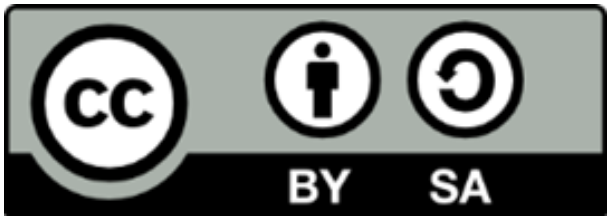
ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

Ενότητα 7α: Μελέτη πολυπλοκότητας των Αλγόριθμων quicksort & mergesort

Μαρία Σατρατζέμη
Τμήμα Εφαρμοσμένης Πληροφορικής

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- 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.

1. **Γρήγορη ταξινόμηση**
2. **Ταξινόμηση με Συγχώνευση**

Γρήγορη Ταξινόμηση

- Η **γρήγορη ταξινόμηση** (*quicksort*), που αλλιώς ονομάζεται και **ταξινόμηση με διαμερισμό και ανταλλαγή** (*partition exchange sort*), θεωρείται ένας από τους top-10 αλγορίθμους ως προς την πρακτική χρηστότητα και το ενδιαφέρον που προκάλεσαν από τη θεωρητική σκοπιά.
- Η γρήγορη ταξινόμηση αποτελεί ένα κλασικό παράδειγμα αλγορίθμου της **οικογενείας Διαίρει και Βασίλευε**.
- Όπως δηλώνει το πρώτο όνομα του αλγορίθμου, πρόκειται για μία πολύ αποτελεσματική μέθοδο ταξινόμησης (πλην ελαχίστων εξαιρέσεων). Όπως δηλώνει το δεύτερο όνομά του, τα βασικά χαρακτηριστικά του είναι δύο: **η ανταλλαγή και ο διαμερισμός**.

Γρήγορη Ταξινόμηση

- Πρώτον, δηλαδή, εκτελούνται ανταλλαγές μεταξύ των στοιχείων του πίνακα έτσι ώστε τα στοιχεία με μικρότερες τιμές να μετακινηθούν προς τη μία πλευρά, ενώ τα στοιχεία με μεγαλύτερες τιμές να μετακινηθούν προς την άλλη πλευρά του πίνακα.
- Έτσι, επακολουθεί η εφαρμογή του δεύτερου χαρακτηριστικού, δηλαδή του διαμερισμού του πίνακα σε δύο μικρότερους που ταξινομούνται ανεξάρτητα μεταξύ τους.
- Είναι προφανές ότι στους δύο υποπίνακες επιφυλάσσεται η ίδια αντιμετώπιση: ανταλλαγή και διαμερισμός.

Αλγόριθμος quicksort

- Δίνεται προς ταξινόμηση ένας πίνακας $A[1 .. n]$, που περιέχει ακεραίους αριθμούς.
- Το pivot (αξονικό) είναι ένα στοιχείο του πίνακα, το οποίο παίρνει την τελική του θέση στον πίνακα, ώστε κατόπιν να γίνει ο διαμερισμός σε δύο υποπίνακες με μικρότερα και μεγαλύτερα στοιχεία από το pivot αντίστοιχα.
- Η επιλογή του pivot μπορεί να γίνει με πολλούς τρόπους.
- Δεχόμαστε ότι ως pivot λαμβάνεται το πρώτο στοιχείο του πίνακα.

- Ο πίνακας σαρώνεται από τα αριστερά προς τα δεξιά αναζητώντας το πρώτο στοιχείο με τιμή μεγαλύτερη ή ίση από την τιμή του pivot.
- 'Υστερα, ο πίνακας σαρώνεται από τα δεξιά προς τα αριστερά αναζητώντας το πρώτο στοιχείο με τιμή μικρότερη ή ίση από την τιμή του pivot.
- Όταν εντοπισθούν δύο τέτοια στοιχεία, τότε αυτά ανταλλάσσονται ώστε κατά το δυνατόν να τείνουν να πλησιάσουν προς την τελική τους θέση.
- Συνεχίζουμε τις σαρώσεις προσπαθώντας να εντοπίσουμε άλλα παρόμοια ζεύγη και να τα ανταλλάξουμε. Κάποια στιγμή οι δύο δείκτες σάρωσης διασταυρώνονται.
- Τότε η σάρωση σταματά και το pivot λαμβάνει την τελική του θέση εκτελώντας άλλη μία ανταλλαγή μεταξύ του pivot και του στοιχείου όπου σταμάτησε ο δείκτης της σάρωσης από δεξιά.
- Έτσι, η διαδικασία μπορεί να συνεχίσει στους δύο υποπίνακες κατά τα γνωστά.

Αλγόριθμος quicksort(*A*, *left*, *right*);

```
1.    if left < right then
2.        lo ← left; hi ← right + 1; pivot ← A[left];
3.        do
4.                do lo ← lo+ 1 while A[lo] <= pivot;
5.                do hi ← hi -1 while A[hi] >= pivot;
6.                if lo < hi then swap(A[lo], A[hi]);
7.        while hi >= lo;
8.        swap(A[left], A[hi]);
9.        quicksort(A, left, hi-1);
10.       quicksort(A, hi+1, right)
```

Επιλογή Pivot

Υπάρχουν διάφοροι τρόποι επιλογής του pivot. Στο παράδειγμα επιλέγεται το πρώτο στοιχείο:

40	20	10	80	60	50	7	30	100

Διαμερισμός πίνακα

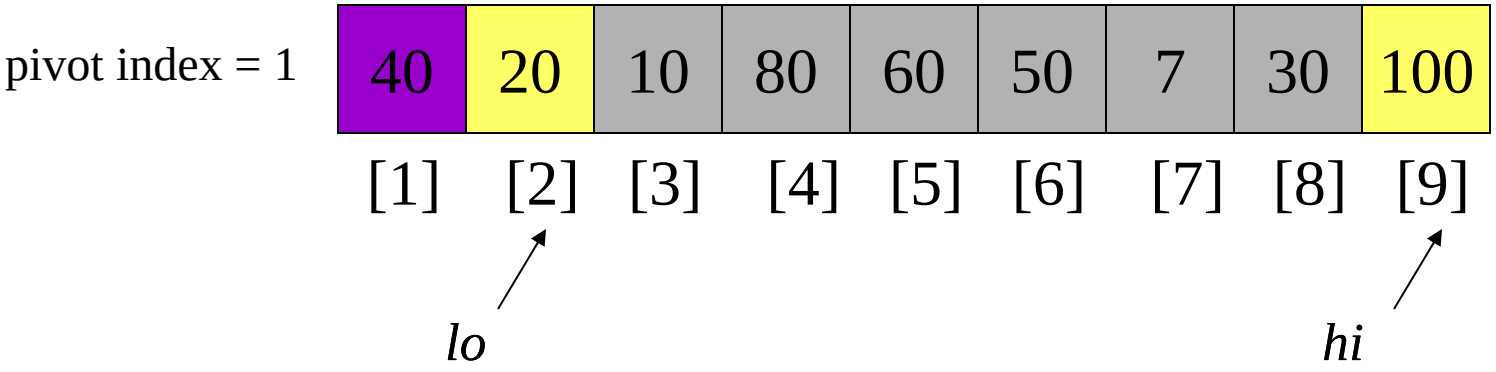
Δοθέντος του pivot διαμερίζουμε τα στοιχεία του πίνακα ώστε ο πίνακας που θα προκύψει να αποτελείται από:

1. Έναν υποπίνακα που περιέχει στοιχεία >= pivot
2. Έναν υποπίνακα που περιέχει στοιχεία < pivot

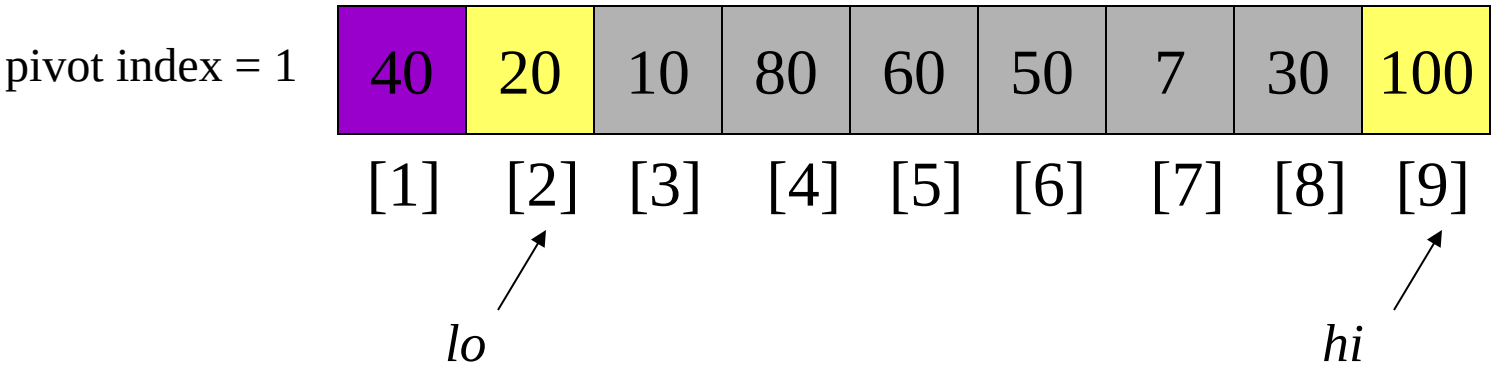
Οι υποπίνακες αποθηκεύονται στον αρχικό πίνακα.

Κατά το διαμοιρασμό γίνεται ανταλλαγή στοιχείων μικρότερων / μεγαλύτερων του pivot.

