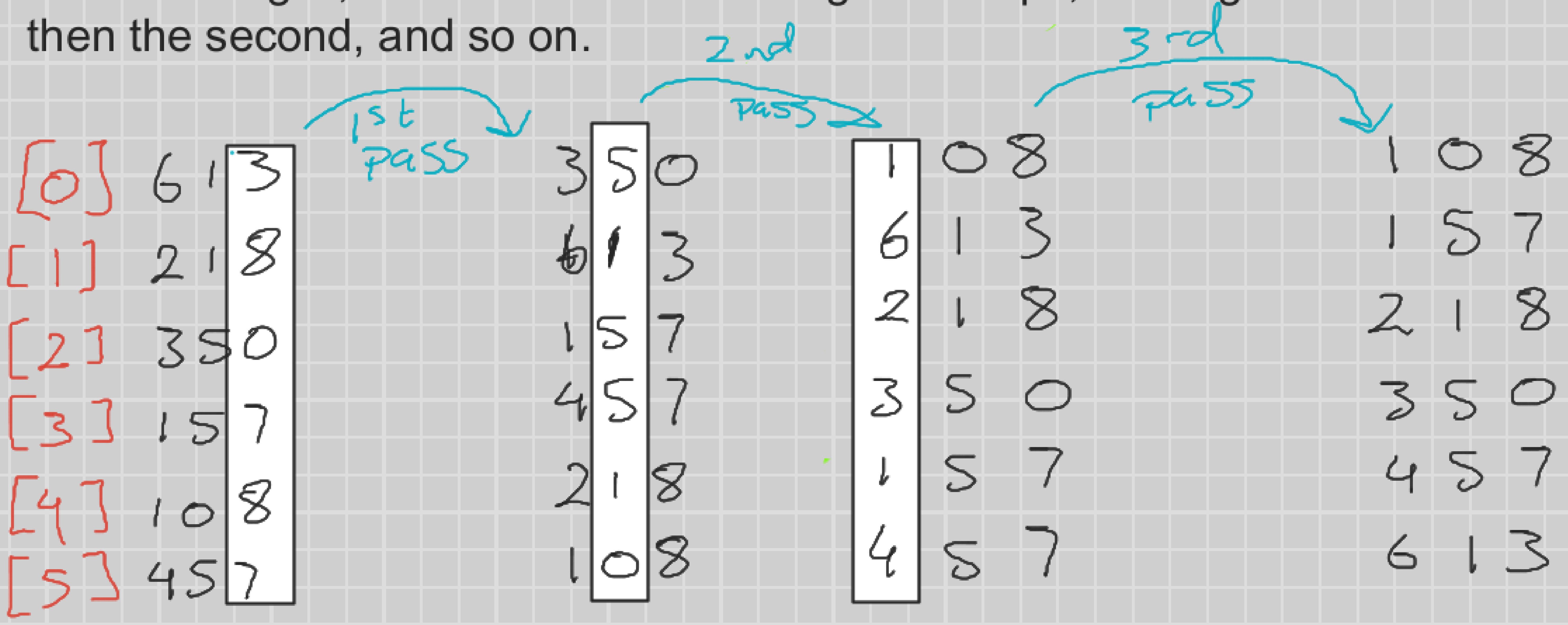
Radix Sort using counting sort, running time θ(d*(N+k))

Radix is a non-comparison sort with a linear worst case running time θ(d*(N+k)).

- It uses a stable* sort, such as counting sort. It works by splitting numbers into their individual digits, and then sorts those digits in steps, starting with the least siginificant digit, then the second, and so on.



We make as many passes as there are digits in the largest number.

*A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted. The "key" is each digit being used to sort.

If you use counting sort to sort the digits the time complexity of Radix sort is given by d times the complexity of counting sort which is equal to (N + k). Where d is the number of digits, N is the length of the array to be sorted, and k is the maximum value of any digit.

