Lecture#5

Data Structures

Dr. Abu Nowshed Chy

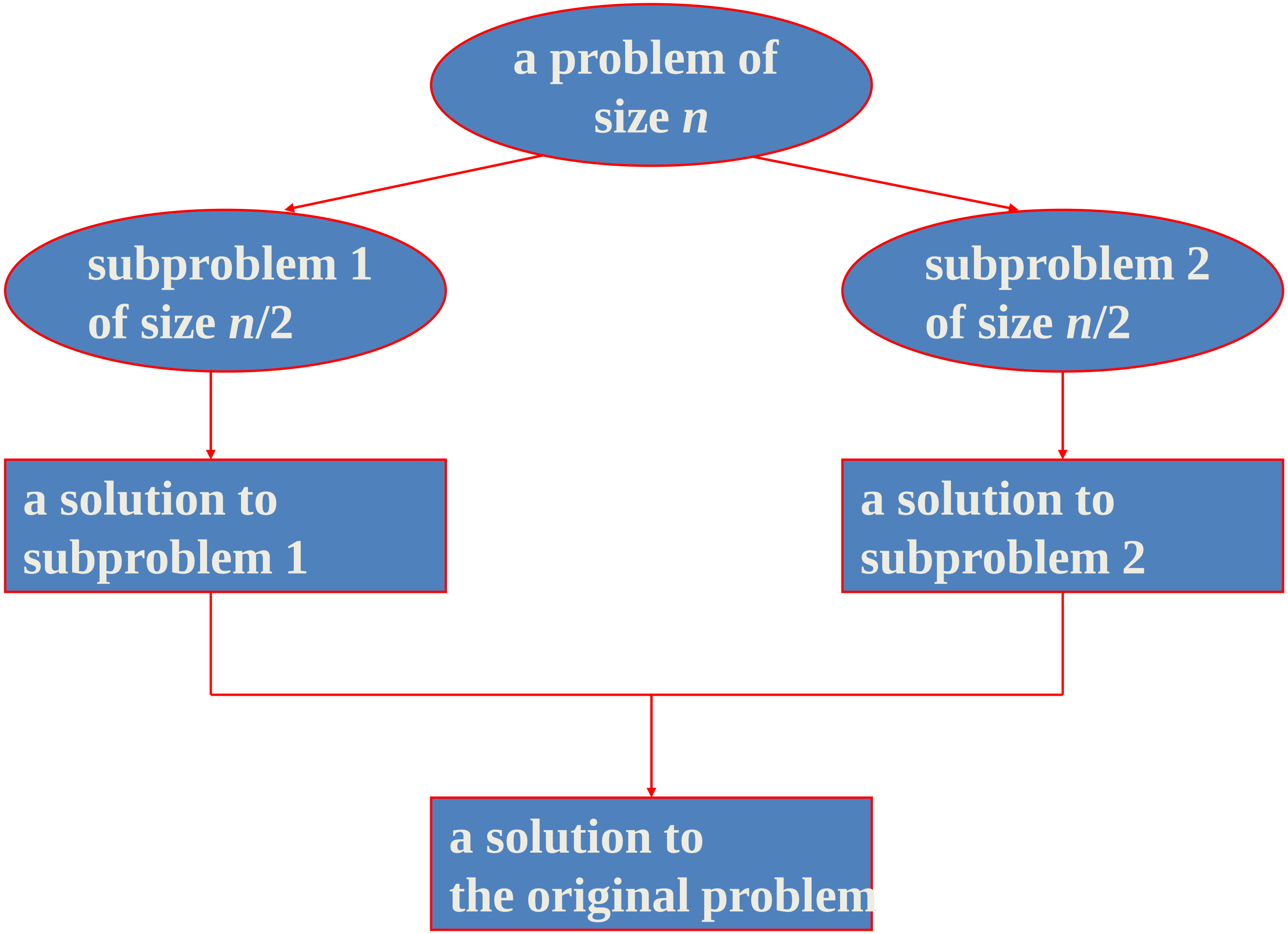
Department of Computer Science and Engineering

University of Chittagong

January 22, 2025

Faculty Profile

Divide and Conquer



a problem of size *n*

subproblem 1 of size *n*/2

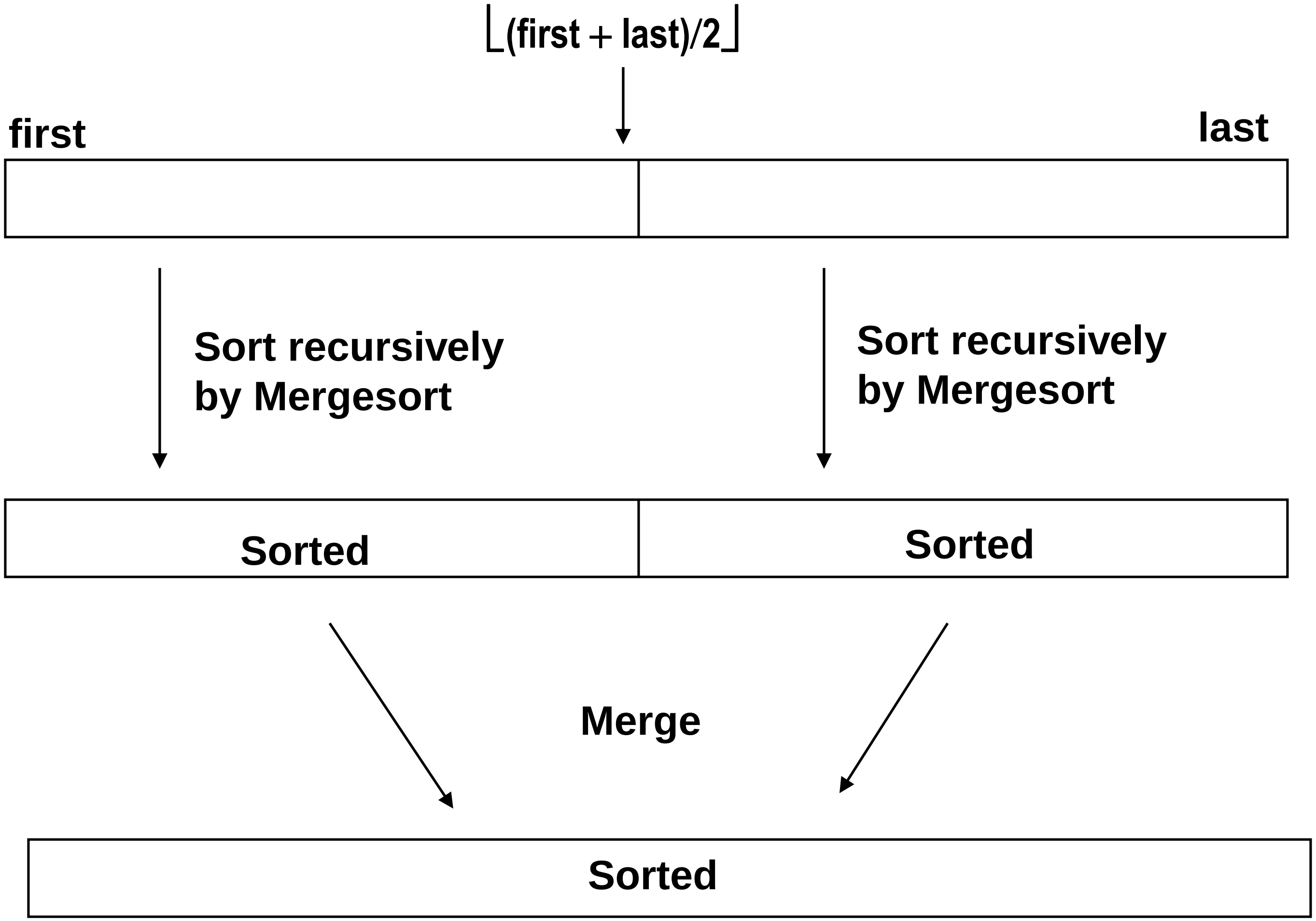
subproblem 2 of size *n*/2

a solution to subproblem 1

a solution to subproblem 2

a solution to the original problem

Divide and Conquer: Merge Sort



$\lfloor (\text{first} + \text{last})/2 \rfloor$

first

last

Sort recursively by Mergesort

Sort recursively by Mergesort

Sorted

Sorted

Merge

Sorted

Divide and Conquer: Merge Sort

- Divide: Partition ‘n’ elements array into two sub lists with n/2 elements each
- Conquer: Sort sub list1 and sub list2
- Combine: Merge sub list1 and sub list2

How Merging Performed!

arrayA

1	13	24	26
indexA			

arrayB

2	15	27	38
indexB			

We compare arrayA[indexA] with arrayB[indexB]. Whichever value is smaller is placed into arrayC[indexC].

1 < 2 so we insert arrayA[indexA] into arrayC[indexC]

arrayC

indexC							

How Merging Performed!

arrayA

1	13	24	26
	indexA		

2 < 13 so we insert arrayB[indexB] into arrayC[indexC]

arrayB

2	15	27	38
indexB			

arrayC

1							
	indexC						

How Merging Performed!

arrayA

1	13	24	26
	indexA		

arrayB

2	15	27	38
	indexB		

13 < 15 so we insert arrayA[indexA] into arrayC[indexC]

arrayC

1	2						
		indexC					

How Merging Performed!

arrayA

1	13	24	26
		indexA	

arrayB

2	15	27	38
	indexB		

15 < 24 so we insert arrayB[indexB] into arrayC[indexC]

arrayC

1	2	13					
			indexC				

How Merging Performed!

arrayA

1	13	24	26
		indexA	

arrayB

2	15	27	38
		indexB	

24 < 27 so we insert arrayA[indexA] into arrayC[indexC]

arrayC

1	2	13	15				
				indexC			

How Merging Performed!

arrayA

1	13	24	26
			indexA

26 < 27 so we insert arrayA[indexA] into arrayC[indexC]

arrayB

2	15	27	38
		indexB	

arrayC

1	2	13	15	24			
					indexC		

How Merging Performed!

arrayA

1	13	24	26

arrayB

2	15	27	38
		indexB	

Since we have exhausted one of the arrays, arrayA, we simply copy the remaining items from the other array, arrayB, into arrayC

arrayC

1	2	13	15	24	26		
						indexC	

How Merging Performed!