Ένα παράδειγμα

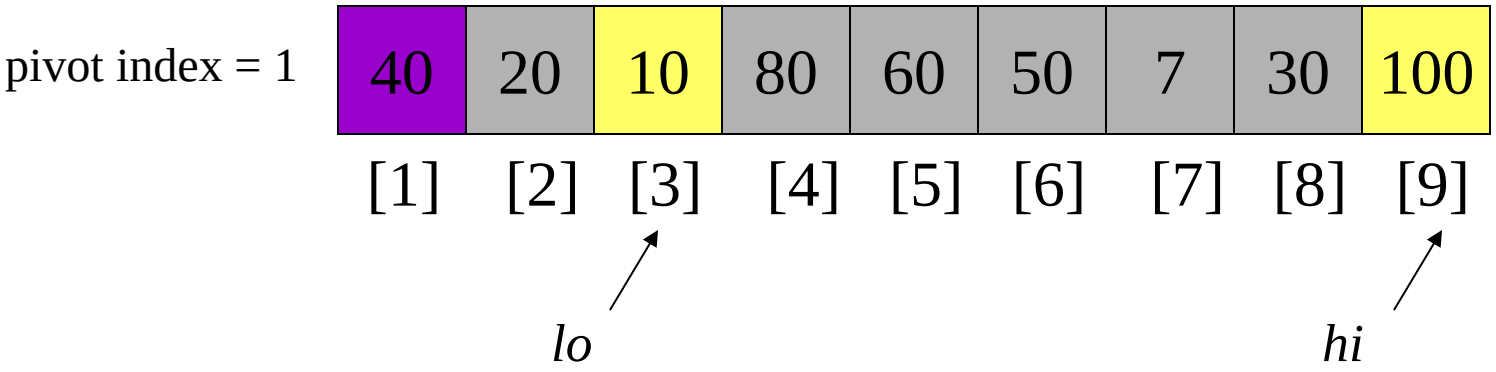


4. do $lo \leftarrow lo + 1$ while $A[lo] <= pivot$



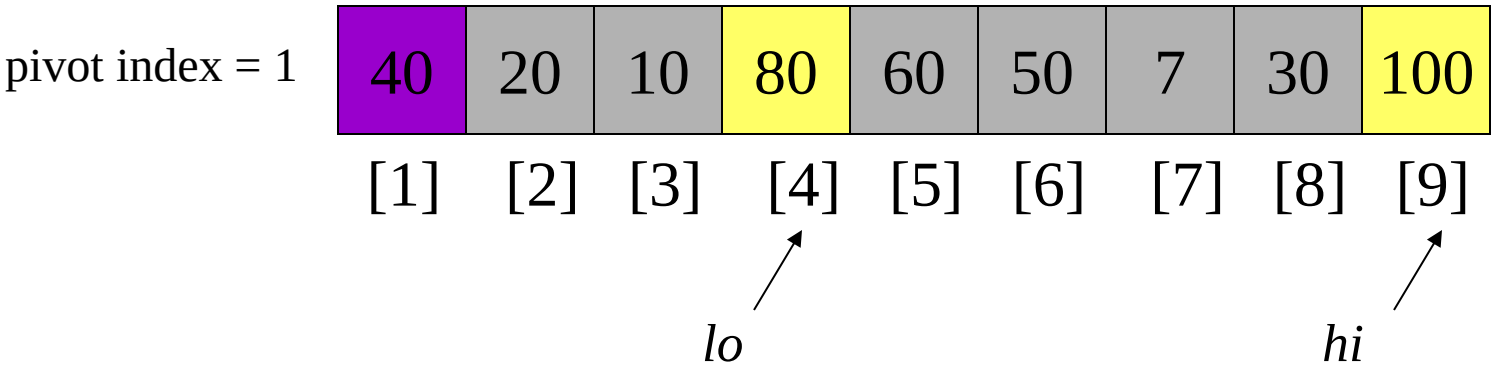
Ένα παράδειγμα

4. do $lo \leftarrow lo + 1$ while $A[lo] <= pivot$



Ένα παράδειγμα

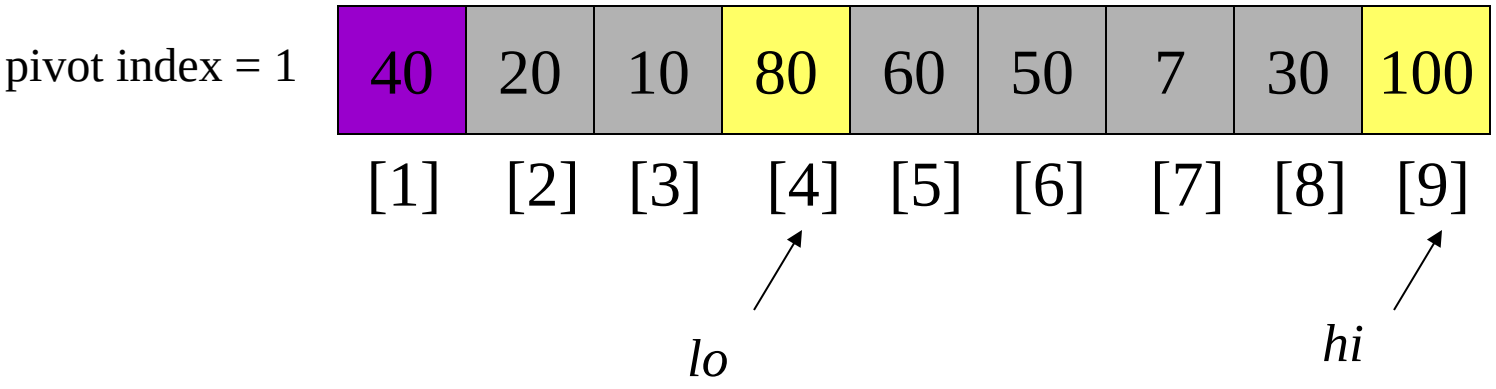
4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*



Ένα παράδειγμα

4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

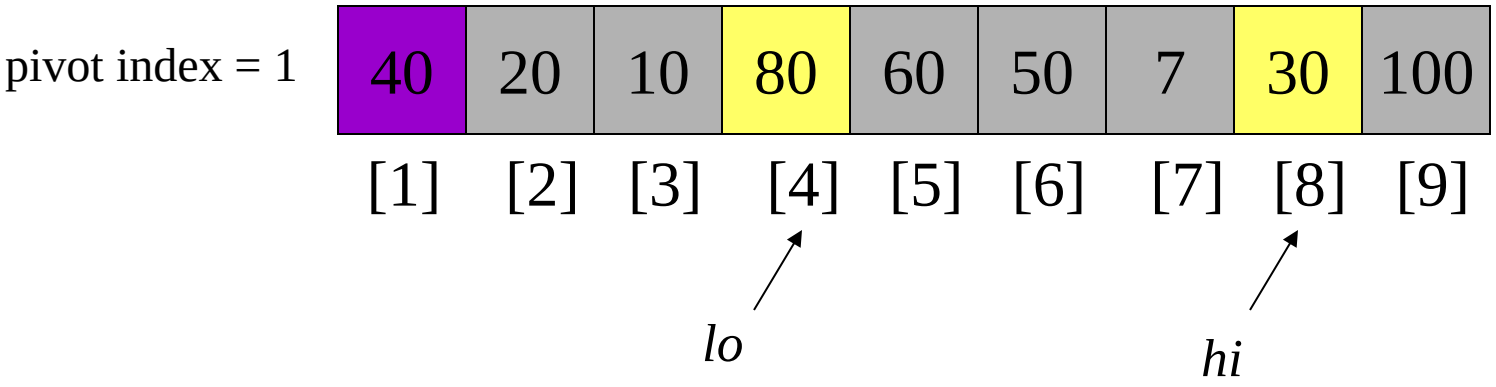
5. do *hi* ← *hi* - 1 while *A*[*hi*] >= *pivot*



