Εργαστήριο 7:
Ο αλγόριθμος ταξινόμησης Radix Sort

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

- Ο αλγόριθμος ταξινόμησης Radix Sort
- Δυο εκδοχές: Most Significant Digit (MSD) και Least Significant Digit (LSD)
- Επίλυση δυο ασκήσεων στο χαρτί

Επανάληψη BucketSort

- Έστω ότι ο πίνακας Α ***n*** στοιχείων περιέχει στοιχεία που ανήκουν στο διάστημα [***1..m***].

- Ο ***αλγόριθμος BucketSort*** βασίζεται πάνω στα ακόλουθα βήματα:
  1. Δημιουργούμε ένα πίνακα **buckets** μήκους **m** και θέτουμε **buckets[i]=0,** για όλα τα **i** (Αυτά τα είναι τα buckets - κάδοι)
  2. Διαβάζουμε τον πίνακα **A** ξεκινώντας από το πρώτο στοιχείο. Αν διαβάσουμε το στοιχείο **α**, τότε αυξάνουμε την τιμή του **buckets[α]** κατά ένα. Επαναλαμβάνουμε το βήμα μέχρι το τελευταίο στοιχείο.
  3. Τέλος, διαβάζουμε γραμμικά τον πίνακα **buckets**, ο οποίος περιέχει αναπαράσταση του ταξινομημένου πίνακα, και θέτουμε τα στοιχεία του πίνακα **A** με την ταξινομημένη ακολουθία.

BUCKETS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Επανάληψη BucketSort: Παράδειγμα

Δεδομένο Εισόδου: Τα στοιχεία είναι στο εύρος m=[0,14], n=8

1	11	1	7	2	14	7	1
0							n-1

Μετά την **1η** εκτέλεση του Bucketsort (Εισαγωγή του 1)

BUCKETS	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

m

Μετά τη **2η** εκτέλεση του Bucketsort (Εισαγωγή του 11)

BUCKETS	0	**1**	0	0	0	0	0	0	0	0	0	**1**	0	0	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

m

Μετά τη **3η** εκτέλεση του Bucketsort (Εισαγωγή του 1)

BUCKETS	0	**2**	0	0	0	0	0	0	0	0	0	**1**	0	0	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

m

Μετά την **8ή** εκτέλεση του Bucketsort

BUCKETS	0	**3**	**1**	0	0	0	0	**2**	0	0	0	**1**	0	0	**1**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

m

Επανάληψη BucketSort: Πρόγραμμα στην C

```
// A: Πίνακας για ταξινόμηση μεγέθους n
// Buckets: Βοηθητικός πίνακας μεγέθους m
void BucketSort(int A[], int buckets[], int n, int m){
  int i, j, k=0;

  // Κατανομή στοιχείων στους σωστούς κάδους
  for (i=0; i<n; i++) {
    buckets[A[i]]++;
  }

  // Αντιγραφή στοιχείων από πίνακα BUCKETS
  // στον πίνακα A
  for (i=0; i<m; i++) {
    for (j=0; j<buckets[i]; j++) {
      A[k] = i;
    k++;
    }
  }
}
```

O(n)

O(n+m)

Συνολικά περνάμε 1 φορά από τα στοιχεία του πίνακα BUCKETS (m) και μια φορά από αυτά του A (n)

Ο αλγόριθμος Bucket Sort

- Bucket Sort
  - Κατανέμουμε τα **n στοιχεία** σε ένα σύνολο από **m κάδους**.
  - Περνούμε μία φορά από κάθε στοιχείο του πίνακα και γεμίζουμε τους κάδους. Ακολούθως τυπώνουμε τους m κάδους μου μας παράγει το ταξινομημένο αποτέλεσμα)
  - **ΜΕΙΟΝΕΚΤΗΜΑ:** Δουλεύει μόνο για ακέραιους αριθμούς.