arrayA

1	13	24	26

arrayB

2	15	27	38

arrayC

1	2	13	15	24	26	27	38

Merge Sort

99	6	86	15	58	35	86	4	0

Merge Sort

99	6	86	15	58	35	86	4	0

99	6	86	15

58	35	86	4	0

Merge Sort

99	6	86	15	58	35	86	4	0

99	6	86	15

58	35	86	4	0

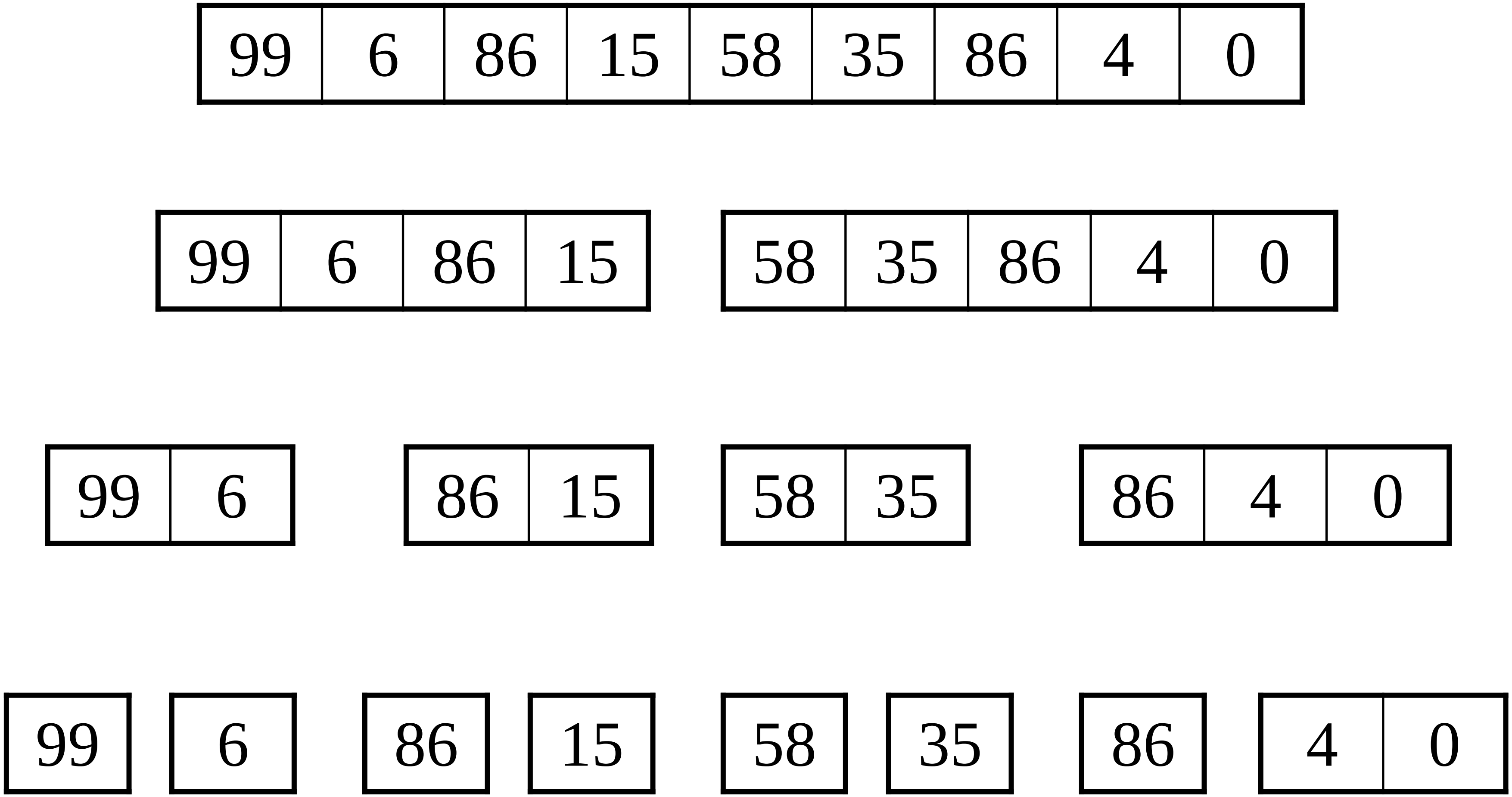
99	6

86	15

58	35

86	4	0

Merge Sort



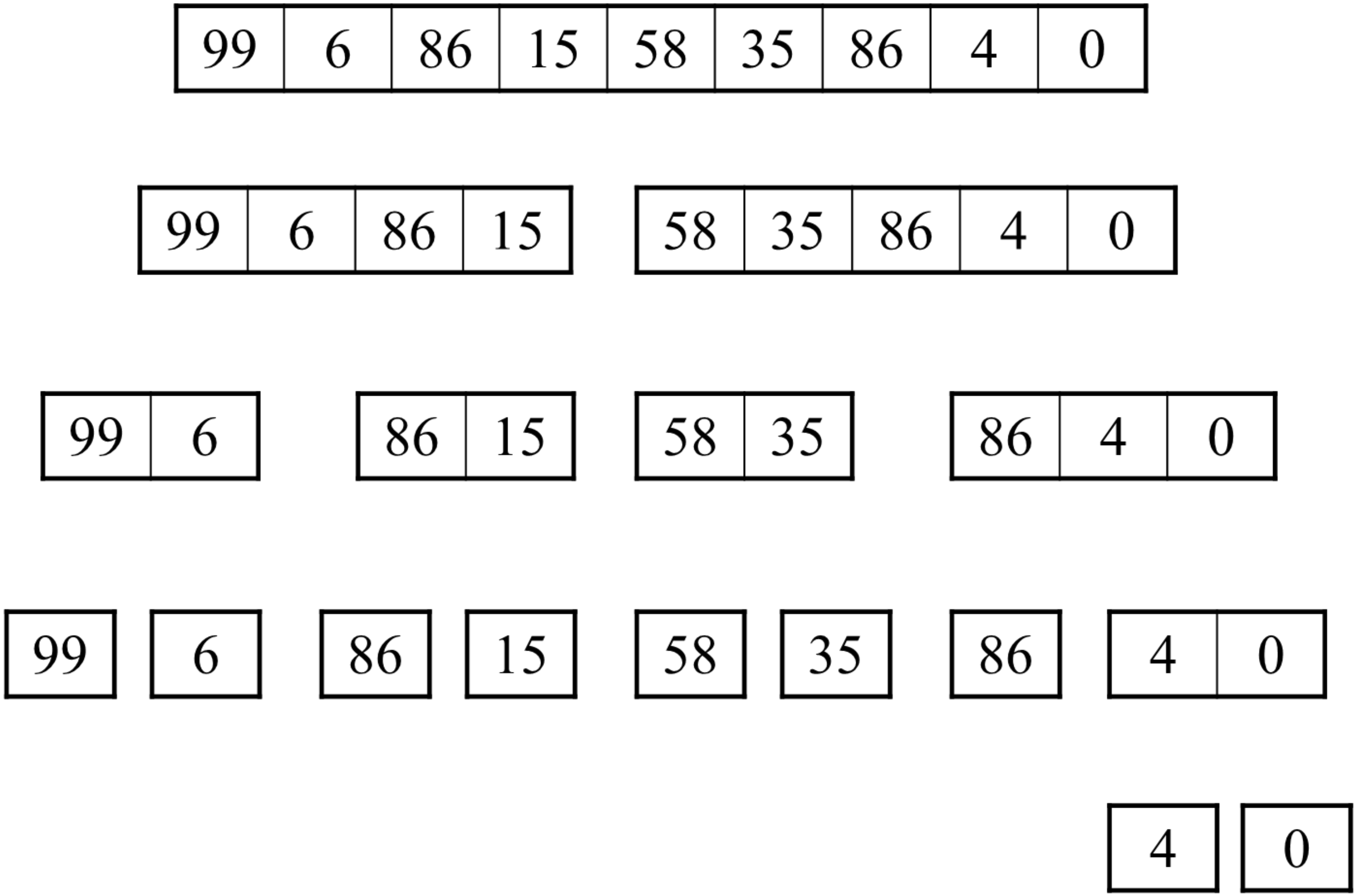
99 | 6 | 86 | 15 | 58 | 35 | 86 | 4 | 0

99 | 6 | 86 | 15   58 | 35 | 86 | 4 | 0

99 | 6   86 | 15   58 | 35   86 | 4 | 0

99   6   86   15   58   35   86   4 | 0

Merge Sort



99 6 86 15 58 35 86 4 0

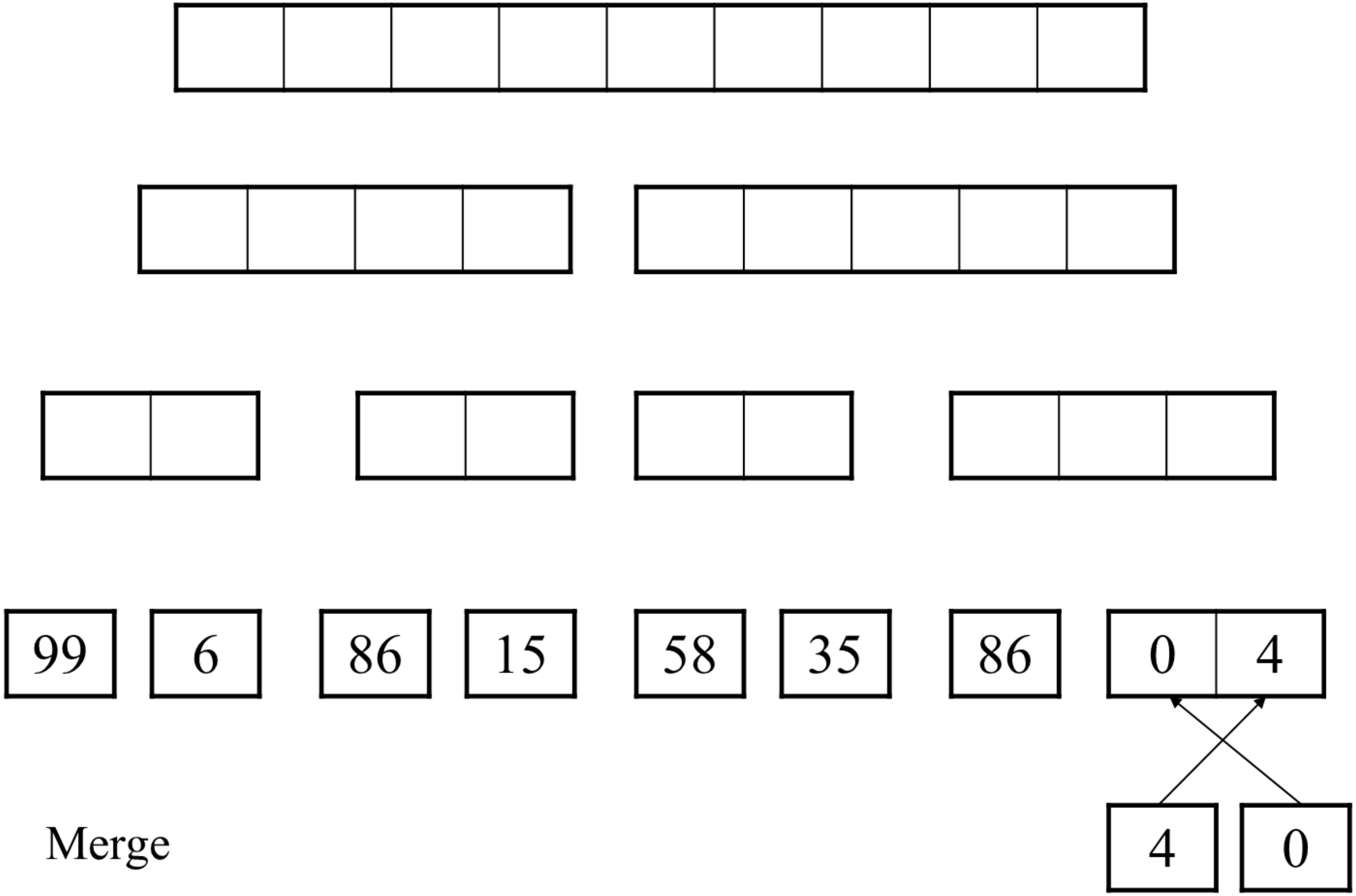
99 6 86 15   58 35 86 4 0

99 6   86 15   58 35   86 4 0

99   6   86   15   58   35   86   4 0

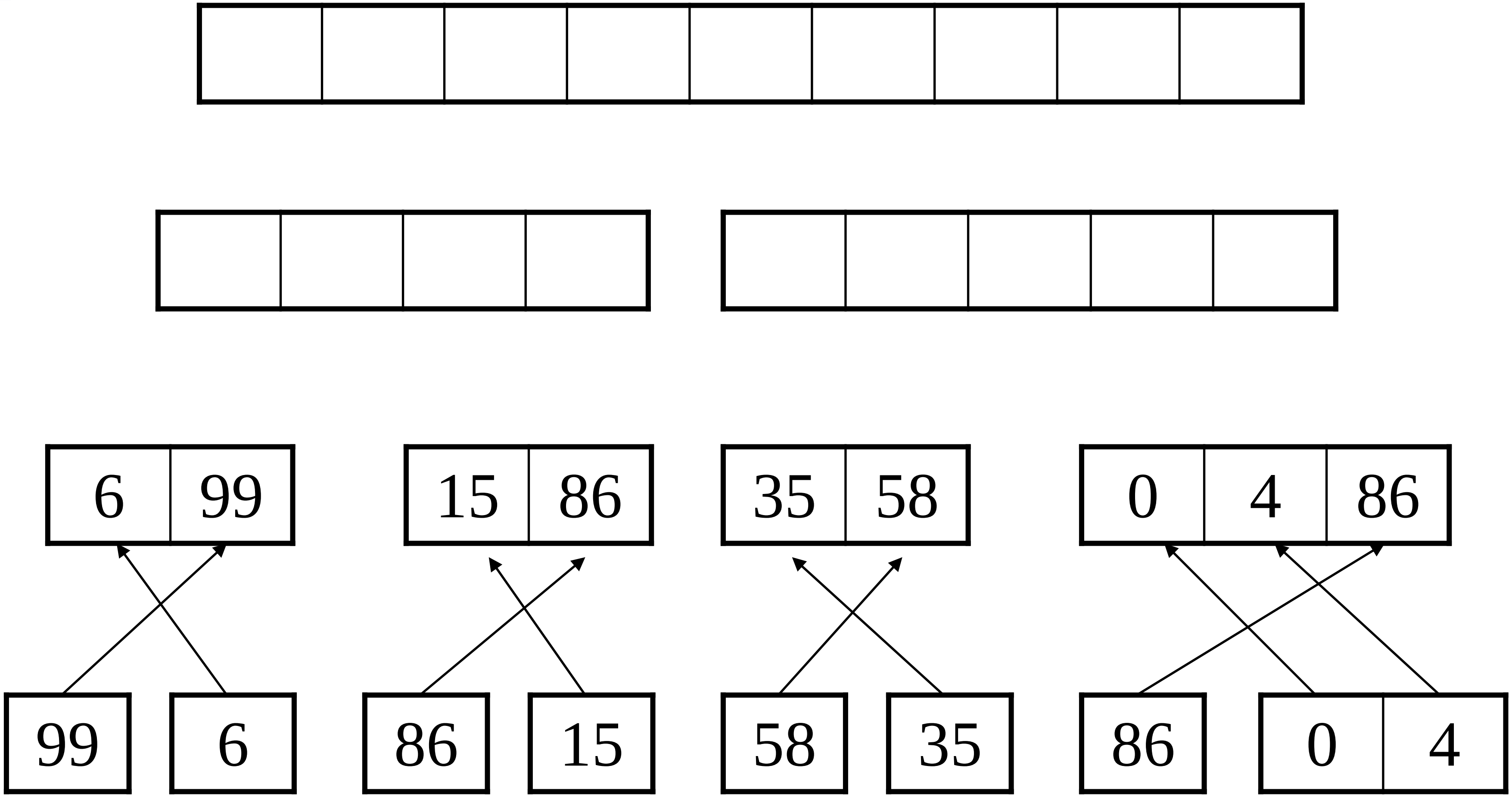
4   0

Merge Sort



Merge

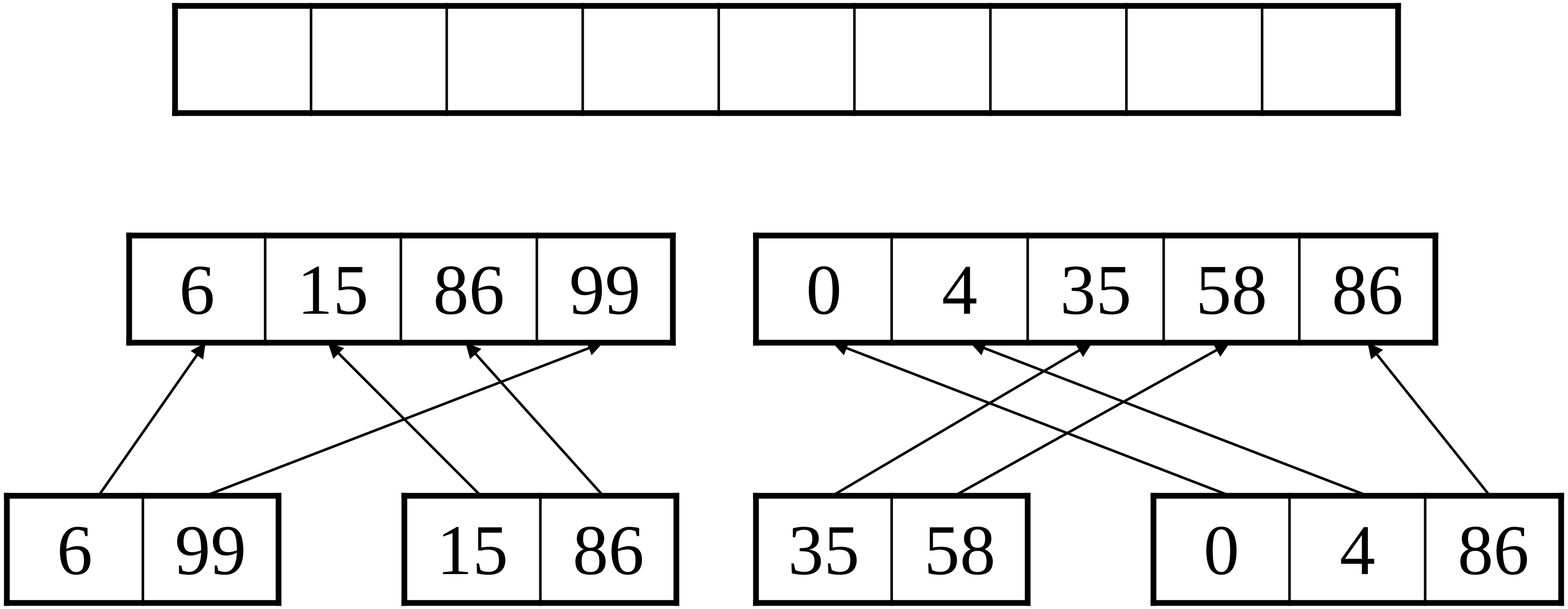
Merge Sort



6 | 99   15 | 86   35 | 58   0 | 4 | 86

99   6   86   15   58   35   86   0 | 4

Merge Sort



6 | 15 | 86 | 99

0 | 4 | 35 | 58 | 86

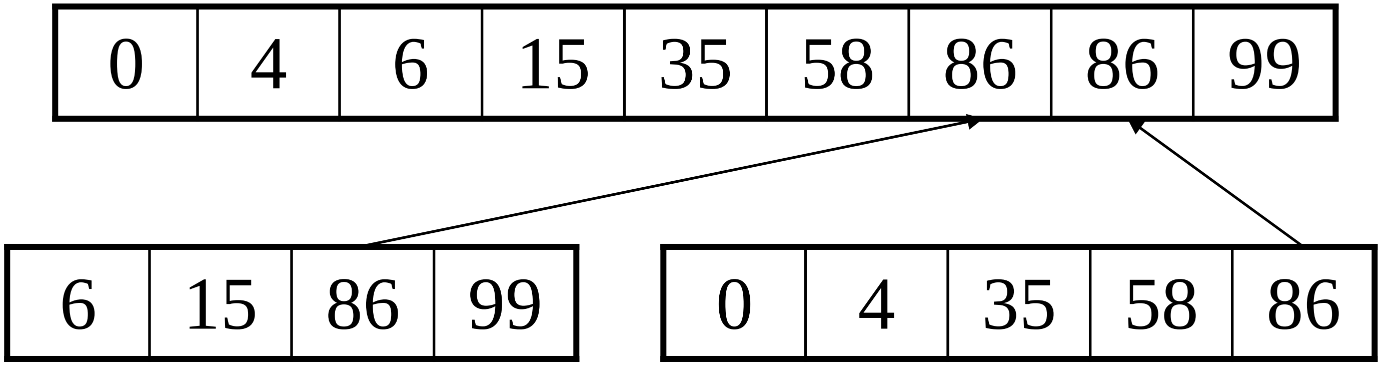
6 | 99

15 | 86

35 | 58

0 | 4 | 86

Merge Sort



0 4 6 15 35 58 86 86 99

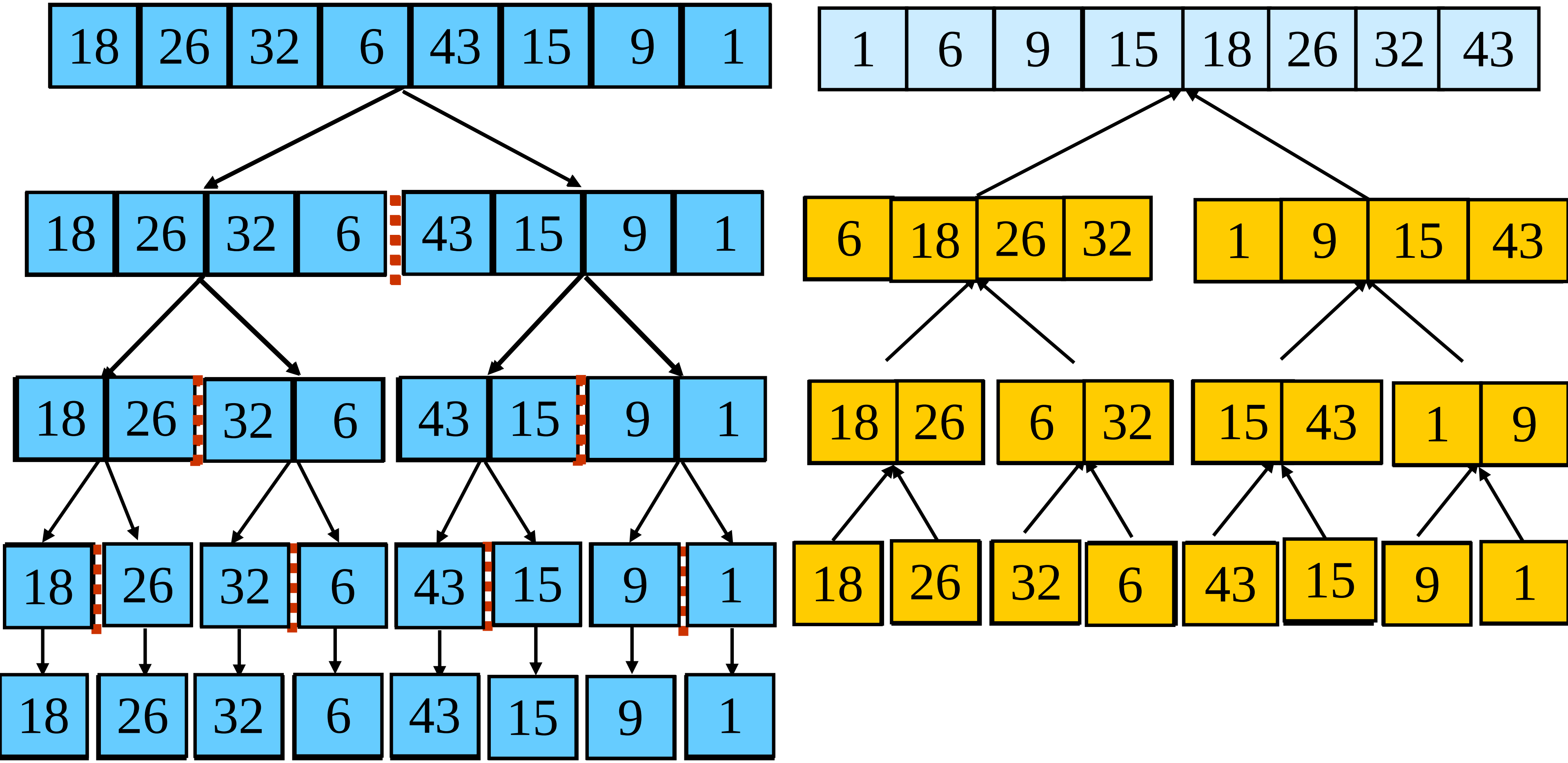
6 15 86 99

0 4 35 58 86

Merge Sort

0	4	6	15	35	58	86	86	99

Merge Sort



Original Sequence

