Bucket Sorting

Bucket Sort

- Bucket sort assumes that the input is generated by a random process and drawn from a uniform distribution.
- In other words the elements are distributed uniformly and independently over the interval [0,1].
- Bucket sort divides the interval [0,1] into n equal sized subintervals or buckets. Then distributes the n inputs into these buckets.
- After that the elements of each buckets are sorted using a sorting algorithm generally using insertion or quick sort.
- Finally the buckets are concatenated together in order.
- Consider that the input is an n-element array A and each element A[i] in the array satisfies the 0<=A[i]<1

Bucket Sort Algorithm

```
Bucket-Sort(A)
Let B[0....n-1] be a new array
n = length[A]
for i = 0 to n-1
    make B[i] an empty list
for i = 1 to n
    do insert A[i] into list B[$\lfloor n A[i] \rfloor$]
for i = 0 to n-1
    do sort list B[i] with Insertion-Sort
Concatenate lists B[0], B[1],…,B[n-1] together in order
```

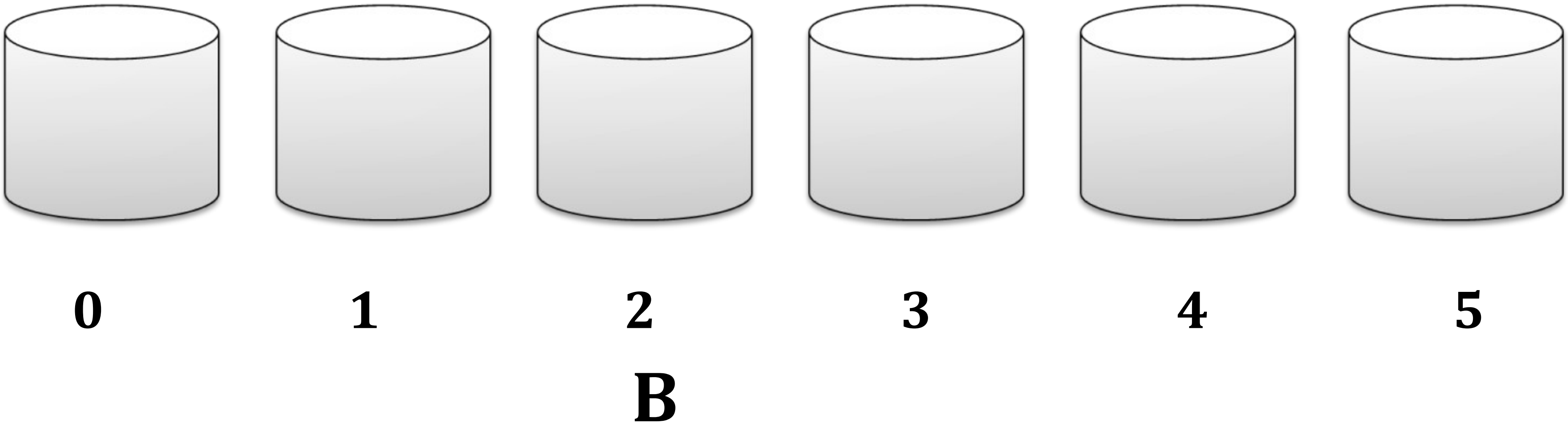
Bucket

	1	2	3	4	5	6
A	.74	.17	.26	.72	.39	.21

Bucket: Loop 1

n=6

	1	2	3	4	5	6
A	.74	.17	.26	.72	.39	.21

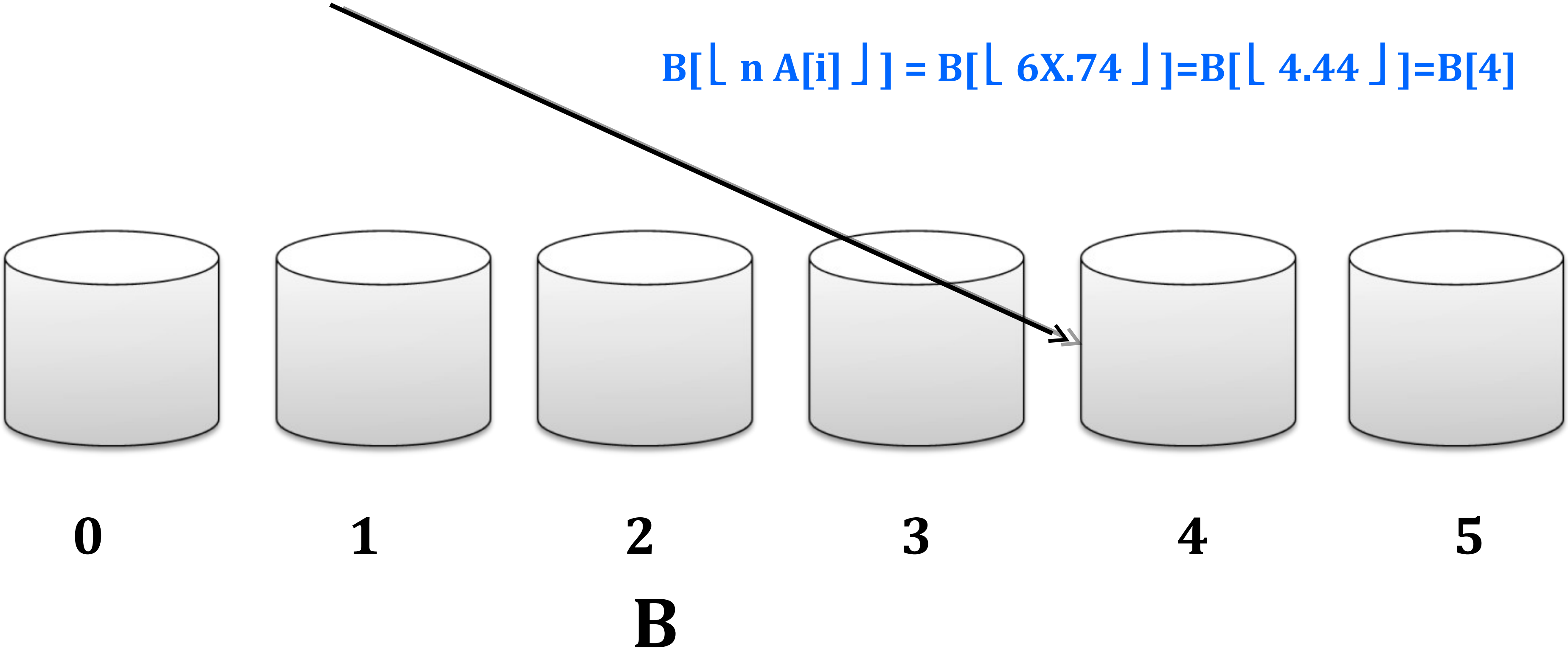


Bucket: Loop 2

FOR n=6, i=1

	1	2	3	4	5	6
A	.74	.17	.26	.72	.39	.21

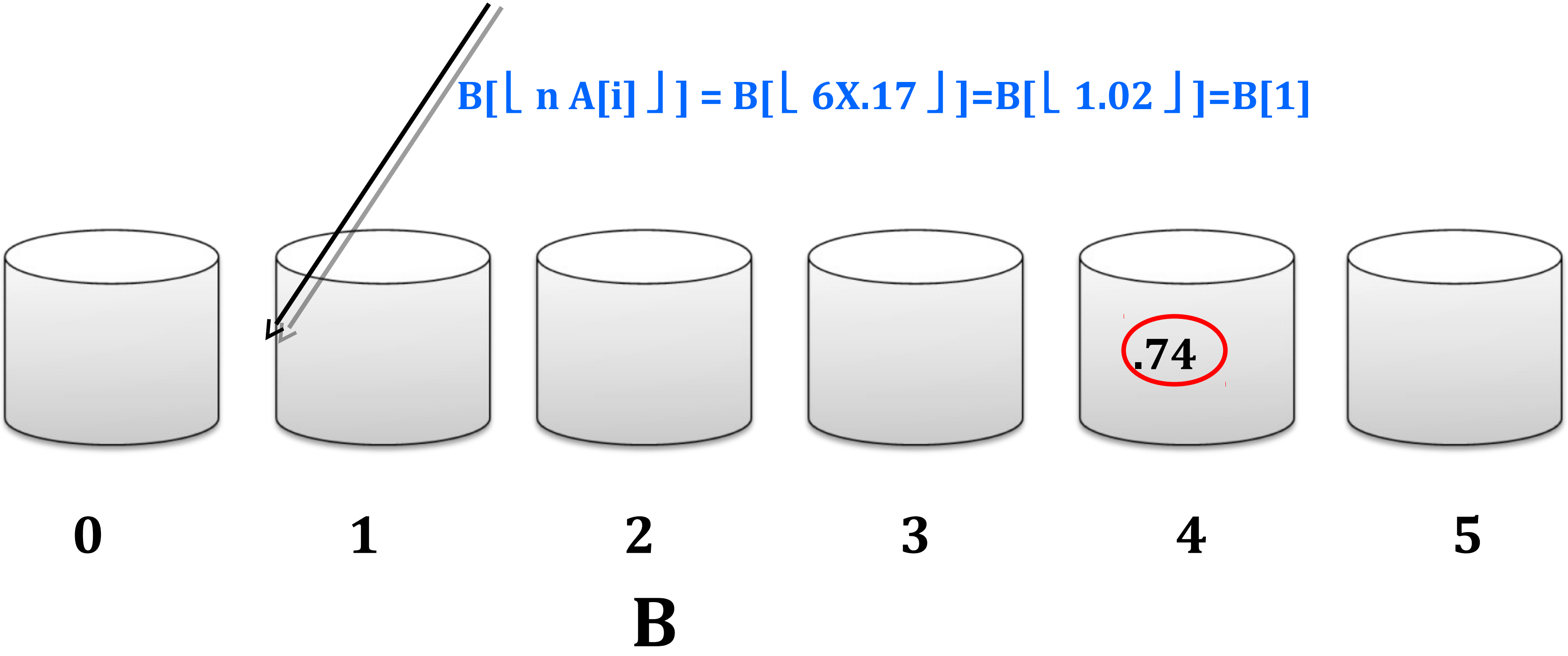
B[⌊ n A[i] ⌋] = B[⌊ 6X.74 ⌋]=B[⌊ 4.44 ⌋]=B[4]



