



Διάλεξη 20: Αλγόριθμοι Ταξινόμησης III

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

- Οι αλγόριθμοι ταξινόμησης:

 - E. QuickSort** – Γρήγορη Ταξινόμηση

- Έμμεση Ταξινόμηση

- Εξωτερική Ταξινόμηση

Διδάσκων: Παναγιώτης Ανδρέου

Ε. Γρήγορη Ταξινόμηση (QuickSort)

- Η **γρήγορη ταξινόμηση (QuickSort)** είναι, όπως ο Mergesort, διαδικασία **διαίρει και βασίλευε** (divide and conquer, δηλ. αναδρομική διαδικασία όπου το πρόβλημα μοιράζεται σε μέρη τα οποία λύνονται ξεχωριστά, και μετά οι λύσεις συνδυάζονται.).
- Δεν χρειάζεται βοηθητικό πίνακα (όπως στην Mergesort)
- **Πρακτικά, ο πιο γρήγορος αλγόριθμος.**
- Στη χειρίστη περίπτωση ο αλγόριθμος Quick Sort είναι $O(n^2)$ αλλά στην καλύτερη περίπτωση χρειάζεται $\Omega(n \log n)$:
- Τα περισσότερα συστήματα χρησιμοποιούν το Quick Sort (π.χ. Unix) και οι περισσότερες γλώσσες προγραμματισμού το προσφέρουν σαν μέρος των βασικών βιβλιοθηκών τους πχ. C, JAVA, C++, etc.

QuickSort: Αλγόριθμος

- Αν το δεδομένο εισόδου περιέχει 0 ή 1 στοιχεία δεν κάνουμε τίποτα.
- Διαφορετικά, αναδρομικά:
 1. διαλέγουμε ένα στοιχείο p (ακόμα δεν ορίζουμε πιο ακριβώς), το οποίο ονομάζουμε το **άξονα (pivot)** στοιχείο και το αφαιρούμε από το δεδομένο εισόδου.
 2. χωρίζουμε τον πίνακα σε δύο μέρη **S1** και **S2**, όπου το **S1** θα περιέχει όλα τα στοιχεία του πίνακα που είναι μικρότερα από το p , και το **S2** θα περιέχει τα υπόλοιπα στοιχεία (όλα τα στοιχεία που είναι μεγαλύτερα ή ίσα από το p).
 3. Καλούμε αναδρομικά τον αλγόριθμο στο **S1**, και παίρνουμε απάντηση το **T1**,
και
στο **S2**, και παίρνουμε απάντηση το **T2**.
 4. Επιστρέφουμε τον πίνακα **[T1, p, T2]**.

QuickSort: Βασική Ιδέα

[72, 12, 1, 34, 3, 50, 28, 6, 5, 22, 91, 73]

χωρίζουμε με ρινοτ το 28

(μπορούσε να είναι οποιοδήποτε άλλο στοιχείο)

Θέτουμε αριστερά του ρινοτ τα μικρότερα και δεξιά τα μεγαλύτερα του

< 28

>= 28

[12, 1, 3, 6, 5, 22] **28** [72, 34, 50, 91, 73]

Quicksort



1, 3, 5, 6, 12, 22

Quicksort



34, 50, 72, 73, 91

Αποτέλεσμα: 1, 3, 5, 6, 12, 22, 28, 34, 50, 72, 73, 91

QuickSort: Ψευδοκώδικας

```
void Quicksort(int A[], int l, int r){  
    if (l>=r) return;
```

```
    int pivotIndex = (l+r)/2;  
    int pivot = A[pivotIndex];
```

Εδώ διαλέξαμε τον μεσαίο. Θα μπορούσαμε να είχαμε διαλέξει οποιονδήποτε άλλο

```
    // κάνουμε swap τον pivot με το τελευταίο.  
    swap(A, pivotIndex, r);
```