Ένα παράδειγμα

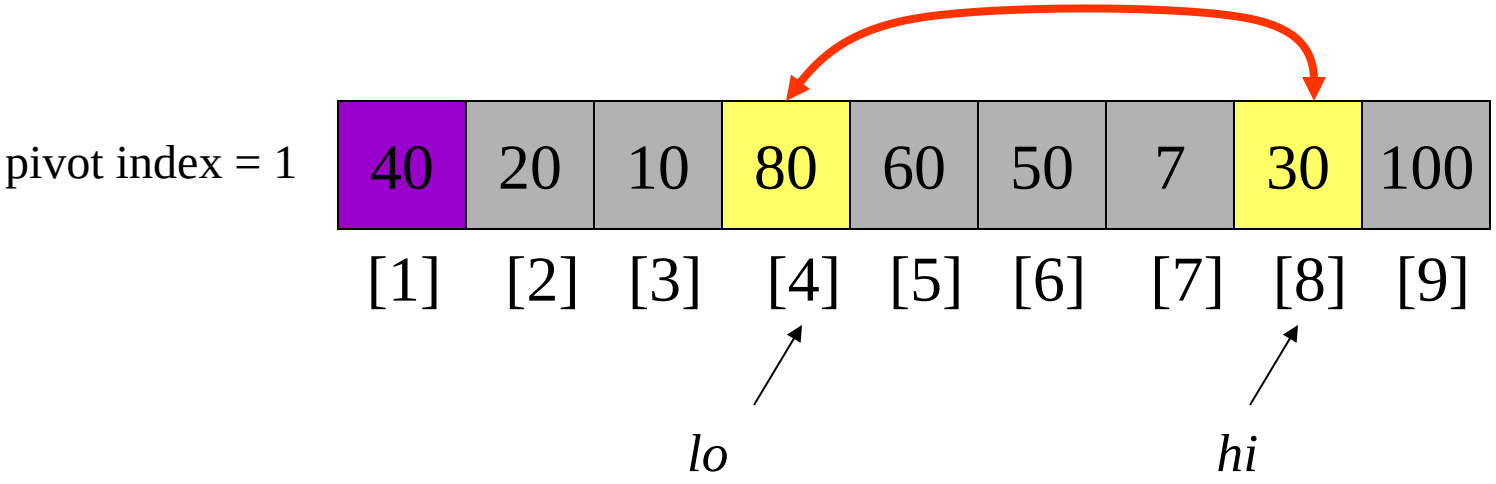
4. do $lo \leftarrow lo + 1$ while $A[lo] <= pivot$

5. do $hi \leftarrow hi - 1$ while $A[hi] >= pivot$



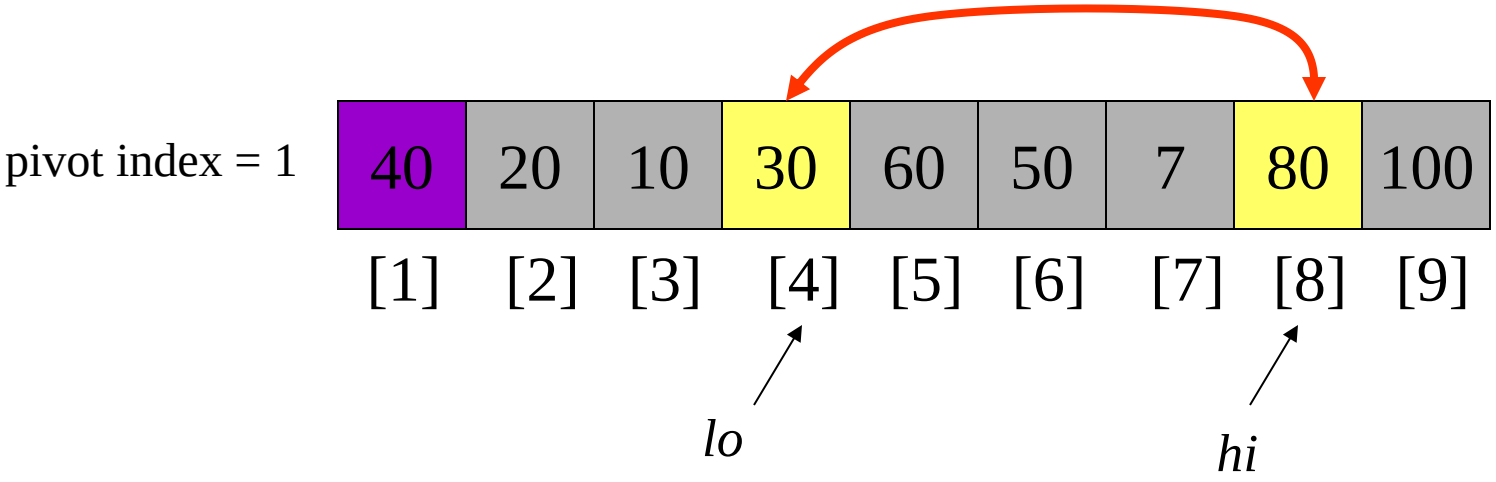
Ένα παράδειγμα

4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*
5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot
6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])



Ένα παράδειγμα

4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*
5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot
6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])



