So the first thing that we're going to need to be doing is having our program read through a list of 8 numbers and then pick the smallest number it sees, then compare and contrast it with each number it sees going down the list. Once it gets to the end it will swap the smallest number to the beginning of the list.

```
#My Pseudocode
FOR beginningnumber in range
   if currentnumber < beginningnumber
      SWAP currentnumber <- beginningnumber
   ELSE
      beginningnumber ->
```

```
#AI Pseudocode
function selectionSort(list):
   for i from 0 to length of list - 1:
      minIndex = i

      for j from i+1 to length of list - 1:
         if list[j] < list[minIndex]:
            minIndex = j

      if minIndex ≠ i:
         swap list[i] and list[minIndex]
```

#My analysis
I didn't think to add the second for loop here as well. I also decided to make this a lot more readable than both my earlier code and the ai's code. The ai had much better logic since it accounted for the second loop. It however didn't have good naming conventions and was hard to
understand. I had to drastically change my pseudocode by the end.

```
#My Updated Pseudocode
#A
FOR currentNumber in numberList - 1:
   smallestNumber = currentNumber  #This by default sets the smallest number to the number
we start on.

   #B
```

```
    FOR check from currentNumber + 1 to length of numberList - 1: #Check our number to the end of the list.


        IF numberList[check] < numberList[smallestNumber]:        #Check if our current number is smaller than


            smallestNumber = check                                #the one we started on.

    #C
    IF smallestNumber != currentNumber: #If the number we checked is smaller than our original number the swap them.
        SWAP numberList[currentNumber] and numberList[smallestNumber]
```

numberList = [26, 6, 90, 55]

```
Step   | currentNumber | check | smallestNumber | Output List
------------------------------------------------------------
A0     |      26       |   -   |       26       | [26, 6, 90, 55]
B0.1   |      26       |   6   |        6       | [26, 6, 90, 55]
B0.2   |      26       |   90  |        6       | [26, 6, 90, 55]
B0.3   |      26       |   55  |        6       | [26, 6, 90, 55]
C0     |      26       |   -   |        6       | [6, 26, 90, 55] #26 and 6 swap

A1     |      26       |   -   |       26       | [6, 26, 90, 55]
B1.1   |      26       |   90  |       26       | [6, 26, 90, 55]
B1.2   |      26       |   55  |       26       | [6, 26, 90, 55]
C1     |      26       |   -   |       26       | [6, 26, 90, 55] #no change

A2     |      90       |   -   |       90       | [6, 26, 90, 55]
B2.1   |      90       |   55  |       55       | [6, 26, 90, 55]
C2     |      90       |   -   |       55       | [6, 26, 55, 90] #90 and 55 swap

A3     |      90       |   -   |       90       | [6, 26, 55, 90]
C3     |      90       |   -   |       90       | [6, 26, 55, 90] #final loop
```

#Final Sorted List: [6, 26, 55, 90]

The program loops through about 10 times in total for about 4 data points
so the algorithmic efficiency is about O(n**2)