Algo Week 7 Bucket Sort

Textbook Countingsort

count number items in the array that have each value. then copy each value in order back into the array

pass in max value as parameter

M - number items in counts array (max_value+1)

spends M steps initializing the array

N - number number of items in values array

spends N steps to count the values in the values array

takes N steps to copy back

so O(2xN+M) simplified to O(N+M)

handles duplication better than quicksort

Wikipedia Countingsort

integer sorting only

uses key values instead of comparisons

textbook Bucketsort

also called binsort

1. divide items into buckets
2. use another algorithm to sort buckets or recursively sort buckets
3. concatenate buckets contents back into original array

N items in array, use M buckets - expect N / M items per bucket

buckets can be any data structure

distributing items takes O(N) steps

sort buckets O(MxF(N/M)) //F is runtime function of sorting algorithm

in total simplifies to O(N). performance really depends upon number of buckets

Wikipedia Bucketsort

could be implemented with comparisons

worst case defined by algorithm used to sort buckets

average case: O(N)

common optimization put buckets back into array then run insertion sort as insertion sort runtime based on how far each element has to go

Lecture

comparison-based algorithms have speed limit of O(nlgn)

countingsort 1. find smallest and largest elements in array 2. create a counting array with enough room for all possible integers 3. scan through the array and count how many times each number appears adjusting value in counting array accordingly 4. use counting array to reconstruct the sorted version of original array

speed: find min and max: traverse array O(n) counting elements traverse again: O(n) assembling sorted array from counting array: O(n) total: O(n) or linear time

bucketsort - can sort doubles, ints, anything really if using hash function

1. find largest and smallest elements in array
2. create n buckets (same number of elements in array) space buckets evenly
3. scan through array and put elements in buckets
4. sort the buckets - usually done with insertion sort
5. put contents of buckets back into array

use arrays for buckets for now

if buckets spaced evenly, probability land in bucket is 1/n

time: 1. finding min and max: O(n) 2. placing items in buckets: O(n) 3. sorting buckets: -n buckets -O(nb^2) = (2-1/n) per bucket when using insertion sort -n x O(nb^2) = O(2n-1) which can be reduced to O(n)

works best if values evenly distributed

slows down when fewer buckets are more full