Rashmika Batra

Programming Project One

First, I worked on my start function. I loaded the stack pointer, num array and length variable in these lines of code:

```
ldr x28, =stack
ldr x0, =nums
ldr x1, =length
```

Then, I made space on my stack, stored the link register and called the sort function in these lines of code:

```
sub x28, x28, #8
stur x30, [x28, #0]
bl sortNums
```

Then I reloaded my linked register, unfreed my stack space in these lines of code.

```
ldur x30, [x28, #0]
add x28, x28, #8
```

Then, I made space on my stack, stored my linked register on the stack, called the print function and reloaded the link register in these lines of code:

```
sub x28, x28, #8
stur x30, [x28, #0]
bl printNums
ldur x30, [x28, #0]
```

Then, I moved on to my swapElements function.

I moved the num array to register 4, found the offset value for index 1, then the offset value for index 2 in these lines of code:

```
mov x4, x2
lsl x5, x0, #3
lsl x6, x1, #3
```

Then, I found the address for element 1, found the address for element 2 and loaded these elements in these lines of code.

```
    ldr x7, [x5, #0]
    ldr x8, [x6, #0]
    stur x7, [x6, #0]
    stur x8, [x5, #0]
```

Then, I worked on my sort nums function.

First, I compared length to counter, and if the length was less than or equal to the counter, I broke out of the loop.

```
    cmp x26, x27
    ble breakLoop
```

Then I moved values into registers to prepare for function call, made space on stack and stored the link register on the stack.

```
    mov x0, x27
    mov x1, x26
    sub x28, x28, #8
    stur x30, [x28, #0]
```

Then I called my innerloop function, reloaded the stack pointer, unfreed stack space, moved values into their registers, made space on the stack and also stored a link register on stack before calling my swap function.

```
    bl inside
    ldur x30, [x28, #0]
    add x28, x28, #8
    mov x1, x27
    mov x2, x25
    sub x28, x28, #0
    stur x30, [x28, #0]
```

Then, I reloaded my stack pointer, unfreed the stack, increment the counter and called the next iteration of my loop.

```
    ldur x30, [x28, #0]
    add x28, x28, #8
    add x27, x27, #1
```

```
	b loop
```

Now I will be explaining the inside and insideLoop functions.

In these lines of code, I compared the length to the counter. If the length was less than or equal to the counter, I broke out of the loop.

```
	cmp x21, x20
	mov x1, x24
	ble breakInsideLoop
```

Then I found the offset value of the current element, loaded the current element, compared the current element to the minimum value so far. If the current element is less than the min, I said go to the newMin, otherwise go to noNewMin.

```
	lsl x19, x20, #3
	add x19, x22, x19
	ldr x19, [x19, #0]
	cmp x19, x24
	blo newMin
	b noNewMin
```

The noNewMin function just increments the counter and calls the next iteration of the loop.

```
	add x20, x20, #1
	b insideLoop
```

The newMin function updates the min value, updates the current min index, increments the counter and calls the next iteration.

```
	mov x24, x19
	mov x0, x20
	add x20, x20, #1
	b insideLoop
```

Now, I will be explaining the print loop function.

In these lines of code, I compared length and counter. If the length is less than or equal to the counter, then break out of the loop.

```
cmp x20, x19
ble breakPrintLoop
```

Then, I found the offset value of the current element, then the address of the current element. I loaded the current element and moved the current value for function call in these lines of code.

```
lsl x22, x19, #3
add x22, x21, x22
ldr x23, [x22, #0]
mov x1, x23
```

Then, I made space on my stack and stored the link register on stack.

```
sub x28, x28, #8
stur x30, [x28, #0]
```

Then, I called the print function, reloaded my link register, unfreed my stack space, increment the counter and called the next iteration.

```
 bl printf
ldur x30, [x28, #0]
add x28, x28, #8
add x19, x19, #1
b printLoop
```