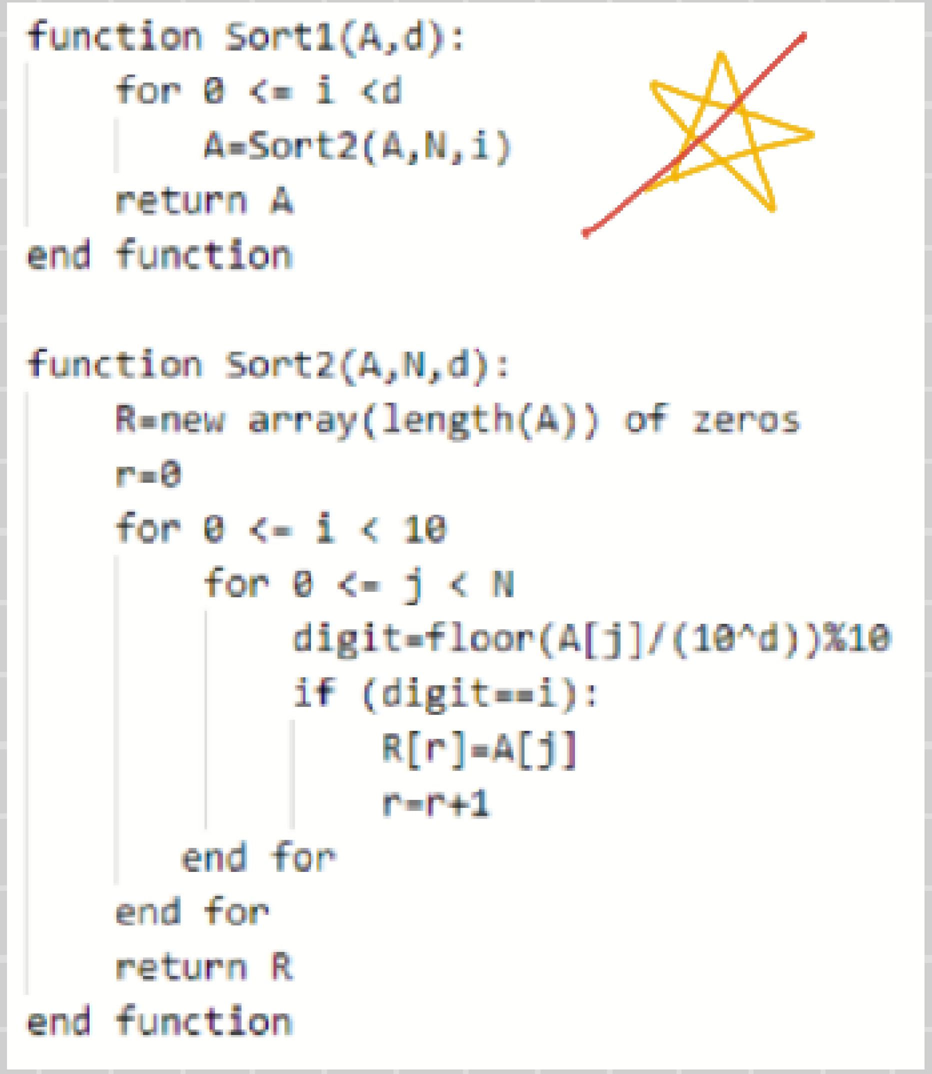
As d is a constant number and it's only recommended to use counting sort when k is in the order of N then we can make Radix sort run a linear time with N.

The time complexity of Radix sort is given by: θ(d*(N+k)) where:
- d: # of digits
- N: is the length of the array
- k: maximum value of digits (buckets = 10, 0-9), based on chosen base.

θ(N+k): the time complexity of counting sort. This run time depends on the length of the largest number.

We can can use other stable sorting algos in Radix sort as subroutines, which means that Radix sort would have a time complexity of θ(d*(f(n)), where f(n) is the time complexity of the sorting algo subroutine you have chosen.

- We can modify counting sort to sort decimal numbers as well within Radix sort.



```
function Sort1(A,d):
    for 0 <= i <d
        A=Sort2(A,N,i)
    return A
end function

function Sort2(A,N,d):
    R=new array(length(A)) of zeros
    r=0
    for 0 <= i < 10
        for 0 <= j < N
            digit=floor(A[j]/(10^d))%10
            if (digit==i):
                R[r]=A[j]
                r=r+1
       end for
    end for
    return R
end function
```