Ένα παράδειγμα

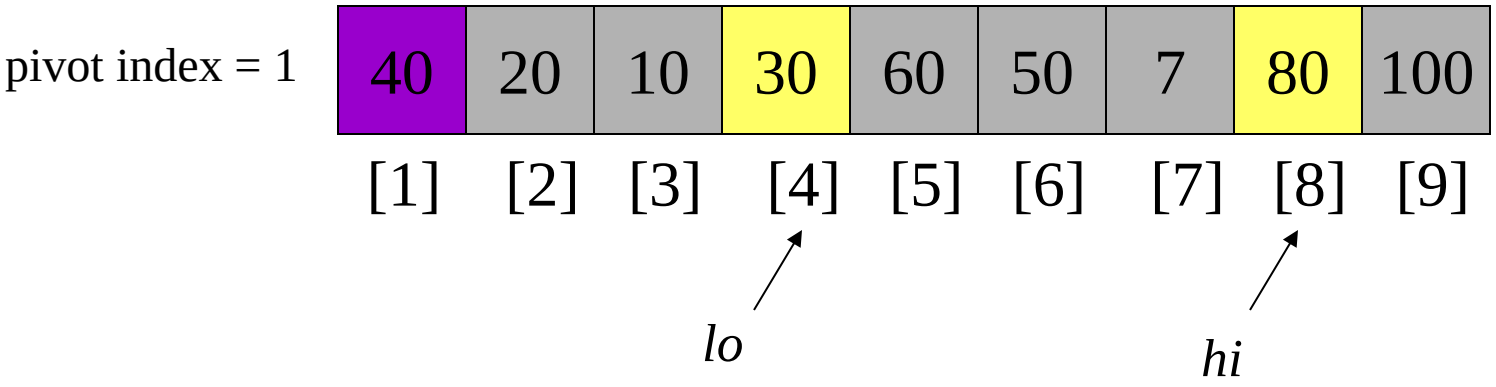
3. do

   4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

   5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

   6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

7. while *hi* >= *lo;*



Ένα παράδειγμα

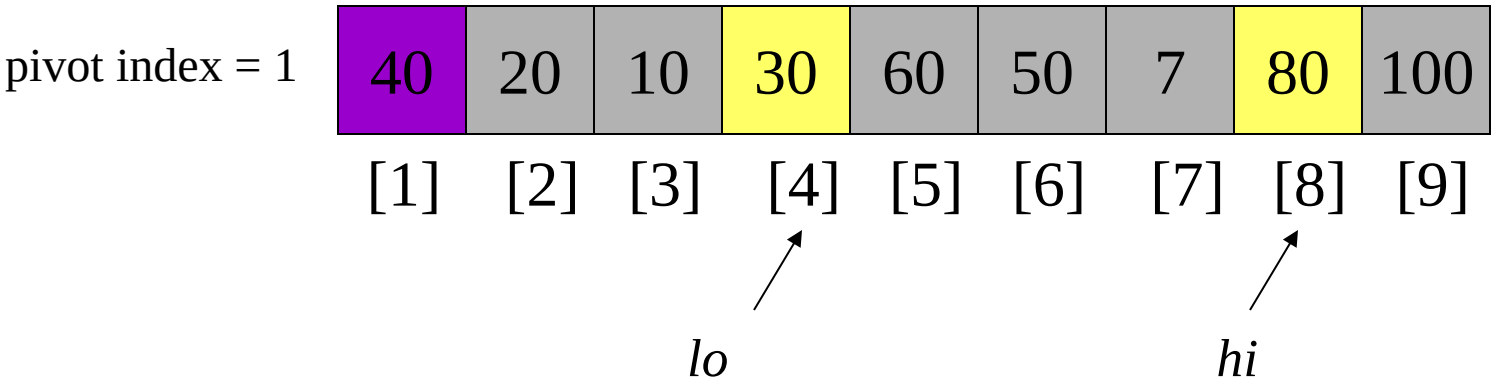
3. do

4. do $lo \leftarrow lo + 1$ while $A[lo] <= pivot$

5. do $hi \leftarrow hi - 1$ while $A[hi] >= pivot$

6. if $lo < hi$ swap($A[lo]$, $A[hi]$)

7. while $hi >= lo$;



Ένα παράδειγμα

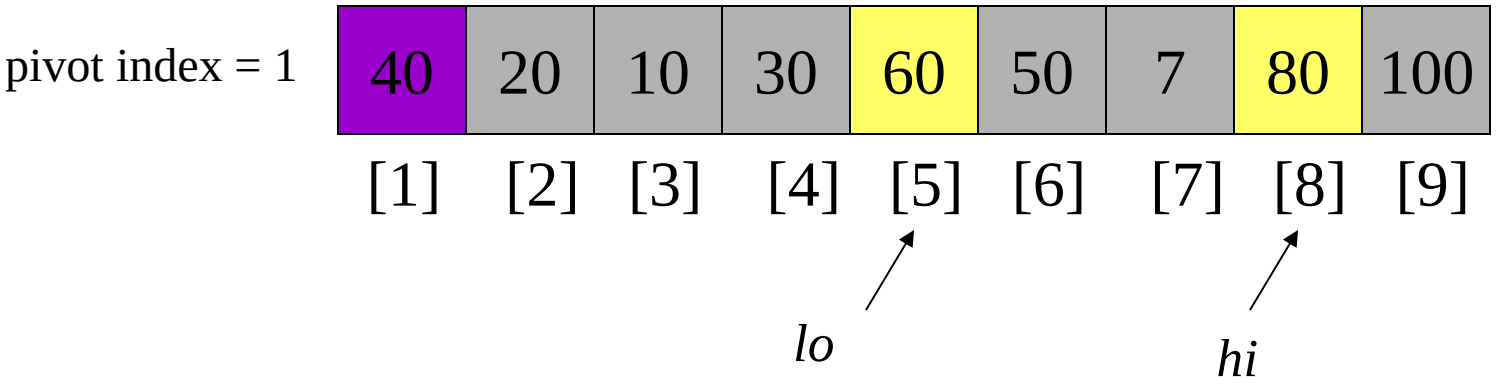
3. do

4. do $lo \leftarrow lo + 1$ while $A[lo] <= pivot$

5. do $hi \leftarrow hi - 1$ while $A[hi] >= pivot$

6. if $lo < hi$ swap($A[lo]$, $A[hi]$)

7. while $hi >= lo$;