18 26 32 6 43 15 9 1

18 26 32 6 43 15 9 1

18 26 32 6 43 15 9 1

18 26 32 6 43 15 9 1

18 26 32 6 43 15 9 1

Sorted Sequence

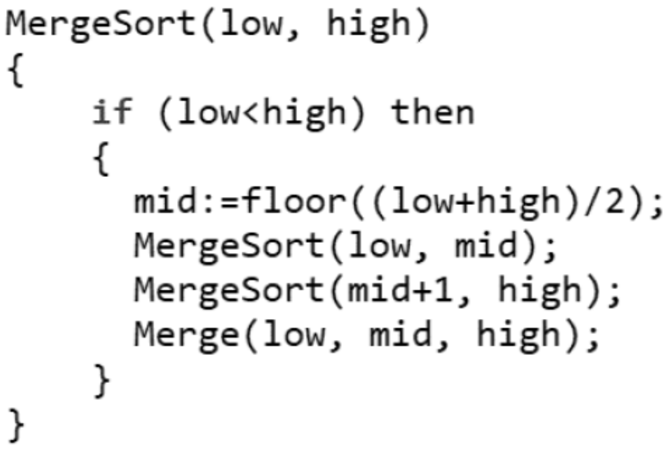
1 6 9 15 18 26 32 43

6 18 26 32 1 9 15 43

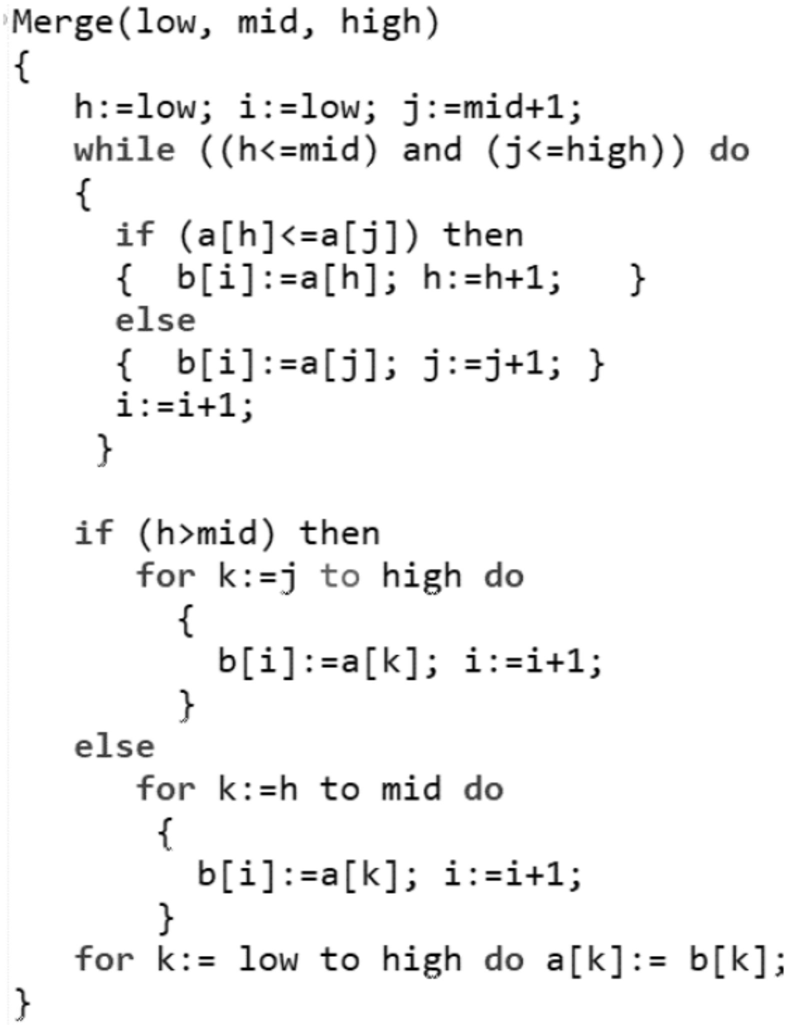
18 26 6 32 15 43 1 9

18 26 32 6 43 15 9 1

Merge Sort



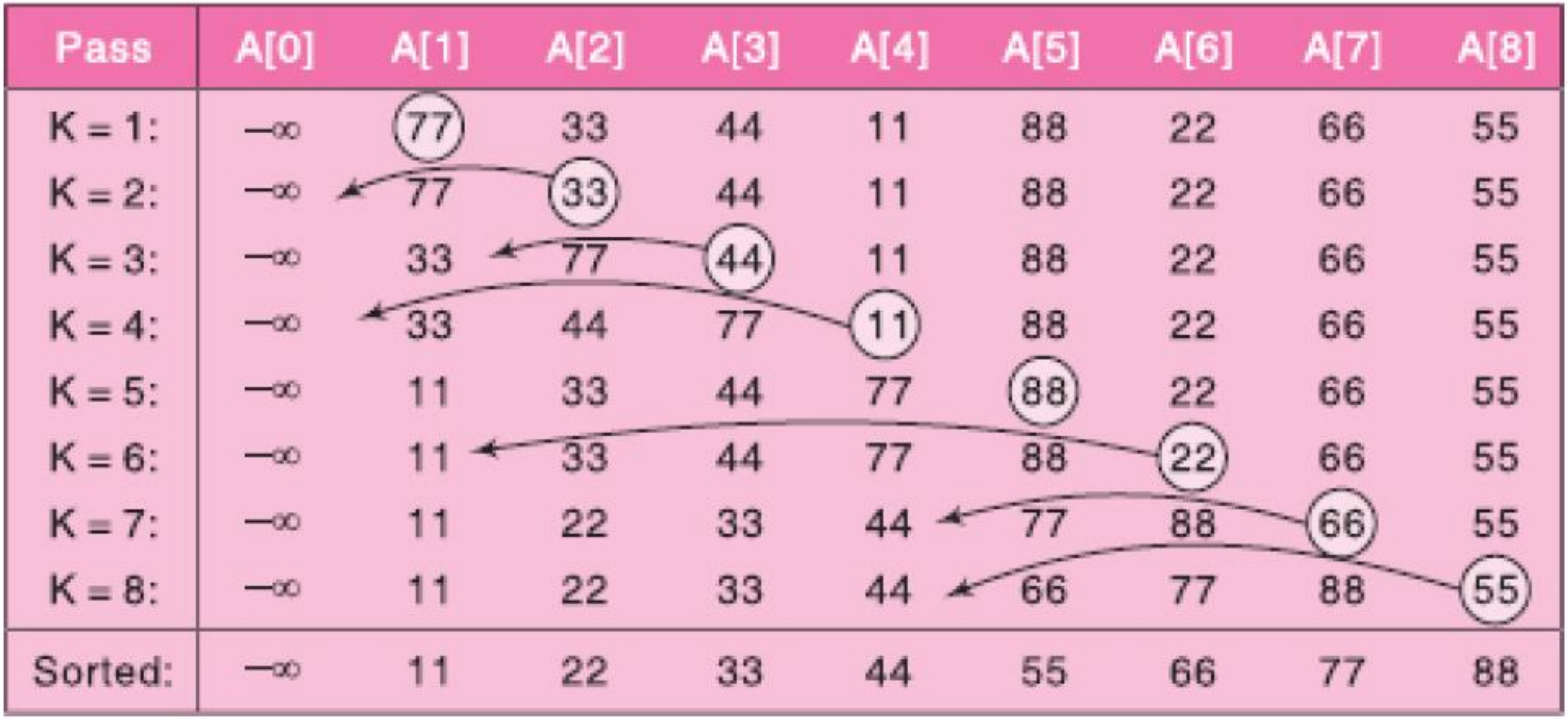
```
MergeSort(low, high)
{
    if (low<high) then
    {
      mid:=floor((low+high)/2);
      MergeSort(low, mid);
      MergeSort(mid+1, high);
      Merge(low, mid, high);
    }
}
```



```
Merge(low, mid, high)
{
   h:=low; i:=low; j:=mid+1;
   while ((h<=mid) and (j<=high)) do
   {
     if (a[h]<=a[j]) then
     {  b[i]:=a[h]; h:=h+1;   }
     else
     {  b[i]:=a[j]; j:=j+1; }
     i:=i+1;
    }

   if (h>mid) then
      for k:=j to high do
        {
          b[i]:=a[k]; i:=i+1;
        }
   else
      for k:=h to mid do
       {
         b[i]:=a[k]; i:=i+1;
       }
   for k:= low to high do a[k]:= b[k];
}
```

Insertion Sort

Suppose an array A contains 8 elements as follows:
77, 33, 44, 11, 88, 22, 66, 55



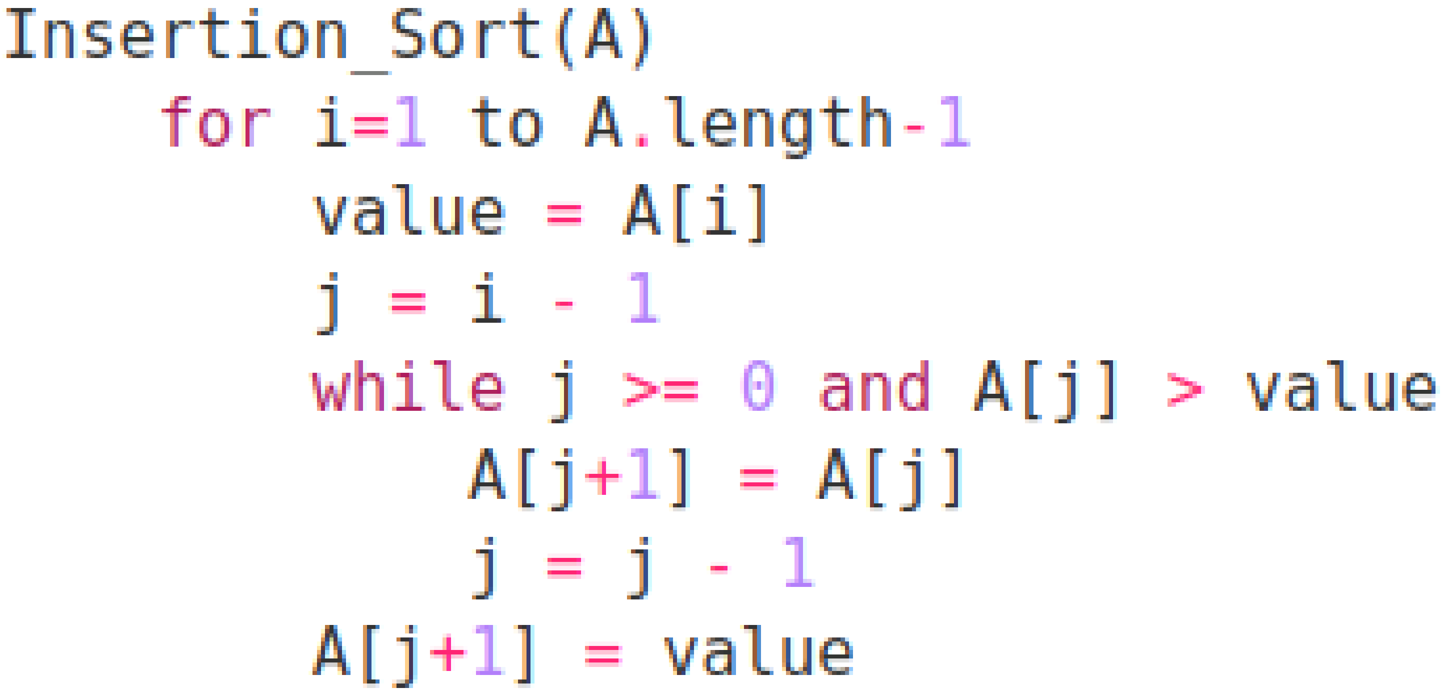
Pass	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K = 1:	−∞	77	33	44	11	88	22	66	55
