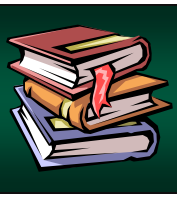
Non-Comparative Sorting

Part 7

1



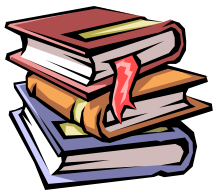
Dictionaries

the "ADT"… not in a human one!

2

Moving Past Arrays....

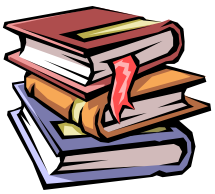
- A *collection* is general term for a group of data items
- So, this can include arrays, linked lists, stacks, queues, and much more
- So far, we have just used arrays – which are indexed by an integer



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

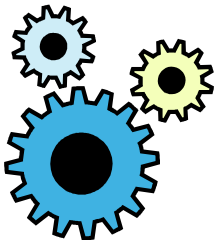
Dictionary Terminology

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

6

Implementing Dictionaries

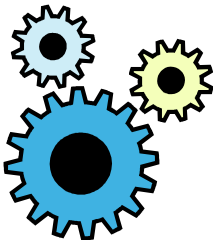
- There are numerous approaches to implementing dictionaries
- *Key-value* structure
  - a class stores a key object and value object
  - this can be stored in any data structure we have covered



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)



