




# Non-Comparative Sorting

Part 7

1



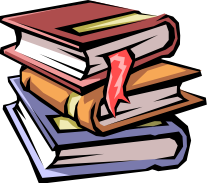
# Dictionaries

the "ADT"... not in a human one!

2

## Moving Past Arrays....

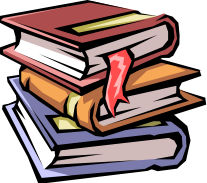
- A *collection* is general term for a group of data items
- So, this can include arrays, linked lists, stacks, queues, and much more
- So far, we have just used arrays – which are indexed by an integer



3

## Moving Past Arrays....


- Are there are other ways to index data?
- Yes.
  - any object can be used as an index
  - e.g. strings, integers, pictures, etc...



4

## Dictionaries


- Collections of objects indexed by other objects are called *dictionaries*
- They have a few alternative names...
  - keyed tables
  - symbol tables
  - maps



5

## Dictionary Terminology

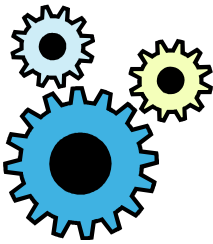
- The objects that are used for indices are called *keys*
- The objects that are accessed using the key are called *values*



6

## Implementing Dictionaries

- There are numerous approaches to implementing dictionaries
- Key-value** structure
  - a class stores a key object and value object
  - this can be stored in any data structure we have covered

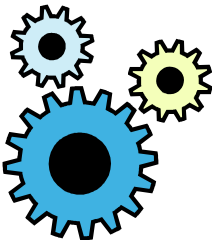


Fall 2024 Sacramento State - CS&E - CS110 7

7

## Implementing Dictionaries

- Using a linked list
  - adding takes  $O(1)$
  - access is  $O(n)$
- Unsorted array
  - add is  $O(n)$  – have to resize
  - access is  $O(n)$
- Sorted array
  - add is  $O(n)$  – have to resize
  - access is  $O(\log n)$



Fall 2024 Sacramento State - CS&E - CS110 8

8

## This Ain't So Good

- So, adding in to an array is  $O(n)$
- Arrays seem like a poor approach
- Is there a better way to store dictionary data? Keeping adding close to  $O(1)$ ?
- ... and keep access at  $O(\log n)$
- Perhaps, we will learn that soon....

Fall 2024 Sacramento State - CS&E - CS110 9


9

## Databases vs. Dictionaries

- Dictionaries...
  - have a single key
  - that key is the only way to access data
  - key returns a single value
- Databases...
  - may have multiple keys (e.g. SSN, name, age, etc...)
  - may return multiple values

Fall 2024 Sacramento State - CS&E - CS110 10

10



## Bucket Sort


The key to sorting is the keys

Fall 2024 Sacramento State - CS&E - CS110 11

11

## Bucket Sort

- The *Bucket Sort* is a fast sorting algorithm that is non-comparative.
- Rather than comparing objects, it uses mathematical properties of their keys



Fall 2024 Sacramento State - CS&E - CS110 12

12

## Bucket Sort

- The most basic algorithm creates a "bucket" for **each** of the different key values
- This "bucket" often takes the form of a queue or list
- Each item in the array is placed into the buckets based on their key



Fall 2024

Sacramento State - Oak - CS5 130

13

13

## Bucket Sort

- Then, each bucket is emptied, in order, back into the array
- This sorts the items, but algorithm has considerable storage requirements – *often making it impactful*



Fall 2024

Sacramento State - Oak - CS5 130

14

14

## Bucket Sort – Fill the Buckets

```
for (i = minKey; i <= maxKey; i++)
    bucket[i] = new Queue()
end for

for (i = 0; i < count; i++)
    bucket[array[i].key].enqueue(array[i])
end for
```