K = 2:	−∞	77	33	44	11	88	22	66	55
K = 3:	−∞	33	77	44	11	88	22	66	55
K = 4:	−∞	33	44	77	11	88	22	66	55
K = 5:	−∞	11	33	44	77	88	22	66	55
K = 6:	−∞	11	33	44	77	88	22	66	55
K = 7:	−∞	11	22	33	44	77	88	66	55
K = 8:	−∞	11	22	33	44	66	77	88	55
Sorted:	−∞	11	22	33	44	55	66	77	88

Insertion Sort

Sort the following sequence using Insertion Sort:

77	**42**	**35**	**12**	**101**	**5**

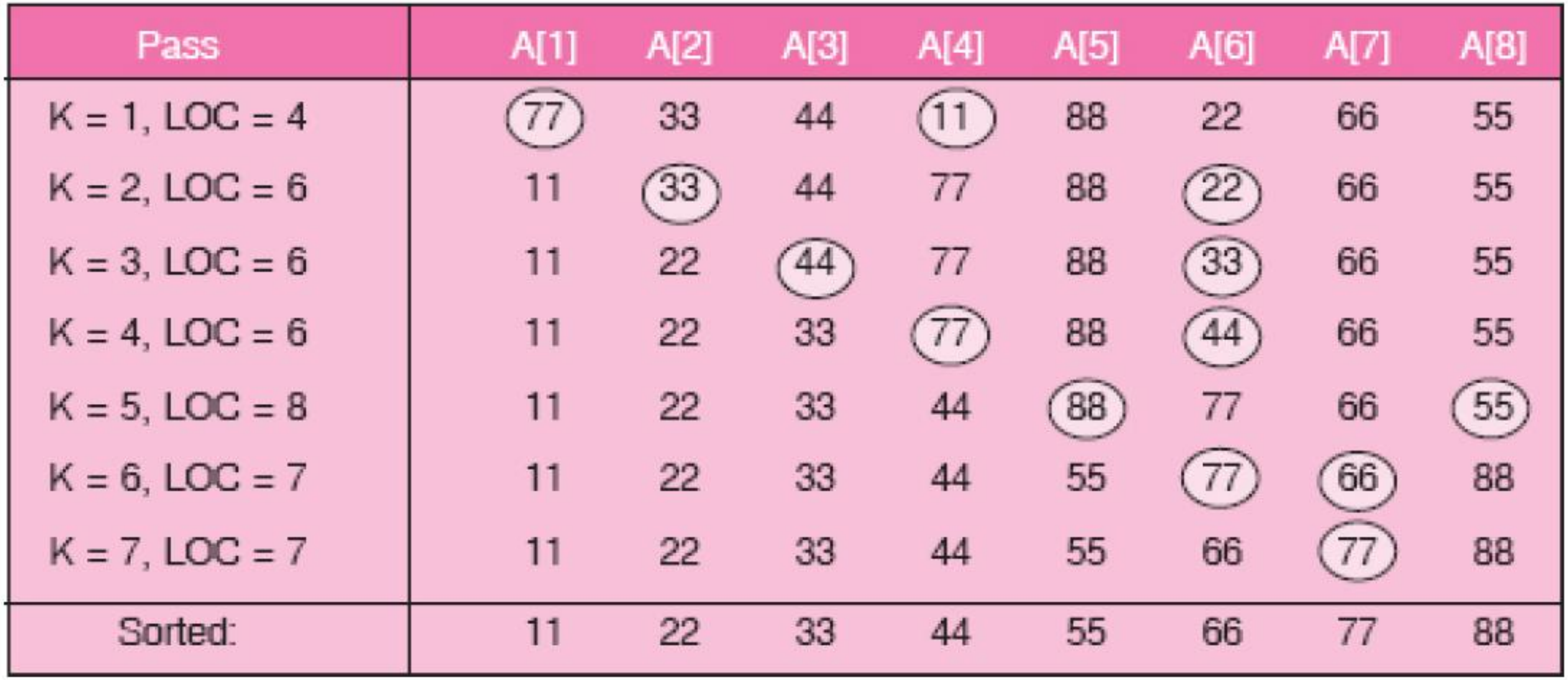
Insertion Sort



```
Insertion_Sort(A)
    for i=1 to A.length-1
        value = A[i]
        j = i - 1
        while j >= 0 and A[j] > value
            A[j+1] = A[j]
            j = j - 1
        A[j+1] = value
```

Selection Sort

Suppose an array A contains 8 elements as follows:
77, 33, 44, 11, 88, 22, 66, 55



Pass	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K = 1, LOC = 4	77	33	44	11	88	22	66	55
K = 2, LOC = 6	11	33	44	77	88	22	66	55
K = 3, LOC = 6	11	22	44	77	88	33	66	55
K = 4, LOC = 6	11	22	33	77	88	44	66	55
K = 5, LOC = 8	11	22	33	44	88	77	66	55
K = 6, LOC = 7	11	22	33	44	55	77	66	88
K = 7, LOC = 7	11	22	33	44	55	66	77	88
Sorted:	11	22	33	44	55	66	77	88

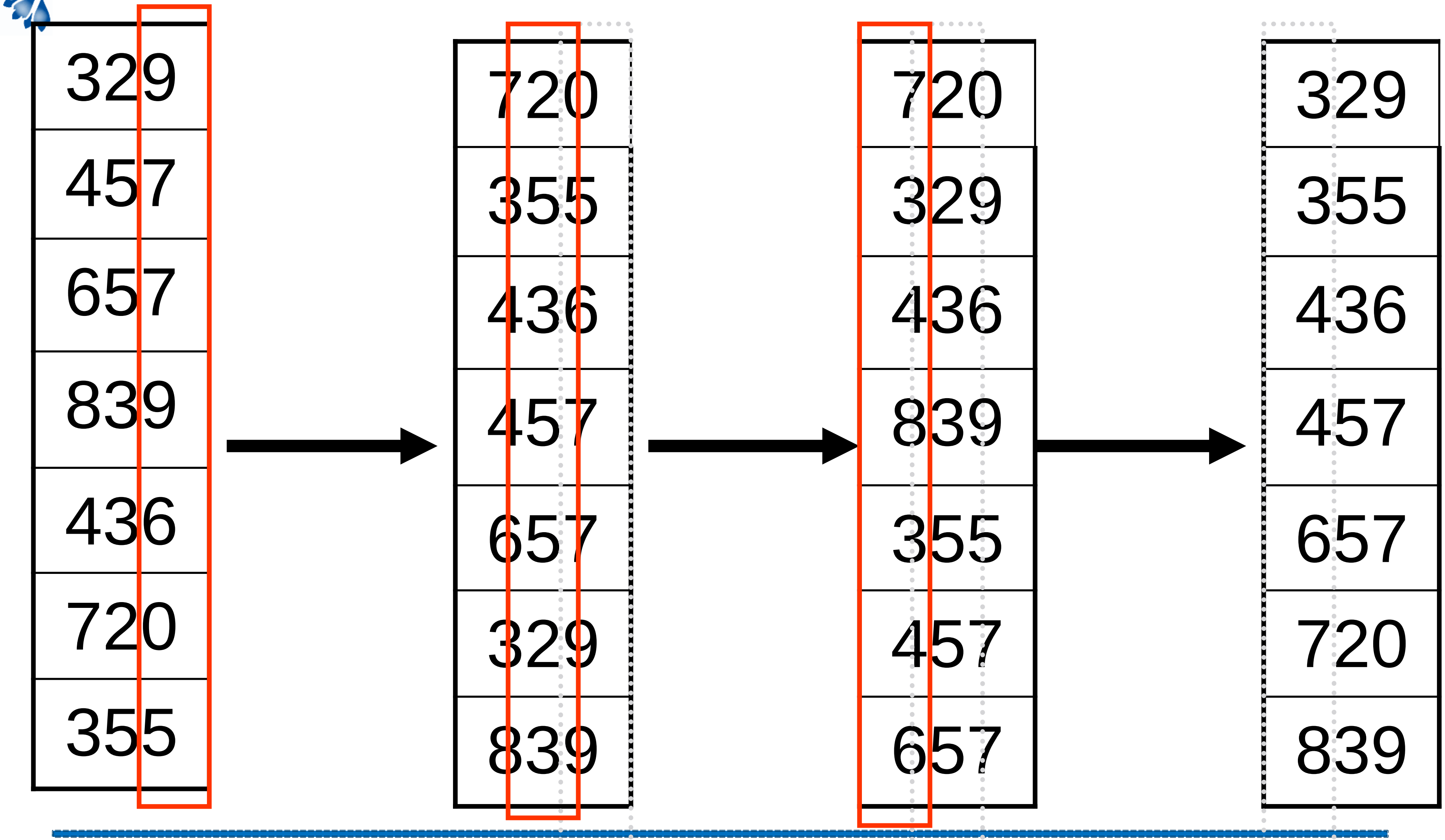
Selection Sort

Input: An array $A[1..n]$ of n elements.