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….*

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

10

Bucket Sort

The key to sorting is the keys

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

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

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 impactable*

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

15

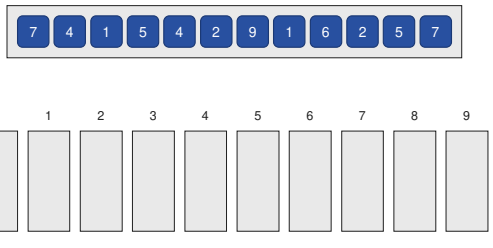
Bucket Sort – Store Back Into Array

```
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

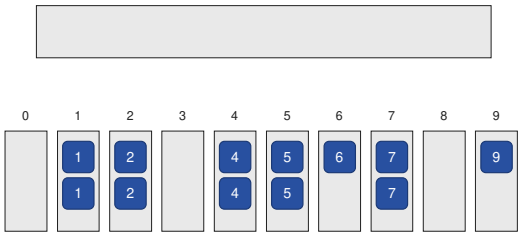
16

Bucket Sort: Scatter to Buckets



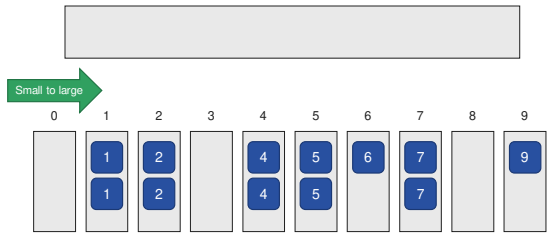
17

Bucket Sort: Buckets Filled



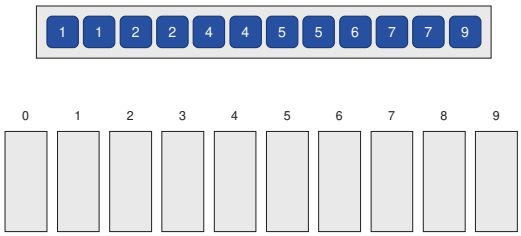
18

Bucket Sort: Gather Into Array



19

Bucket Sort: Complete



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

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

22

22

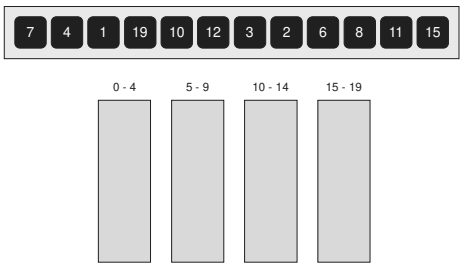
A Better Approach….

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

23

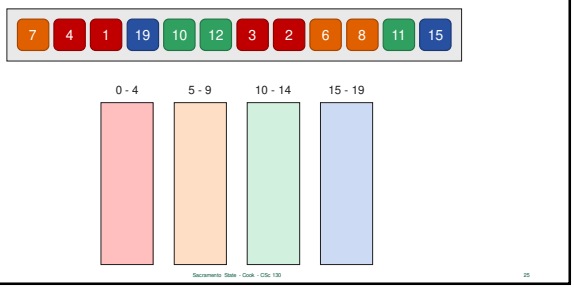
23

Bucket Sort: Fill Buckets



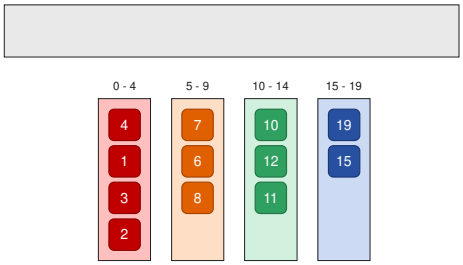
24

Bucket Sort: Fill Buckets



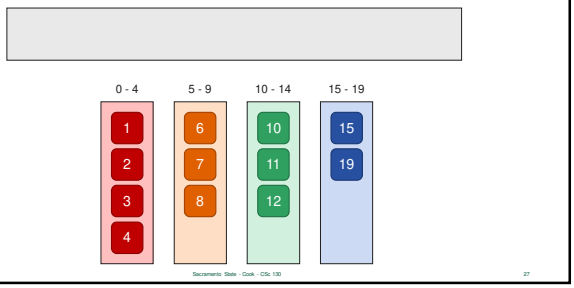
25

Bucket Sort: Buckets Filled



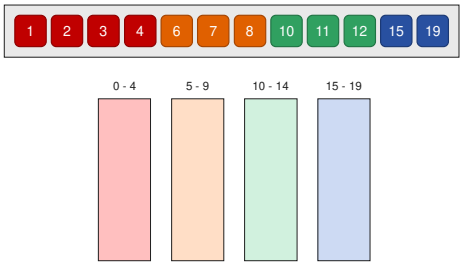
26

Bucket Sort: Sort Buckets



27

Bucket Sort: Emptied Into Array



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 + (n² / b) + b) *where b is the # of buckets*
Time Best	O(n) *when b ≈ n*
Time Worst	O(n²) *if Insertion Sort is used*
Auxiliary space	O(b + n)
Stable	**Yes**
Online?	Yes (bucket filling stage only)

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…

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*

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

35

Inventing a Solution for Sorting

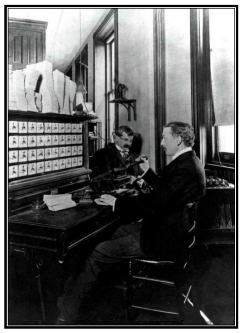
- Invented the idea to Bucket Sort on each digit of a key
- Use multiple passes starting with the 10's digit and move upwards

