Διάλεξη 14: Αλγόριθμοι Ταξινόμησης

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

Οι αλγόριθμοι ταξινόμησης

3) Mergesort – Ταξινόμηση με Συγχώνευση

4) BucketSort – Ταξινόμηση με Κάδους

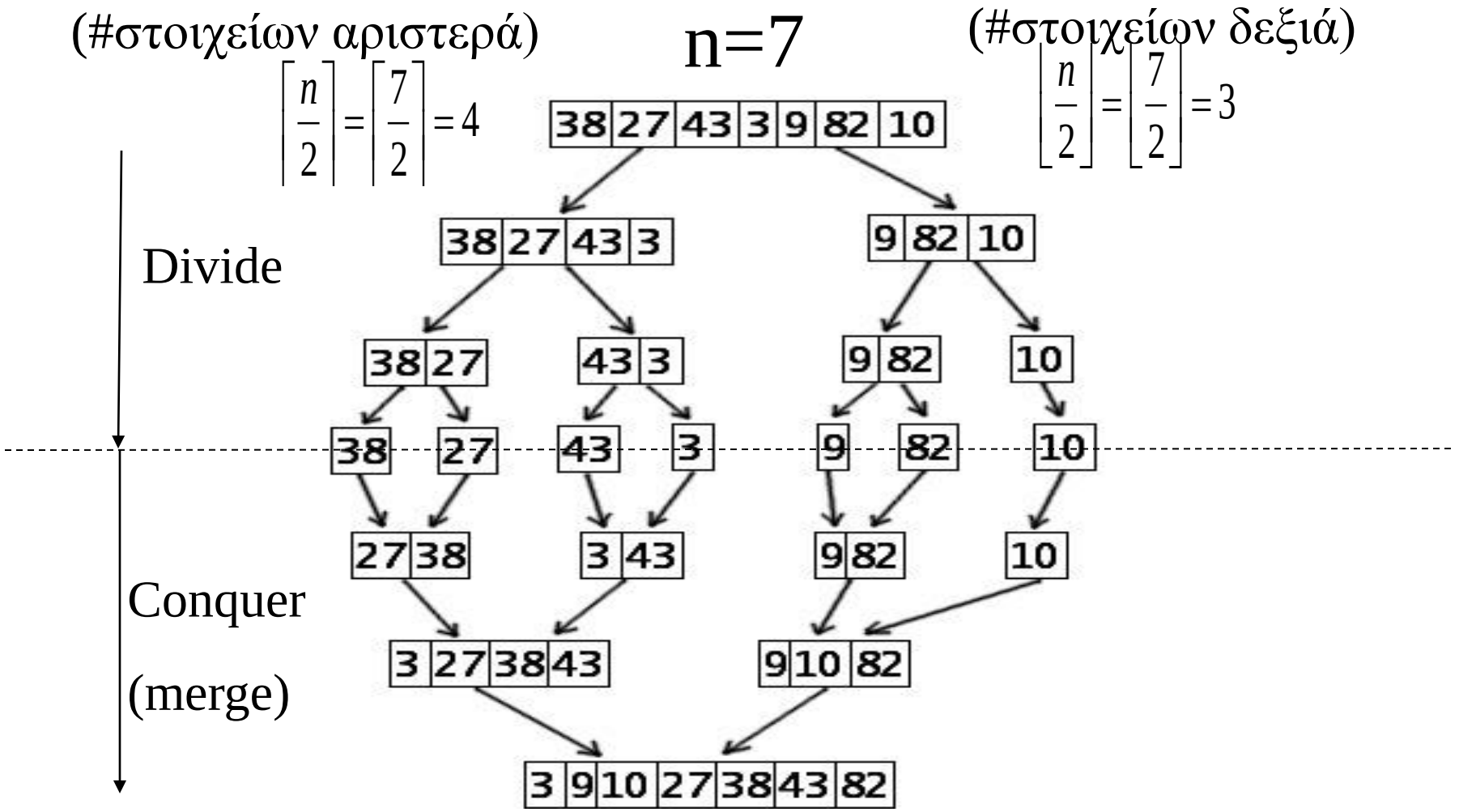
Διδάσκων: Δημήτρης Ζεϊναλιπούρ

3) Ταξινόμηση με Συγχώνευση (Merge sort)

- Η ***ταξινόμηση με συγχώνευση*** είναι διαδικασία ***διαίρει και βασίλευε*** (**Divide and Conquer:** αναδρομική διαδικασία όπου το πρόβλημα μοιράζεται σε μέρη τα οποία λύνονται ξεχωριστά, και μετά οι λύσεις συνδυάζονται.)