Ένα παράδειγμα

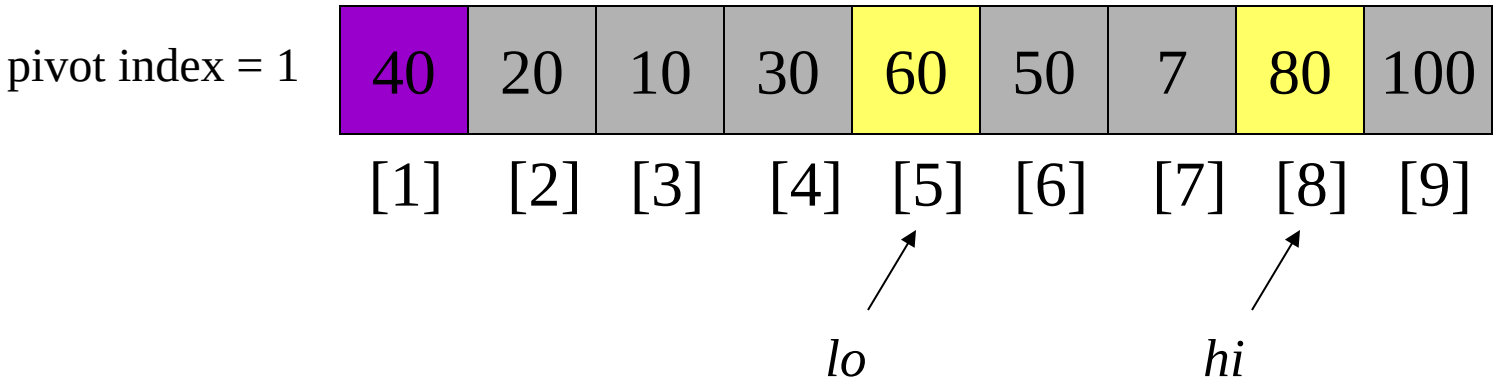
3. do

   4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

   5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

   6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

7. while *hi* >= *lo;*



Ένα παράδειγμα

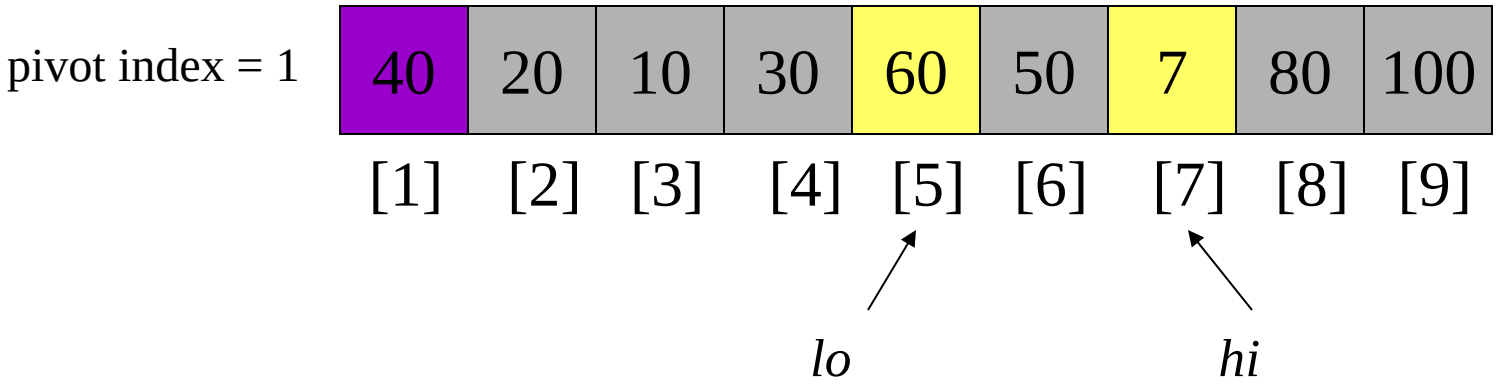
3. do

   4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

   5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

   6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

7. while *hi* >= *lo;*



Ένα παράδειγμα

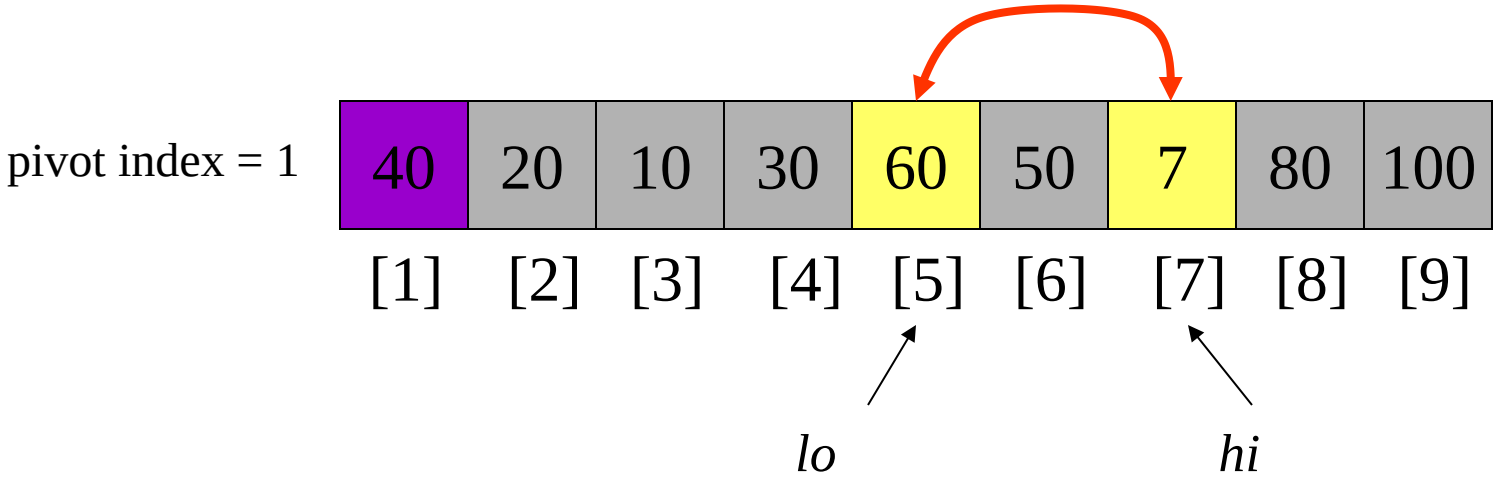
3. do

   4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

   5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

   6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

7. while *hi* >= *lo;*



Ένα παράδειγμα