Bucket: Loop 2

FOR n=6, i=2

	1	2	3	4	5	6
A	.74	.17	.26	.72	.39	.21



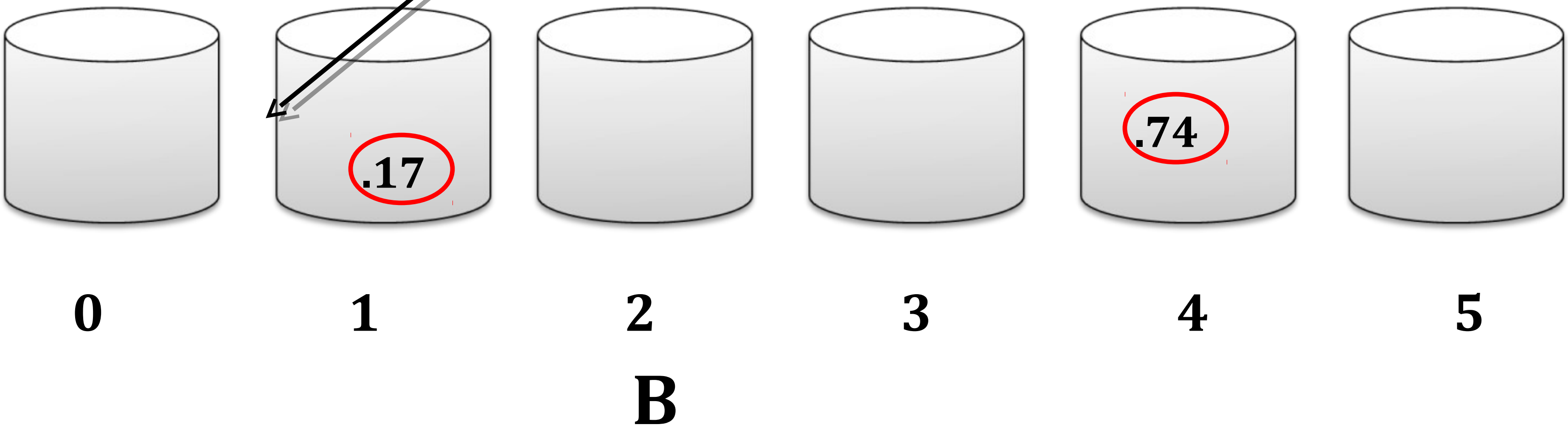
$B[\lfloor n\ A[i] \rfloor] = B[\lfloor 6X.17 \rfloor]=B[\lfloor 1.02 \rfloor]=B[1]$

Bucket: Loop 2

FOR n=6, i=3

	1	2	3	4	5	6
A	.74	.17	.26	.72	.39	.21

$B[\lfloor n A[i] \rfloor] = B[\lfloor 6X.26 \rfloor]=B[\lfloor 1.56 \rfloor]=B[1]$

