

# Lecture 24 Exercise Solutions

Mark Eramian

## Exercise 1

- a) Works best for numbers in a small range.
- b) Time complexity is  $O(n)$  because each key gets put in a bucket, then each key gets removed from a bucket and appended to the final sequence. Let  $m$  be the number of possible values that a key could take on. Since we need a bucket for each possible value, the space complexity of bucket sort is  $O(m)$ . This is why bucket sort is best for keys that have a small range of values.
- c) After putting keys in buckets:

	1, 1	2, 2				6, 6	7	8	9, 9
0	1	2	3	4	5	6	7	8	9

After removing elements from buckets from left to right and appending them to output array:

0	1	2	3	4	5	6	7	8	9
1	1	2	2	6	6	7	8	9	9

## Exercise 2

- a) Integer array sorted by MSD radix algorithm:

0	1	2	3	4	5	6	7
265	28	42	614	7	83	916	95

- b) Integer array sorted by LSD radix algorithm:

0	1	2	3	4	5	6	7
7	28	42	83	95	265	614	916

## Exercise 3

a) String array sorted by MSD radix algorithm:

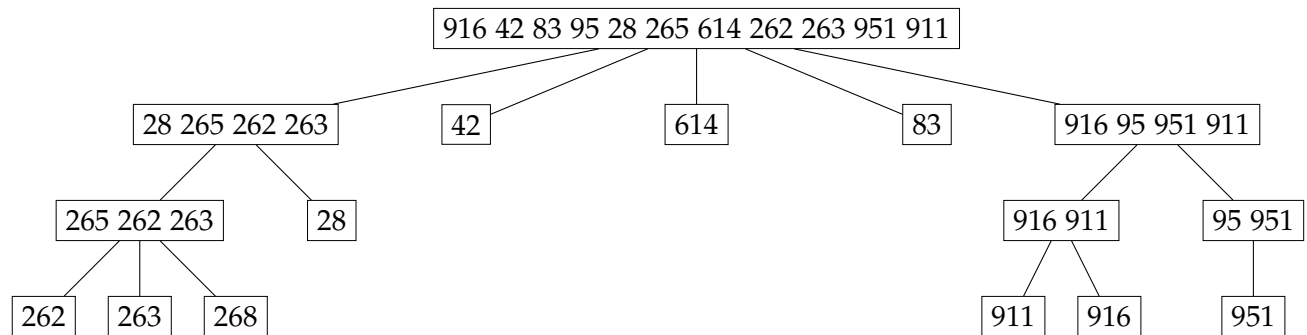
0	1	2	3	4	5	6	7
"coming"	"die"	"is"	"or"	"win"	"winter"	"you"	"you"

b) String array sorted by LSD radix algorithm:

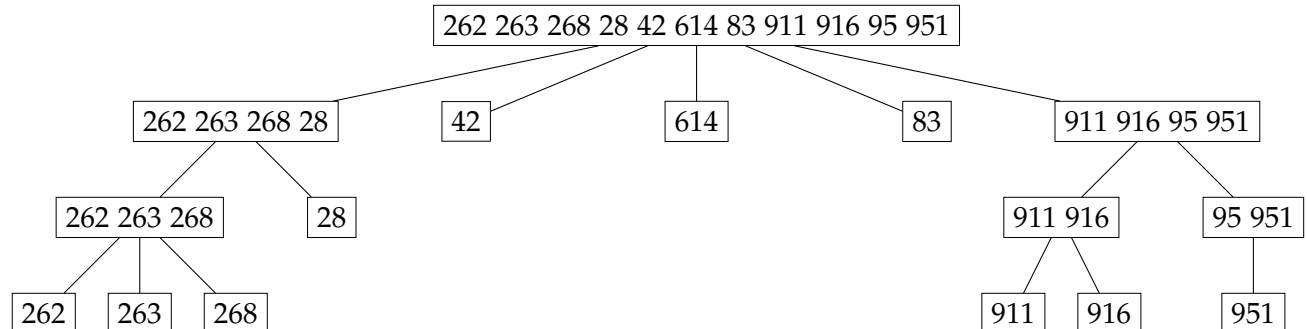
0	1	2	3	4	5	6	7
"is"	"or"	"die"	"win"	"you"	"you"	"winter"	"coming"

## Exercise 4

This is a bit hard to visualize. First, here is the recursion tree showing the structure of recursive calls and their **input** sequences. Note that the left to right order of recursive calls (siblings) corresponds to recursing on the buckets in order from digit 0 to digit 9. Nodes at level 1 are bucket-sorted on the first digit of keys, level 2 on the second digit of keys, and level 3 the third digit of keys (remember: the root is level 0).



Now here is the recursion tree showing the **output** sequences of each recursive call. Basically as the recursion unwinds, siblings are concatenated to sequences in the proper order (like in the recursion tree for merge sort).



## Exercise 5

1. Input Array

0	1	2	3	4	5	6	7	8	9	10
916	42	83	95	28	265	614	262	263	951	911

2. After sorting on the last digit.

0	1	2	3	4	5	6	7	8	9	10
951	911	42	262	83	263	614	95	265	916	28

2. After sorting on the second-last digit. Keys are now sorted by the last 2 digits.

0	1	2	3	4	5	6	7	8	9	10
911	614	916	28	42	951	262	263	265	83	95

2. After sorting on the third-last digit (if no such digit, treat as a leading 0). Keys are now sorted by the last 3 digits.

0	1	2	3	4	5	6	7	8	9	10
028	042	083	095	262	263	265	614	911	916	951