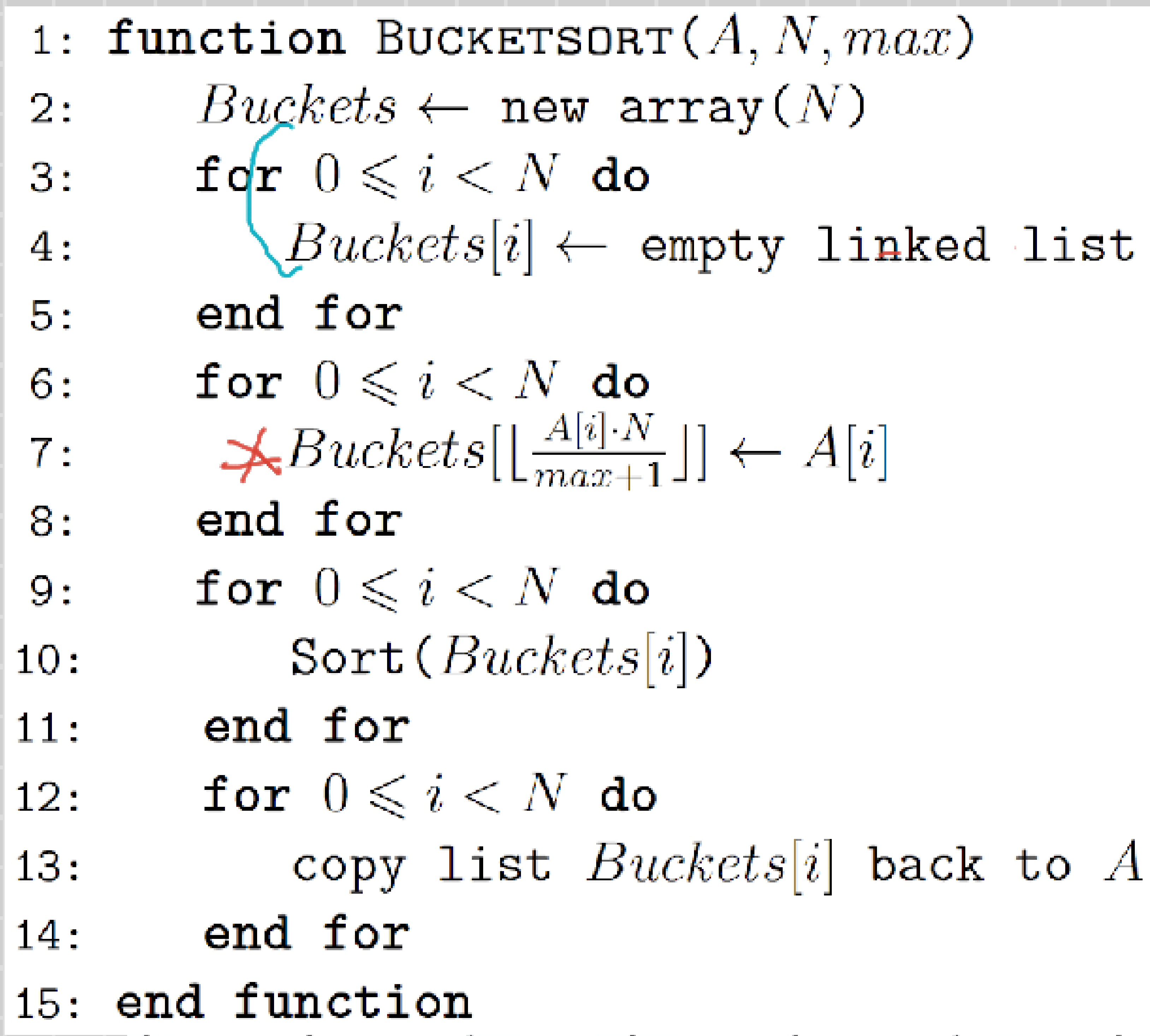
BucketSort

In our lessons is a non-comparison sort.
- We can think of this sorting algo like sorting coins into stacks of the same denomination, pick a coin and stack it according to its value.
- After stacking the coins, we will have N stacks of coins, sorted.

