[](http://www.teilar.gr/)

**Αλγόριθμοι και Στοιχεία Πολυπλοκότητας.**

**Ενότητα 5:** «Διαίρει και Βασίλευε».

Διδάσκων: Ηλίας Κ Σάββας, Αναπληρωτής Καθηγητής.

Τμήμα Μηχανικών Πληροφορικής, Τεχνολογικής Εκπαίδευσης.

**Άδειες χρήσης.**

* Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons (C C). **Αναφορά δημιουργού (B Y), Μη εμπορική χρήση (N C), Μη τροποποίηση (N D), 3.0, Μη εισαγόμενο.**
* Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

[](http://creativecommons.org/licenses/by-nc-nd/3.0/deed.el)

**Χρηματοδότηση.**

* Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
* Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

[](http://www.edulll.gr/)

# Σκοποί ενότητας.

Ο αναγνώστης να μπορεί να:

1) χρησιμοποιεί την τεχνική «Διαίρει και Βασίλευε»,

2) επιλύει προβλήματα εφαρμόζοντας την παραπάνω τεχνική.

# ΠΕΡΙΕΧΌΜΕΝΑ ΕΝΌΤΗΤΑΣ.

[Σκοποί ενότητας. 2](#_Toc368169703)

[ΠΕΡΙΕΧΌΜΕΝΑ ΕΝΌΤΗΤΑΣ. 3](#_Toc368169704)

[ΑΛΓΌΡΙΘΜΟΙ 3](#_Toc368169705)

[ΠΡΟΓΡΆΜΜΑΤΑ 3](#_Toc368169706)

[5. «ΔΙΑΊΡΕΙ ΚΑΙ ΒΑΣΊΛΕΥΕ». 4](#_Toc368169707)

[5.1 Εισαγωγή. 4](#_Toc368169708)

[5.2 Η μέθοδος ταξινόμησης Merge Sort. 4](#_Toc368169709)

[5.3 Πολυπλοκότητα της Merge Sort. 6](#_Toc368169710)

[5.4 Η μέθοδος ταξινόμησης Quick Sort. 7](#_Toc368169711)

[5.5 Quick Sort: Πολυπλοκότητα και υλοποίηση. 9](#_Toc368169712)

# ΑΛΓΌΡΙΘΜΟΙ

|  |  |
| --- | --- |
| ***Αλγόριθμος 1****: Merge Sort.* | 4 |
| ***Αλγόριθμος 2****: Ενοποίηση ταξινομημένων πινάκων.* | 5 |

# ΠΡΟΓΡΆΜΜΑΤΑ

|  |  |
| --- | --- |
| ***Πρόγραμμα 1****: Ενοποίηση ταξινομημένων πινάκων.* | 6 |
| ***Πρόγραμμα 2****: Quick sort.* | 9 |

# 5. «ΔΙΑΊΡΕΙ ΚΑΙ ΒΑΣΊΛΕΥΕ».

## 5.1 Εισαγωγή.

Μία από τις κλασικές μεθόδους επίλυσης προβλημάτων είναι η λεγόμενη «Διαίρει και Βασίλευε» (Divide and Conquer). Η μέθοδος οφείλει την ονομασία της στους αρχαίους Ρωμαίους πολιτικούς, (οι οποίοι προφανώς δεν σκεφτόταν την ανάπτυξη αλγορίθμων). Η τεχνική ήταν απλή: διαίρεσε (divide) τους εχθρούς σου, βάζοντάς τους να φιλονικούν μεταξύ τους, και στη συνέχεια κατέκτησέ τους έναν-έναν (conquer). Στην ανάπτυξη αλγορίθμων, η μέθοδος έγκειται στην διάσπαση ενός προβλήματος, σε μικρά κομμάτια του ίδιου προβλήματος, και στην συνέχεια στην συνένωση των επιμέρους λύσεων στη γενική λύση του προβλήματος. Βέβαια, σε κάθε ένα από τα επιμέρους προβλήματα, πρέπει αναδρομικά να εφαρμοσθεί ή ίδια μέθοδος, μέχρι να καταλήξει σε προβλήματα τάξης μεγέθους ένα ή το πολύ δύο, όπου η λύση είναι εύκολη. Συνοψίζοντας, τα βήματα επίλυσης ενός προβλήματος με αυτή την μέθοδο, είναι τα ακόλουθα:

1. Διαίρει (το πρόβλημα σε μικρότερα επιμέρους προβλήματα),
2. Βασίλευε (λύνοντας το κάθε επιμέρους πρόβλημα, εφαρμόζοντας αναδρομικά την ίδια τακτική), και τέλος
3. Συνένωσε (τις επιμέρους λύσεις σε μία καθολική λύση του προβλήματος).

