Design and Analysis of Algorithm

Linear Time Sorting
(Radix Sort and Bucket Sort)

Lecture -19

Linear Time Sorting
(Radix Sort)

Overview

- Running time of counting sort is $\Theta(d(n+k))$
- Required extra space for sorting.
- Is a stable sorting.

Radix Sort

- Radix sort is non comparative sorting method
- Two classifications of radix sorts are least significant digit (LSD) radix sorts and most significant digit (MSD) radix sorts.
- LSD radix sorts process the integer representations starting from the least digit and move towards the most significant digit. MSD radix sorts work the other way around.

Radix Sort (Algorithm)

```
Radix_Sort(A,d)
$for\ i \leftarrow d\ down\ to\ 1$
    $Use\ a\ stable\ sort\ to\ sort\ the\ array\ A\ on\ digit\ i$
    $(i.e. Counting\ Sort)$
```

Radix Sort

- In input array A, each element is a number of d digit.

```
$\boldsymbol{Radix\_Sort}(\boldsymbol{A, d})$
    $for\ i \leftarrow 1\ to\ d$
        $do$ "use a stable sort to sort array A on digit i;
```

329

457

657

839

436

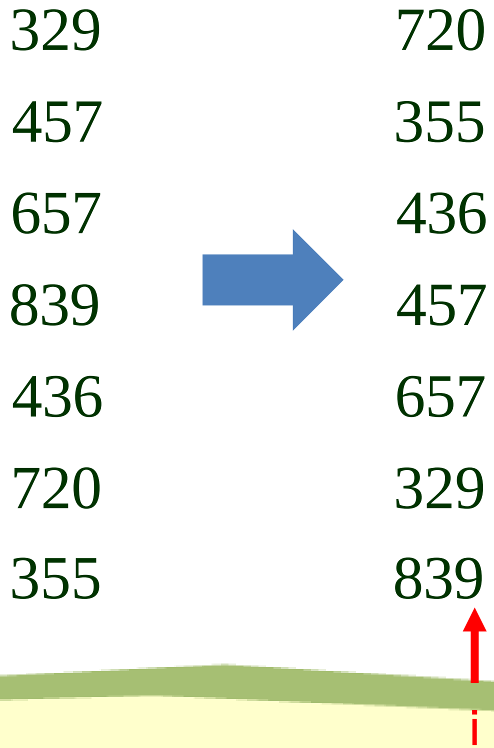
720

355

Radix Sort

- In input array A, each element is a number of d digit.

```
Radix_Sort(A, d)
    for i ← 1 to d
        do "use a stable sort to sort array A on digit i;
```

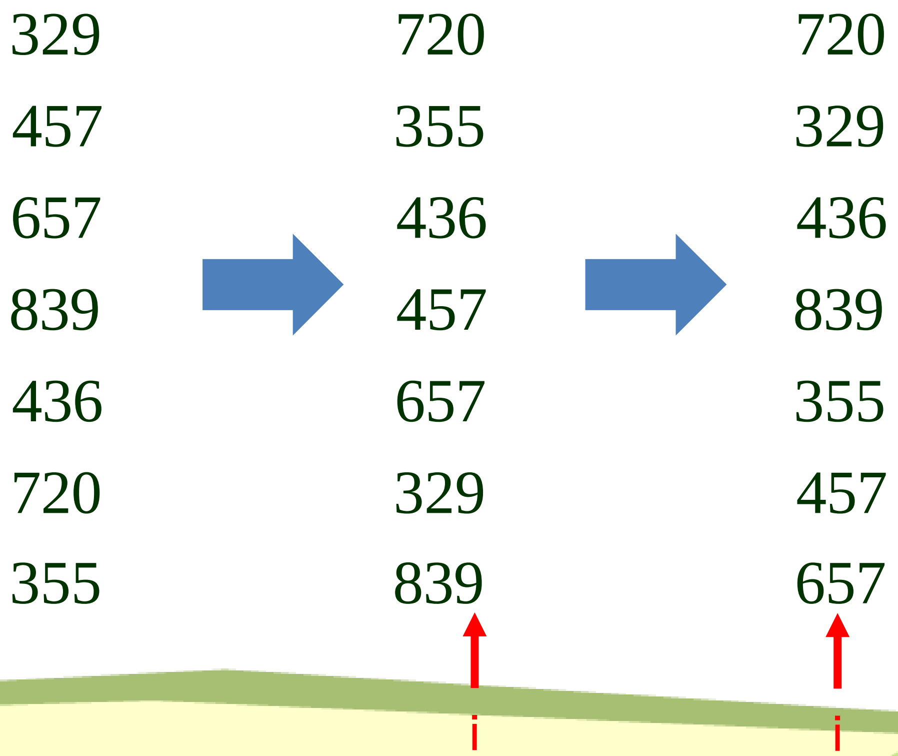


329	720
457	355
657	436
839	457
436	657
720	329
355	839

Radix Sort

- In input array A, each element is a number of d digit.

```
$\boldsymbol{Radix\_Sort}(\boldsymbol{A, d})$
    $for\ i \leftarrow 1\ tod$
        $do$ "use a stable sort to sort array A on digit i;
```

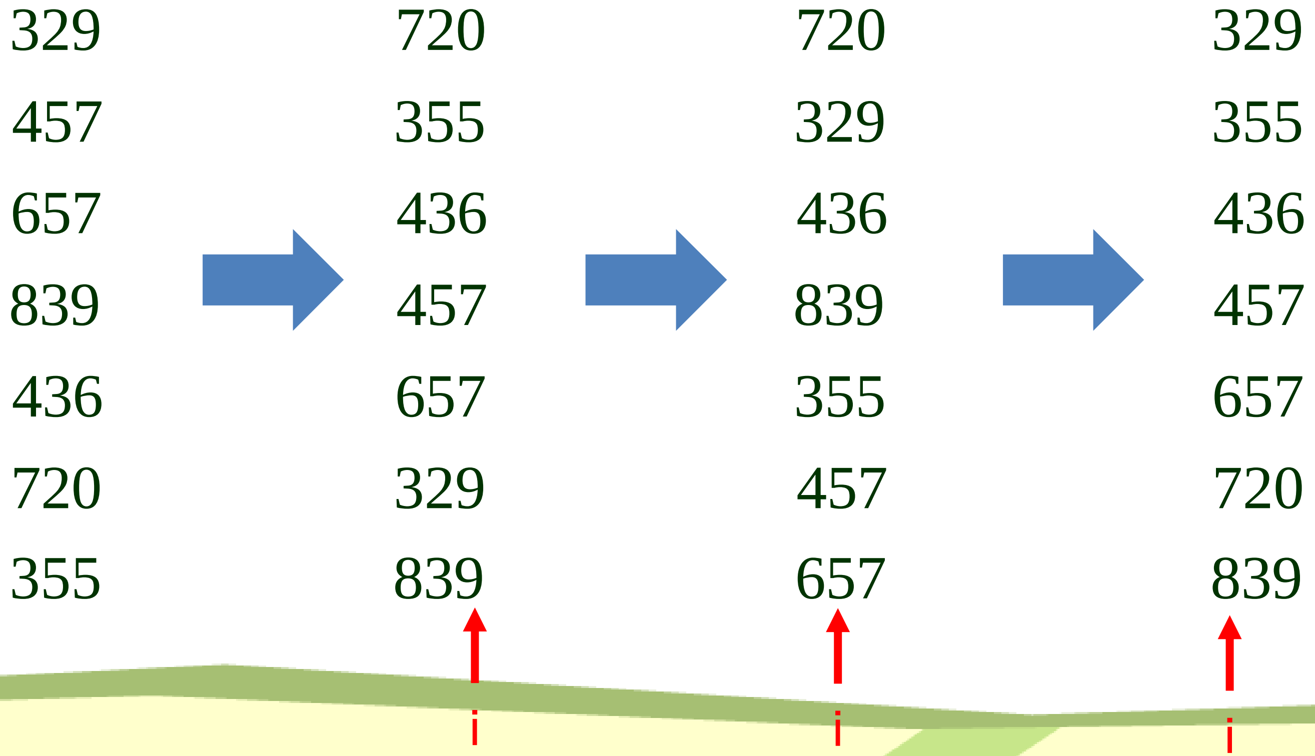


329	720	720
457	355	329
657	436	436
839	457	839
436	657	355
720	329	457
355	839	657

Radix Sort

- In input array A, each element is a number of d digit.

```
$\boldsymbol{Radix\_Sort(A, d)}$
    $for\ i \leftarrow 1\ to\ d$
        $do$ "use a stable sort to sort array A on digit i;
```



329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

Radix Sort (Analysis)

```
Radix_Sort(A,d)
for $i \leftarrow d$ down to 1
    Use a stable sort to sort the array $A$ on digit $i$
    (i.e. Counting Sort)
```

- *Here Counting Sort execute for d times.*
- *The running time of Counting Sort is $\Theta(n+k)$*
- *Hence the running time complexity of Radix Sort is $\Theta(d(n+k))$*

Linear Time Sorting
(Bucket Sort)

Overview

- The average time complexity is $O(n + k)$.
- The worst time complexity is $O(n^2)$.
- Required extra space for sorting.
- Is a stable sorting.

Bucket Sort

- Bucket sort is a comparison sort algorithm that operate on elements by dividing them into different bucket and return the result.
- Buckets are assigned based on each element’s search key.
- A the time of returning the result, First concatenate each bucket one by one and then return the result in a single array.

Bucket Sort

- Some variations
  - Make enough buckets so that each will only hold one element, use a count for duplicates.
  - Use fewer buckets and then sort the contents of each bucket.
- The more buckets you use, the faster the algorithm will run but it uses more memory.

