int array[], const int length) {
   for(int i = 0; i < length - 1; i++) {
       int min_j = i;
       for(int j = i + 1; j < length; j++) {
           if(array[j] < array[min_j]) {
               min_j = j;
           }
       }
       if(min_j != i) {
           swap(array, i, min_j);
       }
   }
}
*/

// Eric Song
// I pledge my honor that I have abided by the Stevens Honor System
.text
.global _start
.extern printf
/*
X20 base address of array
X21 size
X22 i
X23 min_j
X24 j
X25 value of array[j]
X26 value of array[min_j]
X27 length-1
X28 i+1
X29 print index
*/

SelectionSort:
   ADR X20, arr //loads base address of array
   LDR X21, size
   MOV X27, X21
   SUB X27, X27, 1 // length - 1
```

```
   // for(int i = 0; i < length - 1; i++)
   MOV X22, 0
OUTER:
   MOV X23, X22 //int min_j = i;

   MOV X28, X22 // x28 = i
   ADD X28, X28, 1 // x28 = i+1
   MOV X24, X28
INNER:
   LSL X1, X24, 3 // this offset of j * 8
   ADD X9, X20, X1 // base + offset of array
   LDUR X10, [X9, 0] // value of A[j] inside X10
   LSL X2, X23, 3 // this offset of min_j * 8
   ADD X11, X20, X2 // base + offset of array
   LDUR X12, [X11, 0] // value of A[min_j] inside X12

   CMP X12, X10
   B.GT IFMINJGREATERTHANJ // if X12 > X10
   ADD X24, X24, 1
   CMP X24, X21
   B.NE INNER
   B AFTERINNER

IFMINJGREATERTHANJ:
   MOV X23, X24
   ADD X24, X24, 1
   CMP X24, X21
   B.NE INNER
   B AFTERINNER

// end of inner loop
AFTERINNER:
   CMP X23, X22
   B.NE pleaseSwap

pleaseSwap:
   BL SWAP

   ADD X22, X22, 1
   CMP X22, X27 //compares i and (length-1)
   B.NE OUTER // if i != length-1
   B printarr
```

```
SWAP: // swap(array, i, min_j); swap array[i] with array[min_j]
/*
X13 = i * 8
X14 = X20 + X13 ==> base + offset
X15 load from X14 ==> array[i]
X16 = min_j * 8
X17 = X20 + X16
X18 load from X17 ==> array[min_j]
X3 = X15
X15 = X18
X18 = X3
STUR X15, [X14, 0]
STUR X18, [X17, 0]

*/
   SUB SP, SP, 8
   STUR LR, [SP, 0]

   LSL X13, X22, 3
   ADD X14, X20, X13
   LDUR X15, [X14, 0]

   LSL X16, X23, 3
   ADD X17, X20, X16
   LDUR X18, [X17, 0]
   MOV X3, X15
   MOV X15, X18
   MOV X18, X3

   STUR X15, [X14, 0]
   STUR X18, [X17, 0]

   LDUR LR, [SP, 0] //loading return address to start
   ADD SP, SP, 8 //adding it back (pop) == deallocating
   BR LR // or X30



printarr:
   //ideally print array
   ADR X0, str
```

```
    LDUR X1, [X20, 0]
    BL printf
    SUB X21, X21, 1
    ADD X20, X20, 8
    CBNZ X21, printarr
    B EXIT

_start:
    B SelectionSort
    // exit call
EXIT:
    MOV X0, 0
    MOV W8, 93
    SVC 0

.data
size: .quad 20
arr: .quad 14, 50, 7, 45, 18, 39, 19, 4, 10, 28, 29, 12, 6, 26, 11, 21, 16, 33, 48, 15
str: .ascii "%d\n\0" // quad, use %ld to print

.end
```