Ο αλγόριθμος Radix Sort

- Η ιδέα του Αλγόριθμου Radix Sort
  - Είναι κατάλληλος για ταξινόμηση **m-αδικών πλειάδων**
  - Παράδειγμα: m=3, {(1,2,4), (1,5,3),…}
  - Μία m-αδική πλειάδα μπορεί να συμβολίζει συμβολοσειρές (Strings με ASCII), π.χ., cat=(99,97,116)
  - Ο Radix Sort εκτελεί **m διαδοχικά καλέσματα** στον Bucket Sort αρχίζοντας από την αρχή της πλειάδας (Most-Significant-Digit) ή το τέλος (Least-Significant-Digit)
    - Most Significant Digit (MSD): Αρχίζει την επεξεργασία των δεδομένων από το πιο σημαντικό ψηφίο (most significant digit) και προχωρεί μέχρι το λιγότερο σημαντικό στοιχείο (least significant digit)➔ από αριστερά προς δεξιά
    - Least Significant Digit (LSD): το αντίστροφο ➔ από δεξιά προς αριστερά

Radix Sort: Most-Significant-Digit (MSD)

- Ένας αναδρομικό MSD Radix Sort αλγόριθμος δουλεύει ως εξής:
  - Πάρε το πιο σημαντικό ψηφίο από κάθε κλειδί
  - Ταξινόμησε τα στοιχεία βάση αυτού του κλειδιού και παράλληλα ομαδοποίησε τα στοιχεία με το ίδιο κλειδί στον ίδιο κάδο
  - Αναδρομικά, ταξινόμησε κάθε κάδο ξεκινώντας με το πιο σημαντικό ψηφίο
  - Συνένωσε όλους τους κάδους με ταξινομημένη σειρά

Παράδειγμα Αναδρομικού Radix Sort-MSD

- Ταξινόμησε τη λίστα:
  170, 045, 075, 090, 002, 024, 802, 066

- Ταξινόμησε με το πιο σημαντικό ψηφίο (δηλ. εκατοντάδες 100s):

  - Κάδος - Μηδέν εκατοντάδες (0xx):
    045, 075, 090, 002, 024, 066

  - Κάδος - Μία εκατοντάδα (1xx):
    170

  - Κάδος – Οκτώ (8xx):
    802

Παράδειγμα Αναδρομικού Radix Sort-MSD (συν.)

- Ταξινόμησε βάση του επόμενου ψηφίου (δεκάδες 10s): Παρατήρηση: Χρειάζεται να ταξινομήσουμε μόνο τα στοιχεία με 0 εκατοντάδες (οι άλλοι κάδοι έχουν μόνο ένα στοιχείο)
  - Κάδος - Μηδέν δεκάδες (0x): 002
  - Κάδος - Δύο δεκάδες (2x) : 024
  - Κάδος - Τέσσερις δεκάδες (4x) : 045
  - Κάδος - Έξι δεκάδες (6x): 066
  - Κάδος - Επτά δεκάδες (7x): 075
  - Κάδος - Εννέα δεκάδες (9x): 090

Παράδειγμα Αναδρομικού Radix Sort-MSD (συν.)

- Ταξινόμηση με μονάδες (1s) δεν χρειάζεται γιατί δεν υπάρχει κάδος με δεκάδες που να έχει περισσότερο από ένα στοιχείο

- Στο επόμενο βήμα, ο ταξινομημένος κάδος των 0xxάδων (που φαίνεται πιο πάνω) συνενώνεται με τους κάδους των 100άδων και 800άδων για να παράξουν την τελική ταξινομημένη σειρά:

  002, 024, 045, 066, 075, 090, 170, 802

Ο αλγόριθμος Radix Sort

- Δουλεύει ο αλγόριθμος;

- Προφανώς, αν το πιο σημαντικό ψηφίο μεταξύ **α** και **β** είναι διαφορετικό και **α<β** τότε στο τέλος το **α** θα έρθει πριν από το **β**

- Αν το πιο σημαντικό ψηφίο μεταξύ **α** και **β** είναι ίσο και το δεύτερο πιο σημαντικό ψηφίο του β είναι πιο μικρό από το **α** τότε το **β** θα έρθει πριν από το **α**

