RADIXSORT:

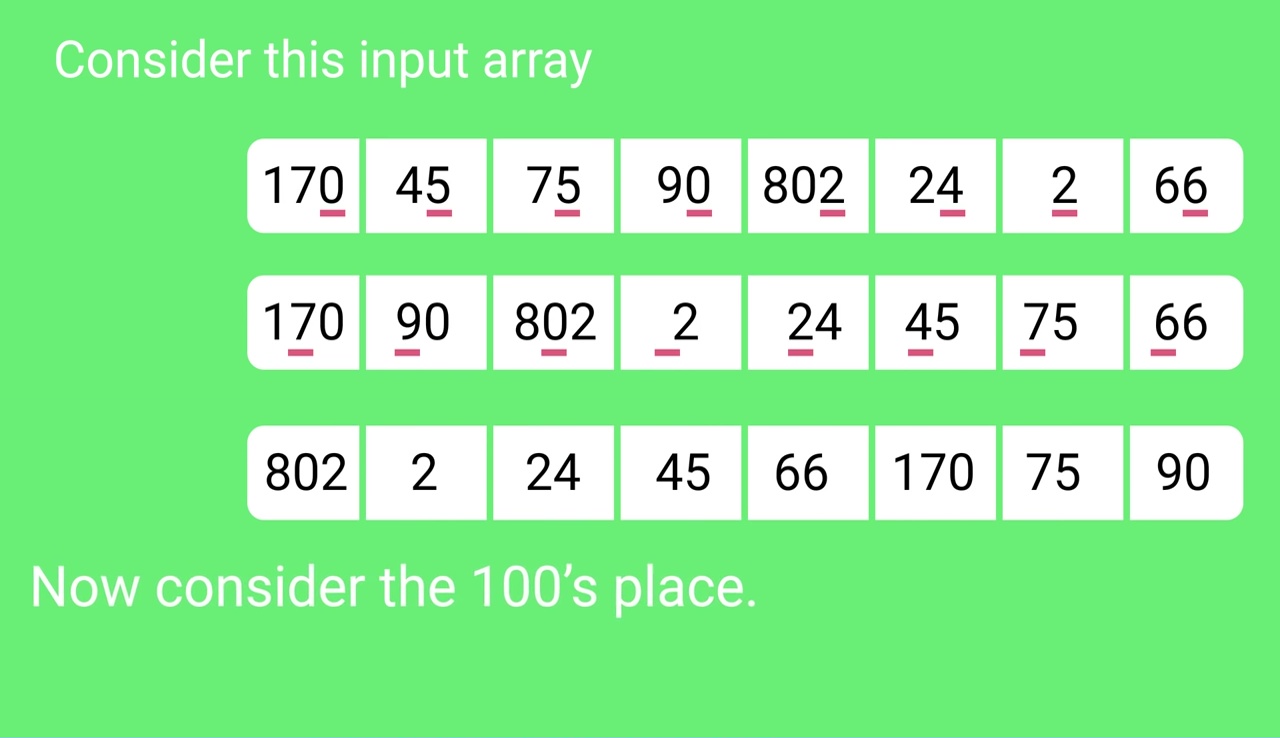
Dicho método consiste en ir verificando la posición de las unidades, luego de las decenas, luego de las centenas y así consecutivamente y en cada una ordenar a partir de dicha cifra.

Cabe destacar que para aquellas cifras que no presenten dígitos en tales posiciones se tomarán como 0 y por ende irán antes que aquellas que tengan dígitos allí.