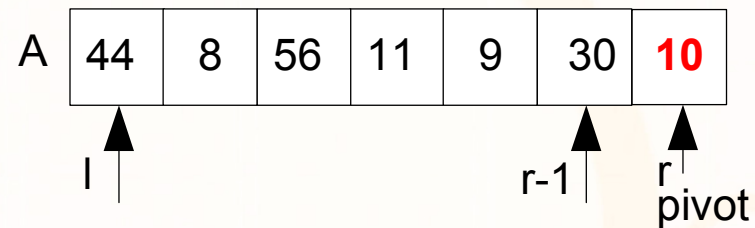
```
    /* Η διαδικασία partition χωρίζει τον πίνακα A[l..r-1] έτσι ώστε  
    A[l..k-1] να περιέχει στοιχεία < pivot, A[k..r-1] να περιέχει  
    στοιχεία >=pivot, και επιστρέφει την τιμή k. */
```

```
    int k = partition (A, l, r-1, pivot);
```

```
    // κάνουμε swap τον k με το τελευταίο.  
    swap(A, k, r);
```

```
    Quicksort(A, l, k-1);  
    Quicksort(A, k+1, r);
```

```
}
```

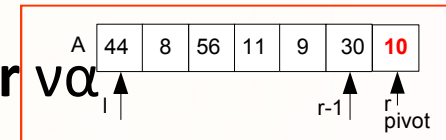


Διαδικασία Partition(A, l, r, p)

- Με δεδομένο εισόδο τον πίνακα $A[l\dots r]$, και **pivot** p , θέλουμε να χωρίσουμε τον πίνακα σε δύο μέρη ως προς το p .
- Το πιο πάνω πρέπει να επιτευχθεί χωρίς τη χρήση δεύτερου πίνακα (το sorting θα γίνει επί τόπου).

- Βασική Ιδέα:

1. Επαναλαμβάνουμε τα εξής μέχρις ότου τα l και r να συναντηθούν.
2. Προχώρα το r προς τα **αριστερά** όσο τα στοιχεία που βρίσκεις είναι μεγαλύτερα (ή ίσα) του p ,
3. Προχώρα το l προς τα **δεξιά** όσο τα στοιχεία που βρίσκεις είναι μικρότερα του p ,
4. αντάλλαξε τα στοιχεία που δείχνονται από τα l και r .



Παράδειγμα Εκτέλεσης Partition

Δεδομένο Εισόδου:

index	0	1	2	3	4	5	6	7
	72	6	37	48	30	42	83	75

pivot = 48, μετακίνηση του pivot στο τέλος (swap(4, 8)):

72	6	37	75	30	42	83	48
						r	

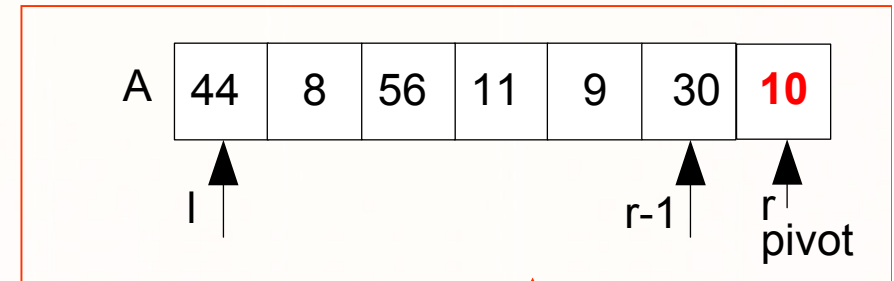
pivot

εκτέλεση του Partition(A, l, r, 48):

72	6	37	75	30	42	83	48
					r		
42	6	37	75	30	72	83	48
				r			
42	6	37	30	75	72	83	48
				r			
42	6	37	30	75	72	83	48
			r				