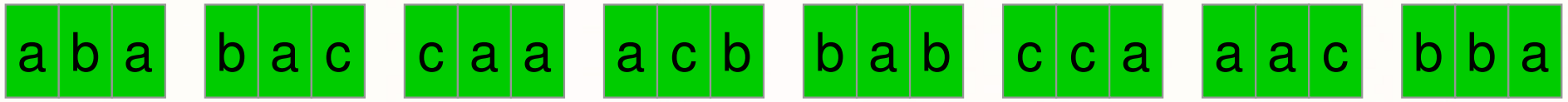
RadixSort: Least-Significant-Digit (LSD)

- Αλγόριθμος
  - Ταξινόμησε βάση του λιγότερο σημαντικού ψηφίου
    - Αριθμοί με το ίδιο ψηφίο καταλήγουν στον ίδιο κάδο
  - Ανακατάταξε όλους τους αριθμούς ως εξής: οι αριθμοί στον κάδο 0 προηγούνται τους αριθμούς στον κάδο 1, οι αριθμοί στον κάδο 1 προηγούνται τους αριθμούς στον κάδο 2, κοκ.
  - Ταξινόμησε βάση του λιγότερο σημαντικού ψηφίου
  - Συνέχισε αυτή την διαδικασία μέχρι όλοι οι αριθμοί να έχουν ταξινομηθεί για όλα τα ψηφία τους

- Σε αντίθεση με τον Radix Sort MSD, στον Radix Sort LSD τα στοιχεία μέσα στον ίδιο κάδο είναι στη σωστή σειρά κατά την διάρκεια της εκτέλεσης

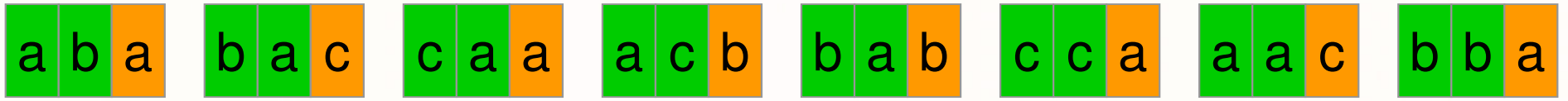
Παράδειγμα Radix Sort-LSD

Input data:



Παράδειγμα Radix Sort-LSD

Pass 1: Looking at rightmost position.

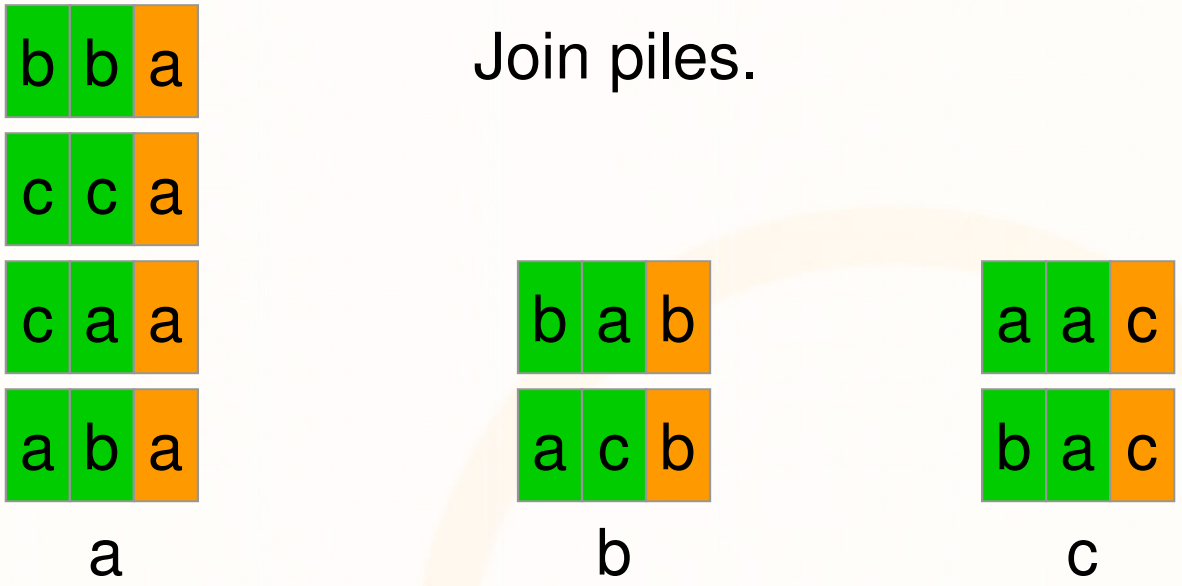


Place into appropriate pile.