Output: $A[1..n]$ sorted in descending order

```
for $i \leftarrow 1$ to $n - 1$
  min $\leftarrow i$
  for $j \leftarrow i + 1$ to $n$   {Find the $i$ th smallest element.}
      if $A[j] < A[min]$ then
          min $\leftarrow j$
  end for
  if min $\neq i$ then interchange $A[i]$ and $A[\text{min}]$
end for
```

RadiX Sort



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

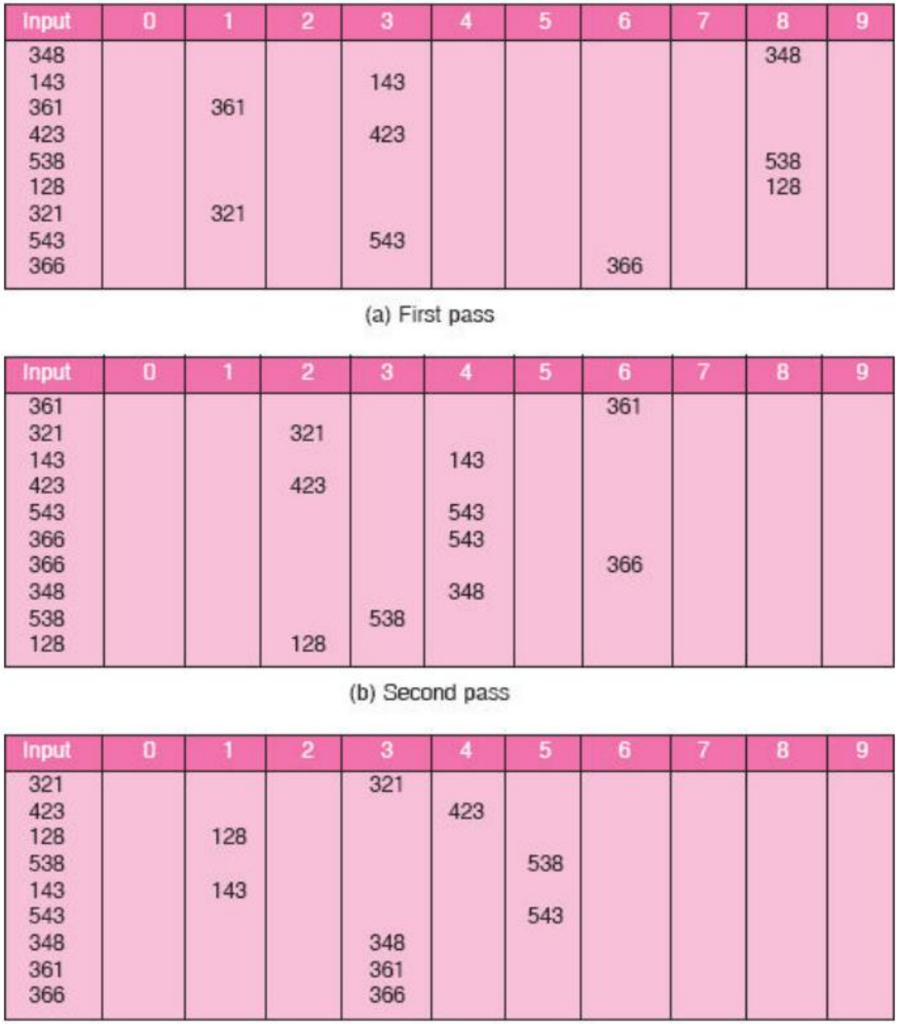
RadiX Sort

Suppose 9 cards are punched as follows:

348, 143, 361, 423, 538, 128, 321, 543, 366

Given to a card sorter, the numbers would be sorted in three phases

RadiX Sort



Input	0	1	2	3	4	5	6	7	8	9
348									348	
143				143						
361		361								
423				423						
538									538	
128									128	
321		321								
543				543						
366							366			

(a) First pass

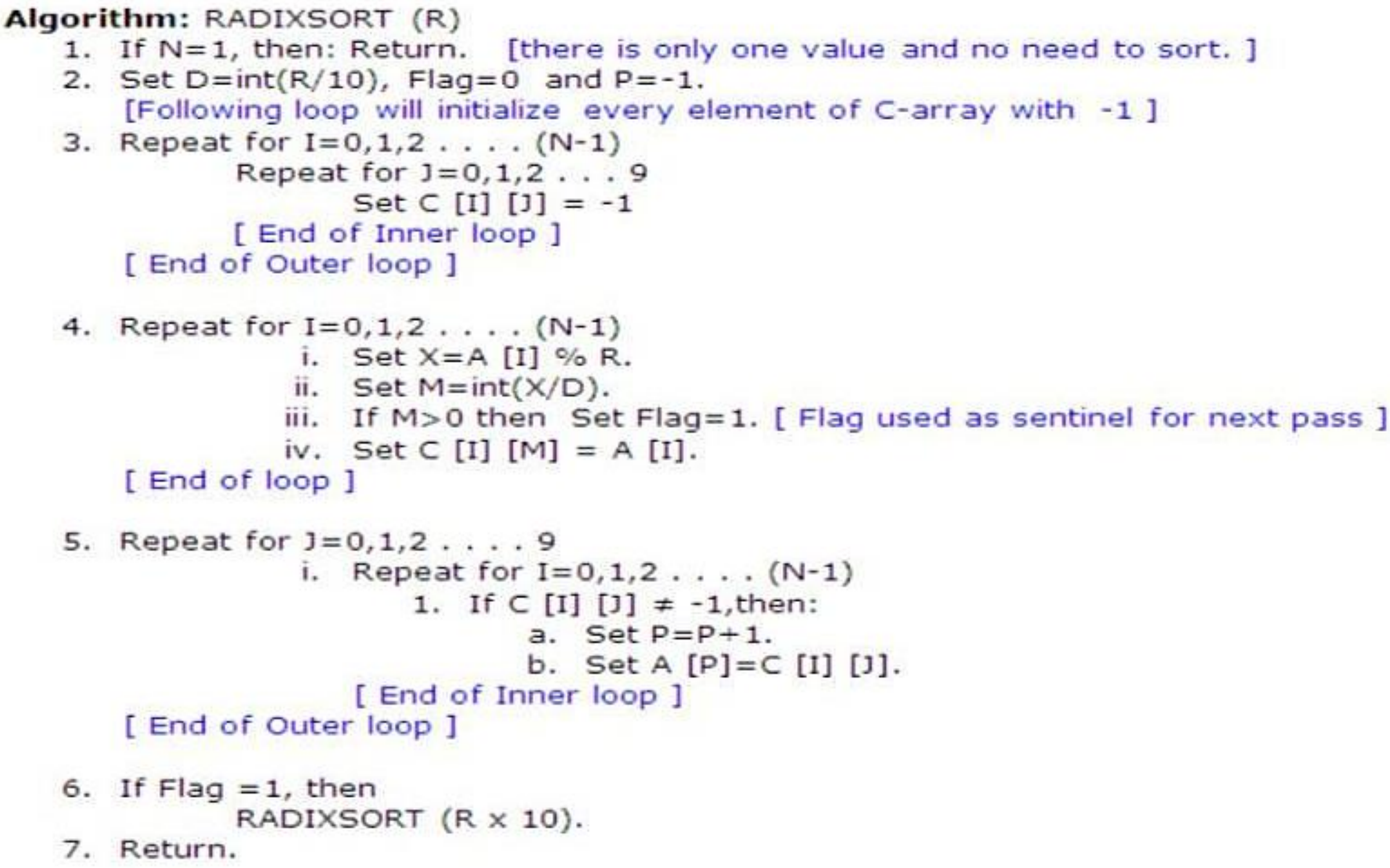
Input	0	1	2	3	4	5	6	7	8	9
361							361			
321			321							
143					143					
423			423							
543					543					
366					543					
366							366			
348					348					
538				538						
128			128							

(b) Second pass

Input	0	1	2	3	4	5	6	7	8	9
321				321						
423					423					
128		128								
538						538				
143		143								
543						543				
348				348						
361				361						
366				366						

(c) Third pass

Radix Sort

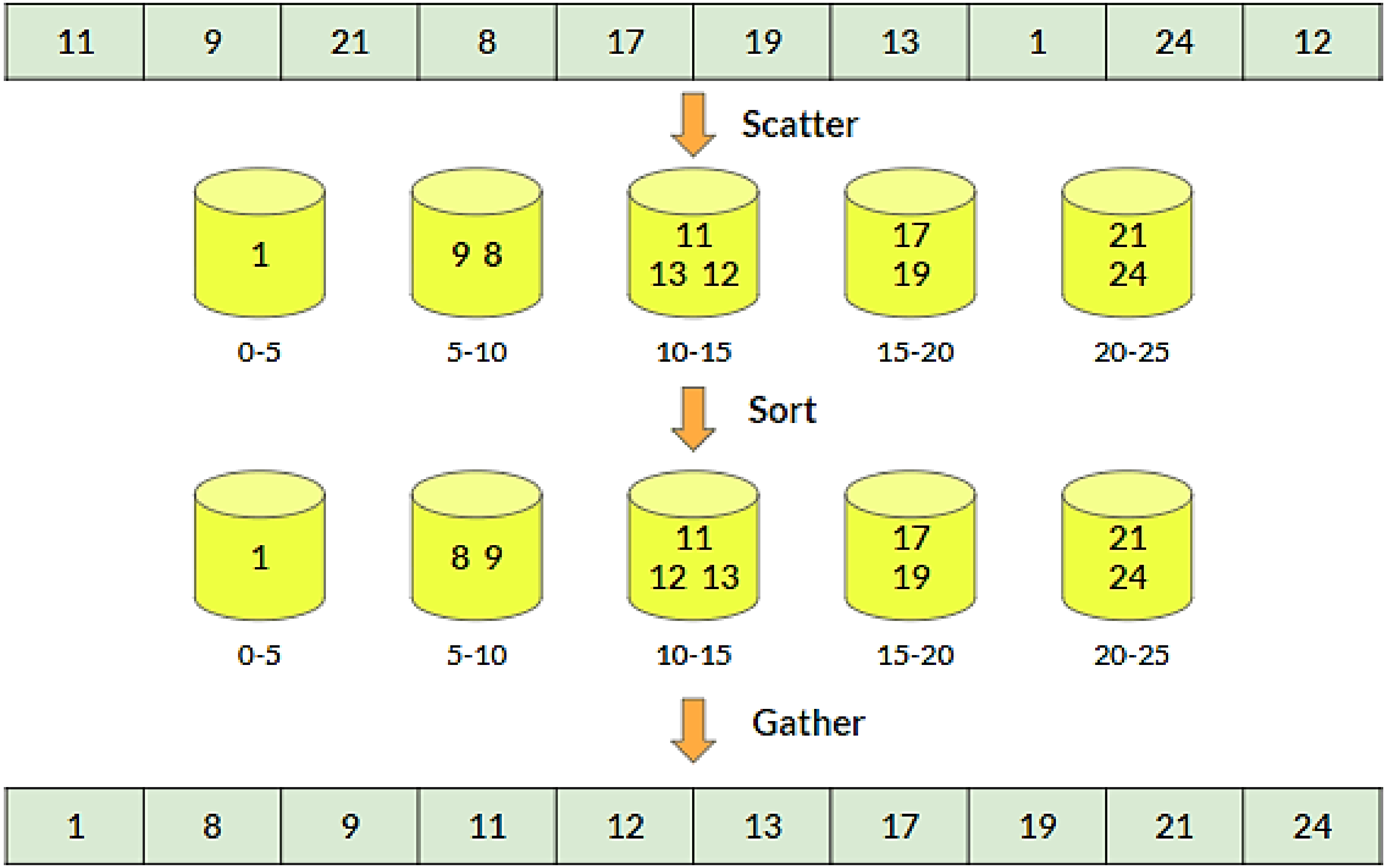


```
Algorithm: RADIXSORT (R)
    1.  If N=1, then: Return.   [there is only one value and no need to sort. ]
    2.  Set D=int(R/10), Flag=0  and P=-1.
        [Following loop will initialize  every element of C-array with  -1 ]
    3.  Repeat for I=0,1,2 . . . . (N-1)
                Repeat for J=0,1,2 . . . 9
                        Set C [I] [J] = -1
                [ End of Inner loop ]
        [ End of Outer loop ]

    4.  Repeat for I=0,1,2 . . . . (N-1)
                     i.   Set X=A [I] % R.
                    ii.   Set M=int(X/D).
                   iii.   If M>0 then  Set Flag=1. [ Flag used as sentinel  for next pass ]
                    iv.   Set C [I] [M] = A [I].
        [ End of loop ]

    5.  Repeat for J=0,1,2 . . . . 9
                     i.   Repeat for I=0,1,2 . . . . (N-1)
                              1.  If C [I] [J] ≠ -1,then:
                                      a.  Set P=P+1.
                                      b.  Set A [P]=C [I] [J].
                          [ End of Inner loop ]
        [ End of Outer loop ]

    6.  If Flag =1, then
                RADIXSORT (R x 10).
    7.  Return.
```