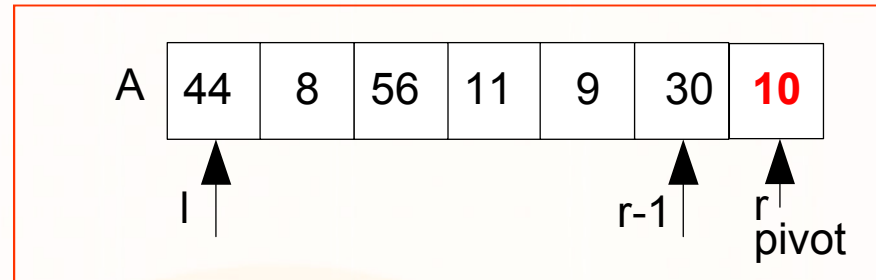
Ο Αλγόριθμος Quicksort στην C

```
void quicksort(int A[], int l, int r) {  
    int pivot, pivotIndex;  
  
    if (l>=r) return;  
  
    // Επιλογή του κατάλληλου pivot  
    pivotIndex = (l+r)/2;  
    pivot = A[pivotIndex];  
  
    // Μετακίνηση του pivot στο τέλος της λίστας  
    swap(A, pivotIndex, r);  
  
    /* Κλήση της συνάρτησης partition (η οποία τοποθετεί αριστερά,  
       δεξιά τα μικρότερα και μεγαλύτερα στοιχεία αντίστοιχα. */  
    pivotIndex = partition(A, l, r-1, pivot);  
  
    // Τοποθέτηση του pivot πίσω στην αρχική του θέση  
    if (A[r] < A[pivotIndex])  
        swap(A, pivotIndex, r);  
  
    quicksort(A, l, pivotIndex-1);  
    quicksort(A, pivotIndex+1, r);  
}
```



Η διαδικασία Partition στην C

```
int partition(int A[], int l, int r, int pivot) {  
  
    while(l<r) {  
        // προχωρούμε από l στο r μέχρι να χρειαστεί ένα swap  
        while (A[l]<pivot && l<r) // leave "<pivot" on left  
            l++;  
        // προχωρούμε από r στο l μέχρι να χρειαστεί ένα swap  
        while (pivot<=A[r] && l<r) //leave ">=pivot" on right  
            r--;  
  
        if (l == r) break;  
  
        // τώρα κάνε το swap  
        if (A[l]>=pivot)  
            swap(A, l, r); // move ">=" to the right  
    }  
  
    // επέστρεψε το σημείο στο οποίο θέλουμε να γίνει η  
    // εισαγωγή του pivot  
    return l;  
}
```



Παράδειγμα Εκτέλεσης QuickSort

```
BEFORE: [72, 6, 37, 48, 30, 42, 83, 75]
         Index: 0  1  2  3  4  5  6  7
```

```
** QuickSort [0,7]
```

```
[72, 6, 37, 48, 30, 42, 83, 75, ]
```

```
PivotIndex: 3(48) => Swapping 48, 75
```

```
[72, 6, 37, 75, 30, 42, 83, 48, ]
```

```
Partitioning [0,6]
```

```
    Swapping 72, 42
```

```
[42, 6, 37, 75, 30, 72, 83, 48, ]
```

```
    Swapping 75, 30
```

```
[42, 6, 37, 30, 75, 72, 83, 48, ]
```

```
Inserting Pivot at Position:4
```

```
Swapping 75, 48
```

```
[42, 6, 37, 30, 48, 72, 83, 75, ]
```

```
** QuickSort [0,3]
```

```
[42, 6, 37, 30, 48, 72, 83, 75, ]
```

```
PivotIndex: 1(6) => Swapping 6, 30
```

```
[42, 30, 37, 6, 48, 72, 83, 75, ]
```

```
Partitioning [0,2]
```

```
Inserting Pivot at Position:0
```

```
Swapping 42, 6
```

```
** QuickSort [0,-1] -> RETURN
```

```
** QuickSort [1,3]
```

```
[6, 30, 37, 42, 48, 72, 83, 75, ]
```

```
PivotIndex: 2(37) => Swapping 37, 42
```

```
[6, 30, 42, 37, 48, 72, 83, 75, ]
```

```
Partitioning [1,2]
```

```
Inserting Pivot at Position:2
```

```
Swapping 42, 37
```

```
** QuickSort [1,1] -> RETURN
```

```
** QuickSort [3,3] -> RETURN
```

```
** QuickSort [5,7]
```

```
[6, 30, 37, 42, 48, 72, 83, 75, ]
```

```
PivotIndex: 6(83) => Swapping 83, 75
```

```
[6, 30, 37, 42, 48, 72, 75, 83, ]
```

```
Partitioning [5,6] with pivot:83
```

```
Inserting Pivot at Position:6
```

```
** QuickSort [5,5] -> RETURN
```

```
** QuickSort [7,7] -> RETURN
```

```
AFTER: [6, 30, 37, 42, 48, 72, 75, 83, ]
```