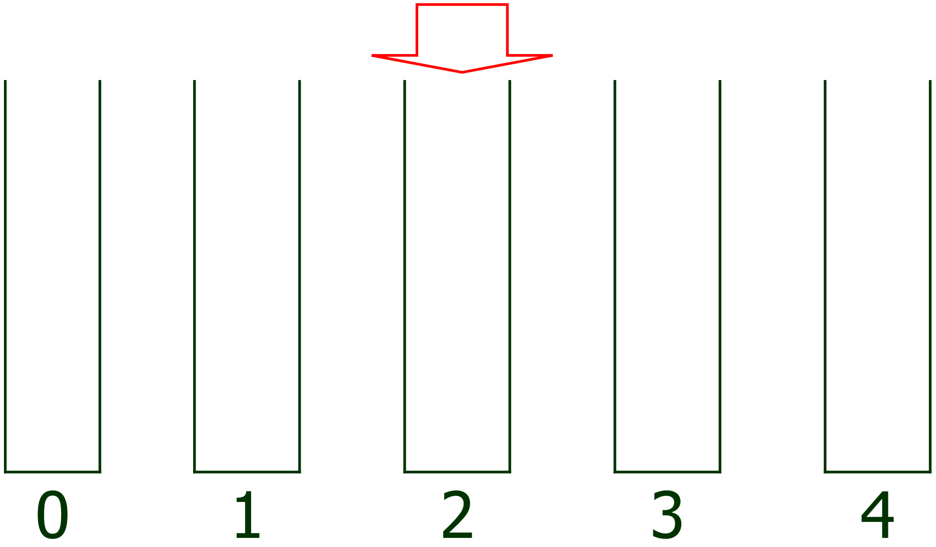
Bucket Sort

- Time complexity is reduced when the number of items per bucket is evenly distributed and it is closed to one item per bucket.
- As buckets require extra space, This algorithm trading increased space consumption for a lower time complexity.
- In general, Bucket Sort beats all other sorting techniques in time complexity but can require a huge of space.

Bucket Sort

- One Value per bucket:

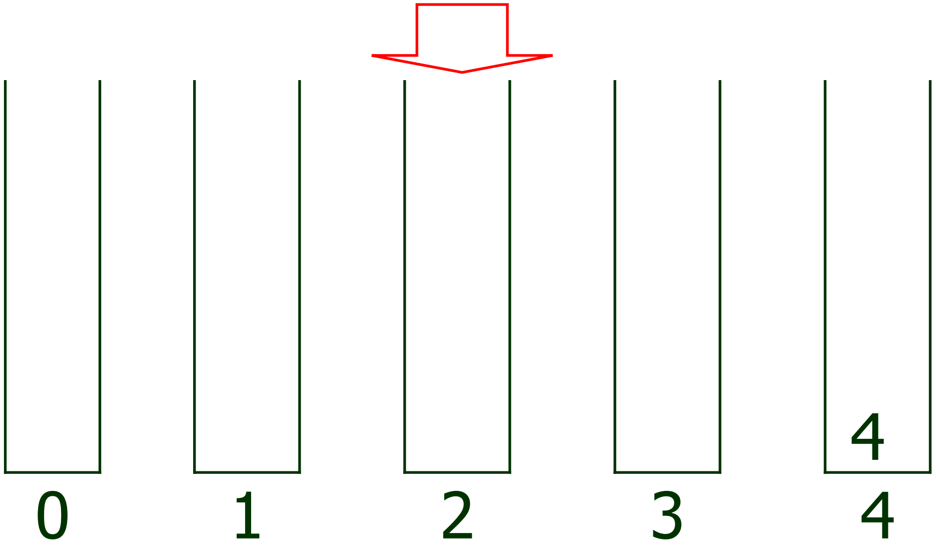
4	2	1	2	0	3	2	1	4	0	2	3	0



Bucket Sort

- One Value per bucket:

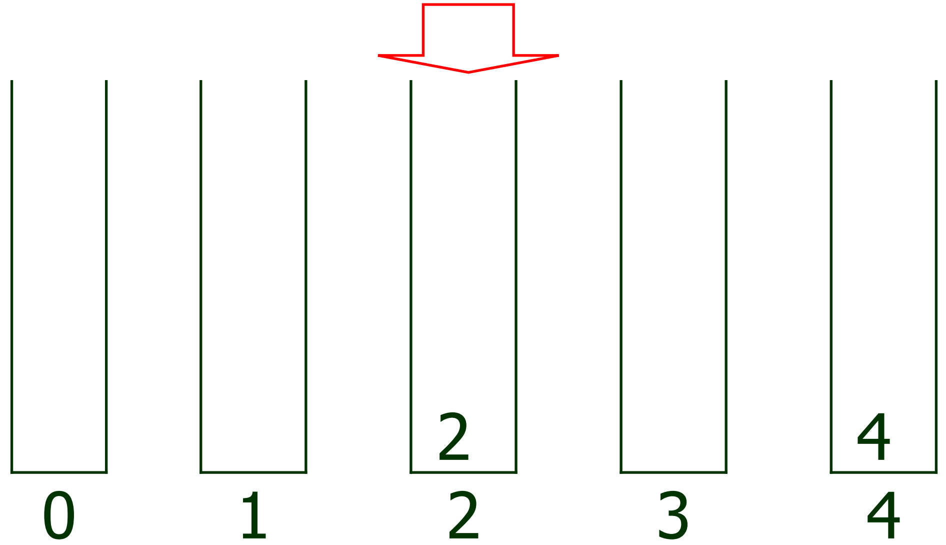
4	2	1	2	0	3	2	1	4	0	2	3	0



Bucket Sort

- One Value per bucket:

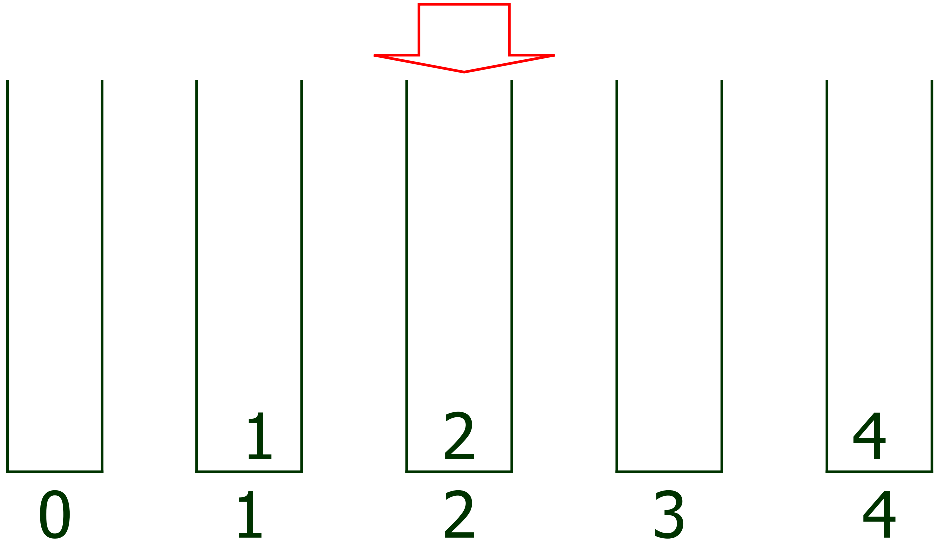
4	2	1	2	0	3	2	1	4	0	2	3	0



Bucket Sort

- One Value per bucket:

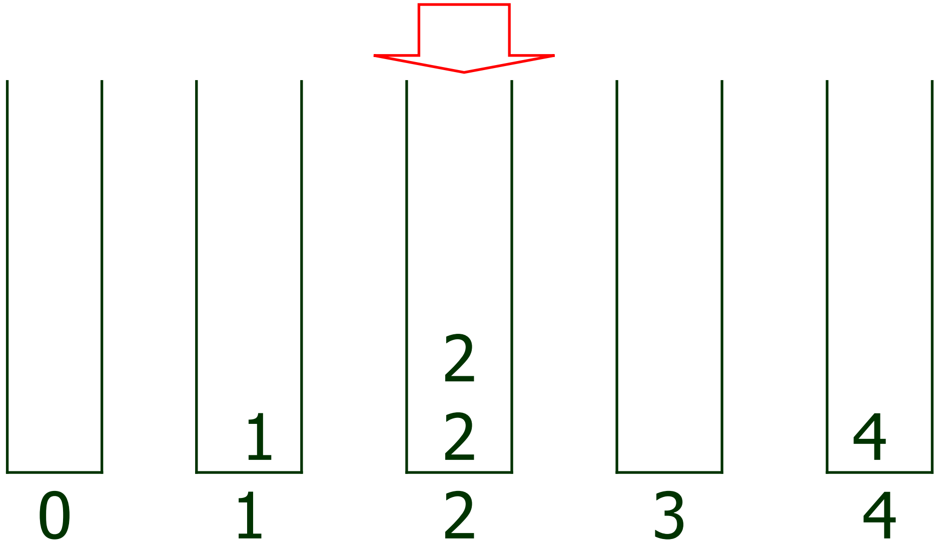
4	2	1	2	0	3	2	1	4	0	2	3	0



Bucket Sort

- One Value per bucket:

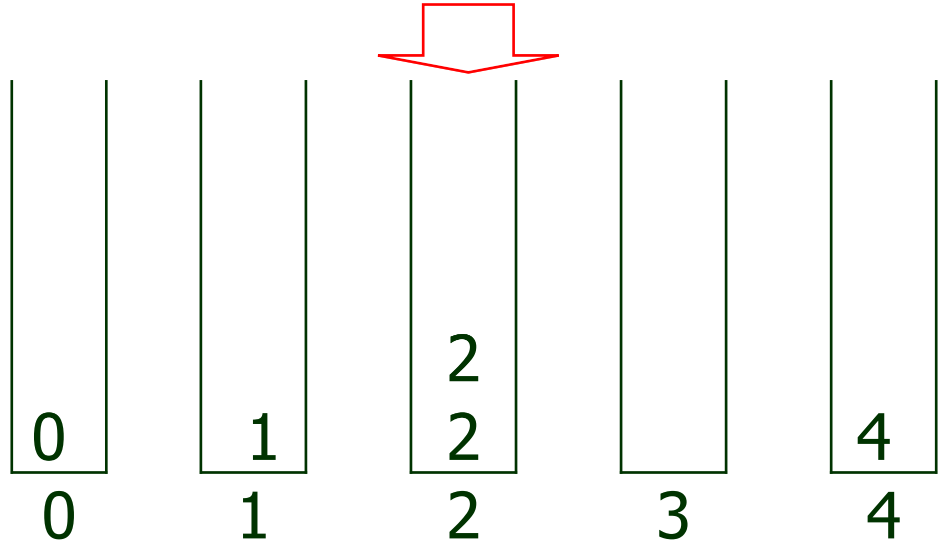
4	2	1	2	0	3	2	1	4	0	2	3	0



Bucket Sort

- One Value per bucket:

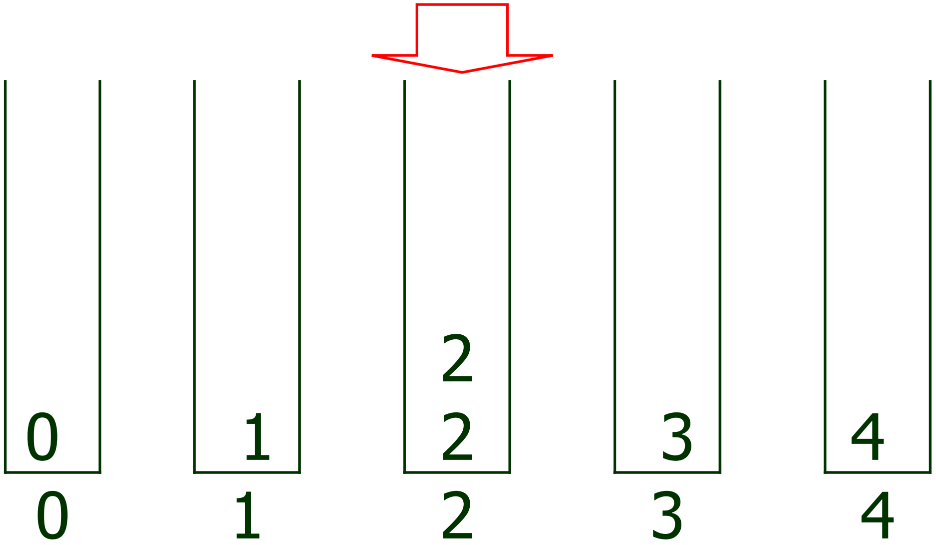
4	2	1	2	0	3	2	1	4	0	2	3	0



Bucket Sort

- One Value per bucket:

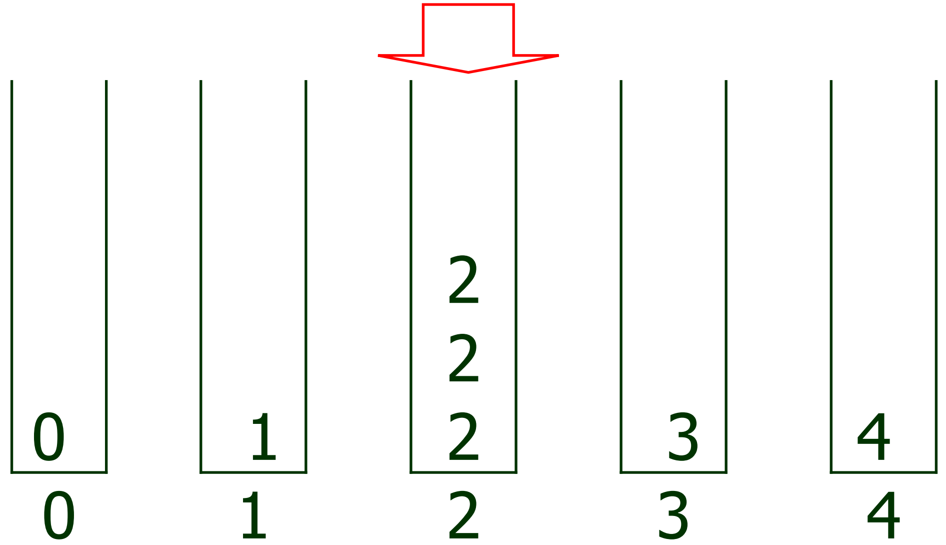
4	2	1	2	0	3	2	1	4	0	2	3	0



Bucket Sort

- One Value per bucket:

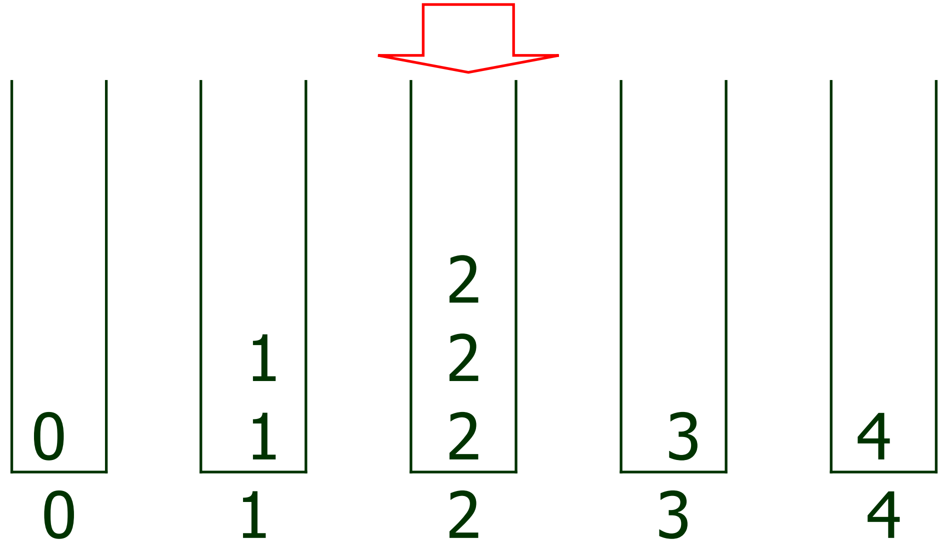
4	2	1	2	0	3	2	1	4	0	2	3	0



Bucket Sort

- One Value per bucket:

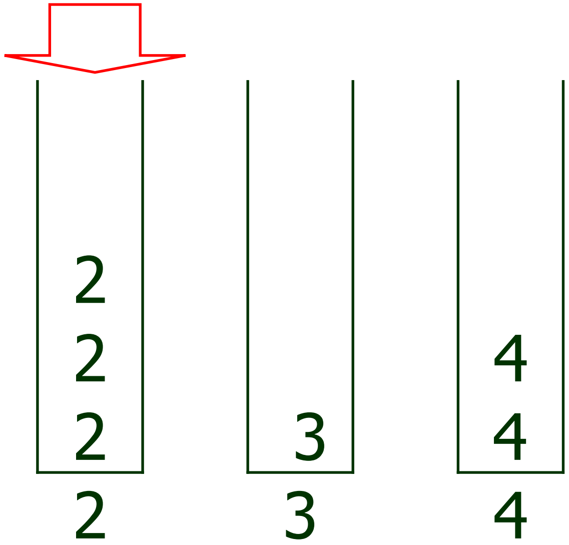
4	2	1	2	0	3	2	1	4	0	2	3	0



Bucket Sort

- One Value per bucket:

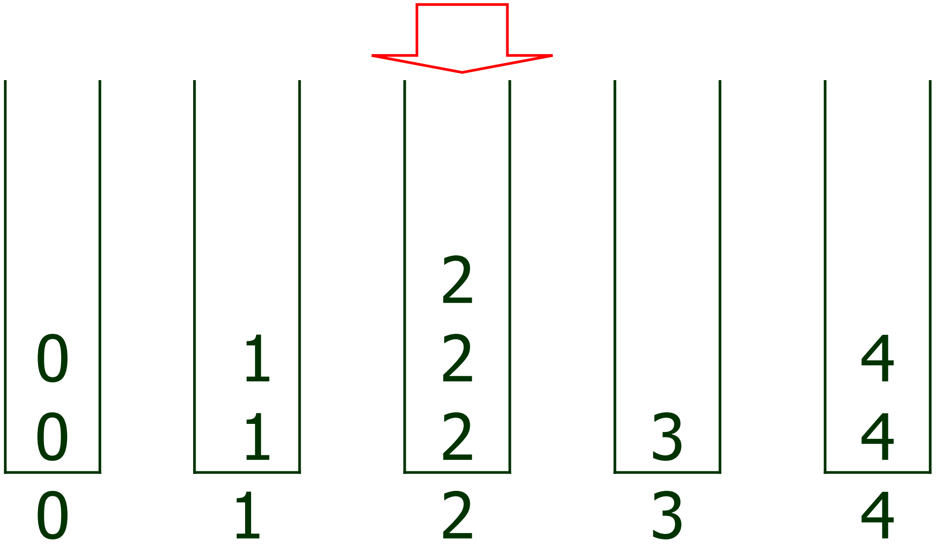
4	2	1	2	0	3	2	1	4	0	2	3	0



Bucket Sort

- One Value per bucket:

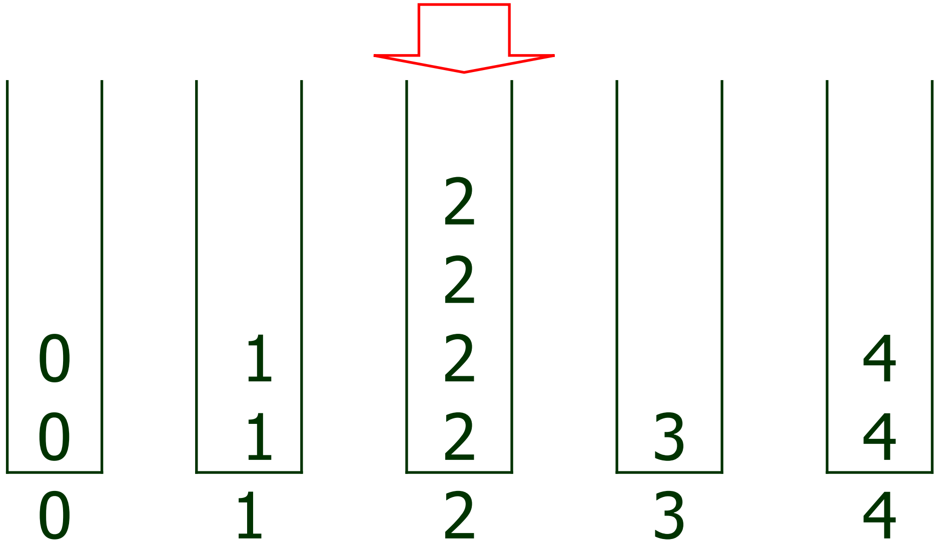
4	2	1	2	0	3	2	1	4	0	2	3	0



Bucket Sort

- One Value per bucket:

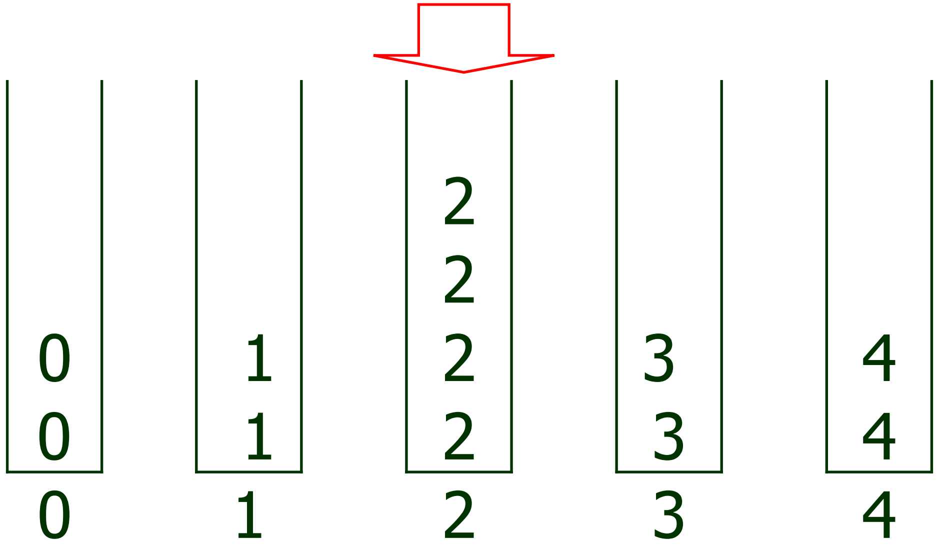
4	2	1	2	0	3	2	1	4	0	2	3	0



Bucket Sort

- One Value per bucket:

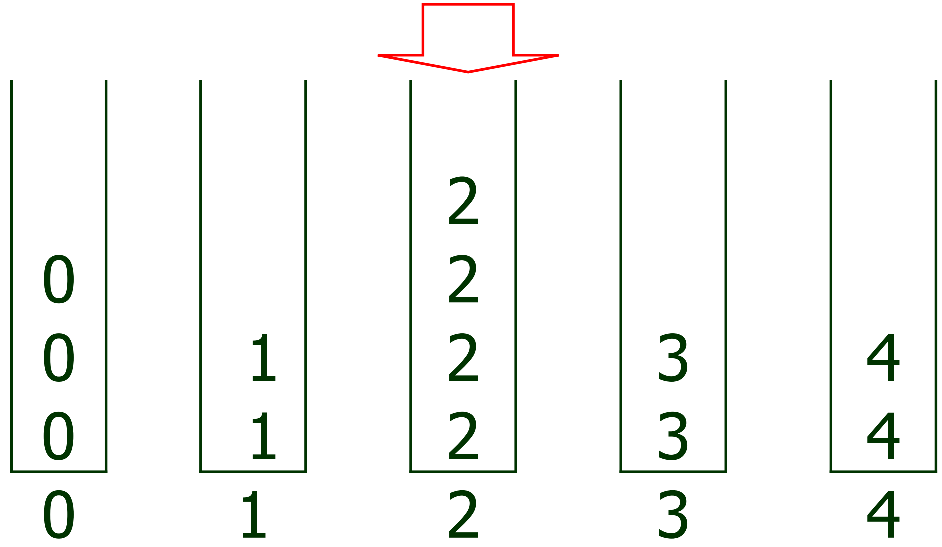
4	2	1	2	0	3	2	1	4	0	2	3	0



Bucket Sort

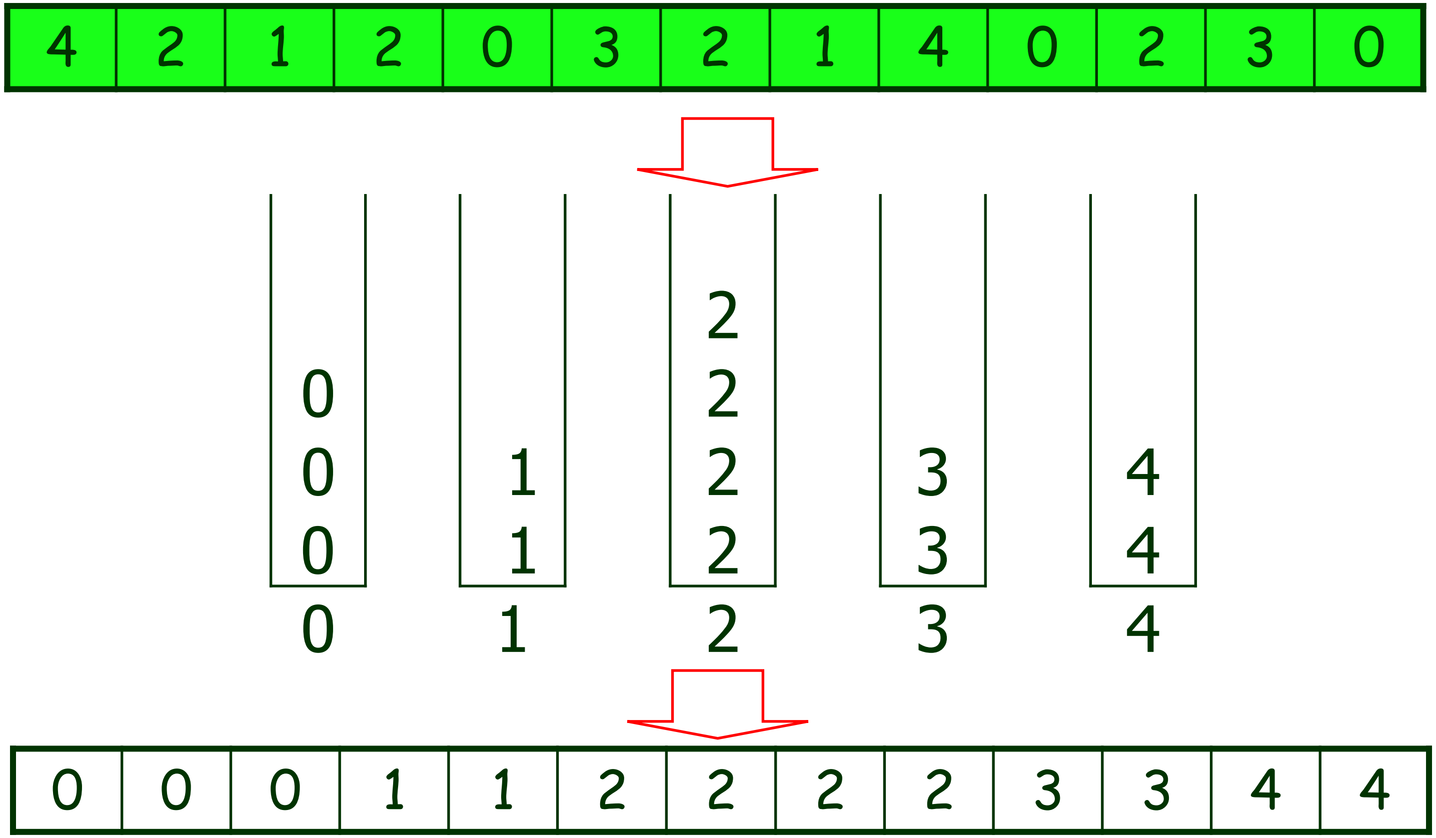
- One Value per bucket:

4	2	1	2	0	3	2	1	4	0	2	3	0



Bucket Sort

- One Value per bucket:



4	2	1	2	0	3	2	1	4	0	2	3	0

0	0	0	1	1	2	2	2	2	3	3	4	4

Bucket Sort

- One Value per bucket:

4	2	1	2	0	3	2	1	4	0	2	3	0

3	2	4	2	2
0	1	2	3	4

0	0	0	1	1	2	2	2	2	3	3	4	4

Bucket Sort

- One Value per bucket:

```
Algorithm BucketSort( S )
( values in S are between 0 and m-1 )
for j ← 0 to m-1 do                // initialize m buckets
    b[j] ← 0
for i ← 0 to n-1 do                // place elements in their
    b[S[i]] ← b[S[i]]  + 1         // appropriate buckets
i ← 0
for j ← 0 to m-1 do                // place elements in buckets
    for r ← 1 to b[j] do           // back in S (Concatination)
        S[i] ← j
        i ← i + 1
```

