JORGE DE GOYENECHE
20.538.979-2

3)

d)

```
1 WEBBSORTER(A):
2     index = [(0, 4), (6, 10)]
3     for i in 0, ..., 1:
4         if i = 0:
5             B = COUNTINGSORT(A, Z, index[i])
6         else:
7             C = COUNTINGSORT(B, Z, index[i]) → 8 return C
```

```
1 COUNTINGSORT(A, Z, range)
2     B[0, ..., n-1] ← vacío
3     C[0, ..., Z] ← vacío
4     for i in 0, ..., Z:
5         C[i] = 0
6     for i in 0, ..., n-1:
7         C[A[i][range[0]:range[1]]] += 1
8     for j in 1, ..., Z:
9         C[j] += C[j-1]
10    for i in n-1, ... 0:
11        B[C[A[i][range[0]:range[1]]] - 1] = A[i]
12        C[A[i][range[0]:range[1]]] -= 1
13    return B
```

orden
Alfanumerico → 0 → 9 → A → Z
10 36

WEBBSORTER = RADIX SORT

Z = indice 36

range se ocupa de solo levisar la slice del string que nos interesa.

Asumo que el computador procesa los valores alfanumericos a indices, lo que toma O(n) para cada dato ∴ O(n), lo que sigue manteniendo linealidad.

Puede hacerse en vez de usar slicing, un pre-procesamiento por dato que defina los valores a un indice y los almacene en un struct.

Este proceso tomaria O(n), por lo que mantendria la linealidad necesaria.

b) Existirá una variable entregada al counting sort que ordene y sume los valores de repeticion C de forma inversa, de la siguiente manera desde la linea 6 de counting sort:

```
6  for i in 0, ..., n-1:
7      C[z-1-A[i].splice] += 1
8  for j in z-1, ..., 0
9      C[j] += C[j+1]
10 for i in 0, ..., n-1:
11     B[C[z-1-A[i].splice]] = A[i]
12     C[z-1-A[i].splice] -= 1
13 return B
```

(.splice) = [range[0] : range[1]]

0 1 2 3 4 5 → 6-4-1 → 5-4 → 1

Se accedera a esta opcion con un simple switch segun la variable inverse ∈ {true, false}

c) Si se utiliza quicksort in place, el algoritmo no cumpliría su función ya que este no es estable, por lo que al revisar el segundo atributo significativo el orden anterior se perderá. Además sea cual sea la version que ese, aumentará el tiempo de ejecución ya que quicksort posee un $O(n\log(n))$ en vez del COUNTINGSORT $O(n)$

∴

RADIX COUNTING ×2 → $O(n) \times 2$ → $O(2n)$

<

RADIX QUICK ×2 → $O(n\log(n)) \times 2$ → $O(2n\log(n))$