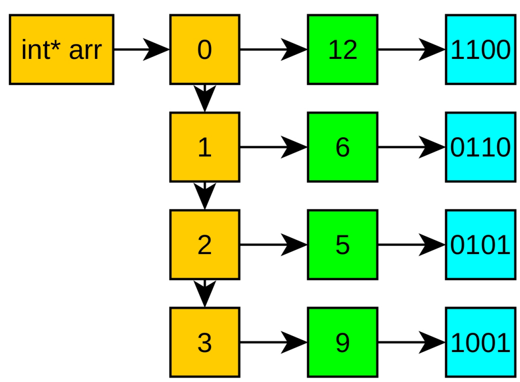
Radix Sort 2020

Name: Emilio Tonix Gleason

Due 26/10/2020

1. We begin from a set of numbers given in an array of size N=4

Lets assign RANDOM numbers in index from 0 to 3, this numbers are 4 bits values.
For 4 bits range is from 0 to 15, (0'b0000-0'b1111)



2. By the given formula, we could iterate between each digit of a number.
Where "i" is the index in a for loop, and N the number we are digiting.
For base 10

$$\frac{N\,mod\,10^{i+i}}{10^{i}} = digit$$

We modify formula for base 2 formula.

$$\frac{N\,mod\,2^{i+i}}{2^{i}} = digit$$

To avoid using powers and mods, which are time expensive in CPU, we could do
a simplifcation with bit shifting, and the algorith will run faster.

$$\frac{N\,mod\,2^{i+i}}{2^{i}} = \frac{N\,and\,2^{i}}{2^{i}} = \frac{N\,and\,(1<<i)}{2^{i}} = (N\,and\,(1 << i)) >> i$$

$$(N\, and\, (1 << i)) >> i = digit$$

To make a short proof, you could open a python terminal, and do fast calculations
to verify it's true.

3. The main idea it will be to iterate numbers like we are scanning row and columns.

Row = arr[i] Value
Column = Digit[i]

arr[i]	dec	b3	b2	b1	b0
arr[0]	12	1	1	0	0
arr[1]	6	0	1	1	0
arr[2]	5	0	1	0	1
arr[3]	9	1	0	0	1

Given that idea we will do the following steps.

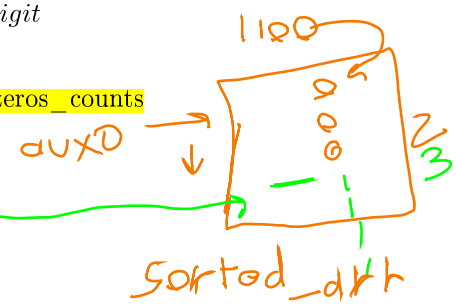
a. Create a copy of the table, and lets called it sorted_arr
b. Scan the b_n column with digit formula and count how many 0s are.

$$(N\, and\, (1 << i)) >> i = digit$$

c. Save this value in a counter. "zeros_counts"
d. Add two new counters aux0=0 and aux1 = zeros_counts
Scan again the b_n column.

if b_n is 0 , sorted_arr[aux0] = arr[i]
Increment aux0

if b_n is 1 , sorted_arr[aux1] = arr[i]
Increment aux1



e. Once scanned you shall copy sorted_arr to arr,
for this task you may use memcpy, this API is on the standar lib in c,
and it will work with #include <string.h>
f. Do again step b and until all digits have been traversed.

4. **Example**:
We may thing in 2 “for” nested loops,
one for the arr[i] and other with the bit size.

NOTE:Until now don’t care about size of bits in value,
to avoid caluculate that, we insert only integers that its space is 4 bits.

i=0

For each shift we will cout all 0s first “zeros_counts”.
Then set aux0=0 and aux1 = zeros_counts

arr[i]	dec	b3	b2	b1	b0
arr[0]	12	1	1	0	0
arr[1]	6	0	1	1	0
arr[2]	5	0	1	0	1
arr[3]	9	1	0	0	1

zeros_counts = 2
aux0=0
aux1 = zeros_counts
sorted_arr[**0**] = arr[0]→0→12
sorted_arr[**1**] = arr[1]→0→6
sorted_arr[**2**] = arr[2]→1→5
sorted_arr[**3**] = arr[3]→1→9
This step when you copy sorted_arr to arr, it reamins the same
because all bits in the col b0 by coincidence had been sorted.

i=1

arr[i]	dec	b3	b2	b1	b0
arr[0]	12	1	1	0	0
arr[1]	6	0	1	1	0
arr[2]	5	0	1	0	1
arr[3]	9	1	0	0	1

zeros_counts = 3
aux0=0
aux1 = zeros_counts
sorted_arr[**0**] = arr[0]→0→12
sorted_arr[**3**] = arr[1]→1→6
sorted_arr[**1**] = arr[2]→0→5
sorted_arr[**2**] = arr[3]→0→9
Next iteration arr will be diferent,
due to the copy from sorted_arr to arr

Sor

i=2

arr[i]	dec	b3	b2	b1	b0
arr[0]	12	1	1	0	0
arr[1]	5	0	1	0	1
arr[2]	9	1	0	0	1
arr[3]	6	0	1	1	0

zeros_counts = 1
aux0=0
aux1 = zeros_counts
sorted_arr[**1**] = arr[0]→1→12
sorted_arr[**2**] = arr[1]→1→5
sorted_arr[**0**] = arr[2]→0→9
sorted_arr[**3**] = arr[3]→1→6

i=3

arr[i]	dec	b3	b2	b1	b0
arr[0]	9	1	0	0	1
arr[1]	12	1	1	0	0
arr[2]	5	0	1	0	1
arr[3]	6	0	1	1	0

zeros_counts = 2
aux0=0
aux1 = zeros_counts
sorted_arr[**2**] = arr[0]→1→9 5
sorted_arr[**3**] = arr[1]→1→12 6
sorted_arr[**0**] = arr[2]→0→5 9
sorted_arr[**1**] = arr[3]→1→6 12

arr[i]	dec	b3	b2	b1	b0
arr[0]	5	1	0	0	1
arr[1]	6	0	1	1	0
arr[2]	9	1	0	0	1
arr[3]	12	1	1	0	0