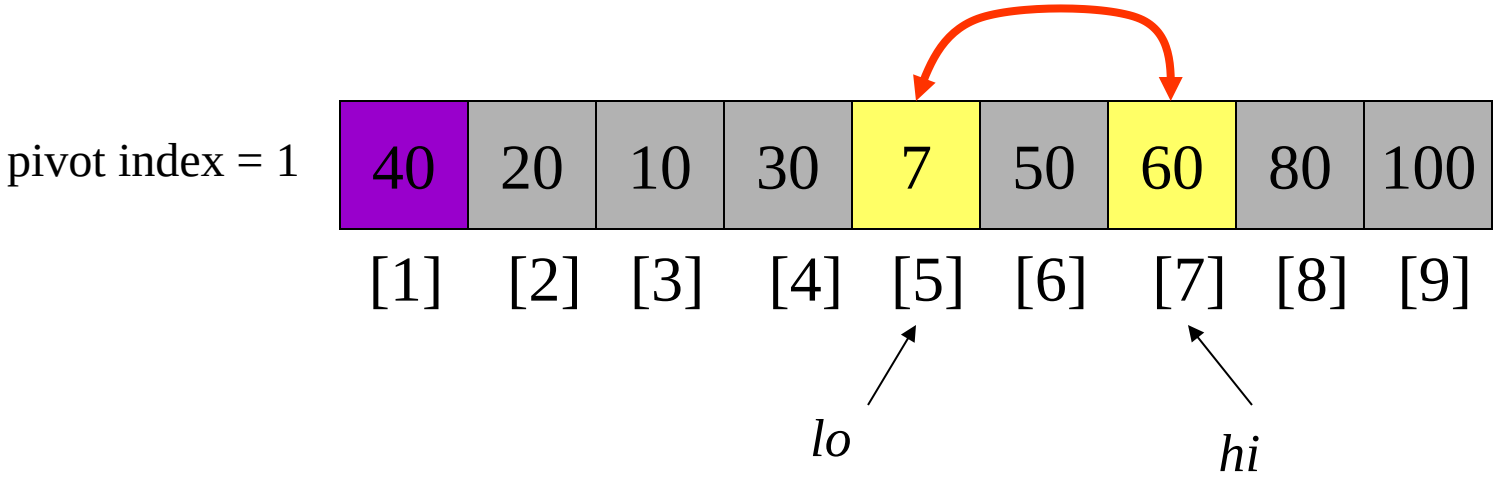
3. do

   4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

   5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

   6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

7. while *hi* >= *lo;*



Ένα παράδειγμα

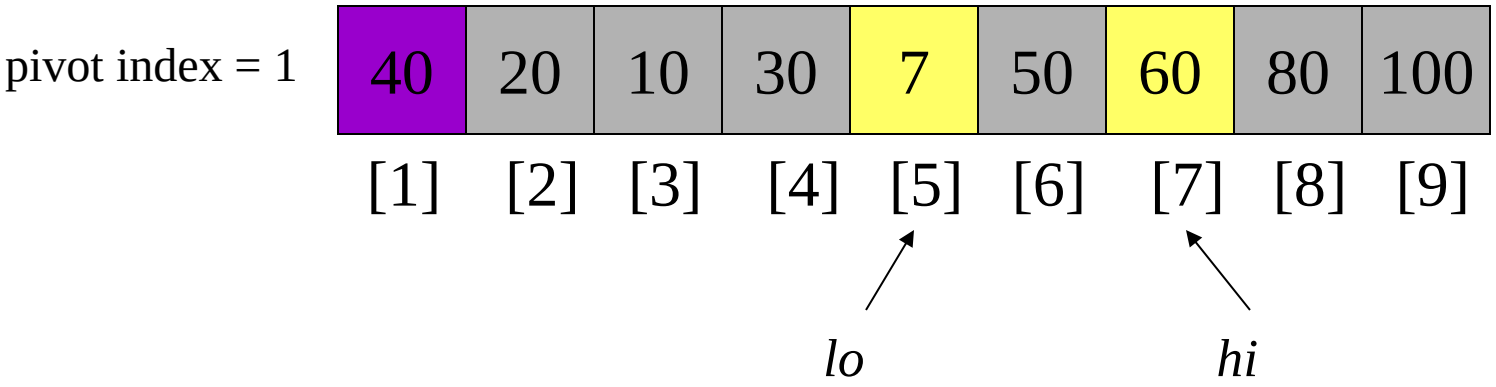
3. do

   4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

   5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

   6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

→ 7. while *hi* >= *lo;*



Ένα παράδειγμα

3. do

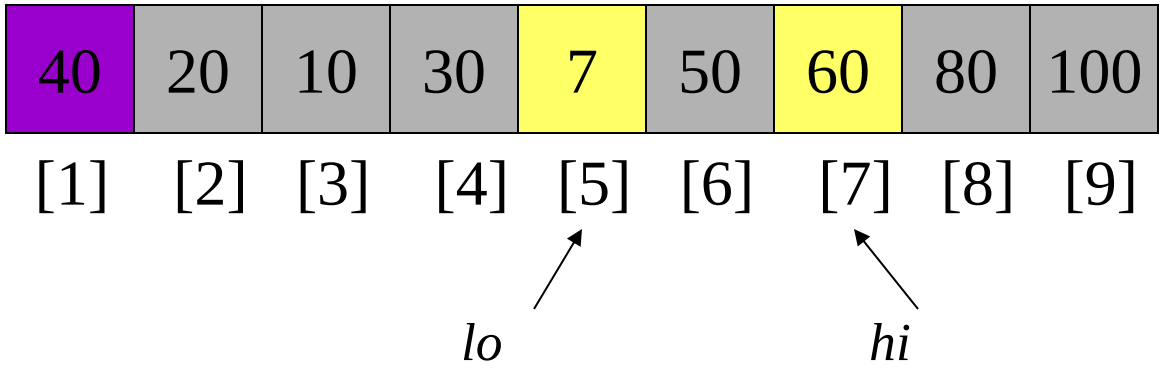
4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

7. while *hi* >= *lo;*

pivot index = 1



Ένα παράδειγμα

3. do

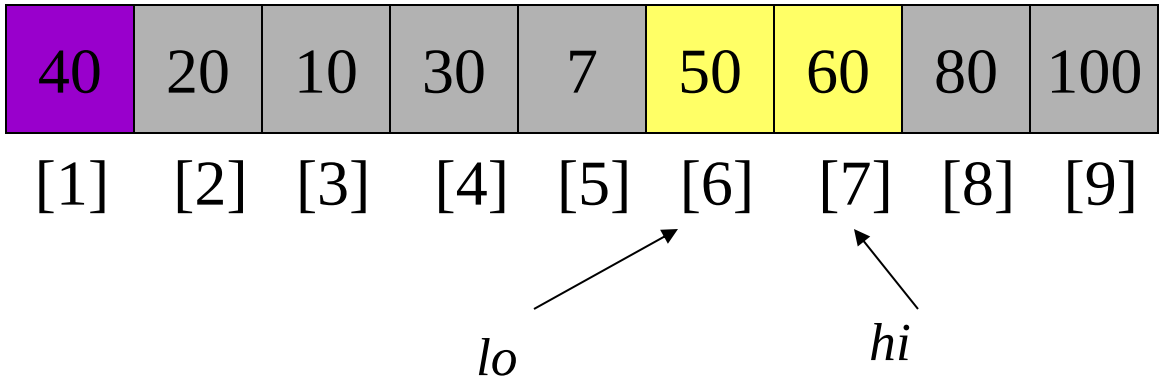
4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