Bucket Sort

- Assumption:
  - the input is generated by a random process that distributes elements uniformly over [0, 1)
- Idea:
  - Divide [0, 1) into k equal-sized buckets ($k=\Theta(n)$)
  - Distribute the n input values into the buckets
  - Sort each bucket (e.g., using quicksort)
  - Go through the buckets in order, listing elements in each one
- **Input:** $A[1 \,.\,.\, n]$, where $0 \le A[i] < 1$ for all i
- **Output:** elements $A[i]$ sorted

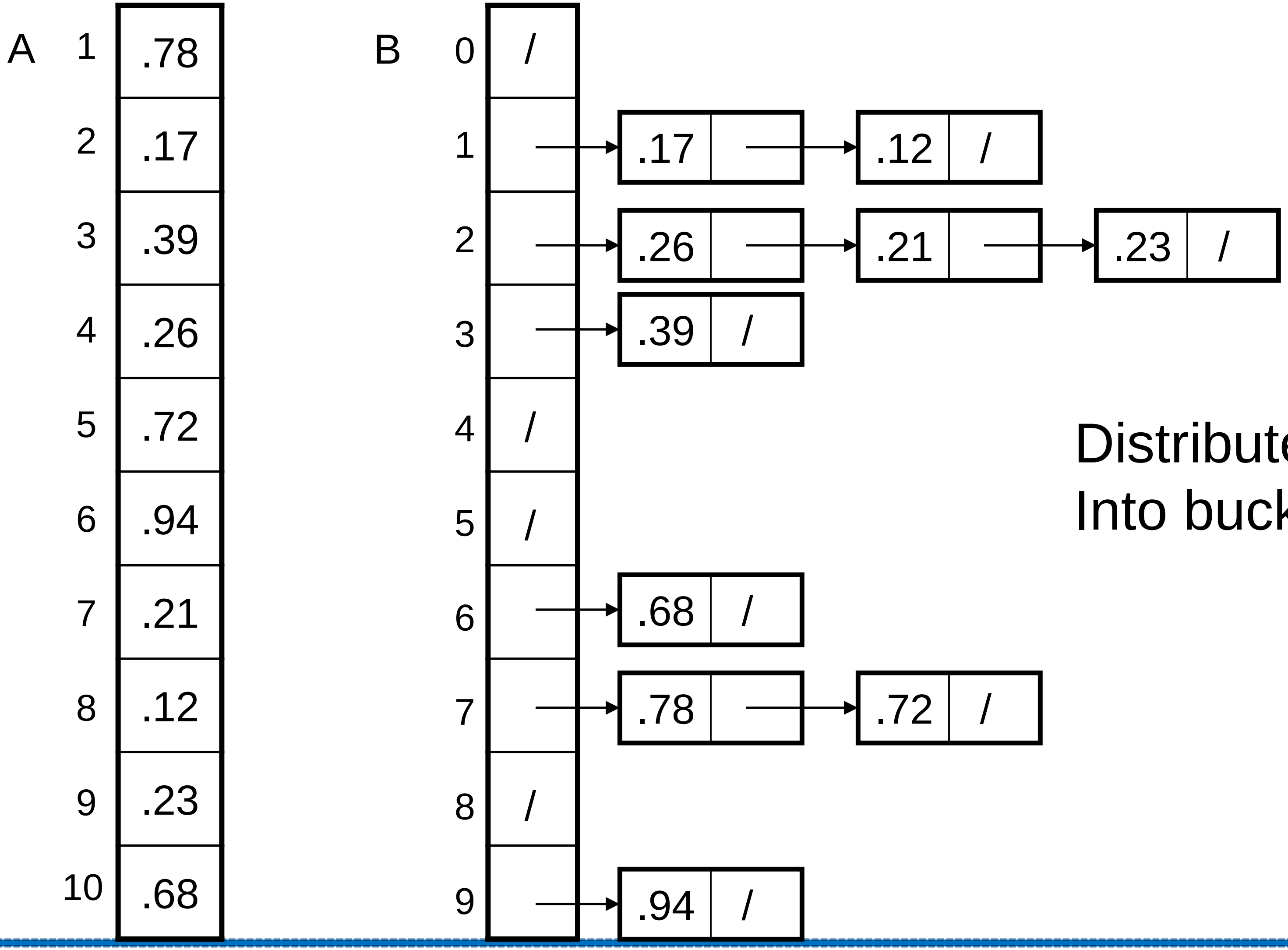
Bucket Sort



11	9	21	8	17	19	13	1	24	12

1	8	9	11	12	13	17	19	21	24

Bucket Sort



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

B

0 /

1 → .17 → .12 /

2 → .26 → .21 → .23 /

3 → .39 /

4 /

5 /

6 → .68 /

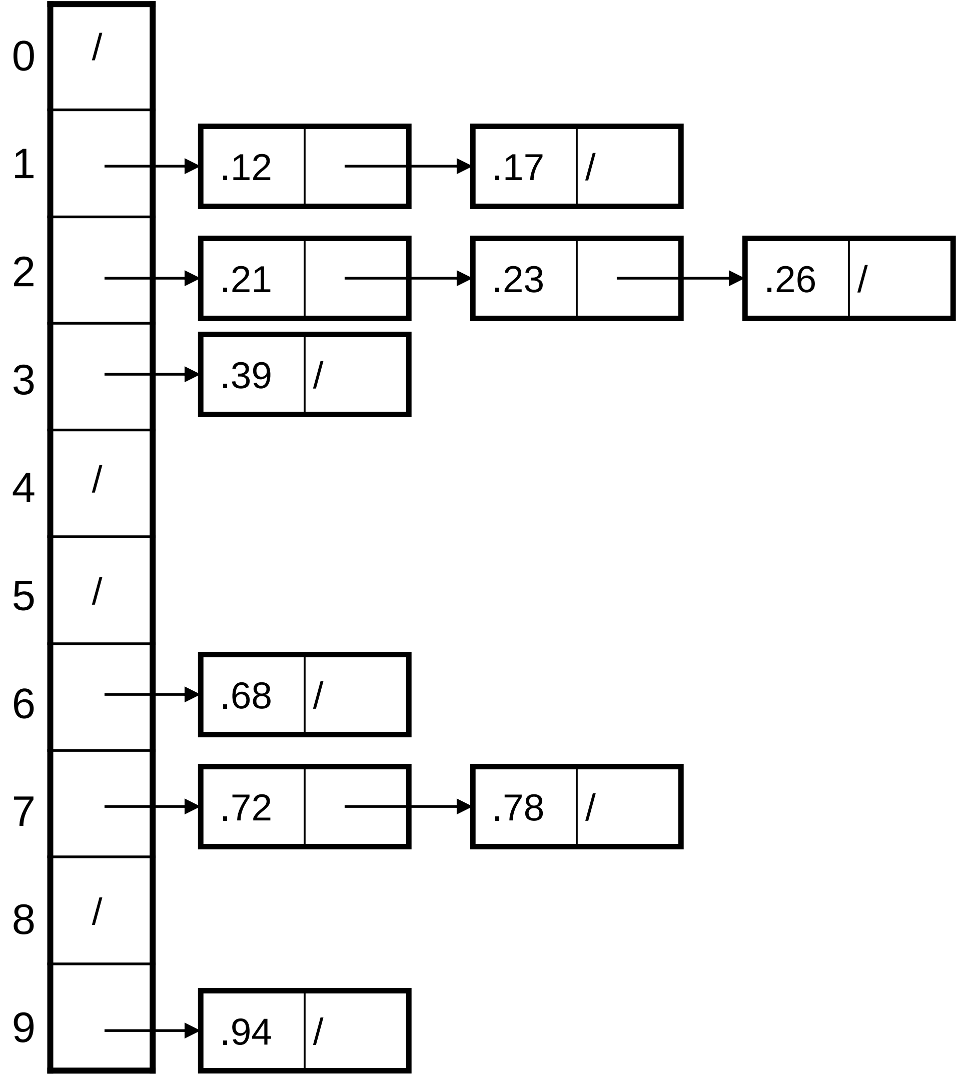
7 → .78 → .72 /

8 /

9 → .94 /

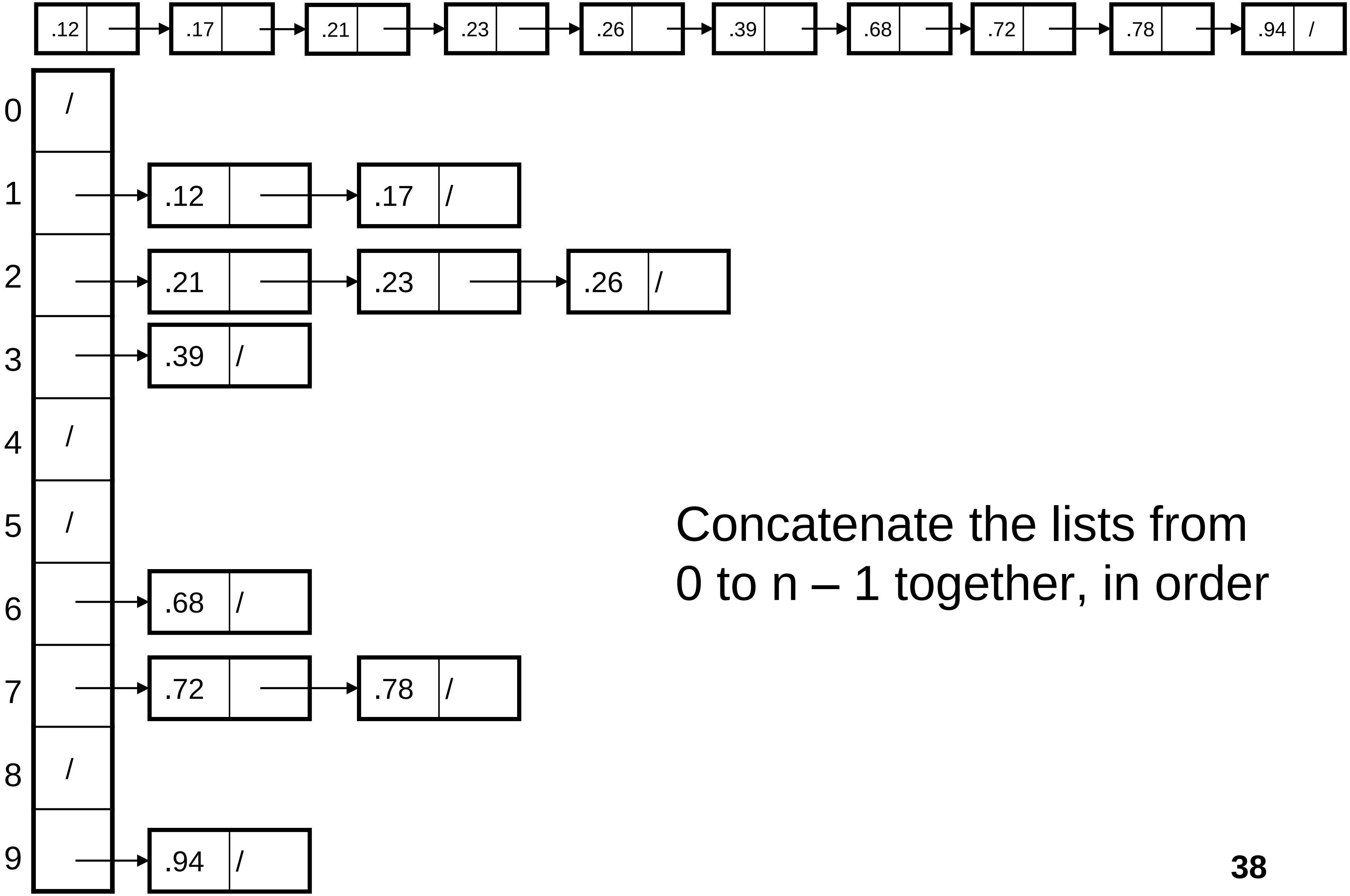
Distribute
Into buckets

Bucket Sort



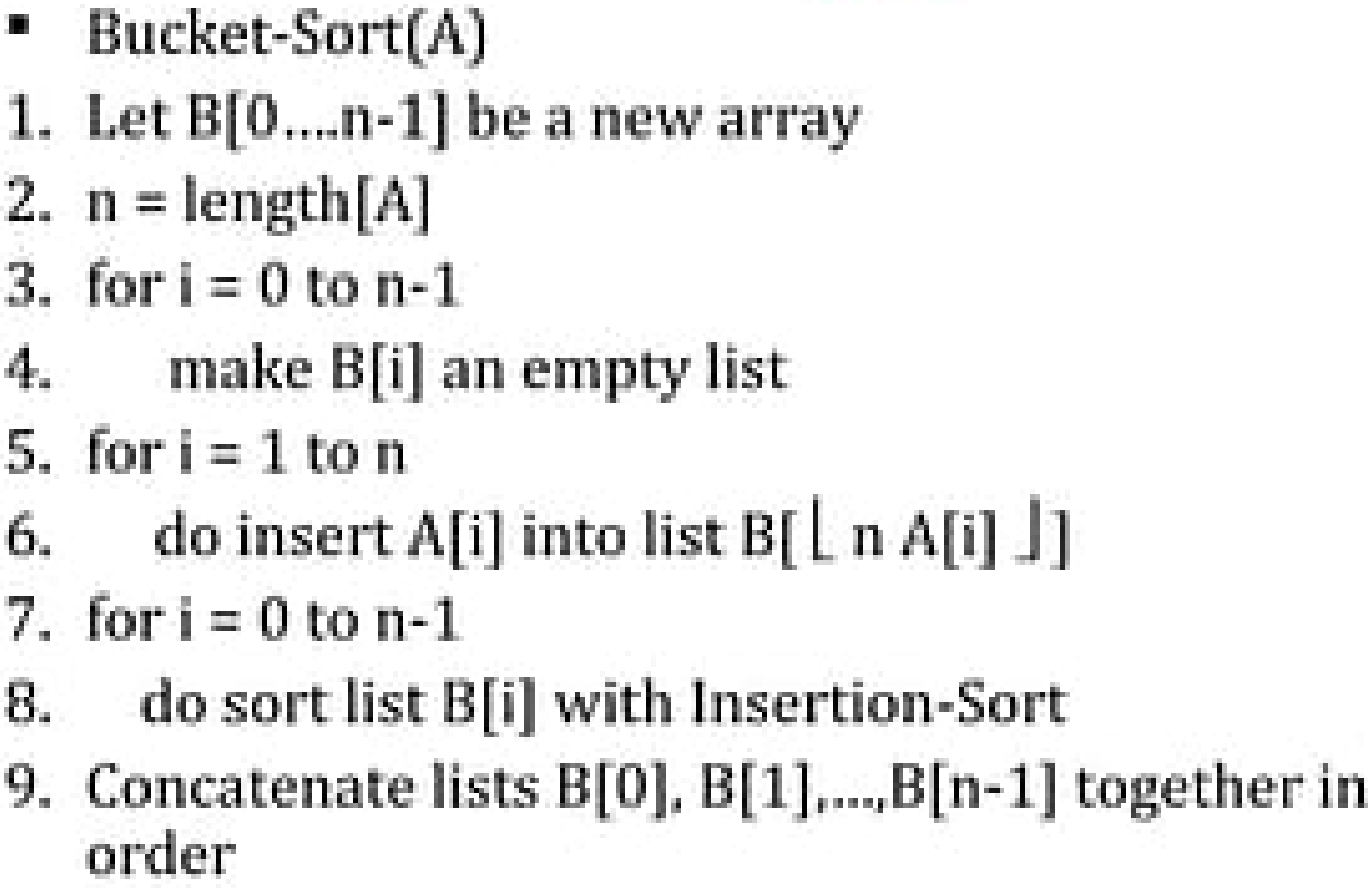
Sort within each bucket

Bucket Sort



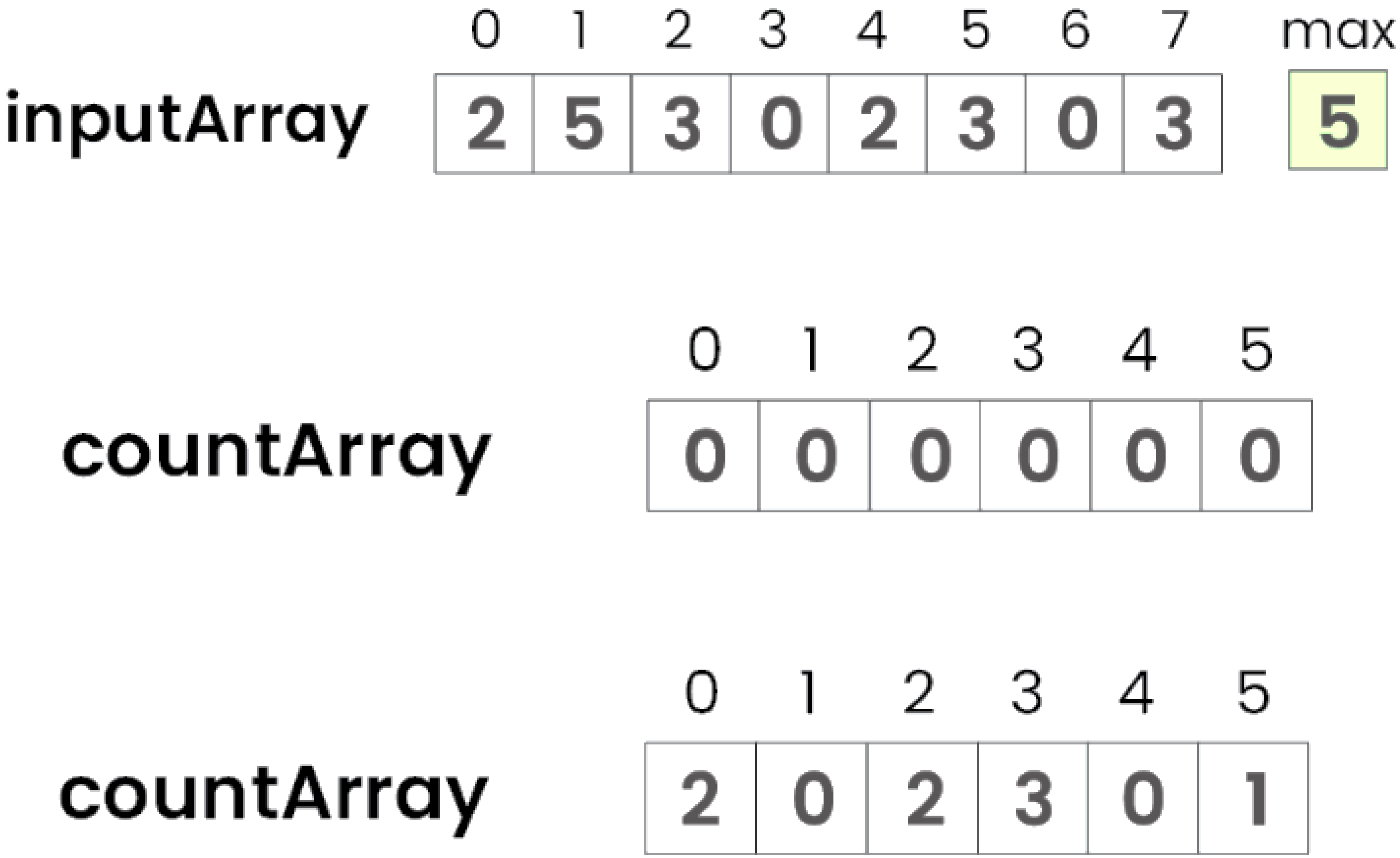
Concatenate the lists from 0 to n – 1 together, in order

