**Bucket Sort**

Context

Suppose we have an array A of **n** distinct integers A= {a1, a2… an} in range of [0 – **m**-1]

Sorting strategy

* Create an array Bucket of size m and all its entries initialized to 0. Create an output array B of size n (requires **O (m + n)** )
* Scan A, when A[i] is encountered, increment the value bucket A[i] (requires **O (n)** )
* Scan Bucket[]. For each j < m for which bucket[j] > 0, copy A[j] into next available slot in B. (requires **O (m)** )

Therefore, BucketSort runs in O (m + n). **When m is O (n), BucketSort runs in O (n)**

Example

Given n integers a1, a2, …, an in the range [0, m-1], possibly with duplicates, and n matched objects o1, o2, …, on. We sort array A whose elements are pairs (a1,o1), (a2,o2),…,(an,on).

Pseudo code

**Algorithm** ***bucketSort***(***A,*** ***m***)

**Input** array ***A*** of (key, element) items with keys in the range [0, ***m*** 1]  
 **Output** array ***B*** sorted by increasing keys

***bucket*** ←array of ***N*** empty lists

**for *i*** ← 0 **to** ***n*** 1

(***k***, ***o***) ← ***A[i]***

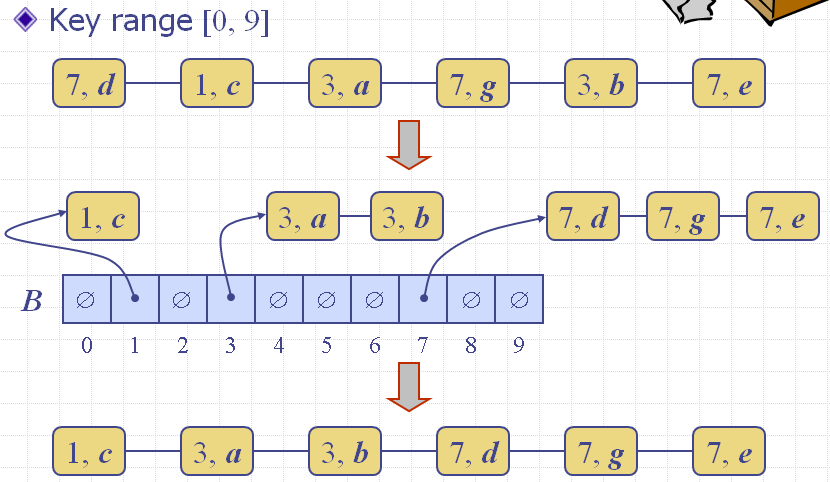
***bucket***[***k***]***.insertLast***((***k***, ***o***))

**for *j*** ← 0 **to** ***m*** 1

**while** ¬***bucket***[***j***]***.isEmpty***()

***(k, o)*** ← ***bucket***[***j***]***.removeFirst***()

***B.insertLast***((***k***, ***o***))



**Radix Sort**

Context

Radix-sort is a generalization of BucketSort that uses multiple bucket arrays. BucketSort doesn’t work because range is too big – would run in Ω(n2)

Example

Sort {48, 1, 6, 23, 37, 19, 21}

Strategy

* + is to use 2 bucket arrays, each of size 7 (7 is the radix)
  + Based on observation that every **k** in [0,48] can be written:  
     **k** = 7**q** + **r** where 0 ≤ **q** < 7, 0 ≤ **r** < 7
  + Procedure
    - Pass #1: Scan initial array and place values in the “remainders” bucket r[] – put x in r[i] if x % 7 = i. (Need to assume bucket array consists of lists)
    - Pass #2: Scan r[], reading from front of each list to back, and place values in the “quotients” bucket q[] – put x in q[i] if x/7 = i.
    - Output: Scan q[], again reading lists front to back

Illustrate

Keys: 48, 1, 6, 23, 37, 19, 21

Pass #1: **Key % 7**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| r[i] | 21 | 1 | 23 |  |  | 19 | 48 |
| 37 | 6 |

Pass #2: **Key / 7**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| j | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| q[j] | 1 |  | 19 | 21 |  |  | 48 |
| 6 | 23 |

Sorted order: 1, 6, 19, 21, 23, 48

Running time is O(d(n+m)) where

* d = # bucket arrays (in example d=2)
* n = size of initial array (example: 7)
* m = size of each bucket array (ex: 7)

RadixSort runs in O(n) whenever the number of bucket arrays is O(1), and size of each bucket array is O(n).

Using 3 Bucket Arrays

Generalizing previous example, we observe that whenever k belongs to [0, 342] (note 342 = 73 – 1), then we can write  
 **k = 49q1 + 7q2 + r**

where 0 ≤ q1 < 7, 0 ≤ q2 < 7, 0 ≤ r < 7.

Example: 340 = 49\*6 + 7\*6 + 4

Therefore to handle sorting approx 7 elements in the range [0, 342], create bucket arrays r[], q1[], q2[], and compute,for each x in input array, values x%7, (x/7)%7, x/49, respectively