- Περιγραφή του Mergesort
  1. **Διαίρεση:** Αναδρομικά μοιράζουμε τον πίνακα στα δύο μέχρι να φτάσουμε σε πίνακες μεγέθους ένα **(DIVIDE)**
  2. **Συγχώνευση:** Ταξινομούμε αναδρομικά τους πίνακες αυτούς με την συγχώνευση γειτονικών πινάκων (χρησιμοποιώντας βοηθητικό πίνακα). **(CONQUER)**

Η Βασική Ιδέα του Αλγόριθμου Merge Sort



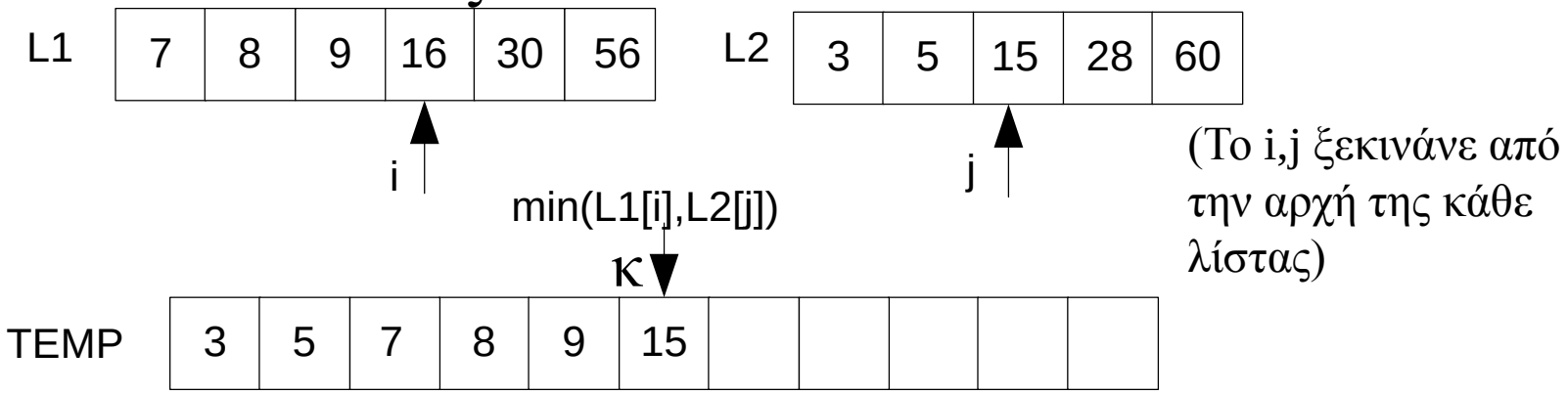
Συγχώνευση 2 Λιστών

- Υποθέστε ότι θέλετε να συγχωνεύσετε 2 ταξινομημένες λίστες L1, L2 και να δημιουργήσετε μια νέα ταξινομημένη λίστα TEMP.

Διαδικασία

1. Τοποθέτησε τους δείκτες i, j στην κεφαλή κάθε λίστας.
2. Διάλεξε το μικρότερο από την λίστα L1 και L2 και τοποθέτησε τον στον πίνακα TEMP στην θέση κ
3. Προχώρησε τον δείκτη i (αν το μικρότερο στοιχείο ήταν από την λίστα L1) ή τον δείκτη j στην αντίθετη περίπτωση.
4. Επανέλαβε τα βήματα 2-4 μέχρι να εισαχθούν όλα τα στοιχεία στον TEMP

Μέτα από 6 εκτελέσεις:



Παράδειγμα Εκτέλεσης Merge Sort

BEFORE:[8,4,8,43,3,5,2,1,10,]

