

5.

a) Pseudocódigo:

```
1  function cycle_sort(array):
2      n = length(array)
3      for i = 0 to n-2 do
4          item = array[i]
5          pos = i
6
7          # Encuentra la posición en la que se debe colocar el elemento en su posición
8 correcta
9          for j = i+1 to n-1 do
10             if array[j] < item then
11                 pos = pos + 1
12
13             # Si el elemento ya está en su posición correcta, continúa con el siguiente
14 elemento
15             if pos == i then
16                 continue
17
18             # Coloca el elemento en su posición correcta
19             while item == array[pos] do
20                 pos = pos + 1
21             swap(array[pos], item)
22
23             # Rota los elementos en el ciclo
24             while pos != i do
25                 pos = i
26                 for j = i+1 to n-1 do
27                     if array[j] < item then
28                         pos = pos + 1
29                 while item == array[pos] do
30                     pos = pos + 1
31                 swap(array[pos], item)
32
33     return array
35
```

b) Prueba de escritorio con {3, 5, 10, 6, 4, 2, 6, 1} usando impresiones en python:

```
-----Iteración 1 -----
item = 3
pos = 0
-----Ciclo para encontrar la posición correcta
j = 1 | pos = 0
j = 2 | pos = 0
j = 3 | pos = 0
j = 4 | pos = 0
j = 5 | pos = 1
j = 6 | pos = 1
j = 7 | pos = 2
-----Colocación del item en la posición correcta
array[2]= 3 | array = [3, 5, 3, 6, 4, 2, 6, 1]
item = 10
-----Rotar los elementos en el ciclo
pos = 0
-----Ciclo for anidado
j = 1 | pos = 1
j = 2 | pos = 2
j = 3 | pos = 3
j = 4 | pos = 4
j = 5 | pos = 5
j = 6 | pos = 6
j = 7 | pos = 7
-----Ciclo while anidado
array[7]= 10 | array = [3, 5, 3, 6, 4, 2, 6, 10]
item = 1
-----
pos = 0
-----Ciclo for anidado
j = 1 | pos = 0
j = 2 | pos = 0
j = 3 | pos = 0
j = 4 | pos = 0
j = 5 | pos = 0
j = 6 | pos = 0
j = 7 | pos = 0
-----Ciclo while anidado
array[0]= 1 | array = [1, 5, 3, 6, 4, 2, 6, 10]
item = 3
-----

-----Iteración 2 -----
```

```

item = 5
pos = 1
-----Ciclo para encontrar la posición correcta
j = 2 | pos = 2
j = 3 | pos = 2
j = 4 | pos = 3
j = 5 | pos = 4
j = 6 | pos = 4
j = 7 | pos = 4
-----Colocación del item en la posición correcta
array[4]= 5 | array = [1, 5, 3, 6, 5, 2, 6, 10]
item = 4
-----Rotar los elementos en el ciclo
pos = 1
-----Ciclo for anidado
j = 2 | pos = 2
j = 3 | pos = 2
j = 4 | pos = 2
j = 5 | pos = 3
j = 6 | pos = 3
j = 7 | pos = 3
-----Ciclo while anidado
array[3]= 4 | array = [1, 5, 3, 4, 5, 2, 6, 10]
item = 6
-----
pos = 1
-----Ciclo for anidado
j = 2 | pos = 2
j = 3 | pos = 3
j = 4 | pos = 4
j = 5 | pos = 5
j = 6 | pos = 5
j = 7 | pos = 5
-----Ciclo while anidado
array[5]= 6 | array = [1, 5, 3, 4, 5, 6, 6, 10]
item = 2
-----
pos = 1
-----Ciclo for anidado
j = 2 | pos = 1
j = 3 | pos = 1
j = 4 | pos = 1
j = 5 | pos = 1
j = 6 | pos = 1
j = 7 | pos = 1
-----Ciclo while anidado
array[1]= 2 | array = [1, 2, 3, 4, 5, 6, 6, 10]
item = 5

```

-----Iteración 3 -----

item = 3

pos = 2

-----Ciclo para encontrar la posición correcta

j = 3 | pos = 2

j = 4 | pos = 2

j = 5 | pos = 2

j = 6 | pos = 2

j = 7 | pos = 2

-----Iteración 4 -----

item = 4

pos = 3

-----Ciclo para encontrar la posición correcta

j = 4 | pos = 3

j = 5 | pos = 3

j = 6 | pos = 3

j = 7 | pos = 3

-----Iteración 5 -----

item = 5

pos = 4

-----Ciclo para encontrar la posición correcta

j = 5 | pos = 4

j = 6 | pos = 4

j = 7 | pos = 4

-----Iteración 6 -----

item = 6

pos = 5

-----Ciclo para encontrar la posición correcta

j = 6 | pos = 5

j = 7 | pos = 5

-----Iteración 7 -----

item = 6

pos = 6

-----Ciclo para encontrar la posición correcta

j = 7 | pos = 6

[1, 2, 3, 4, 5, 6, 6, 10]

- c) Inicialización: Antes de la primera iteración del ciclo, el arreglo tiene un solo ciclo de longitud n , lo que satisface la propiedad de que todos los ciclos excepto el primero están ordenados, ya que solo hay un ciclo en este momento.

Mantenimiento: Durante la i -ésima iteración del ciclo, el elemento i -ésimo se coloca en su posición correcta, lo que implica que el ciclo formado por los elementos anteriores a $i-1$ es ordenado. Además, el algoritmo garantiza que los ciclos son de longitud 1 al terminar cada iteración, ya que cada elemento se coloca en su posición correcta y se forma un ciclo con él mismo.

Terminación: Después de la última iteración del ciclo, todos los elementos están en su posición correcta y el arreglo está completamente ordenado.

- d) La complejidad de tiempo del algoritmo Cycle Sort es $O(n^2)$, donde n es la longitud del arreglo. Esto se debe a que en el peor caso, cada elemento del arreglo debe ser comparado con todos los demás elementos y posiblemente intercambiado con ellos, lo que requiere un número cuadrático de operaciones. Sin embargo, en el mejor caso (cuando el arreglo ya está ordenado), la complejidad es $O(n)$.
- e) La complejidad de espacio del algoritmo Cycle Sort es $O(1)$, ya que no se utiliza memoria adicional para realizar el ordenamiento, sino que se modifican los elementos dentro del mismo arreglo.
- f) Sí, el algoritmo Cycle Sort es in-place, ya que no utiliza una cantidad adicional de memoria para realizar las operaciones de ordenamiento, sino que reordena los elementos dentro del mismo arreglo de entrada.

Además, el algoritmo Cycle Sort es estable, lo que significa que mantiene el orden relativo de los elementos con valores iguales. En otras palabras, si hay dos elementos con el mismo valor, el algoritmo los deja en el mismo orden relativo que tenían antes de la ordenación. Esto se debe a que, en el proceso de encontrar la posición correcta para un elemento, el algoritmo busca todas las posiciones que son menores que la posición actual, lo que garantiza que el elemento se coloque en la última posición disponible antes de cualquier otro elemento que tenga el mismo valor.