Create Buckets

Fill buckets

Fall 2024

Sacramento State - Oak - CS5 130

15

15

## Bucket Sort – Store Back Into Array

```
j = 0;
for (i = minKey; i <= maxKey; i++)
    while (! bucket[i].isEmpty )
        array[j] = bucket[i].dequeue()
        j++
    end while
end for
```

Empty buckets, in order, to array

Fall 2024

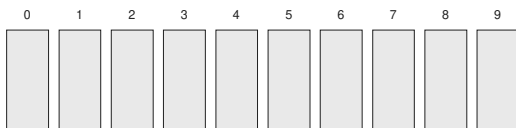
Sacramento State - Oak - CS5 130

16

16

## Bucket Sort: Scatter to Buckets

7 4 1 5 4 2 9 1 6 2 5 7



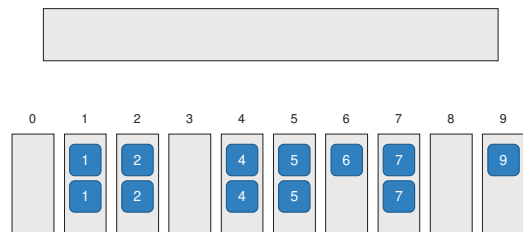
Fall 2024

Sacramento State - Oak - CS5 130

17

17

## Bucket Sort: Buckets Filled



Fall 2024

Sacramento State - Oak - CS5 130

18

18

### Bucket Sort: Gather Into Array

Small to large

0 1 2 3 4 5 6 7 8 9

1 2 4 5 6 7 9

1 2 4 5 7 9

Fall 2024 Sacramento State - Oak - CS110 19

19

### Bucket Sort: Complete

1 1 2 2 4 4 5 5 6 7 7 9

0 1 2 3 4 5 6 7 8 9

Fall 2024 Sacramento State - Oak - CS110 20

20

### Too Many Buckets!

- If we use a bucket for each key, the number of buckets can be **huge!**
- e.g. 32-bit key requires 4,294,967,296 buckets
- So, we need to choose buckets will accept multiple keys within a range

Fall 2024 Sacramento State - Oak - CS110 21

21

### Too Many Buckets

- Naturally, these buckets will contain unsorted keys
- So, we can sort the bucket once it is full
- ... then empty the sorted buckets back into the array

