<2T>

# Bucket Sort Description

Bucket sort assumes that the input is drawn from a uniform distribution and has an average-case running time of O(n). Like counting sort, bucket sort is fast because it assumes something about the input. Whereas counting sort assumes that the input consists of integers in a small range, bucket sort assumes that the input is generated by a random process that distributes elements uniformly and independently over the interval [0, 1).

Bucket sort divides the interval

Bucket sort divides the interval [0, 1) into n equal-sized subintervals, or buckets, and then distributes the n input numbers into the buckets. Since the inputs are uni-formly and independently distributed over [0, 1), we do not expect many numbers to fall into each bucket. To produce the output, we simply sort the numbers in each bucket and then go through the buckets in order, listing the elements in each.

Our code for bucket sort assumes that the input is an n-element array A and that each element A[i] in the array satisfies 0 <= A[i] <1.The code requires an auxiliary array

B[0,..n-1] of linked lists (buckets) and assumes that there is a

mechanism for maintaining such lists.

# Bucket Sort Pseudocode

Bucket-Sort(A)  
1 let B[0…n-1] be a new array  
2 n = A.length  
3 for I = 0 to n-1  
4 make B[i] an empty list  
5 for I = 1 to n  
6 insert A[i] into list B[floor(n\*A[i])]  
7 for I = 0 to n-1  
8 sort list B[i] with insertion sort  
9 concatenate the lists B[0], B[1]…..B[n-1] together in order

# Bucket Sort Code

 def bucketsort( A ):

# get hash codes

code = hashing( A )

buckets = [list() for \_ in range( code[1] )]

# distribute data into buckets: O(n)

for i in A:

x = re\_hashing( i, code )

buck = buckets[x]

buck.append( i )

for bucket in buckets:

insertionsort( bucket )

ndx = 0

# merge the buckets: O(n)

for b in range( len( buckets ) ):

for v in buckets[b]:

A[ndx] = v

ndx += 1

import math

def hashing( A ):

m = A[0]

for i in range( 1, len( A ) ):

if ( m < A[i] ):

m = A[i]

result = [m, int( math.sqrt( len( A ) ) )]

return result

def re\_hashing( i, code ):

return int( i / code[0] \* ( code[1] - 1 ) )