Bucket Sort



```
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
```

Counting Sort



	0	1	2	3	4	5	6	7	max
inputArray	2	5	3	0	2	3	0	3	5

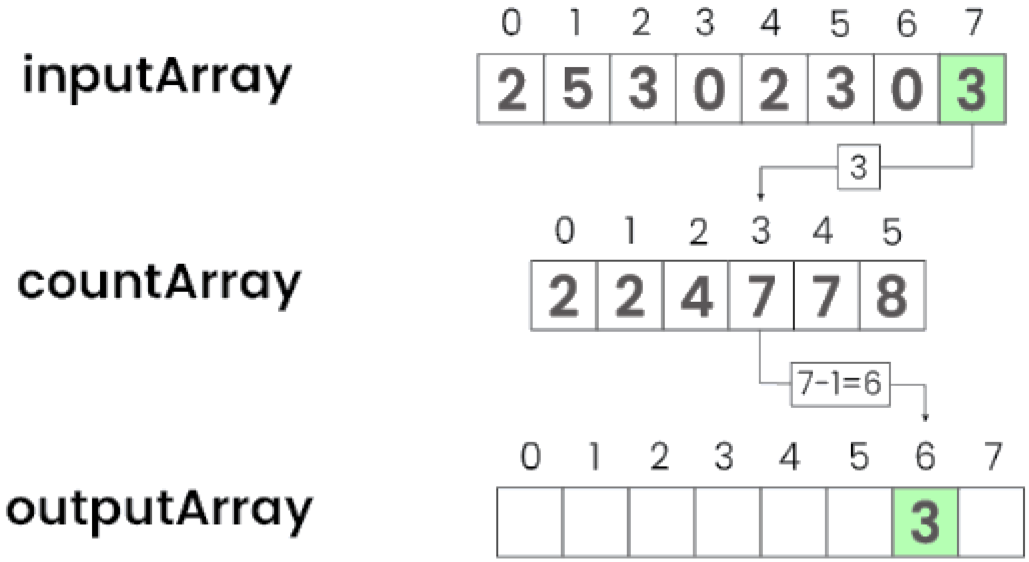
	0	1	2	3	4	5
countArray	0	0	0	0	0	0

	0	1	2	3	4	5
countArray	2	0	2	3	0	1

Counting Sort

countArray

0	1	2	3	4	5
2	2	4	7	7	8



inputArray

0	1	2	3	4	5	6	7
2	5	3	0	2	3	0	3

3

countArray

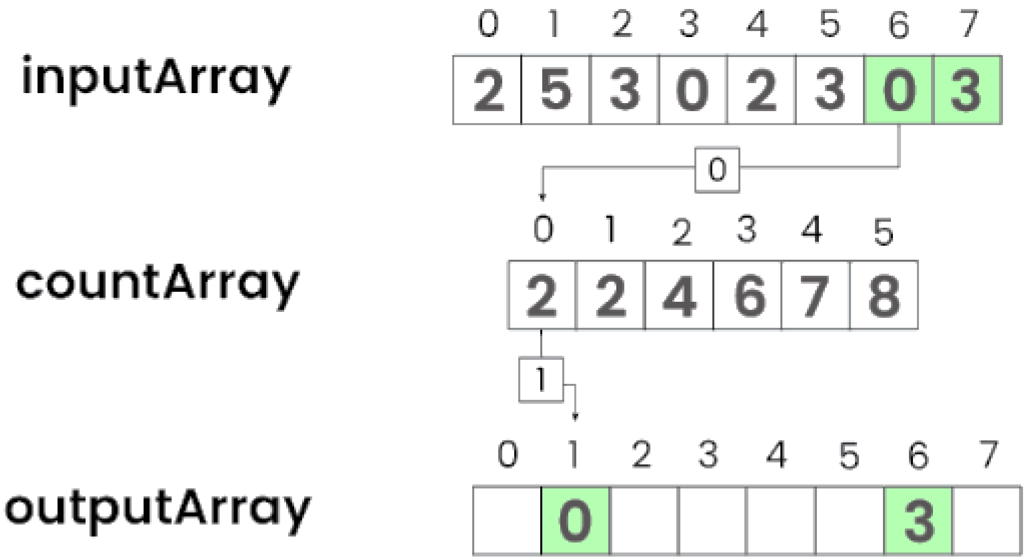
0	1	2	3	4	5
2	2	4	7	7	8

7-1=6

outputArray

0	1	2	3	4	5	6	7
						3	

Counting Sort



inputArray

0	1	2	3	4	5	6	7
2	5	3	0	2	3	0	3

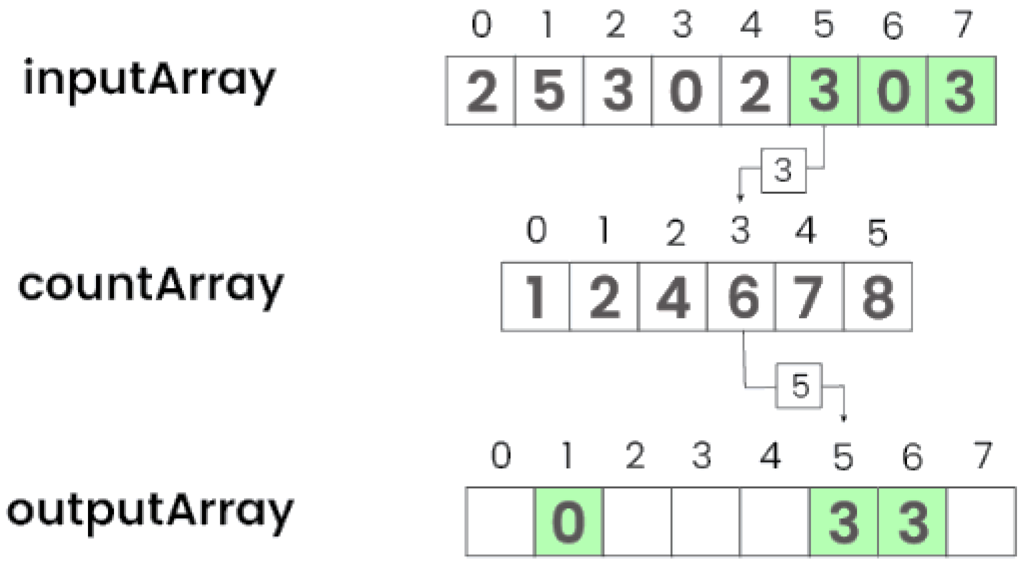
countArray

0	1	2	3	4	5
2	2	4	6	7	8

outputArray

0	1	2	3	4	5	6	7
	0					3	

Counting Sort



inputArray

0	1	2	3	4	5	6	7
2	5	3	0	2	3	0	3

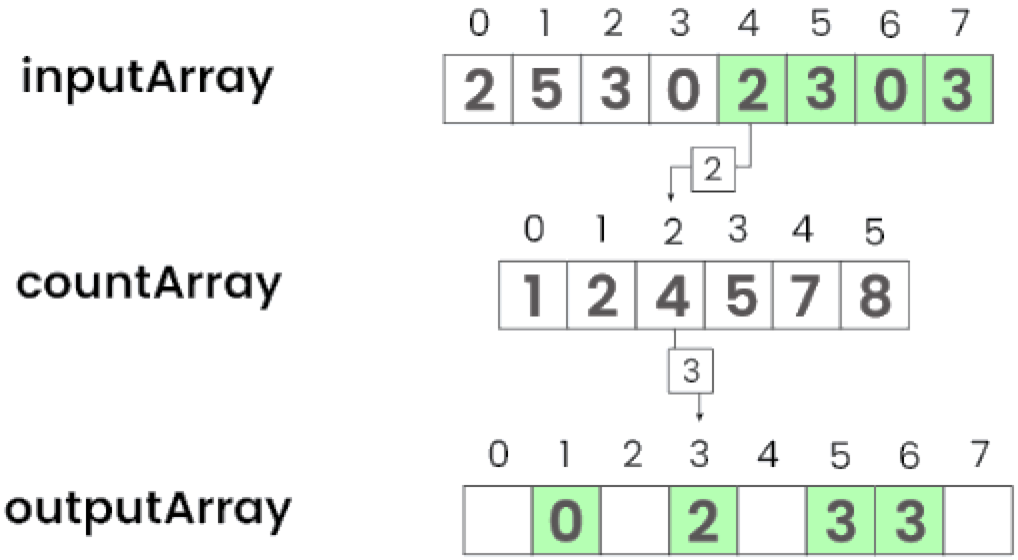
countArray

0	1	2	3	4	5
1	2	4	6	7	8

outputArray

0	1	2	3	4	5	6	7
	0				3	3	

Counting Sort



inputArray

0	1	2	3	4	5	6	7
2	5	3	0	2	3	0	3

2

countArray

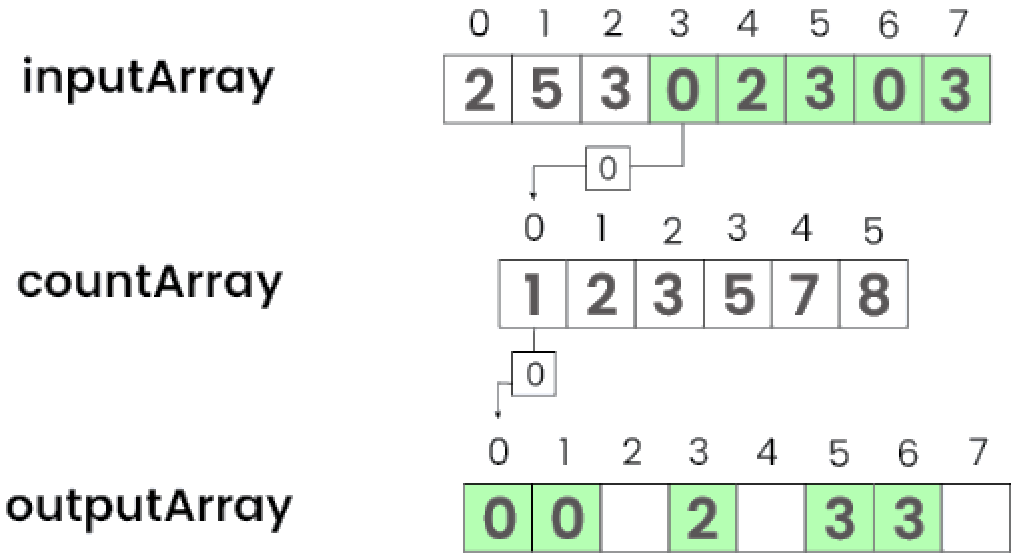
0	1	2	3	4	5
1	2	4	5	7	8

3

outputArray

0	1	2	3	4	5	6	7
	0		2		3	3	

Counting Sort



inputArray

0	1	2	3	4	5	6	7
2	5	3	0	2	3	0	3

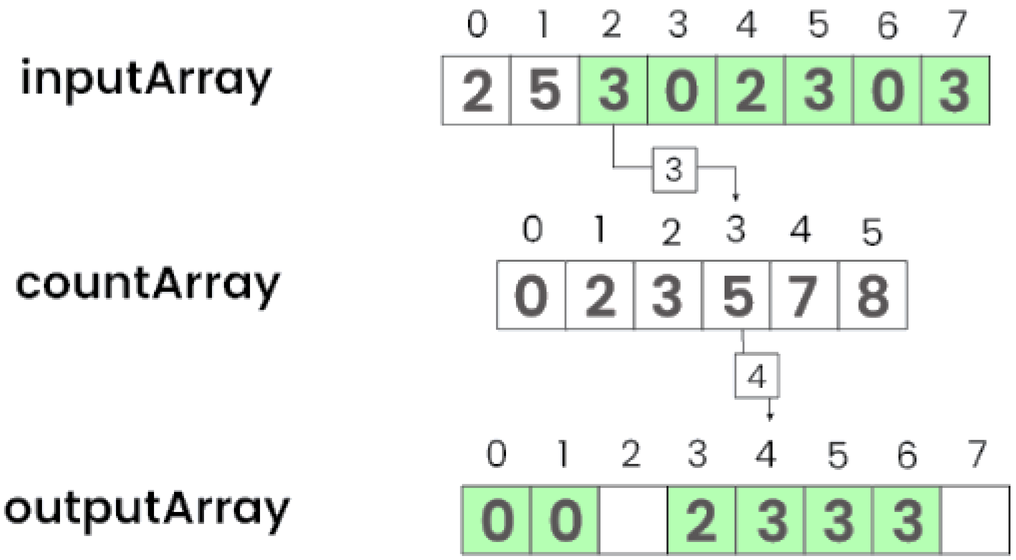
countArray

0	1	2	3	4	5
1	2	3	5	7	8

outputArray

0	1	2	3	4	5	6	7
0	0		2		3	3	

Counting Sort



inputArray

0	1	2	3	4	5	6	7
2	5	3	0	2	3	0	3

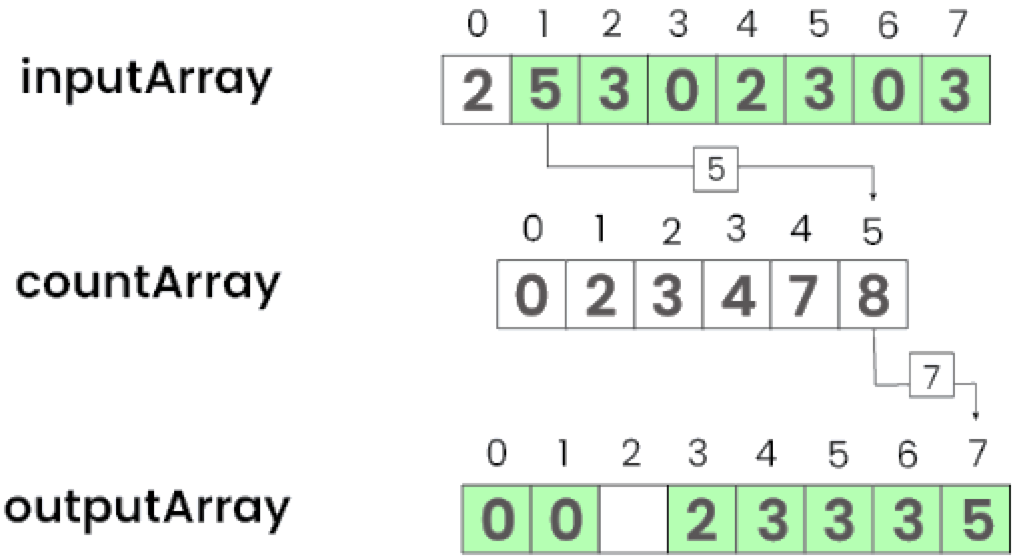
countArray

0	1	2	3	4	5