Ανάλυση του Χρόνου Εκτέλεσης

- Η εύρεση του **ρινοτ** απαιτεί χρόνο $O(1)$ και η διαδικασία $\text{Partition}(A, l, r, p)$ εκτελείται σε **χρόνο $O(n)$** σε κάθε επίπεδο της αναδρομής.
- Η **αναδρομική εκτέλεση** του QuickSort παίρνει στην χειρίστη περίπτωση χρόνο $O(n)$ και στην καλύτερη περίπτωση χρόνο $\Omega(\log n)$

- **Χείριστη περίπτωση:**

Κάθε φορά που επιλέγουμε τον ρινοτ όλα τα στοιχεία τυγχάνει να ταξινομούνται είτε **μόνο** αριστερά του (δηλαδή $< \text{ρινοτ}$) ή **μόνο** δεξιά του (δηλαδή $\geq \text{ρινοτ}$) πχ 9,9,9,9,9,9,9

Συνολικός χρόνος εκτέλεσης $T(n) \in O(n^2)$.

- **Βέλτιστη περίπτωση:**

Κάθε φορά που επιλέγουμε τον ρινοτ τα **μισά** στοιχεία ταξινομούνται αριστερά του (δηλαδή $< \text{ρινοτ}$) και τα **υπόλοιπα μισά** δεξιά του (δηλαδή $\geq \text{ρινοτ}$) πχ 1,2,3,4,5,6,7

Συνολικός χρόνος εκτέλεσης $T(n) \in \Omega(n \log n)$.

Έμμεση Ταξινόμηση (Indirect Sorting)

Έμμεση Ταξινόμηση (Indirect Sorting)

- Οι αλγόριθμοι που έχουμε παρουσιάσει υποθέτουν πως οι πίνακες-δεδομένα εισόδου περιέχουν **ακέραιους αριθμούς** και προϋποθέτουν μετακίνηση των στοιχείων μέσα στον πίνακα.
Τι γίνεται αν θέλουμε να ταξινομήσουμε αρχεία που περιέχουν πολύπλοκα αντικείμενα (π.χ. εγγραφές με πολλαπλές στήλες)?
- Σε τέτοιες περιπτώσεις η μετακίνηση (swapping) στοιχείων είναι δαπανηρή.
- Αυτό μπορεί να αποφευχθεί με τη **χρήση δεικτών**: ως δεδομένο εισόδου χρησιμοποιούμε **πίνακα που περιέχει δείκτες** στα στοιχεία που θέλουμε να ταξινομήσουμε.
- Σύγκριση και ανταλλαγή γίνεται μεταξύ των δεικτών αντί μεταξύ των ιδίων των αντικειμένων. Έτσι έχουμε λιγότερη μετακίνηση στοιχείων.
- Οποιοσδήποτε από τους αλγόριθμους που αναφέραμε μπορεί να υλοποιηθεί με αυτό τον συλλογισμό χωρίς να αλλάξει ο αλγόριθμος.

Εξωτερική ταξινόμηση (External Sorting)

- Τι γίνεται αν έχουμε 256MB RAM αλλά θέλουμε να ταξινομήσουμε ένα αρχείο με 800MB έγγραφες;
- Η ταξινόμηση μπορεί να γίνεται κατά **τμήματα**:
 - Ένα μέρος του αρχείου μεταφέρεται στην κύρια μνήμη, ταξινομείται (**με ένα από τους αλγόριθμους που συζητήσαμε**) και αποθηκεύεται σε ένα προσωρινό αρχείο.
 - Το επόμενο τμήμα μεταφέρεται στην κύρια μνήμη και ταξινομείται και μετά συγχωνεύεται με το προσωρινό αρχείο.
 - Η διαδικασία επαναλαμβάνεται μέχρι εξάντλησης του αρχικού αρχείου.
- Με βάση αυτή την κύρια ιδέα υπάρχουν διάφοροι αλγόριθμοι εξωτερικής ταξινόμησης. Κύριος στόχος τους είναι η αποδοτική επεξεργασία της δευτερεύουσας μνήμης (δηλαδή του μαγνητικού δίσκου).
- Τέτοιοι αλγόριθμοι βρίσκουν εφαρμογές σε Βάσεις Δεδομένων