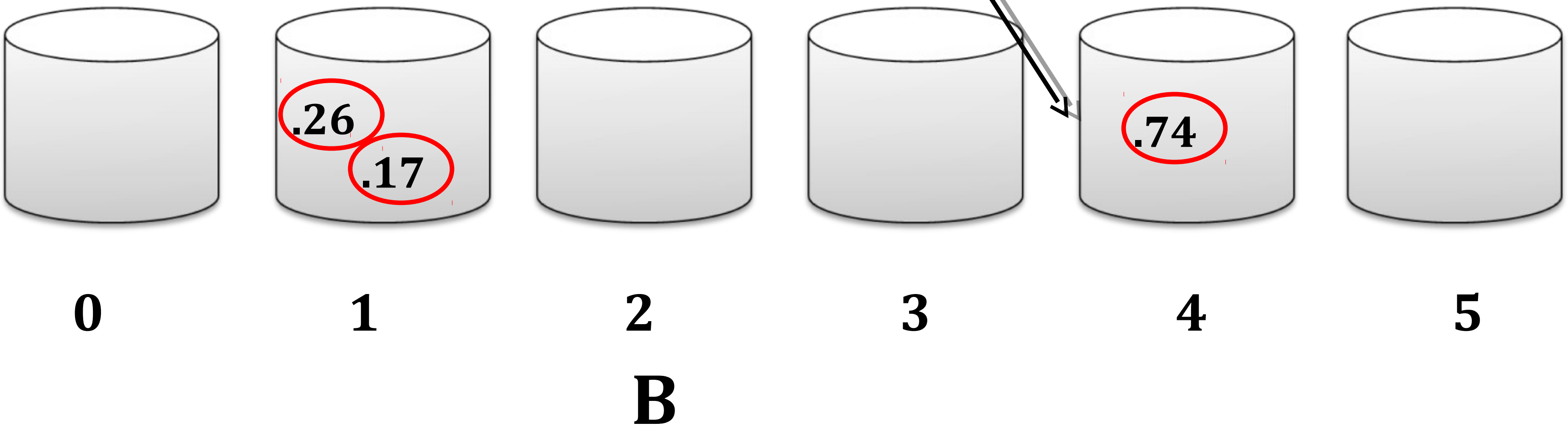


Bucket: Loop 2

FOR n=6, i=4

	1	2	3	4	5	6
A	.74	.17	.26	.72	.39	.21

B[⌊ n A[i] ⌋] = B[⌊ 6X.72 ⌋]=B[⌊ 4.32 ⌋]=B[4]

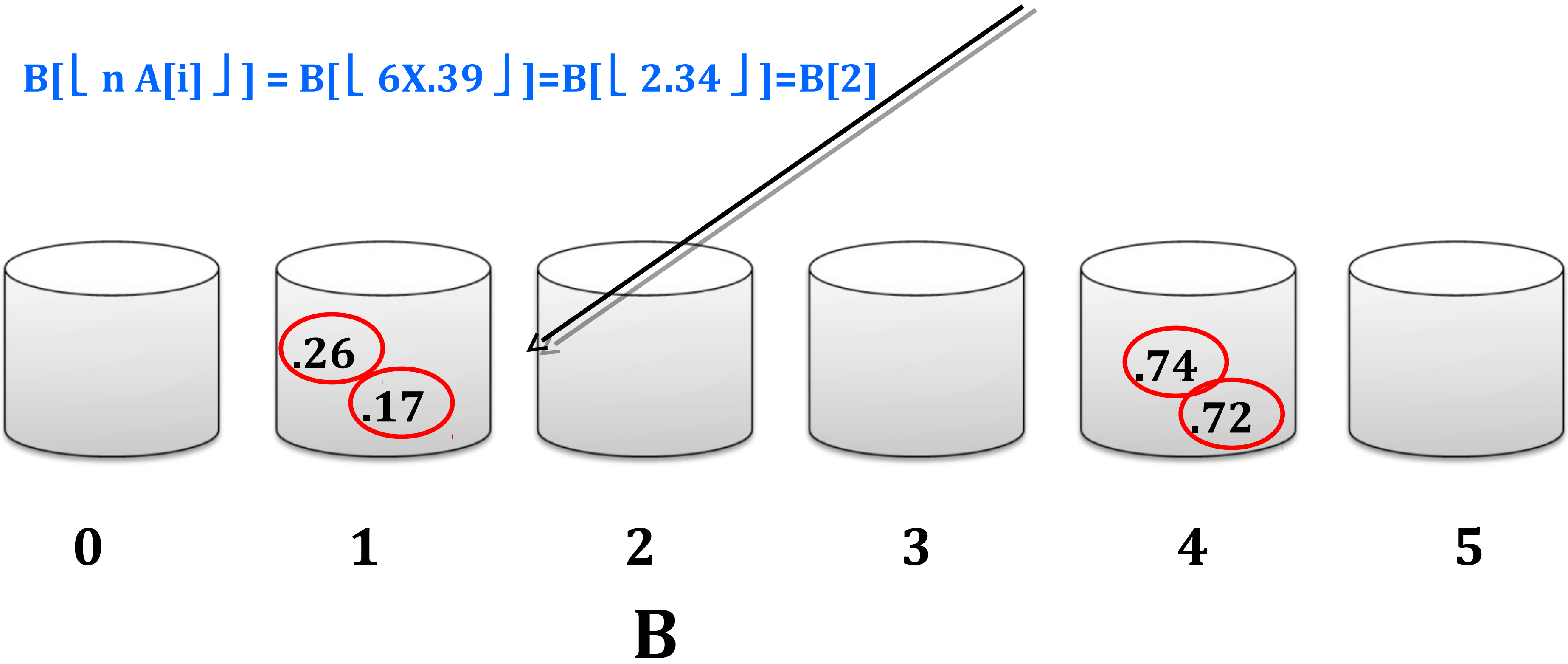


Bucket: Loop 2

FOR n=6, i=5

	1	2	3	4	5	6
A	.74	.17	.26	.72	.39	.21

B[⌊ n A[i] ⌋] = B[⌊ 6X.39 ⌋]=B[⌊ 2.34 ⌋]=B[2]