Fall 2024 Sacramento State - Oak - CS110 22

22

### A Better Approach....

1. Fill each bucket with a range of keys
2. Sort each bucket
3. Empty the buckets, in order, into the array

Fall 2024 Sacramento State - Oak - CS110 23

23

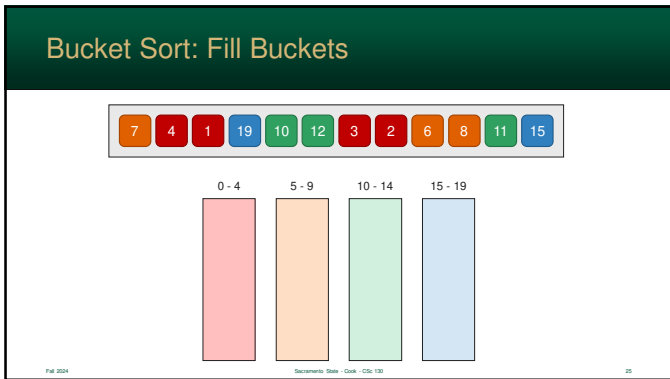
### Bucket Sort: Fill Buckets

7 4 1 19 10 12 3 2 6 8 11 15

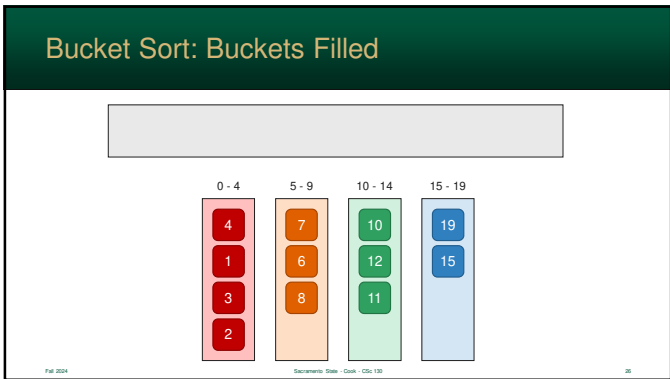
0 - 4 5 - 9 10 - 14 15 - 19

Fall 2024 Sacramento State - Oak - CS110 24

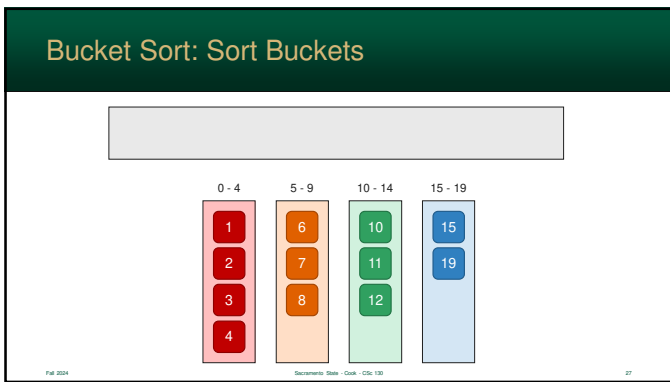
24



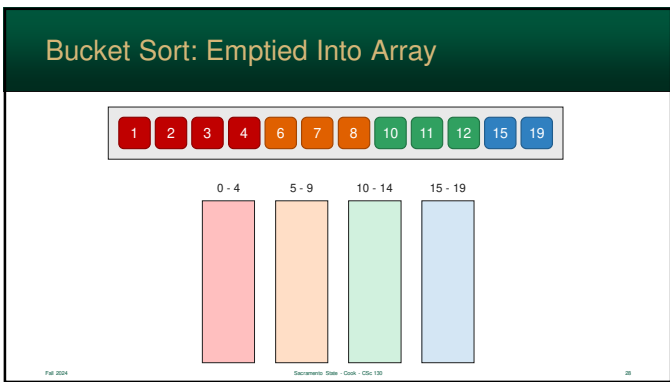
25



26



27



28

### Other Bucket Sort Variations

- Proxmap Sort
  - almost identical to the basic Bucket Sort
  - items are sorted immediately when placed in the bucket – usually an Insertion Sort
- Histogram Sort (aka Counting Sort)
  - does an initial scan of the array and creates buckets the exact size that they will be filled
  - greatly minimalizes overhead

29

### Other Bucket Sort Variations

- Postman's Sort
  - very similar to the next sort we cover: Radix
  - sorts items by "category" of the key
- Shuffle Sort
  - array is recursively sub-divided, sorted, and merged/concatenated when complete
  - 2-bucket Shuffle Sort is essentially a Quick Sort with the pivot acting as divider between the two buckets

30

## Bucket Summary

Bucket Sort	
Time Average	$O(n + (n^2 / b) + b)$ <i>where b is the # of buckets</i>
Time Best	$O(n)$ <i>when <math>b \approx n</math></i>
Time Worst	$O(n^2)$ <i>if Insertion Sort is used</i>
Auxiliary space	$O(b + n)$
Stable	Yes
Online?	Yes (bucket filling stage only)

Fall 2024

Sacramento State - CS&E - CS110

31

31



## The 1890 Census Crisis

Computer Science to the rescue!

32

## The 1890 Census Crisis

- United States Constitution:
  - population must be calculated every 10 years
  - used in the House of Representatives
- Before the 1890 Census Crisis, all this counting was **done by hand**...