Index: 0 1 2 3 4 5 6 7 8

0,8: [8,4,8,43,3,5,2,1,10,]

0,4: [8,4,8,43,3,]

0,2: [8,4,8,]

0,1: [8,4,]

divide

0,0: [8,]

1,1: [4,]

Merging: *[A0,A0] [A1,A1] => [4,8,]*

2,2: [8,]

divide

Merging: *[A0,A1] [A2,A2] => [4,8,8,]*

3,4: [43,3,]

3,3: [43,]

divide

4,4: [3,]

Merging: *[A3,A3] [A4,A4] => [3,43,]*

Merging: *[A0,A2] [A3,A4] => [3,4,8,8,43,]*

5,8: [5,2,1,10,]

5,6: [5,2,]

5,5: [5,]

divide

6,6: [2,]

Merging: *[A5,A5] [A6,A6] => [2,5,]*

7,8: [1,10,]

7,7: [1,]

divide

8,8: [10,]

Merging: *[A7,A7] [A8,A8] => [1,10,]*

Merging: *[A5,A6] [A7,A8] => [1,2,5,10,]*

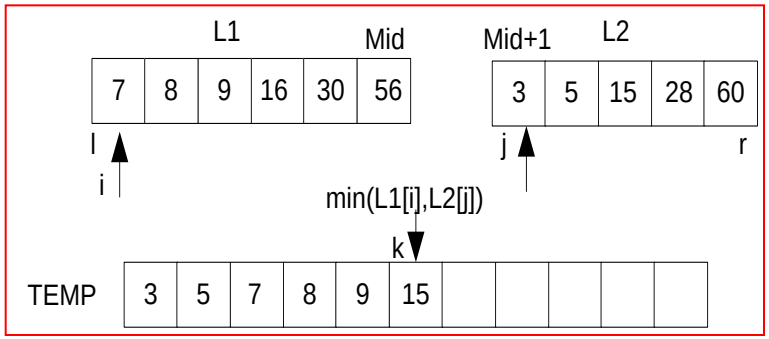
Merging: *[A0,A4] [A5,A8] =>*

[1,2,3,4,5,8,8,10,43,]

AFTER:[1,2,3,4,5,8,8,10,43,]

Αλγόριθμος Merge Sort

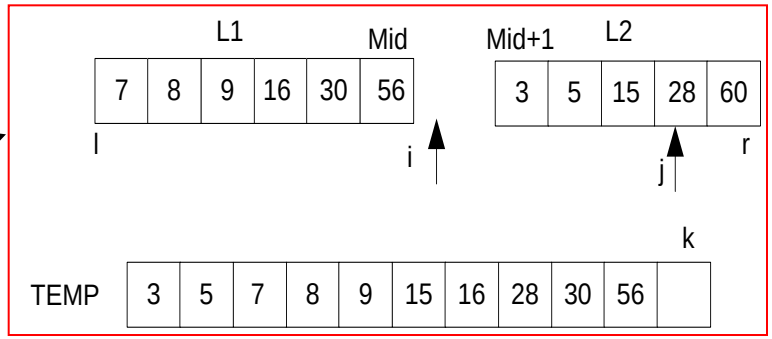
```
void MergeSort(int A[], int temp[],int l, int r){
  // η συνθήκη τερματισμού της ανάδρομης
  if (l==r) return;
  int mid = (l+r)/2;
  // για ελαχιστοποίηση overflow(για μεγάλα l,r)
  // int mid = l + ((r - l) / 2);

  // μοιράζουμε αναδρομικά τον πίνακα
  Mergesort(A, temp, l, mid);
  Mergesort(A, temp, mid+1, r);

  // Τώρα οι πίνακες [l..mid] και [mid+1..r] είναι ταξινομημένοι
  // Η διαδικασία συγχώνευσης
  k=l, i=l; j=mid+1;

  // συγχώνευση στον TEMP μέχρι μια από τις λίστες να είναι κενή
  while ((i<=mid) && (j<=r)) {
        if (A[i]<A[j])
                { temp[k] = A[i]; i++; }
        else
                { temp[k] = A[j]; j++; }
        k++;
   }
```



