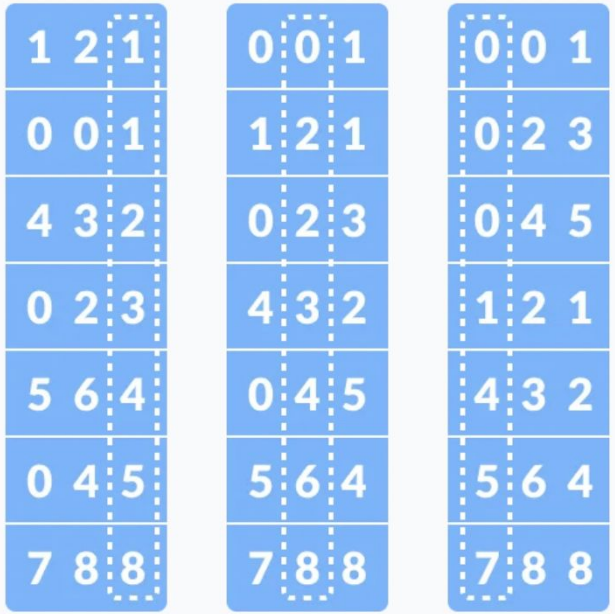
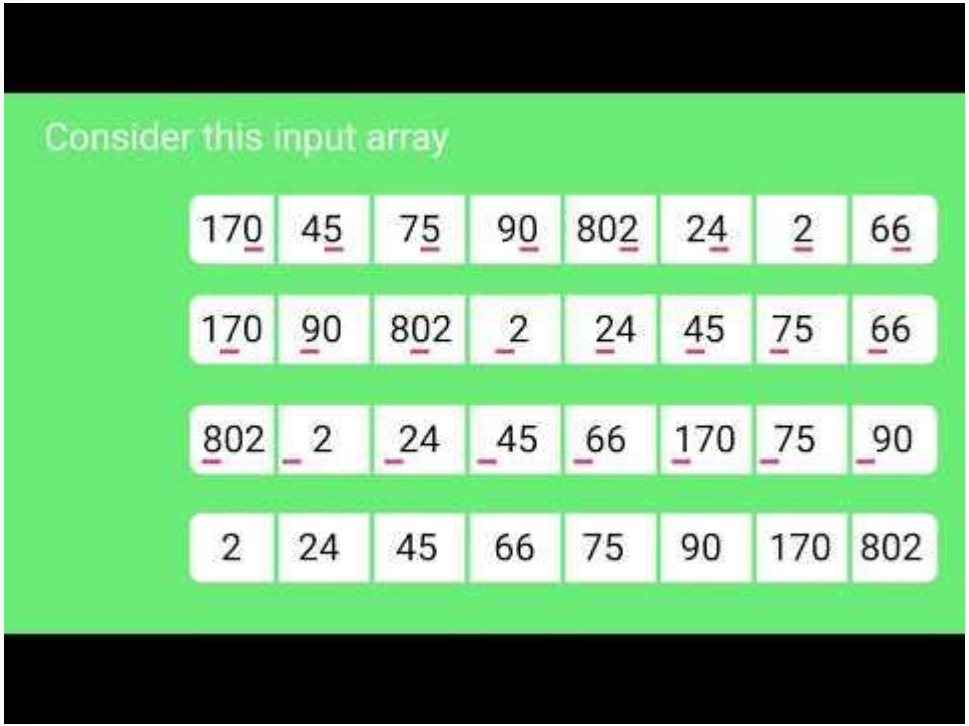
Ayudantía 8: Sorting en O(n)

Radix Sort



sorting the integers according to units, tens and hundreds place digits



Radix Sort

```
radixSort(array)
  d <- maximum number of digits in the largest element
  create d buckets of size 0-9
  for i <- 0 to d
    sort the elements according to ith place digits using countingSort

countingSort(array, d)
  max <- find largest element among dth place elements
  initialize count array with all zeros
  for j <- 0 to size
    find the total count of each unique digit in dth place of elements and
    store the count at jth index in count array
  for i <- 1 to max
    find the cumulative sum and store it in count array itself
  for j <- size down to 1
    restore the elements to array
    decrease count of each element restored by 1
```

Ejemplo: I2-2022-1

Una central telefónica registra en un archivo LOG las llamadas realizadas a través de ella durante un mes dado. En dicho archivo, en cada fila, registra: fecha_llamada, hora_llamada, número_origen, número_destino, duración_llamada.