Bucket Sort

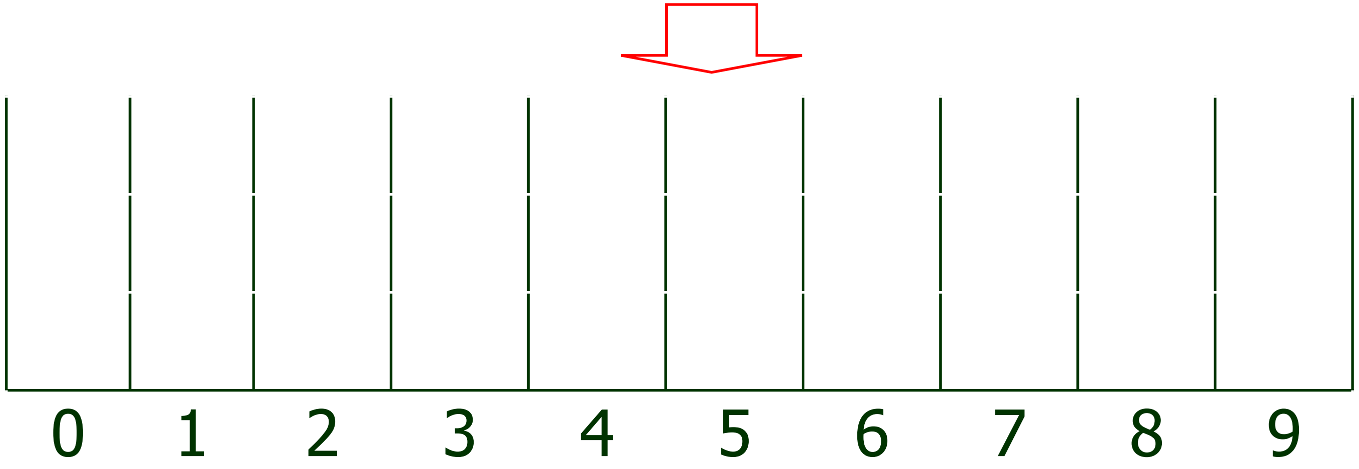
One Value per bucket (Analysis)

- Bucket initialization: $O(m)$
- From array to buckets: $O(n)$
- From buckets to array: $O(n)$
  - Due to the implementation of dequeue.
- Since m will likely be small compared to n, Bucket sort is $O(n)$
- Strictly speaking, time complexity is $O(n + m)$

Bucket Sort

- Multiple items per bucket:

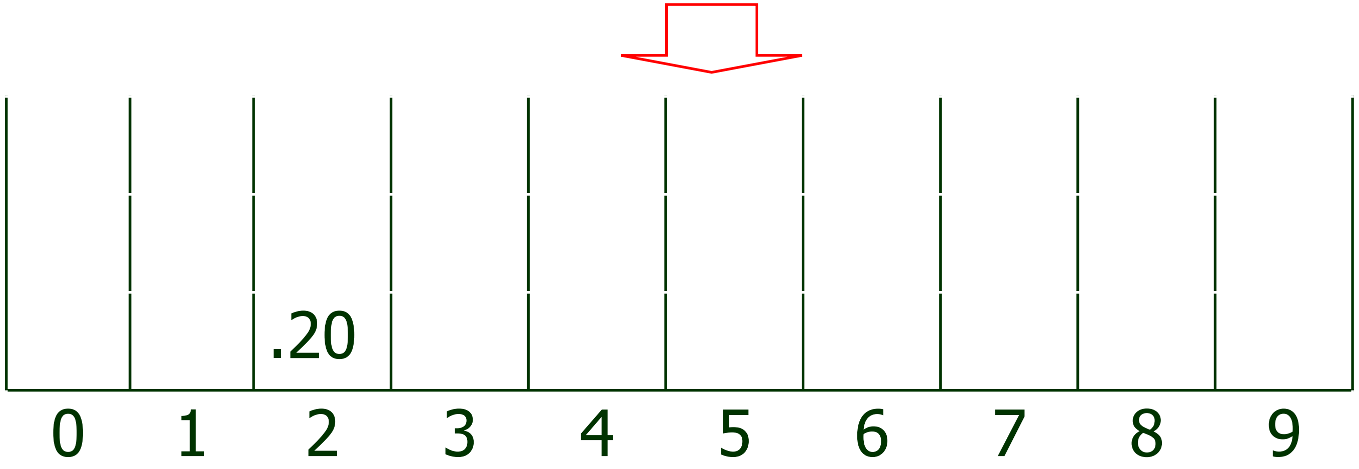
.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99



Bucket Sort

- Multiple items per bucket:

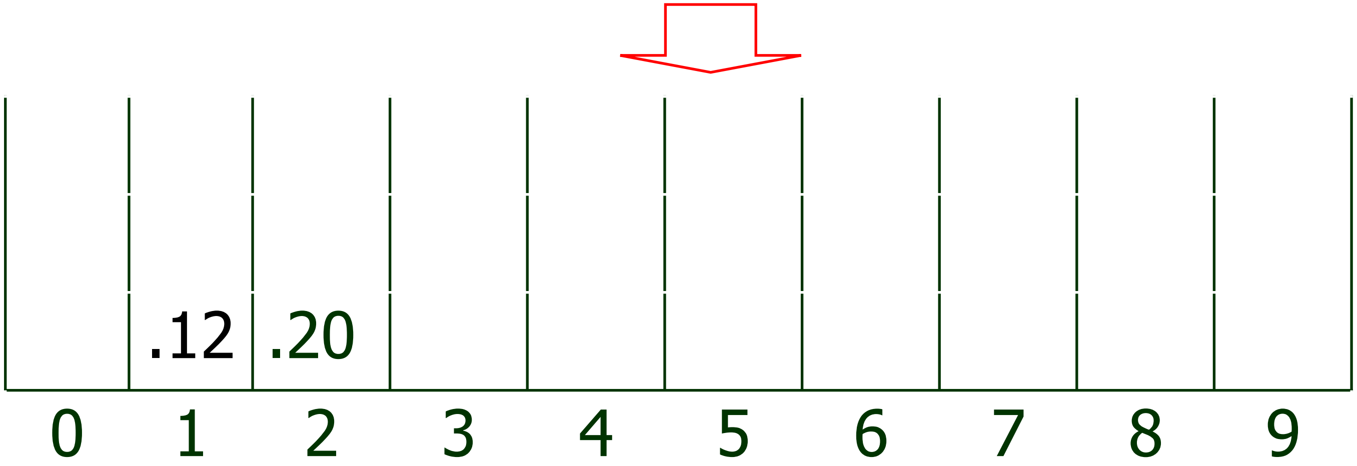
.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99



Bucket Sort

- Multiple items per bucket:

.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99



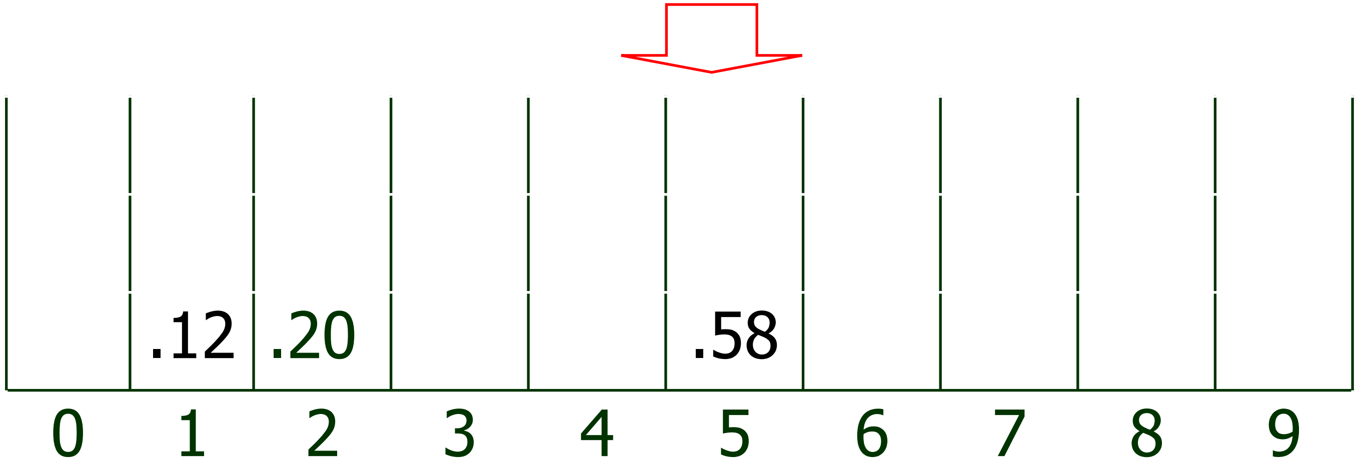
.12 .20

0 1 2 3 4 5 6 7 8 9

Bucket Sort

- Multiple items per bucket:

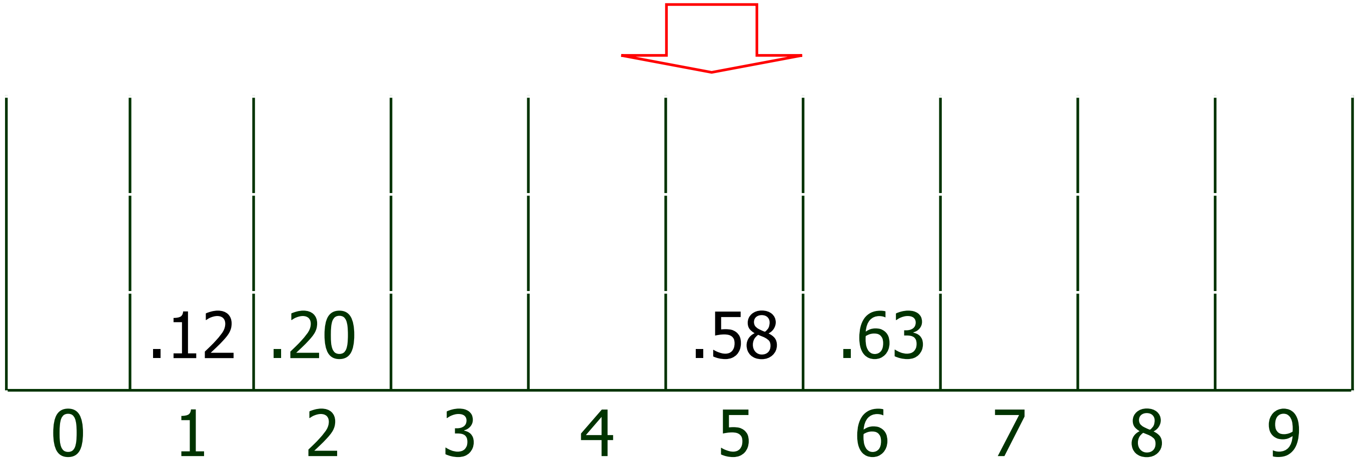
.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99