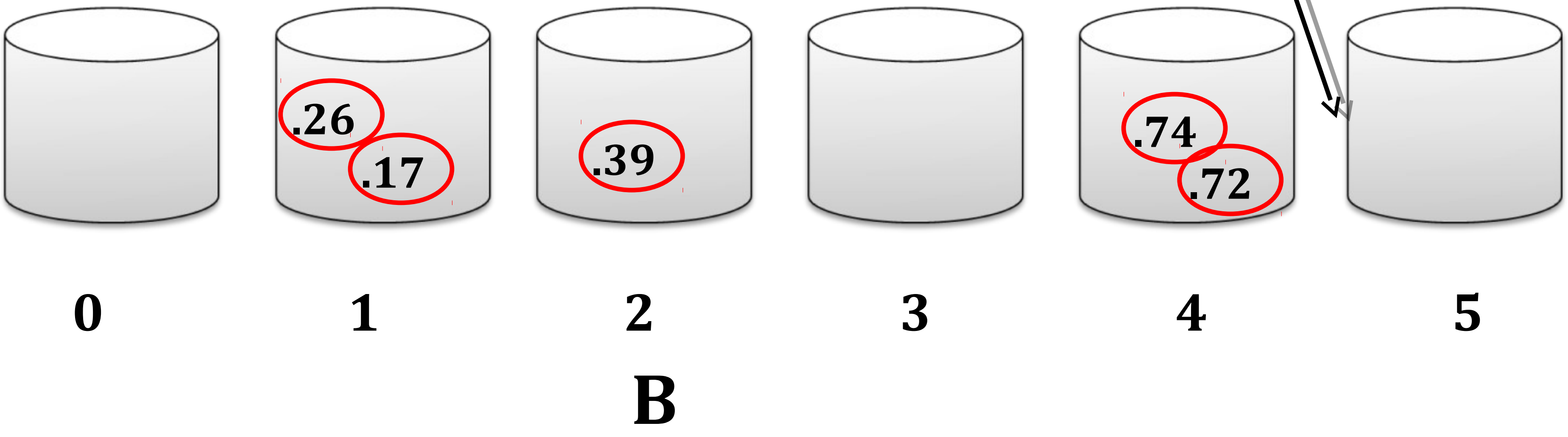


Bucket: Loop 2

FOR n=6, i=6

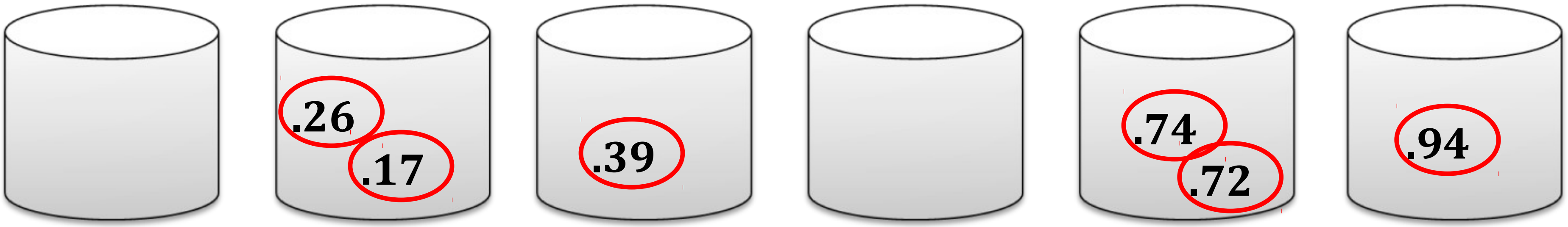
	1	2	3	4	5	6
A	.74	.17	.26	.72	.39	.94

B[⌊ n A[i] ⌋] = B[⌊ 6X.94 ⌋]=B[⌊ 5.64 ⌋]=B[5]