a b c

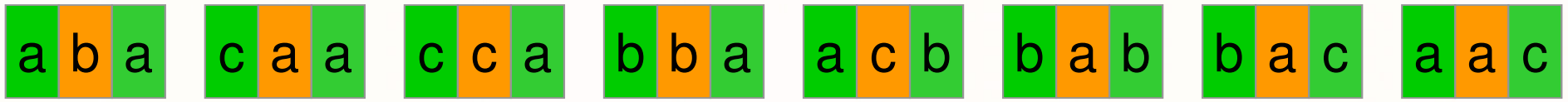
Παράδειγμα Radix Sort-LSD

Pass 1: Looking at rightmost position.



Παράδειγμα Radix Sort-LSD

Pass 2: Looking at next position.

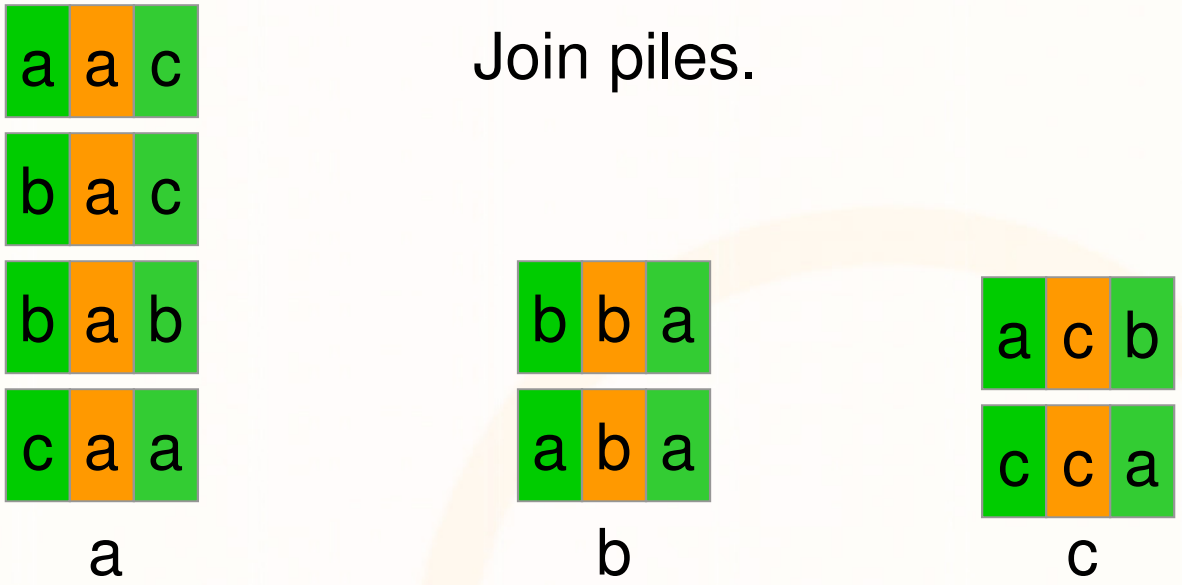


Place into appropriate pile.

a b c

Παράδειγμα Radix Sort-LSD

Pass 2: Looking at next position.



Παράδειγμα Radix Sort-LSD

Pass 3: Looking at last position.