36

Herman Hollerith

- His system...
  - was used for 1890 census
  - it only took **9 months**!
- Hollerith then founded:
  - Tabulating Machine Company
  - it later became International Business Machines

37



38

Radix Sort

A dance of buckets

39

Radix Sort

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

40

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

41

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 the emptied into another set of buckets based on the next digit

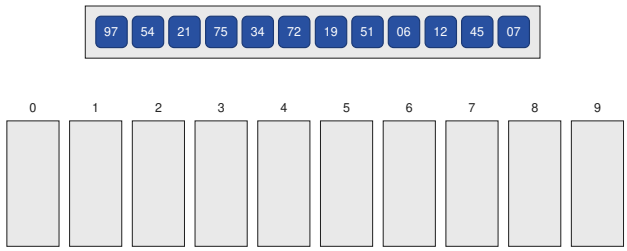
42

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

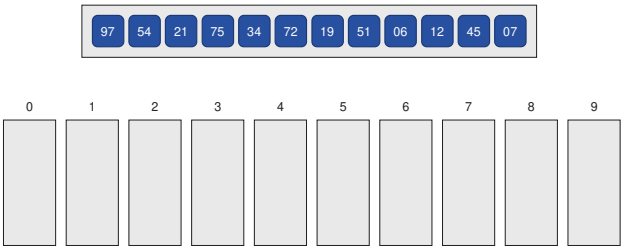
43

Radix Sort Example



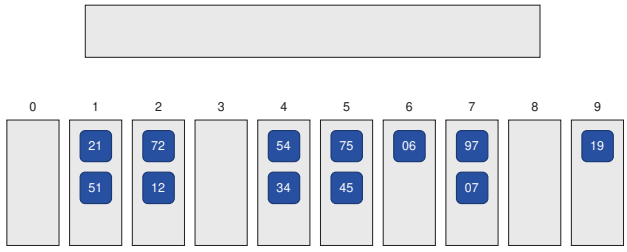
44

First Digit: Scatter into Buckets



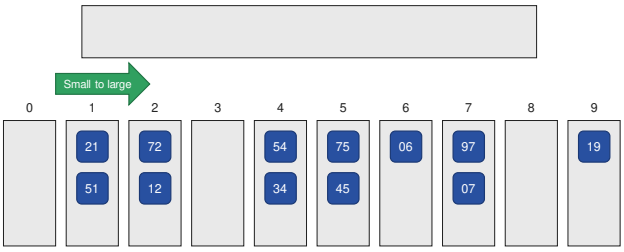
45

First Digit: Scattered



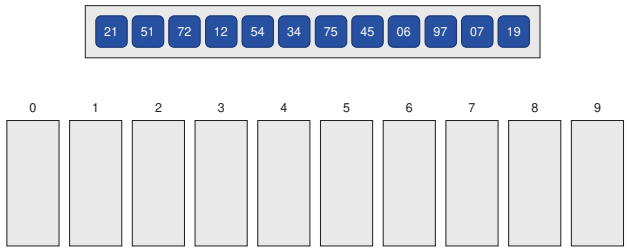
46

First Digit: Gather First Digit



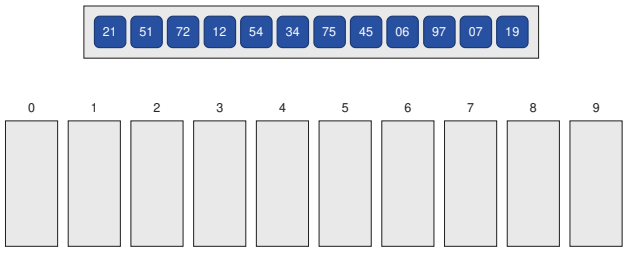
47

First Digit: Sorted



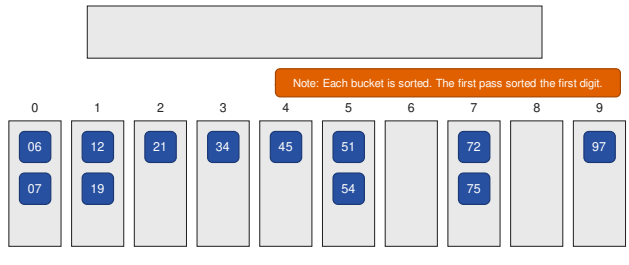
48

Second Digit: Scatter into Buckets



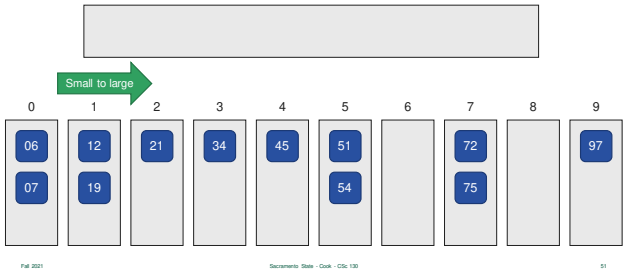
49

Second Digit: Scattered



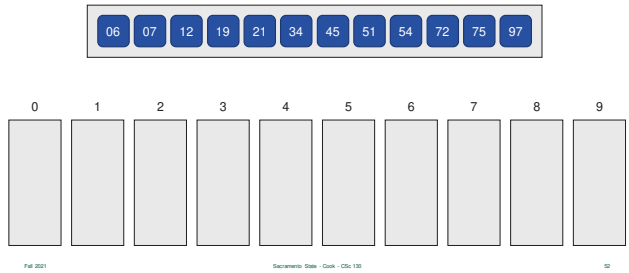
50

Second Digit: Gather



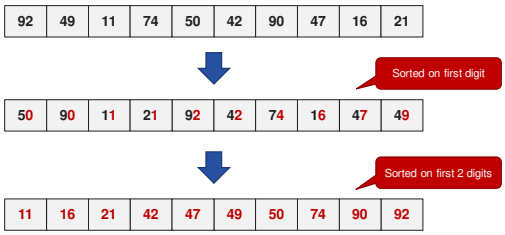
51

Second Digit: Completed



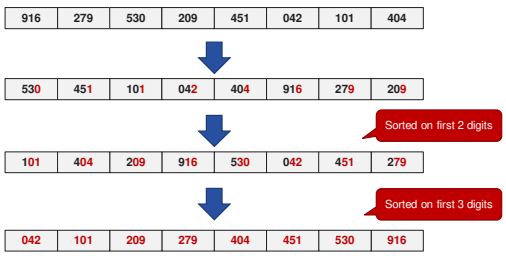
52

Example 3: LSD Base 10



53

Example 3: LSD Base 10



54

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 × n)

55

Auxiliary Storage of Radix Sort

- How many buckets?
  - the number of buckets will be equal to the number of unique values each digit can have
  - for a base-10 number, b = 10
- So…
  - we need auxiliary storage for each array element and for a bucket for each digit
  - so, it will be O(b + n)

56

Radix Summary

Radix Sort	
Time Average	O(k × n) *where k is the # of key digits*
Time Best	O(k × n) *actually a slow O(n)*
Time Worst	O(k × n) *actually a slow O(n)*
Auxiliary space	O(b + n)
Stable	**Yes**
Online?	No

57