Bucket: End of Loop 2

	1	2	3	4	5	6
A	.74	.17	.26	.72	.39	.94



.26 .17

.39

.74 .72

.94

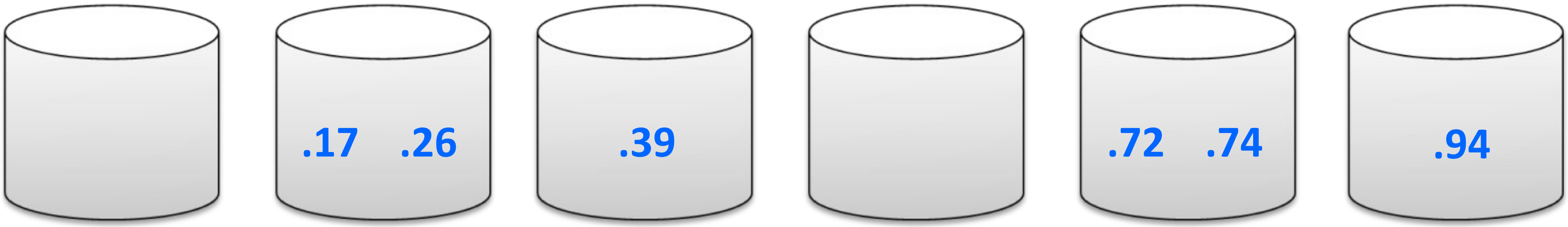
0 1 2 3 4 5

B

Bucket: Loop 3

Apply insertion sort on each bucket

	1	2	3	4	5	6
A	.74	.17	.26	.72	.39	.94



	.17 .26	.39		.72 .74	.94
0	1	2	3	4	5

B

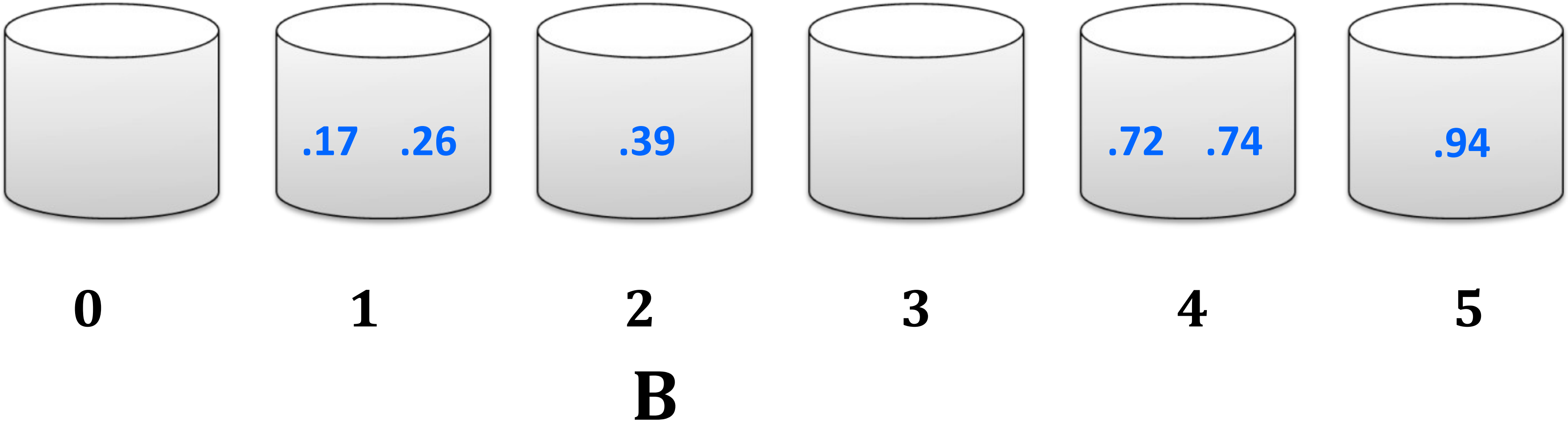
Bucket

Concatenate the buckets in order

	1	2	3	4	5	6
A	.74	.17	.26	.72	.39	.94

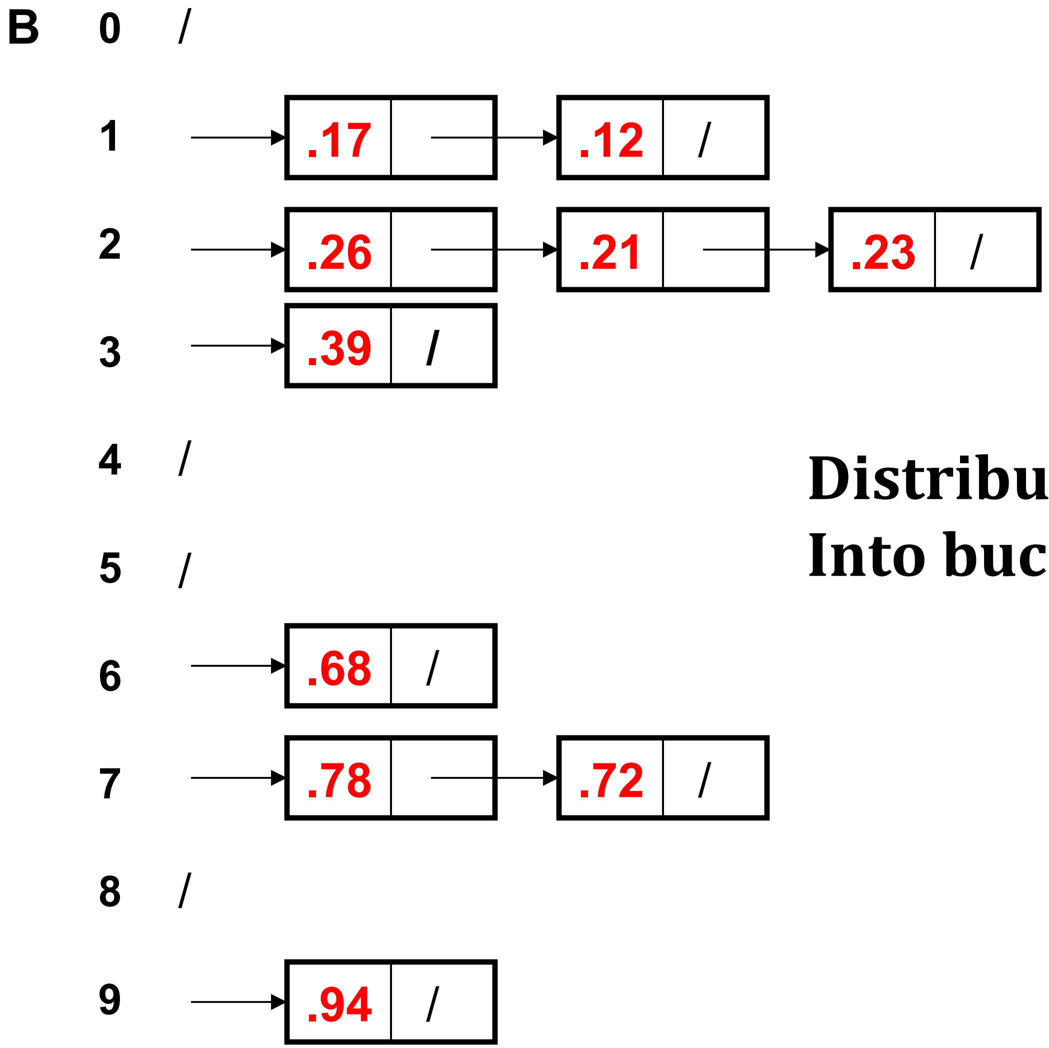
Sorted output

	0	1	2	3	4	5
B	.17	.26	.39	.72	.74	.94



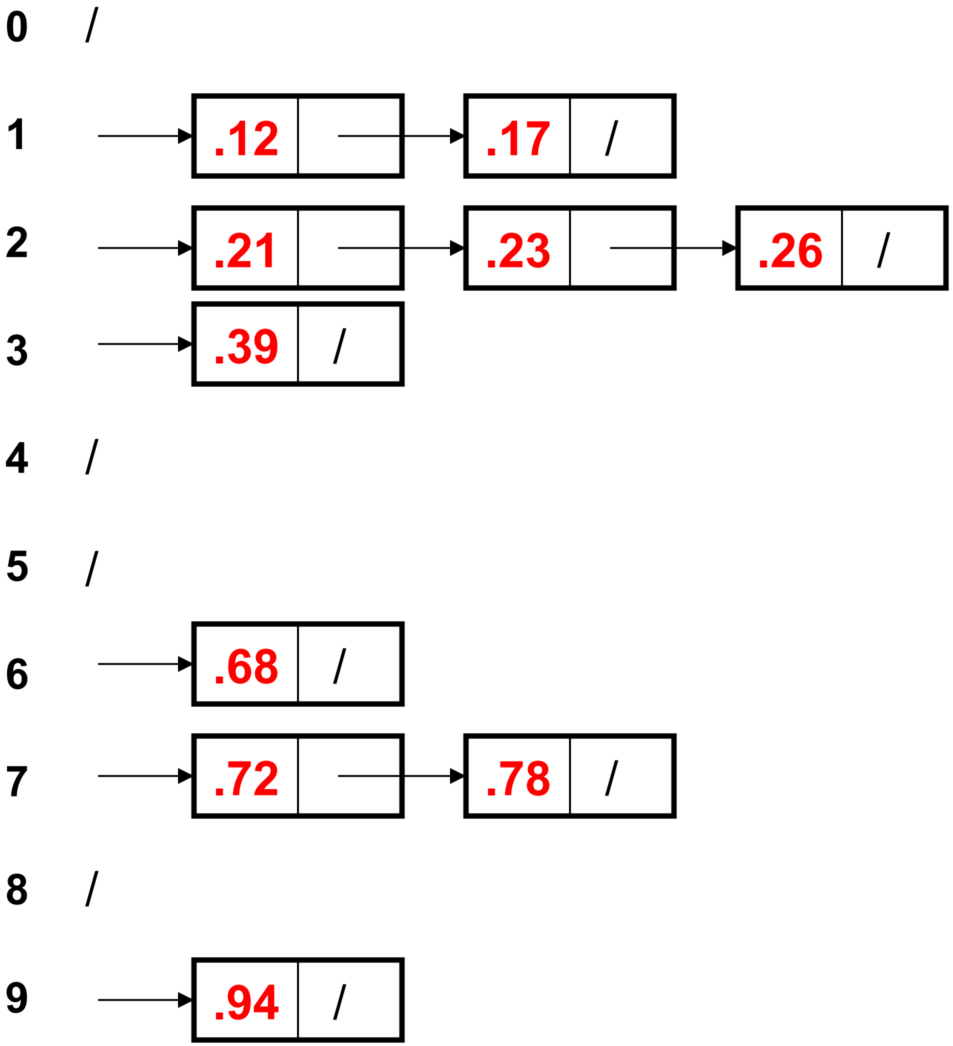