Se desea utilizar el archivo LOG para realizar la facturación del servicio telefónico, indicando separadamente para cada número de origen el detalle en orden cronológico de cada llamada realizada (fecha, hora, destino, duración) y el costo total del servicio calculado como la suma de las duraciones de las llamadas multiplicado por el valor del minuto.

Ejemplo: I2-2022-1

a) Proponga, utilizando la materia vista en el curso hasta el momento, una solución para obtener la facturación indicada, de la manera más eficiente en tiempo posible.
b) Muestre que su solución cumple con los requisitos indicados para cada característica de la facturación.

a)

El input es fecha llamada, hora llama, número origen, número destino y duración.
El output por cada número de origen, entregar sus llamadas en forma cronológica y su costo total.

Para esto se utilizaría el algoritmo de ordenamiento radixSort, donde se aplica por cada campo, esto usando como subrutina un algoritmo de ordenación estable (por ejemplo, countingSort o insertionSort).
Para tener un orden final respecto a los números de origen se puede aplicar como último campo de ordenamiento a origen. El tiempo es lineal.

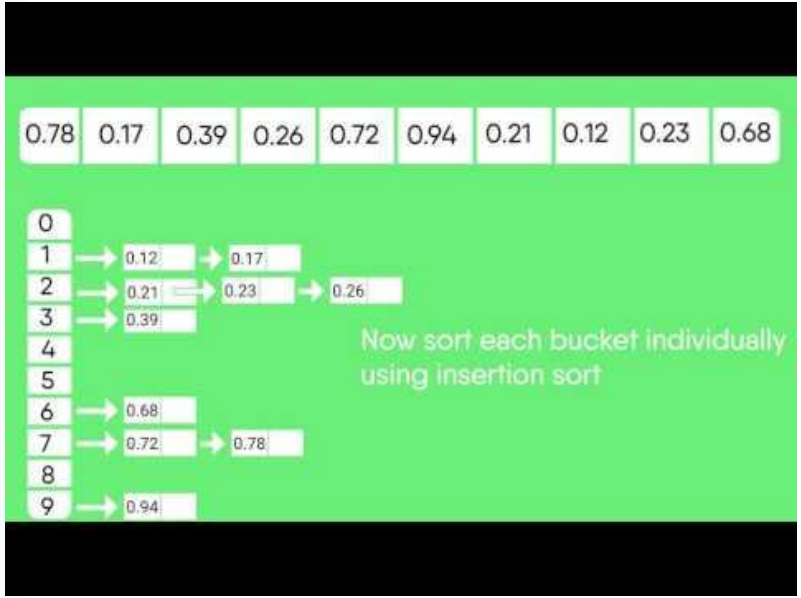
Ya con el listado de números ordenados de forma cronológica se realiza una última pasada por todos los registros para calcular el costo total respecto a cada número, esto es lineal.

b)

Para mostrar que cumple con los requisitos de cada característica, se tiene que cumplir un orden relativo a dicha característica, por ejemplo, que para un mismo día, número de destino, número de origen, duración, pero distintas horas exista este orden. Esto se cumple si se utiliza un algoritmo estable. Es decir se respeta la separación por número de origen, orden cronológico y cálculo de costo total.

Bucket Sort


11 9 21 8 17 19 13 1 24 12
Scatter
1
9 8
11 13 12
17 19
21 24
0-5
5-10
10-15
15-20
20-25
Sort
1
8 9
11 12 13
17 19
21 24
0-5
5-10
10-15
15-20
20-25
Gather
1 8 9 11 12 13 17 19 21 24


0.78 0.17 0.39 0.26 0.72 0.94 0.21 0.12 0.23 0.68
0
1
2
3
4
5
6
7
8
9
0.12
0.17
0.21
0.23
0.26
0.39
0.68
0.72
0.78
0.94
Now sort each bucket individually using insertion sort

Bucket Sort

```
bucketSort()
  create N buckets each of which can hold a range of values
  for all the buckets
    initialize each bucket with 0 values
  for all the buckets
    put elements into buckets matching the range
  for all the buckets
    sort elements in each bucket
  gather elements from each bucket
end bucketSort
```

¡Muchas gracias! <3

Sesión de ayuda para la Tarea