-

$*$

Buckets[floor((A[i]*N)/max + 1) - 1] <-- A[i]



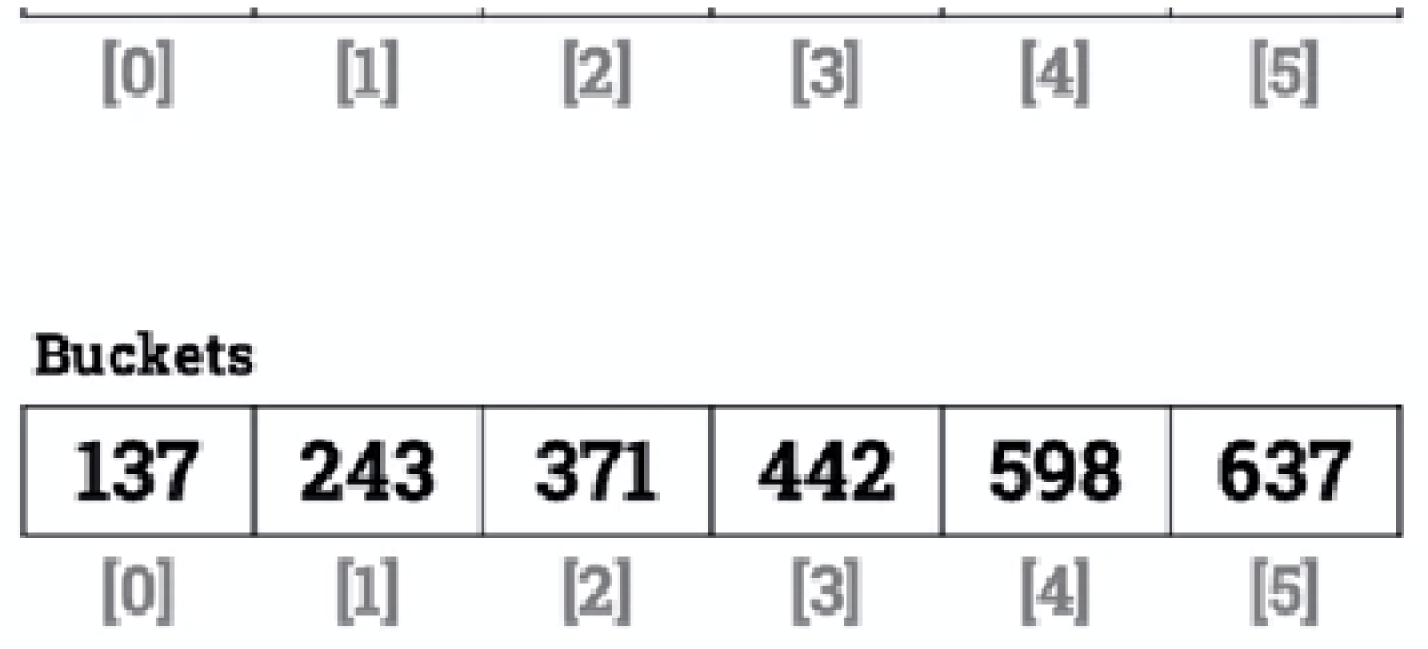
```
function BUCKETSORT($A, N, max$)
    $Buckets \leftarrow$ new array($N$)
    for $0 \leqslant i < N$ do
        $Buckets[i] \leftarrow$ empty linked list
    end for
    for $0 \leqslant i < N$ do
        $* Buckets[\lfloor \frac{A[i] \cdot N}{max+1} \rfloor] \leftarrow A[i]$
    end for
    for $0 \leqslant i < N$ do
        Sort($Buckets[i]$)
    end for
    for $0 \leqslant i < N$ do
        copy list $Buckets[i]$ back to $A$
    end for
end function
```

$$B\left[\left\lfloor \frac{A[0] \cdot 6}{637+1} \right\rfloor - 1\right] \leftarrow A[0]$$

$$B[1] \leftarrow 243$$

$$B\left[\left\lfloor \frac{A[1] \cdot 6}{638} \right\rfloor - 1\right] \leftarrow A[1]$$

$$B[\cancel{5}] \leftarrow 637$$



[0]	[1]	[2]	[3]	[4]	[5]

Buckets

137	243	371	442	598	637
[0]	[1]	[2]	[3]	[4]	[5]

```
A: array to be sorted
N: size of array A
max: maximum value in A
```

```
function Bucketsort(A,N,max)
   Buckets ← new array(N)
   for 0 ≤ i < N
      Buckets[i] ← empty linked list
   for 0 ≤ i < N
      Buckets[floor(A[i]*N/(max+1))] ← A[i]
   for 0 ≤ i < N
      sort(B[i])
   for 0 ≤ i < N
      copy list B[i] back to A
end function
```

243	637	371	~~598~~	~~442~~	~~137~~
[0]	[1]	[2]	[3]	[4]	[5]

Buckets

137	243	371	442	598	637
[0]	[1]	[2]	[3]	[4]	[5]