Fall 2024

Sacramento State - CS&E - CS110

33

33

## The 1890 Census Crisis

- There were too many people...
  - 1880 Census **barely** made it within the 10-year window
  - U.S. population had continued to grow, and it could **not** be counted in 10 years
- The U.S. was still healing from the Civil War... *failing to represent each state fairly could have resulted in another war*

Fall 2024

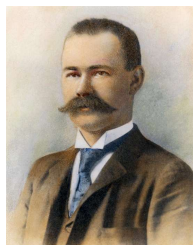
Sacramento State - CS&E - CS110

34

34

## Herman Hollerith to the Rescue

- Herman Hollerith** developed a machine (and concepts) that saved the U.S.
- The machine used electricity (a new idea for the time)
- Could automatically read cards and quickly, accurately tabulate results



Fall 2024

Sacramento State - CS&E - CS110

35

35

## Inventing a Solution for Sorting

- Invented the idea to Bucket Sort on **each digit of a key**
- Use multiple passes starting with the 1's digit and move upwards



Fall 2024

Sacramento State - CS&E - CS110

36

36

## Herman Hollerith

- His system was used for the 1890 Census
- And, it only took **9 months!**
- Ever since, some form of tabulating machine has been used



Fall 2024

Sacramento State - Oak - CSI 130

37

37

## Herman Hollerith & IBM

- The World took notice & wanted his machines
- Hollerith founded the Tabulating Machine Company
- Later it was renamed International Business Machines

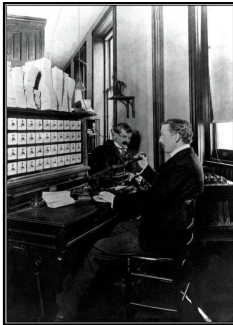


Fall 2024

Sacramento State - Oak - CSI 130

38

38



39

39



## Radix Sort

A dance of buckets

40

40

## Radix Sort

- The *Radix Sort* was developed by *Herman Hollerith* in 1887 (*77 BBW*)
- The sort is completely non-comparative
- It uses a multiple Bucket Sort passes to sort data



Fall 2024

Sacramento State - Oak - CSI 130

41

41

## Radix Sort

- Hollerith observed:
  - Bucket Sort was **stable**
  - i.e. items **did not change relative positions**
- He took advantage of this to sort data regardless of the size of the key



Fall 2024

Sacramento State - Oak - CSI 130

42

42

## How it Works

- Radix Sort uses a Bucket Sort on each digit on the key
- This is done from the Least Significant Digit (LSD) to the most (MSD)
- After each pass, the buckets are emptied into another set of buckets based on the next digit

Fall 2024

Sacramento State - Oak - CS 130

43

43

## How it Works

- So, the number of buckets is equal to the number of possible digits
- Different "digits" can also be used:
  - base-10 digits for numbers (10 buckets)
  - or a single bit in the key (2 buckets)
  - or several binary bits as a group – e.g. every 4 bits for  $2^4 = 16$  buckets

