**2.1-1 :**

From

31 41 59 26 41 58 to 26 31 41 41 58 59

**2.1-2**

1 **for** j = 2 **to** A.*length do*

2 *key =* A[j]

3 **//**Insert A[j] into the sorted sequence A[1 : : j -1].

4 i = j - 1

5 **while** i > 0 and A[i] < *key do (changed from A[i] > key)*

6 A[i+ 1] =A[i]

i = i -1

7 end while

8 A[i+1]= *key*

9 end for

Changed A[i] while from greater than to less than and then added the end while section and end for

**2.1-3**

for j = a.length

key = v

while(v != a[i])

go through array

else

return match or nil

meets three criteria because it is true before and during and exits with nil or the correct output

**2.1-4**

Input: two n-element arrays A

**2.2-1**

Express n ^3 /1000 − 100n ^ 2 − 100n + 3 in terms of Θ notation.

n ^3 /1000 − ~~100~~n ^ 2 ~~− 100n + 3~~

n^3 > n^2 so we are left with: Θ(n^3) <https://web.cs.wpi.edu/~guttman/cs2223/seven_rules.pdf>

**2.2-1**

Consider sorting n numbers stored in array A by first finding the smallest element

of A and exchanging it with the element in A[1]. Then find the second smallest

element of A, and exchange it with A[2]. Continue in this manner for the first n-1

elements of A. Write pseudocode for this algorithm, which is known as ***selection***

***sort***.

For I = n to n – 1 do

min = i

for j = i + 1 to n

do

if A[j] < A[min] then

min = j

End if

End for

Swap A[min] and A[i]

End for

What loop invariant does this algorithm maintain?

Selection sort swaps a[1] with the smallest element in the array.

1. A[1] to a[i-1] is already sorted

Why does it need to run for only the first n - 1 elements, rather than for all n elements?

Because it is already greater than the elements that are earlier

Give the best-case and worst-case running times of selection sort in O notation.

Best and worst are O(n^2)

**2.2-3**

O(A.length) explanation found here <https://sites.math.rutgers.edu/~ajl213/CLRS/Ch2.pdf>

**2.2-4**

We could improve almost any algorithm by first checking the random output and then see if if satisfies the goal of the algorithm. For instance with selection sort we can first check the elements of A and then check if they are sorted if they are then the output is a and that leaves us with a O(n) running time.

**2.3-1**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **3** | **9** | **26** | **38** | **41** | **49** | **52** | **57** |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 26 | 41 | 52 |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | 38 | 49 | 57 |

|  |  |
| --- | --- |
| 26 | 52 |

|  |  |
| --- | --- |
| 38 | 57 |

|  |  |
| --- | --- |
| 9 | 49 |

|  |  |
| --- | --- |
| 3 | 41 |

|  |
| --- |
| 52 |

|  |
| --- |
| 26 |

|  |
| --- |
| 38 |

|  |
| --- |
| 57 |

|  |
| --- |
| 9 |

|  |
| --- |
| 49 |

|  |
| --- |
| 3 |

|  |
| --- |
| 41 |

**2.3-2**

We can just add a condition that stops when the array is empty

**2.3-3**

Since n is a power of two, we may write n = 2k . If k = 1, T(2) = 2 = 2 lg(2). Suppose it is true for k, we will show it is true for k + 1. T(2k+1) = 2T 2 k+1 2 + 2k+1 = 2T 2 k + 2k+1 = 2(2k lg(2k )) + 2k+1 = k2 k+1 + 2k+1 = (k + 1)2k+1 = 2k+1 lg(2k+1) = n lg(n)