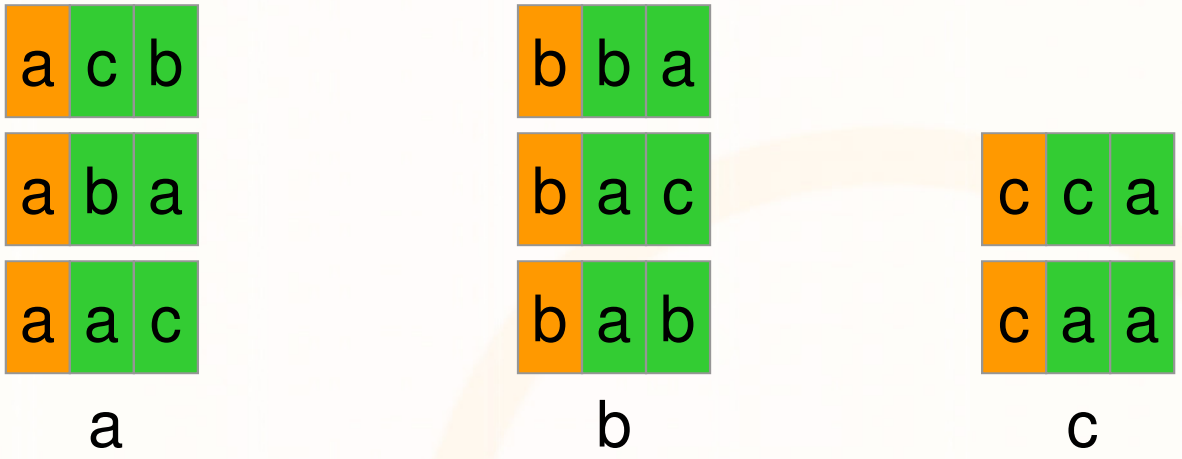
Place into appropriate pile.

a b c

Παράδειγμα Radix Sort-LSD

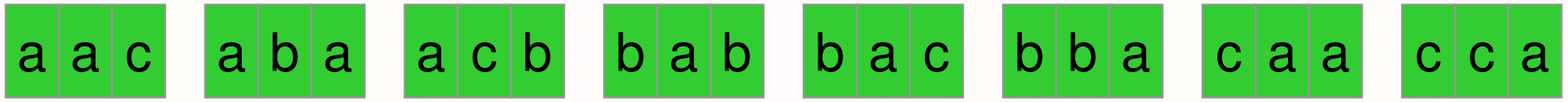
Pass 3: Looking at last position.

Join piles.



Παράδειγμα Radix Sort-LSD

Result is sorted.



Άσκηση 1

- Υποθέστε ότι για είσοδο μας δίνονται δύο ταξινομημένες ακολουθίες Α και Β με n στοιχεία
- Οι ακολουθίες μπορεί να περιέχουν διπλά, τριπλά, κοκ. στοιχεία
- **Περιγράψετε ένα αποδοτικό αλγόριθμο για να καθορίσετε αν η Α και Β περιέχουν τον ίδιο σύνολο από στοιχεία.**
- Τι είναι η χρονική πολυπλοκότητα του αλγόριθμου που προτείνετε;

Άσκηση 2

- Υποθέστε ότι για είσοδο μας δίνονται δύο ταξινομημένες ακολουθίες Α και Β με n στοιχεία
- **Περιγράψετε ένα αποδοτικό αλγόριθμο της τάξης O(nlgn) ο οποίος καθορίζει αν υπάρχουν στοιχεία a∈A και b∈B ώστε m=a+b.**

Λύσεις

Άσκηση 1

We will move along the elements of A. For each new element we will compare the element with the current element of B. If they match, we will traverse A and B until we find a new element in A and a new element in B accordingly. If the elements are the same we repeat the previous step until the end of the lists. Else we report that the lists do not have the same set of elements.

The above algorithm is O(n)

Λύσεις

Άσκηση 2

Examine linearly all the elements a of A. For each element a, perform a binary search in the elements of B looking for an element b such that b=m-a

```
Ex2(m, A, B) {
      Foreach a∈A {
            b=binarysearch(m-a, B);
            if (found(b)) {
                  print (“%d + %d = %d”, a, b, m)
                  return;
            }
      }
      print (“element was not found in b”)
}
```