Fall 2024

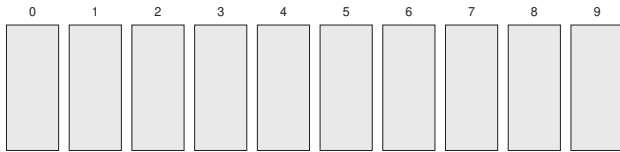
Sacramento State - Oak - CS 130

44

44

## Radix Sort Example

97 54 21 75 34 72 19 51 06 12 45 07



Fall 2024

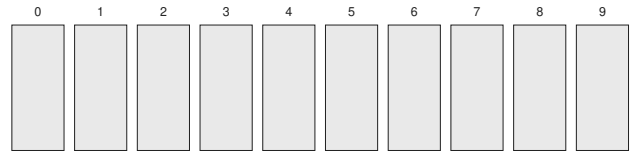
Sacramento State - Oak - CS 130

45

45

## First Digit: Scatter into Buckets

97 54 21 75 34 72 19 51 06 12 45 07



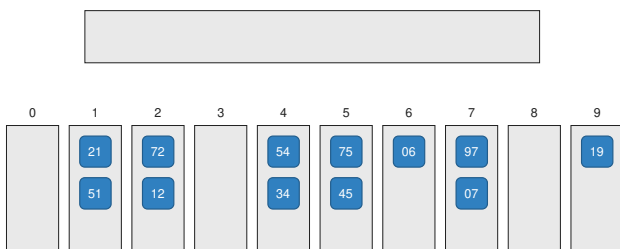
Fall 2024

Sacramento State - Oak - CS 130

46

46

## First Digit: Scattered



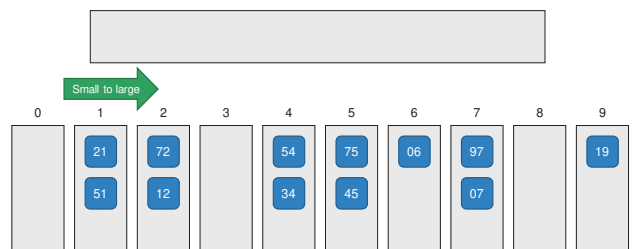
Fall 2024

Sacramento State - Oak - CS 130

47

47

## First Digit: Gather First Digit



Fall 2024

Sacramento State - Oak - CS 130

48

48



### First Digit: Sorted

Initial array: 21, 51, 72, 12, 54, 34, 75, 45, 06, 97, 07, 19

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

49

### Second Digit: Scatter into Buckets

Initial array: 21, 51, 72, 12, 54, 34, 75, 45, 06, 97, 07, 19

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

50

### Second Digit: Scattered

Note: Each bucket is sorted. The first pass sorted the first digit.

Buckets: 0: [06, 07], 1: [12, 19], 2: [21], 3: [34], 4: [45], 5: [51, 54], 6: [], 7: [72, 75], 8: [], 9: [97]

51

### Second Digit: Gather

Small to large

Buckets: 0: [06, 07], 1: [12, 19], 2: [21], 3: [34], 4: [45], 5: [51, 54], 6: [], 7: [72, 75], 8: [], 9: [97]

52

### Second Digit: Completed

Sorted array: 06, 07, 12, 19, 21, 34, 45, 51, 54, 72, 75, 97

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

53

### Example 3: LSD Base 10

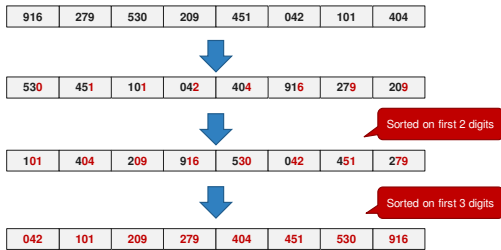
Initial array: 92, 49, 11, 74, 50, 42, 90, 47, 16, 21

Sorted on first digit: 50, 90, 11, 21, 92, 42, 74, 16, 47, 49

Sorted on first 2 digits: 11, 16, 21, 42, 47, 49, 50, 74, 90, 92

54

### Example 3: LSD Base 10



55

### Time Complexity of Radix Sort

- How many passes?
  - the algorithm will pass over the array equal to the total number of digits in the key ( $k$ )
  - e.g. for, a phone number,  $k = 10$
- So...
  - we will exam  $n$  array elements a total  $k$  number of times
  - so, it will be  $O(k \times n)$

56

### Auxiliary Storage of Radix Sort

- How many buckets?
  - we need auxiliary storage for each array element and for a bucket for each digit
  - for a base-10 number, the number of buckets  $b = 10$
- So...
  - we will need an extra  $n$  array elements and  $b$  buckets
  - so, it will be  $O(b + n)$

57

### Radix Summary

Radix Sort	
Time Average	$O(k \times n)$ where $k$ is the # of key digits
Time Best	$O(k \times n)$ <i>actually a slow <math>O(n)</math></i>
Time Worst	$O(k \times n)$ <i>actually a slow <math>O(n)</math></i>
Auxiliary space	$O(b + n)$
Stable	Yes
Online?	No

58