## 5.2 Η μέθοδος ταξινόμησης Merge Sort.

Αυτή η μέθοδος ταξινόμησης αποτελεί ένα πολύ καλό παράδειγμα εφαρμογής της τεχνικής «διαίρει και βασίλευε». Είναι ένας καλός και πολύ αποτελεσματικός αλγόριθμος ταξινόμησης. Η εφαρμογή της μεθόδου συνοψίζεται ως εξής:

1. Διαίρει: Διαχωρισμός των στοιχείων στην μέση ώστε να προκύψουν δύο πίνακες, με τα μισά περίπου στοιχεία ο καθένας,
2. Βασίλευε: Ταξινόμηση κάθε υπό-πίνακα στοιχείων (καλώντας την ίδια μέθοδο αναδρομικά), και
3. Συνένωσε: Ένωση των ταξινομημένων υπό-πινάκων σε ένα νέο ταξινομημένο.

***Αλγόριθμος 1****: Merge Sort.*

1: Διαδικασία MergeSort(πίνακας Α, ακέραιος p, ακέραιος r).

2: Αρχή,

3: Εάν (p < r). Τότε,

4: q = (p+r) / 2,

5: MergeSort(A, p, q),

6: MergeSort(A, q+1, r),

7: Merge(A, p, q, r),

8: Τέλος Εάν.

9: Τέλος Διαδικασίας «MergeSort».

Επομένως, για να ταξινομηθεί ολόκληρος ο πίνακας, ο αλγόριθμος πρέπει να κληθεί: MergeSort(A, 1, N), εάν Ν είναι το πλήθος των στοιχείων του.

Βέβαια, για να ολοκληρωθεί η διαδικασία της ταξινόμησης, πρέπει να λυθεί και το πρόβλημα της συνένωσης (merge), των δύο ταξινομημένων υπό-πινάκων. Ο αλγόριθμος επίλυσης αυτού του προβλήματος, δίνεται στον *Αλγόριθμο 2,* καθώς επίσης και η υλοποίησή του στο *Πρόγραμμα 1*.

***Αλγόριθμος 2****: Ενοποίηση ταξινομημένων πινάκων.*

1: Διαδικασία Merge(πίνακας Α, ακέραιος p, ακέραιος q, ακέραιος r).

2: Αρχή,

3: Πίνακας ακεραίων Βr-p.

4: Ακέραιοι: i, j, k.

5: i 🡨 p.

6: k 🡨 p.

7: j 🡨 q+1.

8: Εφόσον (i ≤ q KAI j ≤ r),

9: Εάν (Ai ≤ Aj). Τότε,

10: Βk 🡨 Ai,

11: k 🡨 k+1,

12: i 🡨 i+1.

13: Αλλιώς,

14: Βk 🡨 Aj,

15: k 🡨 k+1,

16: j 🡨 j+1,

17: Τέλος Εάν.

18: Τέλος Εφόσον.

19: Εφόσον (i ≤ q ),

20: Βk 🡨 Ai,

21: k 🡨 k+1,

22: i 🡨 i+1,

23: Τέλος Εφόσον.

24: Εφόσον (j ≤ r),

25: Βk 🡨 Aj,

26: k 🡨 k+1,

27: j 🡨 j+1,

28: Τέλος Εφόσον.

29: Για i=1 μέχρι r,

30: Ai 🡨 Bi,

31: Τέλος Για (i).

32: Τέλος Διαδικασίας «Merge».

***Πρόγραμμα 1****: Ενοποίηση ταξινομημένων πινάκων.*

void merge(A[], int p, int q, int r) {

int B[r-p];

int i = k = p;

int j = q+1;

while (i <= q && j <= r)

if (A[i] <= A[j])

B[k++] = A[i++];

else

B[k++] = A[j++];

while (i <= q) B[k++] = A[i++];

while (j <= r) B[k++] = A[j++];

for (i=0; i < r; i++)

A[i] = B[i];

}

Το μειονέκτημα της μεθόδου είναι ότι χρησιμοποιεί ένα βοηθητικό πίνακα (Β), και τον οποίο στην συνέχεια τον μεταφέρει στον αρχικό, αλλά κατά τα άλλα είναι μία γρήγορη μέθοδος ταξινόμησης, που η τάξη πολυπλοκότητάς της είναι .

## 5.3 Πολυπλοκότητα της Merge Sort.

Εάν υποτεθεί ότι ο χρόνος ταξινόμησης ενός πίνακα N στοιχείων είναι Τ(Ν), τότε επειδή ο πίνακας πρέπει να διαιρεθεί σε δύο υπό-πίνακες, και επιπλέον να γίνει και η συγχώνευση των Ν στοιχείων, είναι προφανές ότι:

.