Example - Bucket Sort

A	
1	.78
2	.17
3	.39
4	.26
5	.72
6	.94
7	.21
8	.12
9	.23
10	.68



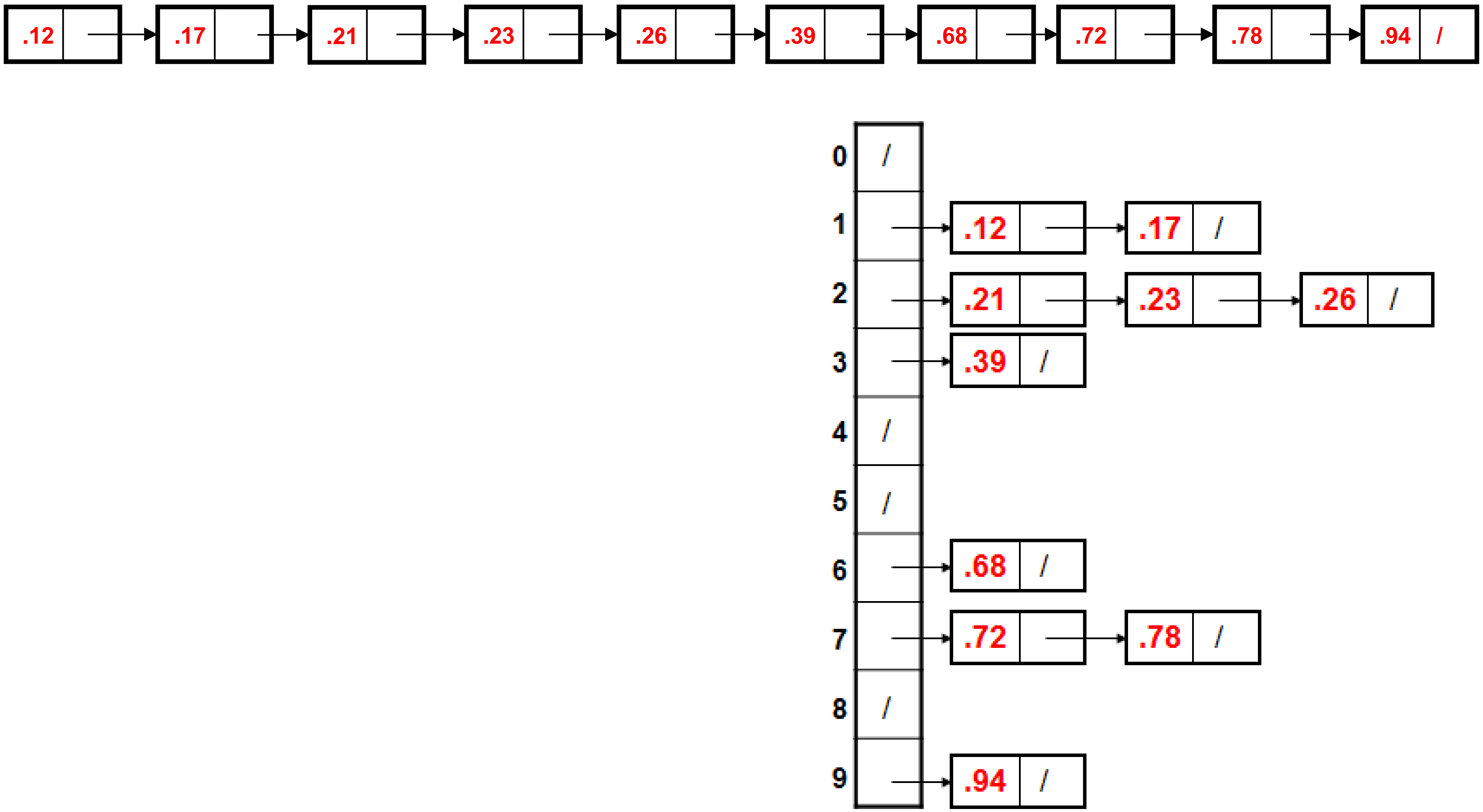
Distribute
Into buckets

Example - Bucket Sort



Sort within each bucket

Example - Bucket Sort



Analysis of Bucket Sort

- Bucket-Sort(A)

```
1. Let B[0….n-1] be a new array
2. n = length[A]
3. for i = 0 to n-1
4.       make B[i] an empty list
5. for i = 1 to n
6.      do insert A[i] into list B[ floor of n A[i] ]
7. for i = 0 to n-1
8.     do sort list B[i] with Insertion-Sort
9. Concatenate lists B[0], B[1],…,B[n-1]
   together in order
```

Step 5 and 6 takes O(n) time

Step 7 and 8 takes O(n log(n/k) time

Step 9 takes O(k) time

In total Bucket sort takes : O(n) (if k=Θ(n))

Bucket Sort Review

- **Assumption:** input is uniformly distributed across a range
- Basic idea:
  - Partition the range into a fixed number of buckets.
  - Toss each element into its appropriate bucket.
  - Sort each bucket.
- Pro's:
  - Fast
  - Asymptotically fast (i.e., O(n) when distribution is uniform)
  - Simple to code
  - Good for a rough sort.
- Con's:
  - Doesn't sort in place