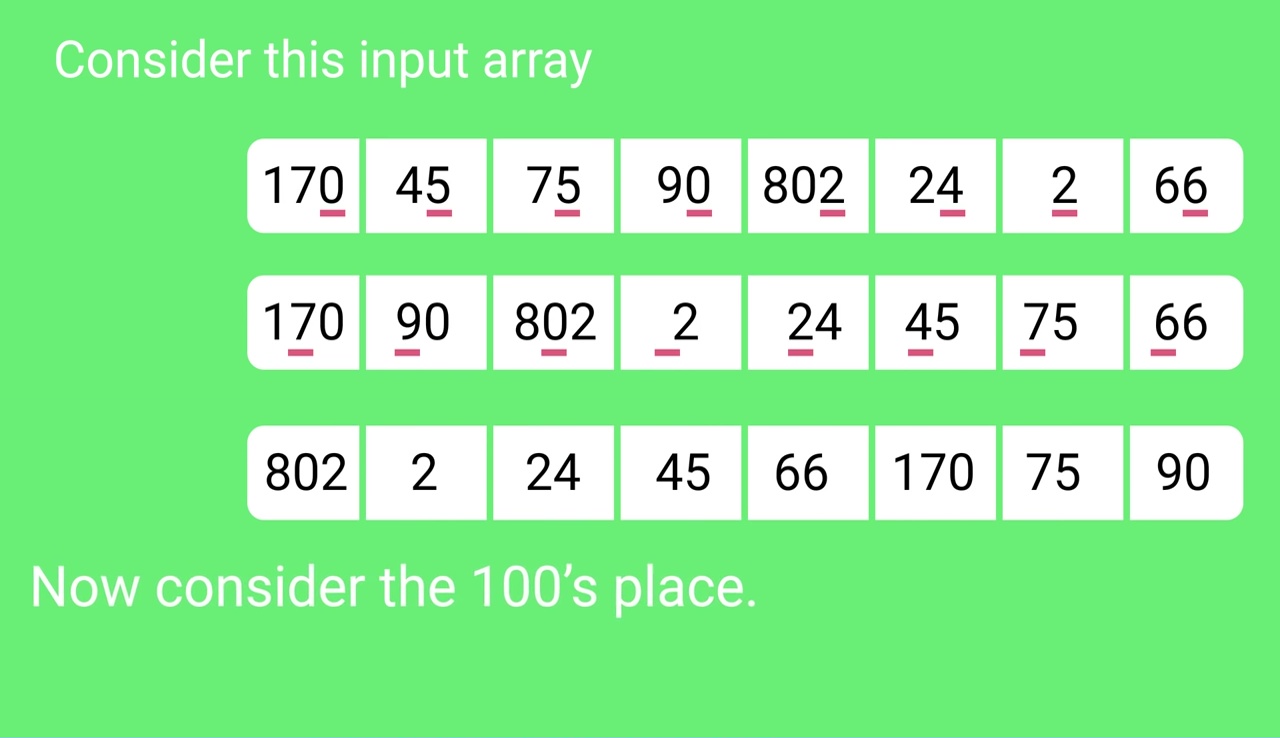
Es importante saber que este ordenamiento es un acoplamiento para el counting Sort, el cual trabaja ordenando números por su clave, es decir el valor, solo que al trabajar con números de muchas cifras el rango de revisión se vuelve muy grande, por lo que se vuelve ineficiente. Lo que hace el radixSort es acotar dicho rango entre 0 y 9 en cada cifra y generar el ordenamiento a partir de este.

Este método es útil incluso para generar ordenamientos lexicográficos, puesto que funciona como si miráramos “letra a letra” y ordenáramos a partir de estas (claramente deberíamos pasar en programación las letras a números, pero eso se puede simplificar sabiendo que cada letra tiene su representación numérica en el código ASCII.

Así:

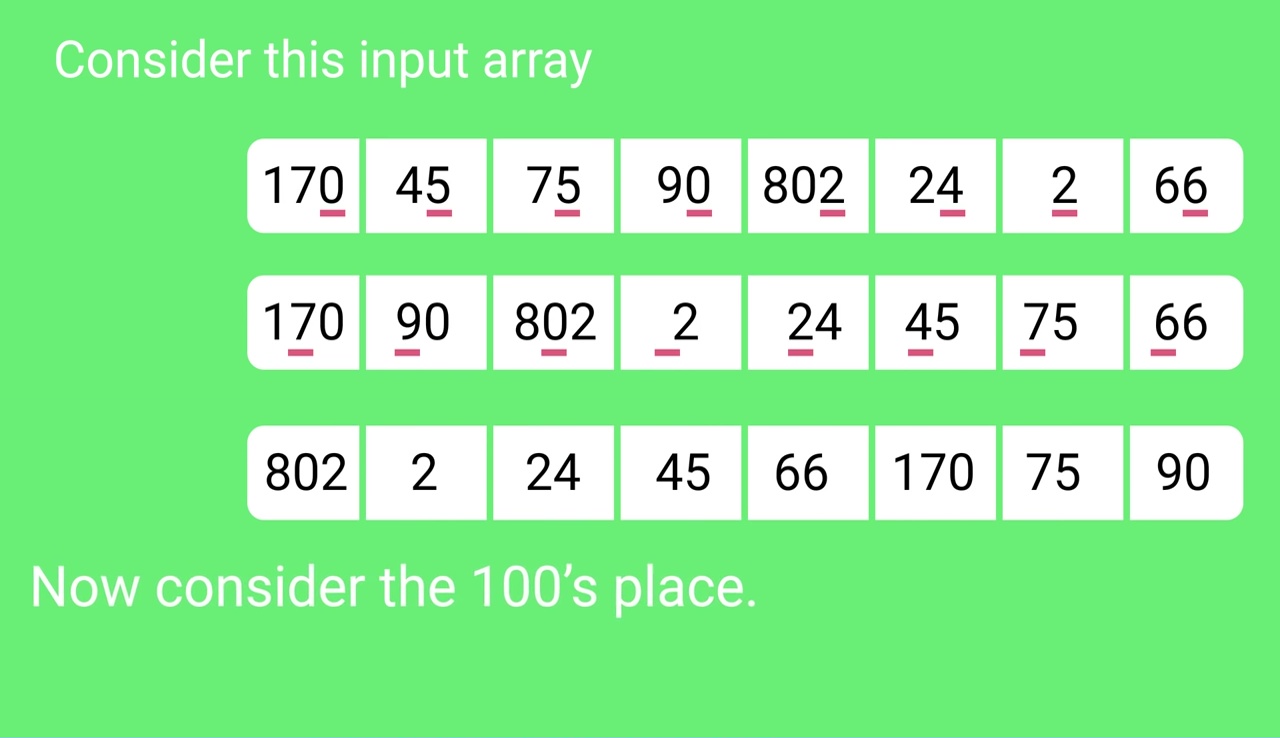


AL REVISAR LAS UNIDADES, NOS QUEDA:

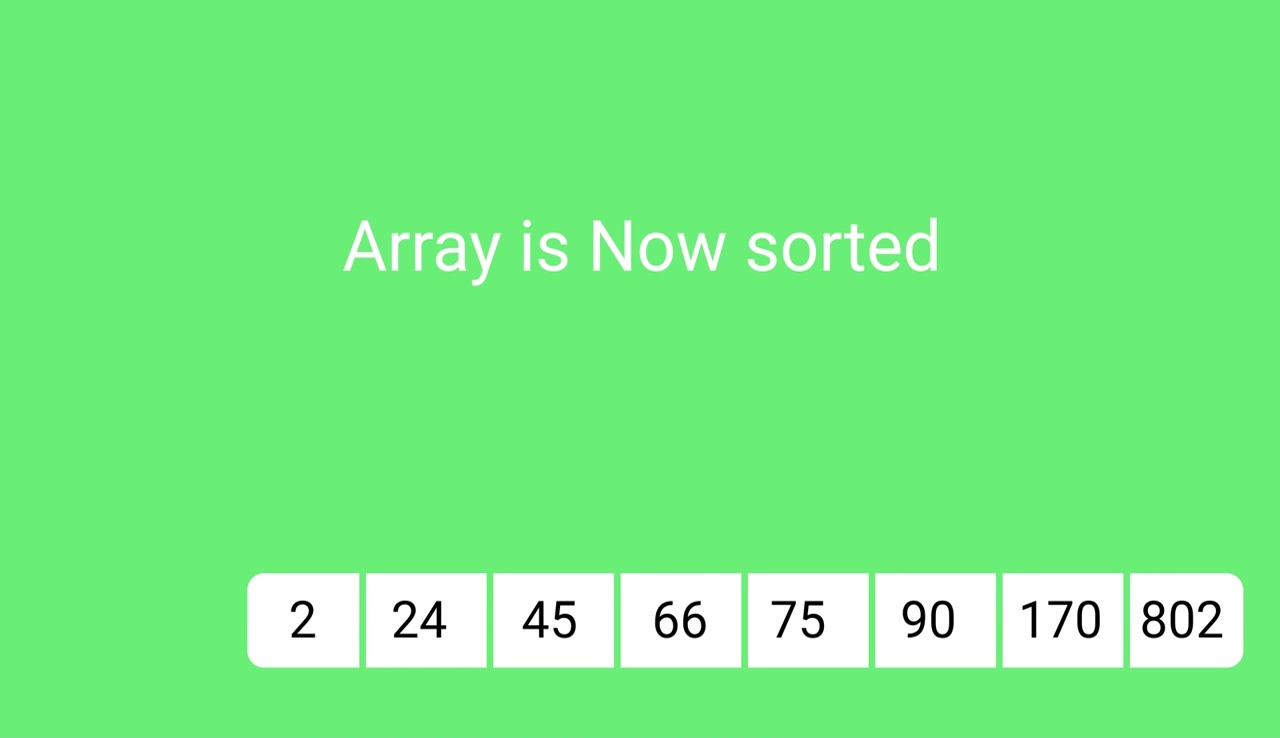


AL REVISAR LAS DECENAS, NOS QUEDA:

(NOTESE QUE EL 2 AL IGUAL QUE EL 802 TIENEN EN LAS DECENAS UN 0)



FINALMENTE, REVISANDO LAS CENTENAS NOS QUEDA



# Ejemplo:



BINSORT:

Dicho método, también conocido por el nombre de bucket sort, consiste en la creación de casilleros o cubetas, de ahí su nombre (bin=casillero y bucket=cubeta), que serán ordenados individualmente y posteriormente reagrupados dejando así un orden completo.

El procedimiento es el siguiente:

Se crea el numero de casilleros igual al numero de elementos que tenga la estructura, siendo estos casilleros comúnmente listas enlazadas o doblemente enlazadas.

A cada casillero entraran números de acuerdo al rango que estos presenten. Este rango se forma tomando el dato mayor del arreglo, el dato menor y restamos ambos, y posteriormente dividimos dicho resultado entre el numero de intervalos que se tienen (el número de elementos en la estructura).

Teniendo lo anterior podemos recorrer la estructura inicial e ir insertando los datos en los casilleros que correspondan y finalizado esto se organizan por un método cualquiera, normalmente el insertion sort, cada uno de los casilleros.

Finalmente, solo resta reunir los casilleros ya arreglados en la estructura inicial.

BucketSort

En el ejemplo anterior solo se muestra el arreglo entrante y los casilleros creados y ya ordenados, para finalizar el proceso debe notarse que solo resta ingresar dichos datos en el arreglo original y ya quedaría arreglado.

# Ejemplo:



# Datos importantes:

* El método Radixsort es notablemente más rápido puesto que las pasadas que debe de hacer se limitarán solo a el número de dígitos que tenga el mayor número de la lista.
* El método del bucketSort se suele usar cuando el rango es mucho mayor que la longitud de la lista, si no es así el ordenamiento sigue siendo eficiente pero no tanto como lo planteado anteriormente.
* Es importante saber que los dos métodos enunciados anteriormente tienen un comportamiento lineal, es decir que su complejidad es O(n) inclusive teniendo el insertion sort, cuya complejidad es de , puesto que al trabajar con sub-arreglos se espera que en una tasa muy alta (casi del 99.5% de las veces) los sub-arreglos sean de a lo sumo 5 posiciones.
* En el bucket sort también se presta para una variante la cual solo toma los elementos que le corresponden a cada casillero, y finalizada la asignación se retorna los elementos a la lista, lo que la tendrá parcialmente ordenada y al aplicar el insertion sort también se ordenaría la estructura de manera rápida y eficaz.

Fuentes:

* <https://youtu.be/pJ1IQD5rv4o>
* <https://www.geeksforgeeks.org/radix-sort/>
* <https://www.geeksforgeeks.org/bucket-sort/>