συνέχεια στην επόμενη διαφάνεια…

Αλγόριθμος Merge Sort

```
// copy όλων τα υπόλοιπων στοιχείων της λίστας L1
while (i<=mid) {
     temp[k] = A[i];
     k++;i++;
}
```




```
// copy όλων τα υπόλοιπων στοιχείων της λίστας L2
while (j<=r) {
     temp[k] = A[j];
     k++;j++;
}

// αντιγραφή όλων των στοιχείων από TEMP -> A
for (i=l; i<=r; i++) {
     A[i] = temp[i];
}
}
```

Ανάλυση Χρόνου (αναδρομής)

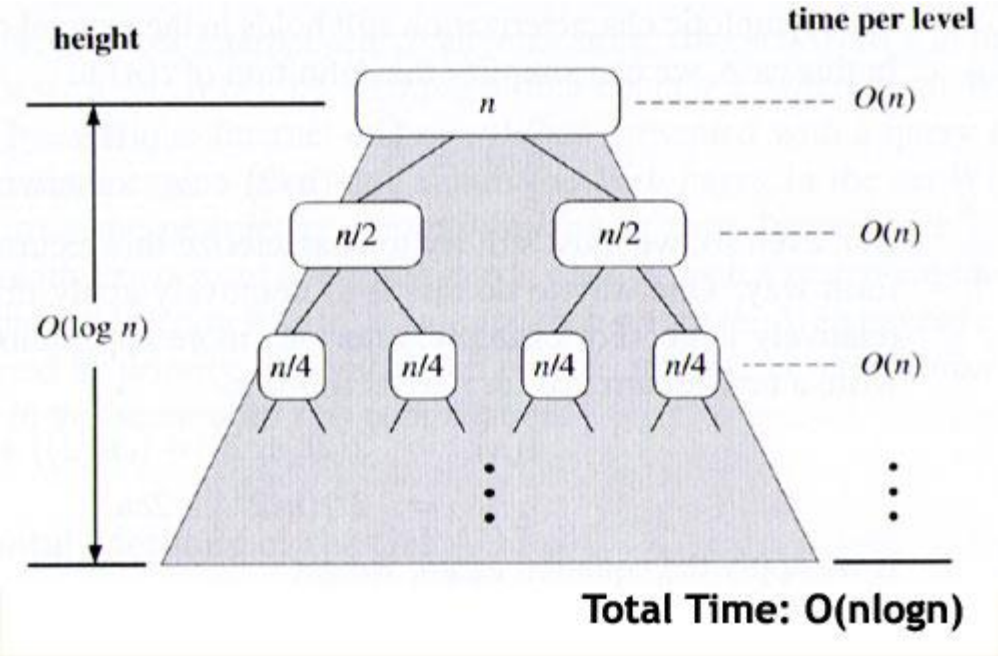
- Το πρόβλημα μοιράζει αναδρομικά σε δυο μέρη την λίστα που θέλουμε να ταξινομήσουμε.
- Όταν φτάσουμε στην λίστα που έχει μέγεθος 1 τότε σταματά η αναδρομή και το πρόγραμμα αρχίζει να συνδυάζει (merge) τις επιμέρους λίστες.
- Παρατηρούμε ότι πάνω σε μια λίστα μεγέθους N η αναδρομή εκτελείται $2\log_2 n$ φορές. Δηλαδή ο πίνακας μοιράζεται ως εξής: **n, n/2, n/4, …, 2, 1**

```
void MergeSort(int A[], int temp[],int l, int r){
  if (l==r) return;
  int mid = (l+r)/2;

  Mergesort(A, temp, l, mid);
  Mergesort(A, temp, mid+1, r);
…
```

Θ(logn)

Ανάλυση Χρόνου (συγχώνευση)

- **Σε κάθε επίπεδο της ανάδρομης περνάμε μια φορά από το κάθε στοιχείο.**
- Επομένως η συγχώνευση των στοιχείων σε κάθε επίπεδο της εκτέλεσης χρειάζεται **γραμμικό** χρόνο **Θ(n).**
- Σημειώστε ότι η διαδικασία απαιτεί τη χρήση βοηθητικού πίνακα.
- Μπορούμε να χρησιμοποιούμε τον ίδιο βοηθητικό πίνακα temp για όλες τις (αναδρομικές) κλήσεις του MergeSort.