Πιο αναλυτικά για να δοθεί μία εικόνα της πολυπλοκότητας είναι:

T(1) = 1,

T(2) = 2T(1) + 2 = 4,

T(3) = T(1) + T(2) + 3 = 8,

T(4) = T(2) + T(2) + 4 = 12,

T(5) = T(2) + T(3) + 5 = 17,

…………

Αν και είναι αρκετά δύσκολο να βρεθεί ένας γενικός τύπος από τις παραπάνω τιμές, δεν είναι ακατόρθωτο, και αυτός είναι . Η απόδειξη πάλι με την μέθοδο της επαγωγής (όπως σε σχεδόν όλες τις περιπτώσεις αναδρομικών αλγόριθμων).

**Απόδειξη.**

Για Ν=1, ισχύει γιατί Τ(1) = 1 = 1 lg1.

Έστω ότι ισχύει για κάθε αριθμό μικρότερο του Ν. Τότε για Ν έχουμε:

.

Επομένως



και επομένως αποδείχθηκε.

Δηλαδή, η πολυπλοκότητα της merge sort, που είναι της τάξης , δείχνει ότι πρόκειται για μία γρήγορη μέθοδο ταξινόμησης.

## 5.4 Η μέθοδος ταξινόμησης Quick Sort.

Η ιδέα της γρήγορης ταξινόμησης (quick sort), βασίζεται στο ότι εάν βρεθεί ένα κατάλληλο στοιχείο *οδηγό,* και αναδιαταχθεί ο αρχικός πίνακας σε δύο υπό-πίνακες, όπου όλα τα στοιχεία του πρώτου μπορεί μεν να μην είναι ταξινομημένα, αλλά είναι μικρότερα του οδηγού στοιχείου, και όλα τα στοιχεία του δεύτερου να είναι μεγαλύτερα του οδηγού, τότε δεν αρκεί να επαναληφθεί αυτή η διαδικασία για τους δύο μικρότερους πίνακες, ώσπου να καταλήξει σε πίνακες στοιχεία, και τότε η διαδικασία θα έχει ολοκληρωθεί, (μέθοδος διαίρει και βασίλευε). Για παράδειγμα στον παρακάτω πίνακα:

26, 18, 4, 9, 37, 119, 220, 47, 74.

Ο αριθμός 37 είναι το ζητούμενο στοιχείο.

Βέβαια, το πρόβλημα τώρα τίθεται στην εύρεση του οδηγού στοιχείου. Αρχικά πάντως η μέθοδος της γρήγορης ταξινόμησης, θα μπορούσε να περιγραφεί ως εξής:

ΓρήγορηΤαξινόμηση1(x: πίνακας Ν στοιχείων).

Εάν Ν >= 2. Τότε,

Εύρεση του οδηγού στοιχείου x[i],

ΓρήγορηΤαξινόμηση1(του υποπίνακα ),

ΓρήγορηΤαξινόμηση1(του υποπίνακα ),

Τέλος Εάν.

Τέλος {ΓρήγορηΤαξινόμηση1}.

Βέβαια, αυτός ο αλγόριθμος δεν δουλεύει, αλλά δείχνει καθαρά ότι πρόκειται για έναν αναδρομικό αλγόριθμο (recursive). Για να γίνει λίγο πιο γενικός, θα μπορούσε να δοθεί με τρόπο, ώστε να ταξινομεί ένα πίνακα με αρχικό δείκτη *left,* και τελικό *right*. Δηλαδή:

ΓρήγορηΤαξινόμηση2(x: πίνακας, left, right).

Εάν right -left >= 1. Τότε,

Εύρεση του οδηγού στοιχείου x[i],

ΓρήγορηΤαξινόμηση2(x, left, i-1),

ΓρήγορηΤαξινόμηση2(x, i+1, right),

Τέλος Εάν.

Τέλος {ΓρήγορηΤαξινόμηση2}.

Και φυσικά για την ταξινόμηση ολόκληρου του πίνακα δεν έχουμε παρά να τεθεί,

left = 1, right = N.

Το επόμενο ζητούμενο είναι ο εντοπισμός του οδηγού στοιχείου. Η επιλογή του στοιχείου αυτού είναι όμως πάρα πολύ απλή, είναι τυχαία. Απλώς επιλέγεται με χρήση κάποιας συνάρτησης παραγωγής τυχαίων αριθμών, ένα στοιχείο του προς ταξινόμηση πίνακα έστω *Τ*. Μόλις επιλεγεί ο οδηγός, η επόμενη διαδικασία είναι η αναδιάταξη του πίνακα, ώστε αριστερά να υπάρχουν στοιχεία μόνο μικρότερα του οδηγού, ενώ δεξιά μόνο μεγαλύτερα. Αυτή η διαδικασία μπορεί να περιγραφεί με τον παρακάτω αλγόριθμο:

**ΔιαχωρισμόςΠίνακα**(x, left, right, i),

{i είναι η τελική θέση του T}.