0	2	3	5	7	8

outputArray

0	1	2	3	4	5	6	7
0	0		2	3	3	3	

Counting Sort



inputArray

0	1	2	3	4	5	6	7
2	5	3	0	2	3	0	3

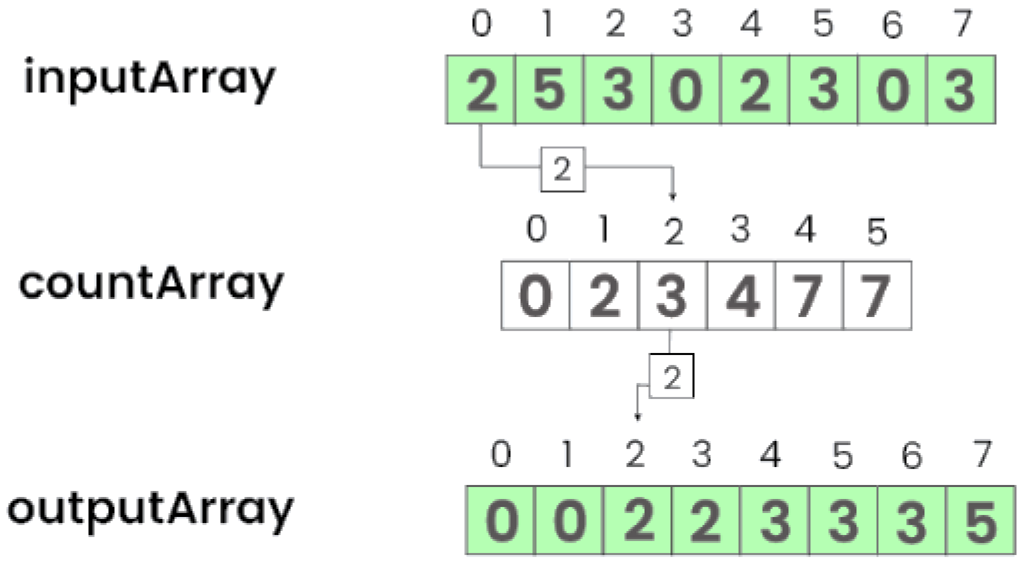
countArray

0	1	2	3	4	5
0	2	3	4	7	8

outputArray

0	1	2	3	4	5	6	7
0	0		2	3	3	3	5

Counting Sort



inputArray

0	1	2	3	4	5	6	7
2	5	3	0	2	3	0	3

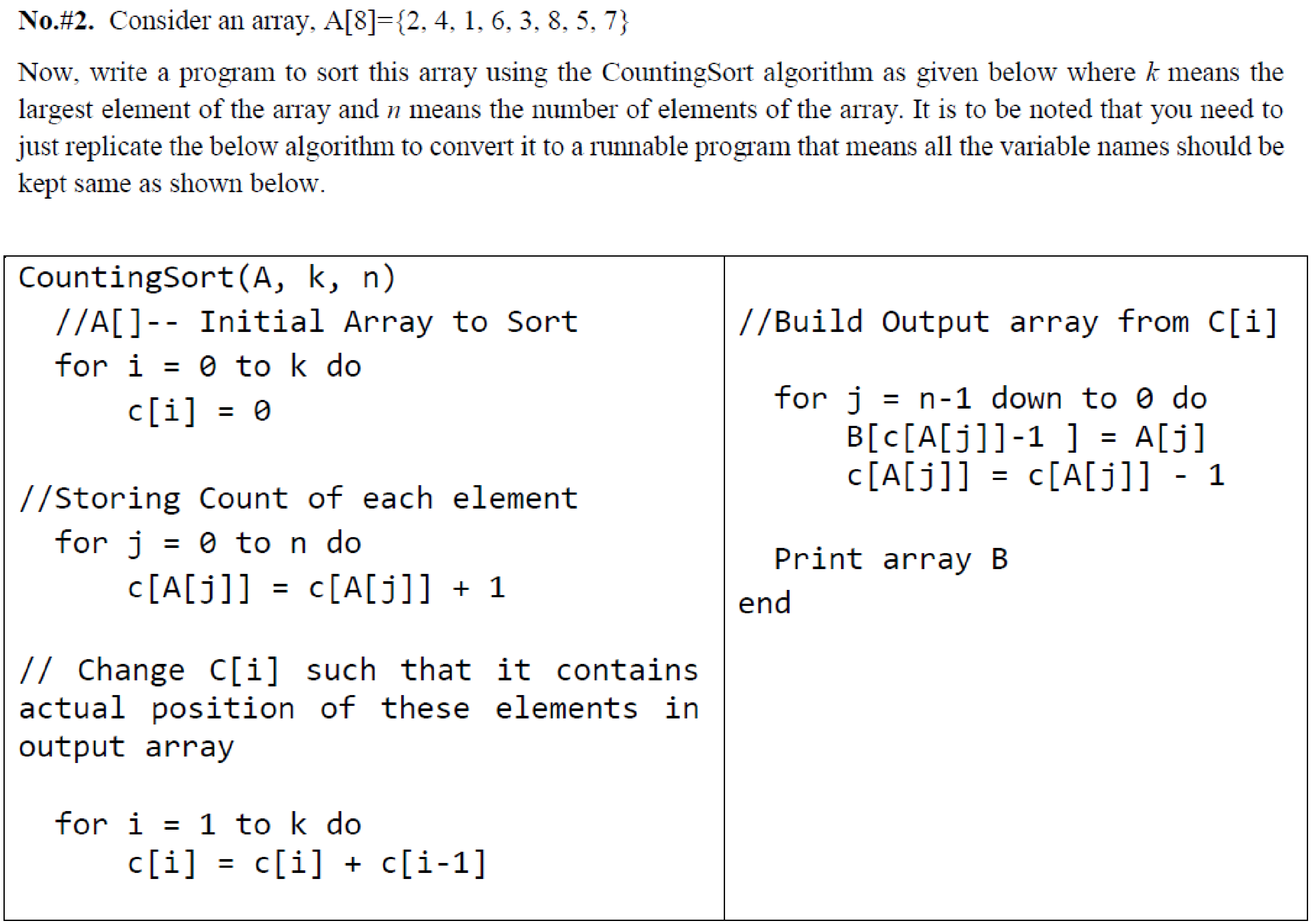
countArray

0	1	2	3	4	5
0	2	3	4	7	7

outputArray

0	1	2	3	4	5	6	7
0	0	2	2	3	3	3	5

Counting Sort



No.#2. Consider an array, A[8]={2, 4, 1, 6, 3, 8, 5, 7}

Now, write a program to sort this array using the CountingSort algorithm as given below where *k* means the largest element of the array and *n* means the number of elements of the array. It is to be noted that you need to just replicate the below algorithm to convert it to a runnable program that means all the variable names should be kept same as shown below.

```
CountingSort(A, k, n)
  //A[]-- Initial Array to Sort
  for i = 0 to k do
      c[i] = 0

//Storing Count of each element
  for j = 0 to n do
      c[A[j]] = c[A[j]] + 1

// Change C[i] such that it contains
actual position of these elements in
output array

  for i = 1 to k do
      c[i] = c[i] + c[i-1]
```

```
//Build Output array from C[i]

  for j = n-1 down to 0 do
      B[c[A[j]]-1 ] = A[j]
      c[A[j]] = c[A[j]] - 1

  Print array B
end
```

Thank You!