Bucket Sort

- Multiple items per bucket:

.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99

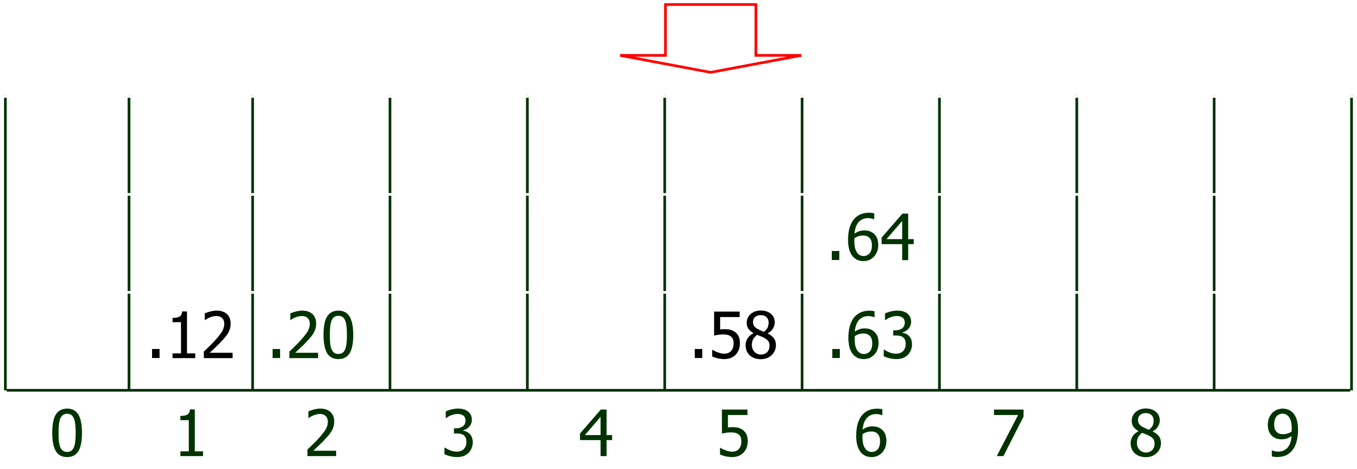


0	1	2	3	4	5	6	7	8	9
	.12	.20			.58	.63			

Bucket Sort

- Multiple items per bucket:

.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99

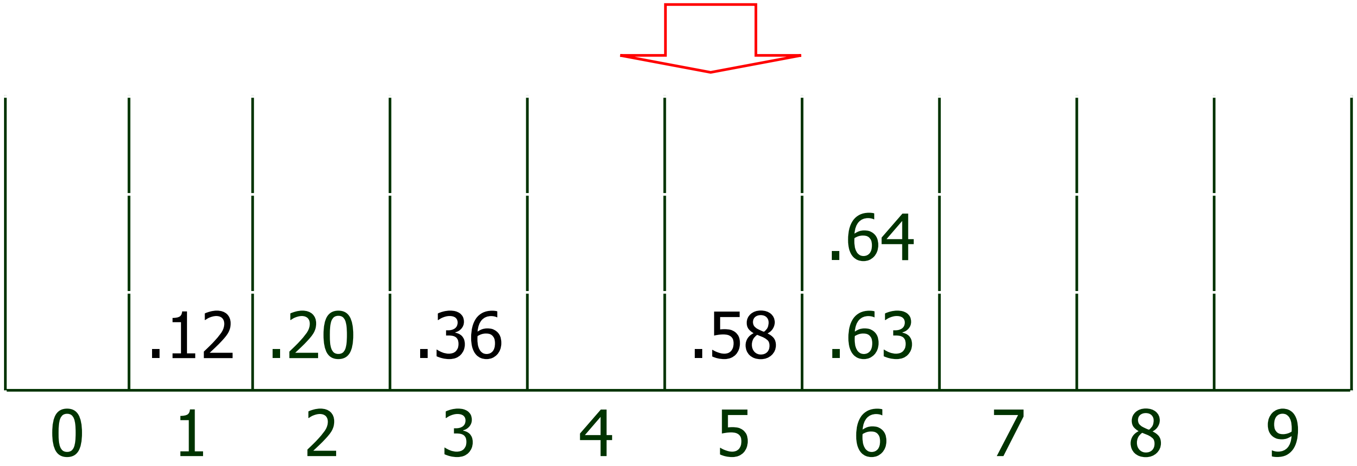


0	1	2	3	4	5	6	7	8	9
	.12	.20			.58	.63 .64			

Bucket Sort

- Multiple items per bucket:

.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99

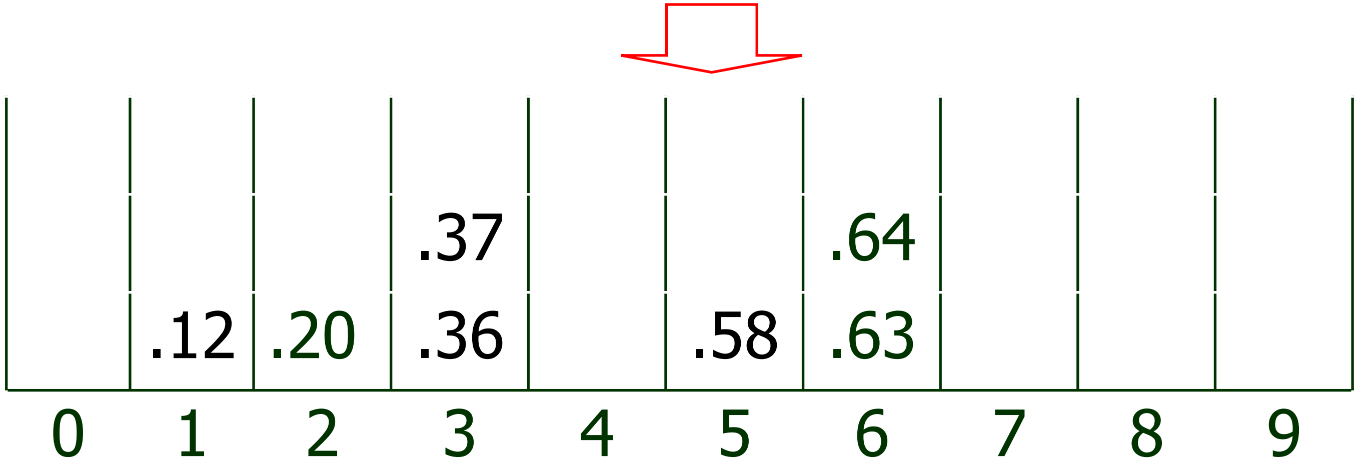


						.64			
	.12	.20	.36		.58	.63			
0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Multiple items per bucket:

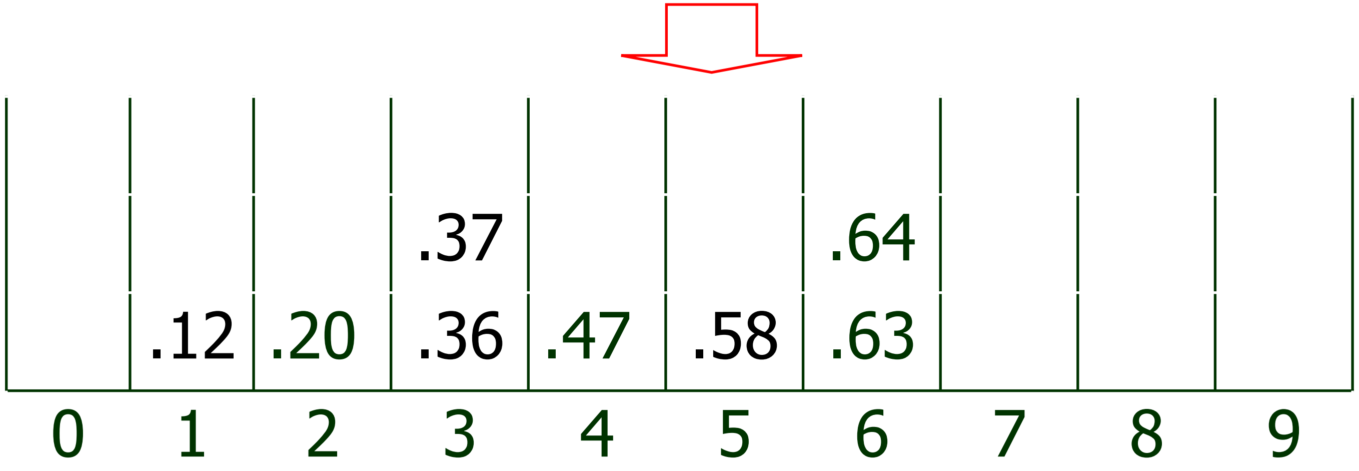
.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99



Bucket Sort

- Multiple items per bucket:

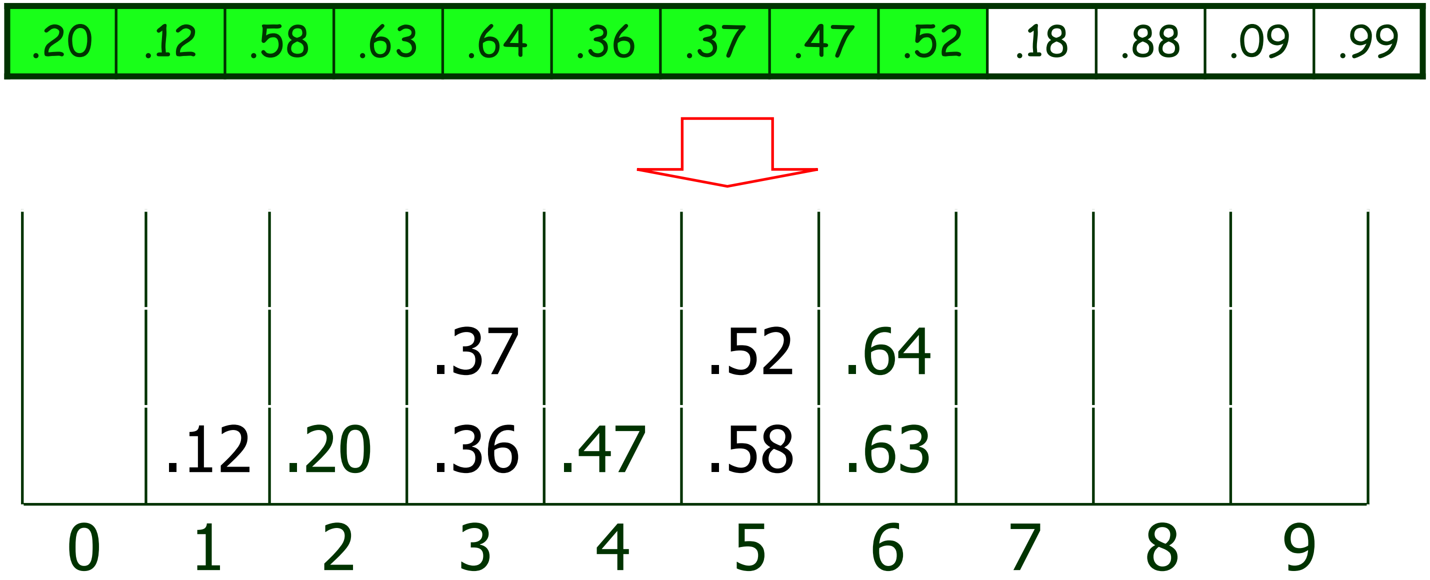
.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99



0	1	2	3	4	5	6	7	8	9
			.37			.64			
	.12	.20	.36	.47	.58	.63			

Bucket Sort

- Multiple items per bucket:



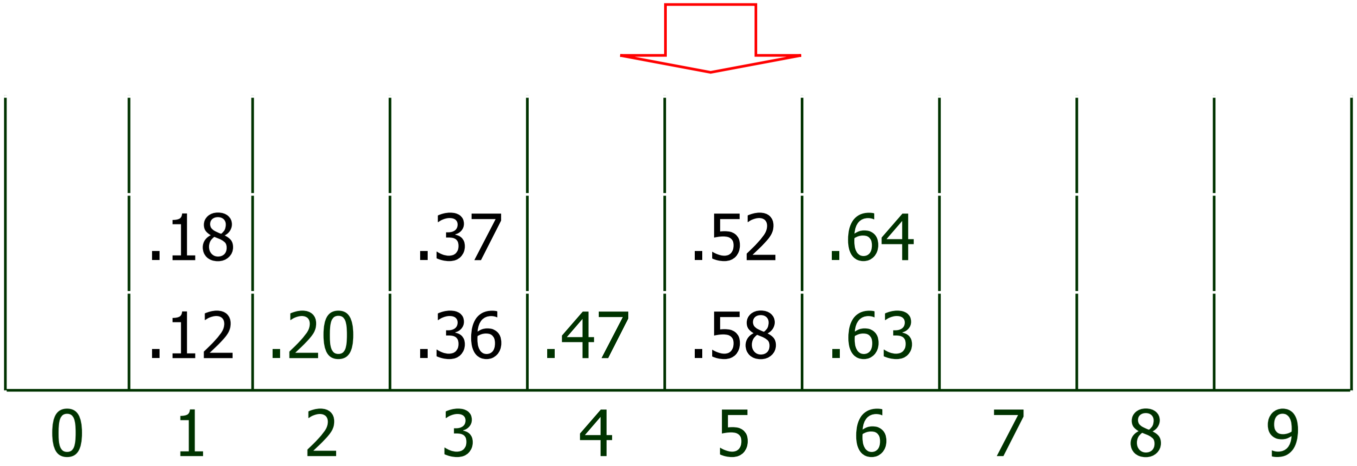
.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99

			.37		.52	.64			
	.12	.20	.36	.47	.58	.63			
0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Multiple items per bucket:

.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99

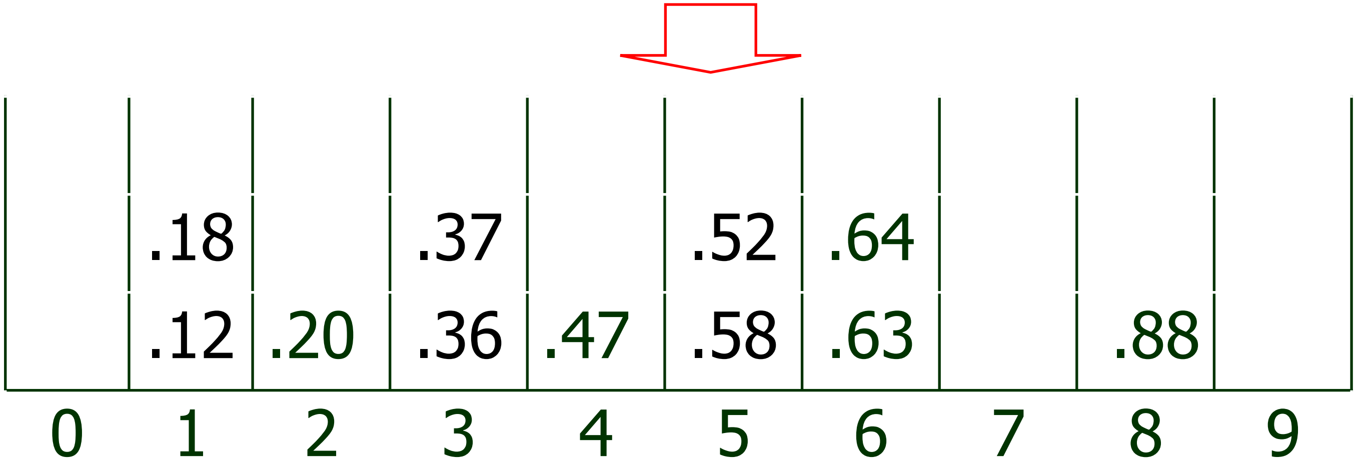


	.18		.37		.52	.64			
	.12	.20	.36	.47	.58	.63			
0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Multiple items per bucket:

.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99

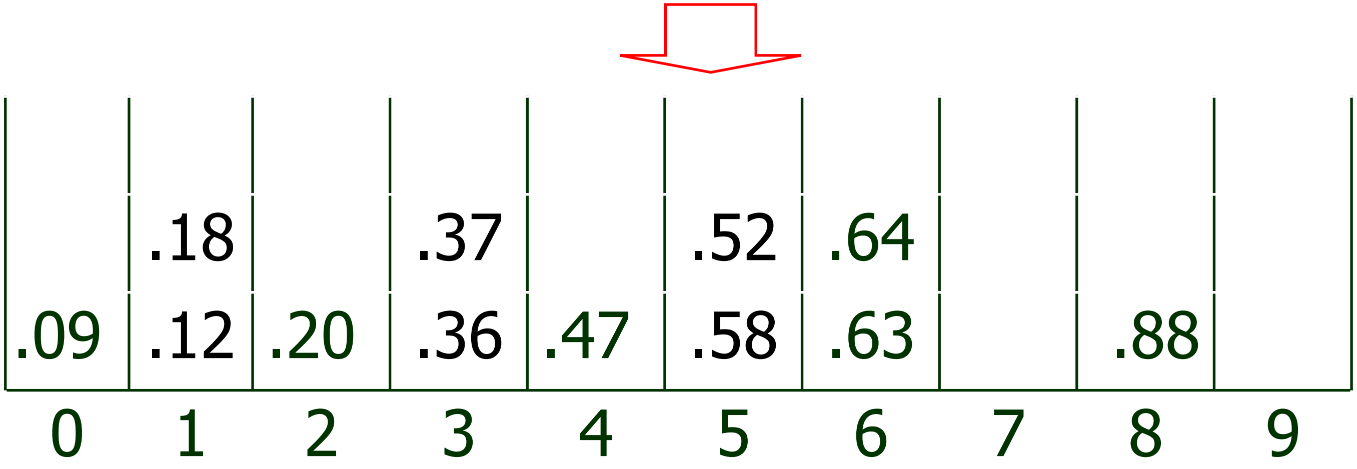


	.18		.37		.52	.64			
	.12	.20	.36	.47	.58	.63		.88	
0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Multiple items per bucket:

.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99



	.18		.37		.52	.64			
.09	.12	.20	.36	.47	.58	.63		.88	
0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Multiple items per bucket:

.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99

	.18		.37		.52	.64			
.09	.12	.20	.36	.47	.58	.63		.88	.99
0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Multiple items per bucket:

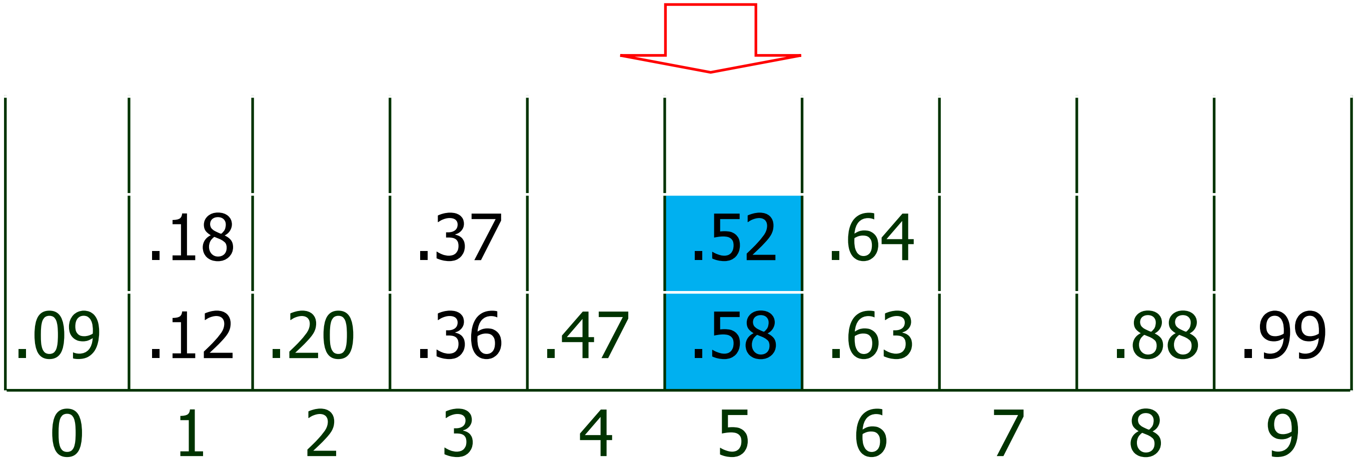
.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99

	.18		.37		.52	.64			
.09	.12	.20	.36	.47	.58	.63		.88	.99
0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Multiple items per bucket:

.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99



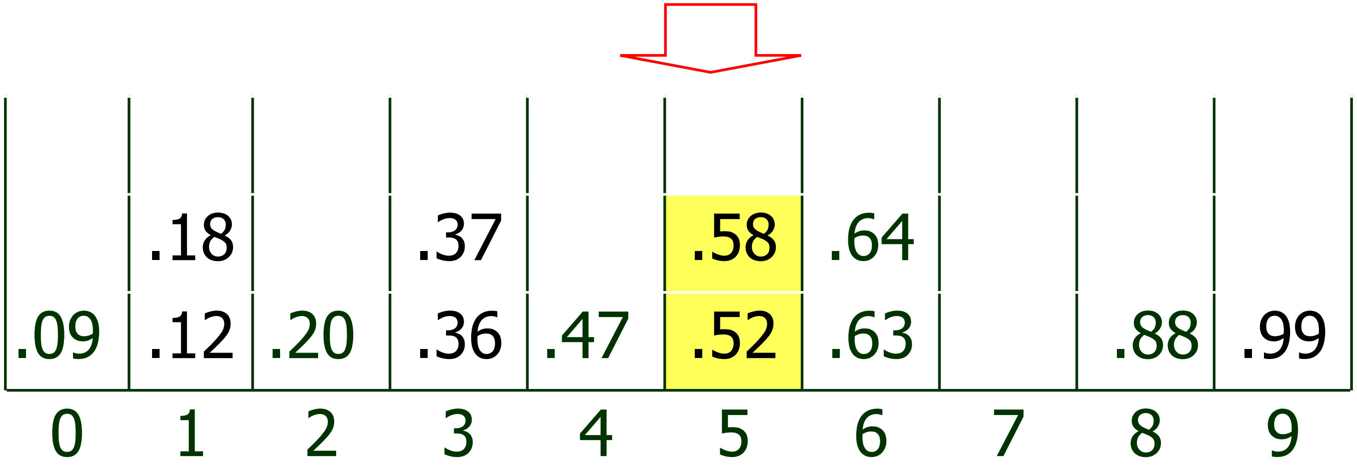
	.18		.37		.52	.64			
.09	.12	.20	.36	.47	.58	.63		.88	.99
0	1	2	3	4	5	6	7	8	9

Apply Internal sorting(stable) on highlighted data

Bucket Sort

- Multiple items per bucket:

.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99



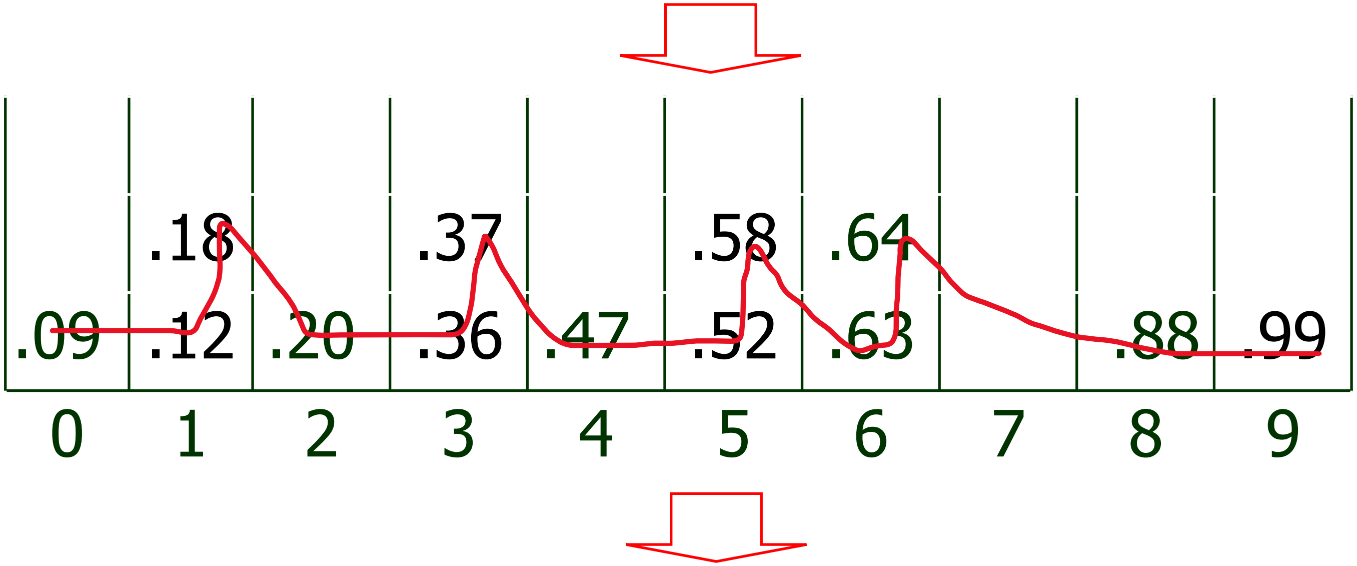
	.18		.37		.58	.64			
.09	.12	.20	.36	.47	.52	.63		.88	.99
0	1	2	3	4	5	6	7	8	9

After Internal sorting(stable) on highlighted data

Bucket Sort

- Multiple items per bucket:

.20	.12	.58	.63	.64	.36	.37	.47	.52	.18	.88	.09	.99



	.18		.37		.58	.64			
.09	.12	.20	.36	.47	.52	.63		.88	.99
0	1	2	3	4	5	6	7	8	9

.09	.12	.18	.20	.36	.37	.47	.52	.58	.63	.64	.88	.99

Bucket Sort

- Multiple items per bucket:

Algorithm BucketSort(S)

```
Let $B[0..(n-1)]$ be a new array.
$n \leftarrow A.length$
for $i \leftarrow 0$ to $n-1$
    make $B[i]$ an empty list
for $i \leftarrow\leftarrow 1$ to $n$
    insert $A[i]$ into list $B[n * A[i]]$
for $i \leftarrow 0$ to $n-1$
    sort list $B[i]$ with a stable sorting(insertion sort)
Concatenate the list $B[0], B[1], B[2], \dots\dots, B[n-1]$
 together in order.
```

Bucket Sort

- Multiple items per bucket:

Algorithm BucketSort(S)

```
$Let\ B[0..(n-1)]be\ a\ new\ array.$   $O(1)$
$n \leftarrow A.length$   $O(1)$
$for\ i \leftarrow 0\ to\ n-1$
    $make\ B[i]an\ empty\ list$   $\}\ O(n)$
$for\ i \leftarrow\leftarrow 1\ to\ n$
    $insert\ A[i]into\ list\ B[n * A[i]]$   $\}\ O(n)$
$for\ i \leftarrow 0\ to\ n-1$
  $sort\ list\ B[i]\ with\ a\ stable\ sorting(insertion\ sort)$   $\}\ O(n^2)$
$Concatenate\ the\ list\ B[0], B[1], B[2], \ldots\ldots, B[n-1]$
  $together\ in\ order.$   $\}\ O(n)$
```

if all the elements belongs to one bucket.

Bucket Sort

Multiple items per bucket (Analysis)

- It was observed that except line no 8 all other lines take $\mathrm{O}(n)$ time in worst case.
- Line no. 8 (i.e. insertion sort) takes $\mathrm{O}(n^2)$, if all the elements belongs to one bucket.
- The average time complexity for Bucket Sort is $O(n + k)$ in uniform distribution of data.

Bucket Sort

Characteristics of Bucket Sort

- Bucket sort assumes that the input is drawn from a uniform distribution.
- The computational complexity estimates involve the number of buckets.
- Bucket sort can be exceptionally fast because of the way elements are assigned to buckets, typically using an array where the index is the value.

Bucket Sort

Characteristics of Bucket Sort

- This means that more auxiliary memory is required for the buckets at the cost of running time than more comparison sorts.
- The average time complexity is $O(n + k)$.
- The worst time complexity is $O(n^2)$.
- The space complexity for Bucket Sort is $O(n + k)$.

Thank u