L = τυχαίος αριθμός στο διάστημα [left, right].

Αντιμετάθεσε(x[left], x[L]),

{επομένως, τώρα το οδηγό στοιχείο είναι στην πρώτη θέση}.

T = x[left],

i = left.

Για j = left+1 μέχρι right,

Εάν x[j] < T. Τότε,

i = i+1;

αντιμετάθεσε(x[i], x[j]),

Τέλος Εάν.

Τέλος Για,

Αντιμετάθεσε(x[left], x[i]).

Τέλος {Διαχωρισμός Πίνακα}.

Οπότε, τελικά η μέθοδος της γρήγορης ταξινόμησης μπορεί να περιγραφεί ως εξής:

**ΓρήγορηΤαξινόμηση3**(x:πίνακας, left, right).

Εάν right -left >= 1. Τότε,

ΔιαχωρισμόςΠίνακα(x, left, right, i),

ΓρήγορηΤαξινόμηση3(x: πίνακας, left, i-1),

ΓρήγορηΤαξινόμηση3(x: πίνακας, i+1, right),

Τέλος Εάν.

Τέλος {ΓρήγορηΤαξινόμηση3}.

## 5.5 Quick Sort: Πολυπλοκότητα και υλοποίηση.

Όπως αποδεικνύεται (η απόδειξη είναι πέραν του σκοπού αυτού του μαθήματος), η πολυπλοκότητα της quick sort είναι στην χειρότερη περίπτωση πάλι *O*(*N2*), αλλά η μέση περίπτωση είναι *O*(*N log*(*Ν*)), και αυτή θεωρείται και ως βασική. Δηλαδή, η γρήγορη ταξινόμηση είναι ασύγκριτα πιο γρήγορη από τις αργές μεθόδους, με πολυπλοκότητες της τάξης του *O*(*N2*), και μάλιστα σε τάξη μεγέθους  Για παράδειγμα, εάν μία τράπεζα ήθελε να ταξινομήσει τους 5.000.000 καταθέτες της, η αργή ταξινόμηση θα χρειαζόταν 25.000.000.000.000 συγκρίσεις, ενώ η γρήγορη, μόλις 105.000.000, δηλαδή 238.000 φορές πιο γρήγορη.

Στο *Πρόγραμμα 2* δίνεται μία υλοποίηση της μεθόδου quick sort, με κάπως διαφορετικό τρόπο επιλογής του οδηγού στοιχείου.

***Πρόγραμμα 2****: Quick sort.*

#include <stdio.h>

#define N 20, /\* Μέγεθος Πίνακα \*/,

void quicksort(int \*, int, int);

int pivot(int, int);

int partition(int \*, int, int);

main()

{

int A[N] = {4, 7, 9, 0, 11, 6, 2, -8, 3, 6, -7, 23, 5, 12, -8, -6, 3, 9, -1, 10};

int i;

/\* Εκτύπωση αταξινόμητου πίνακα \*/,

printf("\n Αταξινόμητος Πίνακας \n");

for (i=0; i<N; i++)

printf("%10d", A[i]);

printf("\n\n");

system("pause");

quicksort(&A[0], 0, N-1);

/\* Ταξινομημένος Πίνακας \*/,

printf("\n\n Ταξινομημένος Πίνακας \n");

for (i=0; i<N; i++)

printf("%10d", A[i]);

printf("\n\n");

}

/\* ΣΥΝΑΡΤΉΣΕΙΣ \*/,

void quicksort(int P[], int p, int r)

{

int temp;

int q, i;

if (r <= p) return;

i = pivot(p, r);

/\* αντιμετάθεση P[i] με P[p] \*/,

temp = P[i];

P[i] = P[p];

P[p] = temp;

q = partition(&P[0], p, r);

quicksort(&P[0], p, q-1);

quicksort(&P[0], q+1, r);

}

int pivot(int a, int b)

{

int x;

x = rand() % (b -a) + a;

return x;

}

int partition(int P[], int p, int r)

{

int x, q, temp, i;

x = P[p];

q = p;

for (i=p; i<=r; i++)

if (P[i] < x) {

q++;

/\* Αντιμετάθεση \*/,

temp = P[q];

P[q] = P[i];

P[i] = temp;

}

/\* Αντιμετάθεση P[p] με P[q] \*/,

temp = P[p];

P[p] = P[q];

P[q] = temp;

return q;

}

Τέλος πέμπτης ενότητας.

[](http://creativecommons.org/licenses/by-nc-nd/3.0/deed.el)[](http://www.edulll.gr/)