Συνολικός Χρόνος Εκτέλεσης Mergesort

- Η αντιγραφή και η συγχώνευση παίρνουν χρόνο Θ(n) και η αναδρομή παίρνει χρόνο O(log n). Συνολικά **Θ(n log n).**
- Ο χρόνος εκτέλεσης εκφράζεται και από την αναδρομική εξίσωση

$$T(0) = T(1) = 1$$
$$T(n) = 2 \cdot T(n/2) + n$$

- … η οποία μπορεί να λυθεί με το Master Theorem ή με την μέθοδο της αντικατάστασης**.**

- **Πλεονέκτημα MergeSort:**

  Ο συνολικός χρόνος εκτέλεσης είναι **Θ(n log n)**

  (σε αντίθεση με το SelectionSort ($\Theta(n^2)$) και το InsertionSort ($O(n^2)$)

- **Μειονέκτημα**:

  Απαιτεί τη χρήση βοηθητικού πίνακα (δηλαδή χρειάζεται διπλάσιο χώρο αποθήκευσης για την εκτέλεση του). Αυτό δεν καθιστά την μέθοδο πολύ εύχρηστη.

4) Ταξινόμηση με Κάδους - 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
ΕΠΛ	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Η Βασική Ιδέα του Αλγόριθμου Bucket Sort

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 **πρώτη** 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 **τρίτη** 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

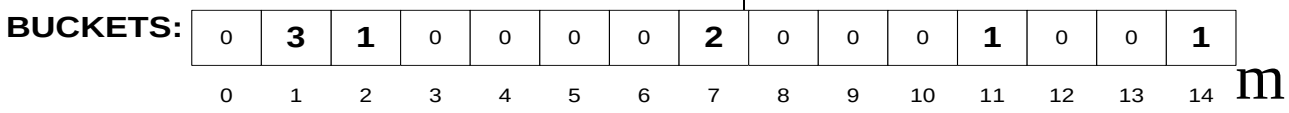
Αλγόριθμος BucketSort

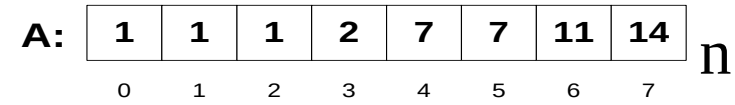
```
// A: Πίνακας για ταξινόμηση μεγέθους n
// Buckets: Βοηθητικός πίνακας μεγέθους m

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(m+n)

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





Χρόνος Εκτέλεσης

- **Ο *αλγόριθμος BucketSort* πετυχαίνει ταξινόμηση του Α σε χρόνο Θ(n+m):**
- **Σημαίνει ότι ο αλγόριθμος είναι καλύτερος από τον Mergesort Θ(nlogn)?**
  (Κατακρίβειαν μπορεί να αποδειχθεί ότι όλοι οι αλγόριθμοι ταξινόμησης, με **δυαδική σύγκριση**, έχουν σαν κάτω φράγμα Ω(n log n))
- **ΟΧΙ,** γιατί το μοντέλο είναι διαφορετικό. Μέχρι τώρα υποθέσαμε ότι η μόνη πράξη που μπορούμε να εφαρμόσουμε στα δεδομένα είναι η δυαδική σύγκριση στοιχείων. Ο αλγόριθμος BucketSort όμως στο Βήμα 2 ουσιαστικά εφαρμόζει ***m-αδική (m-ary) σύγκριση***, σε χρόνο Ο(1).
- Αυτό μας υπενθυμίζει πως σχεδιάζοντας ένα αλγόριθμο και λαμβάνοντας υπόψη κάποια αποδεδειγμένα κάτω φράγματα πρέπει πάντα να αναλύουμε το μοντέλο στο οποίο δουλεύουμε.
- Η ύπαρξη και αξιοποίηση περισσότερων πληροφοριών πιθανόν να επιτρέπουν τη δημιουργία αποδοτικότερων αλγορίθμων.