7. while *hi* >= *lo;*

pivot index = 1



Ένα παράδειγμα

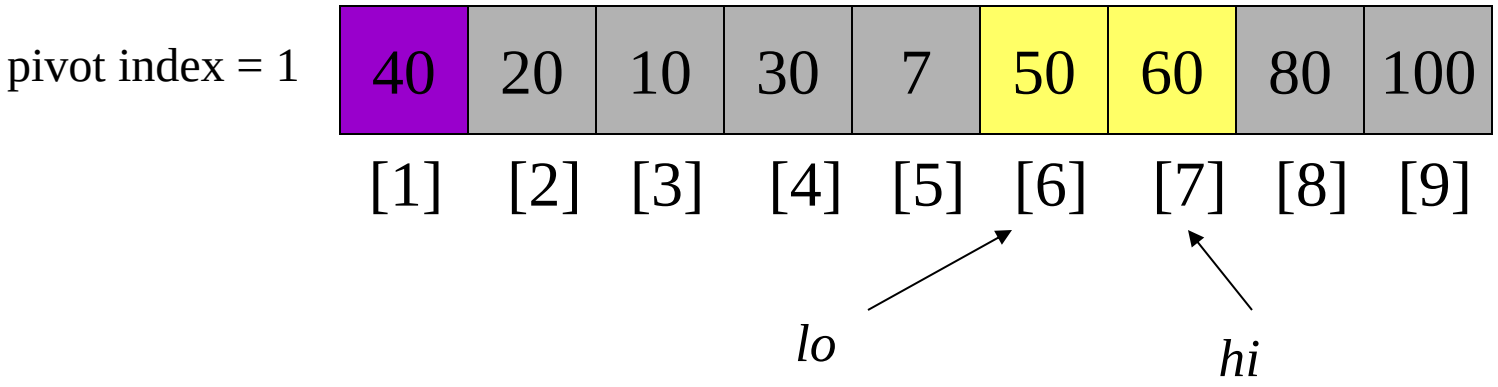
3. do

   4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

   5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

   6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

7. while *hi* >= *lo;*



Ένα παράδειγμα

3. do

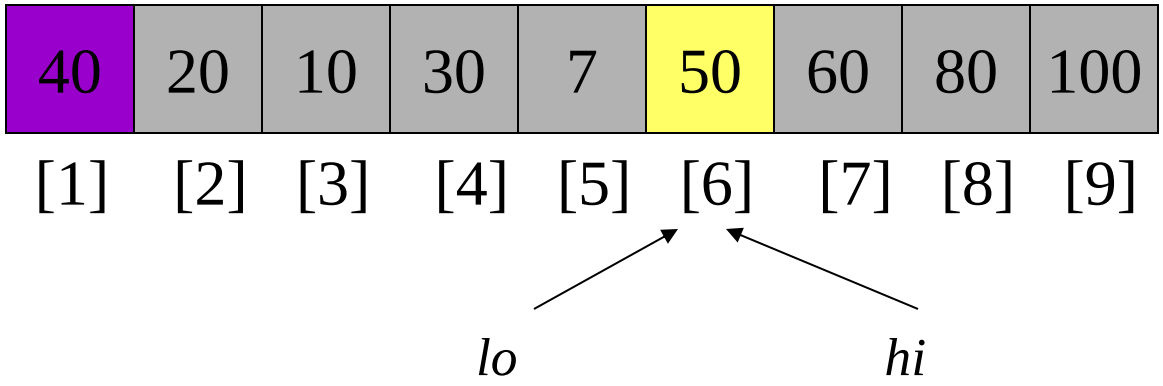
   4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

   5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

   6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

7. while *hi* >= *lo;*

pivot index = 1



Ένα παράδειγμα

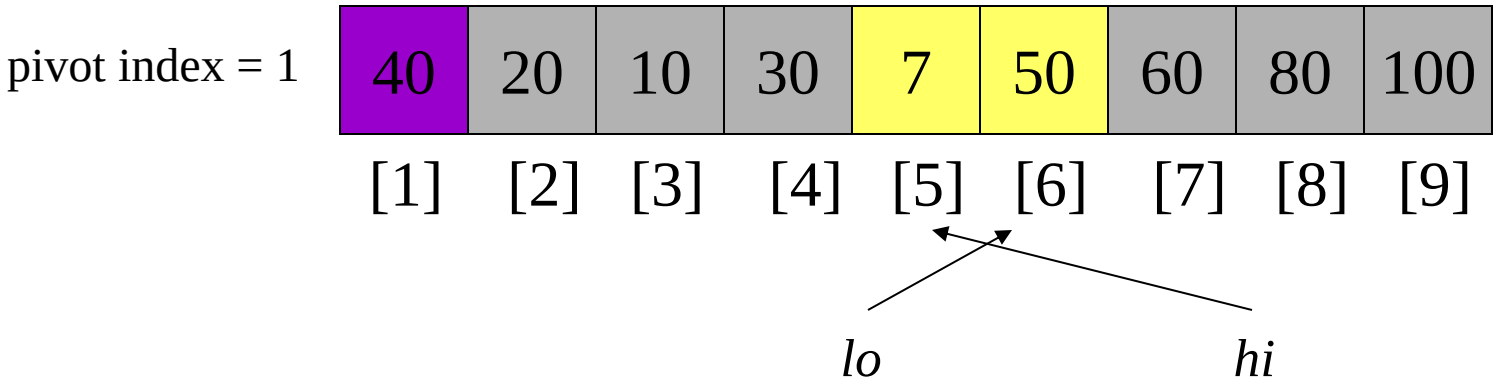
3. do

   4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

   5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

   6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

7. while *hi* >= *lo;*



Ένα παράδειγμα

3. do

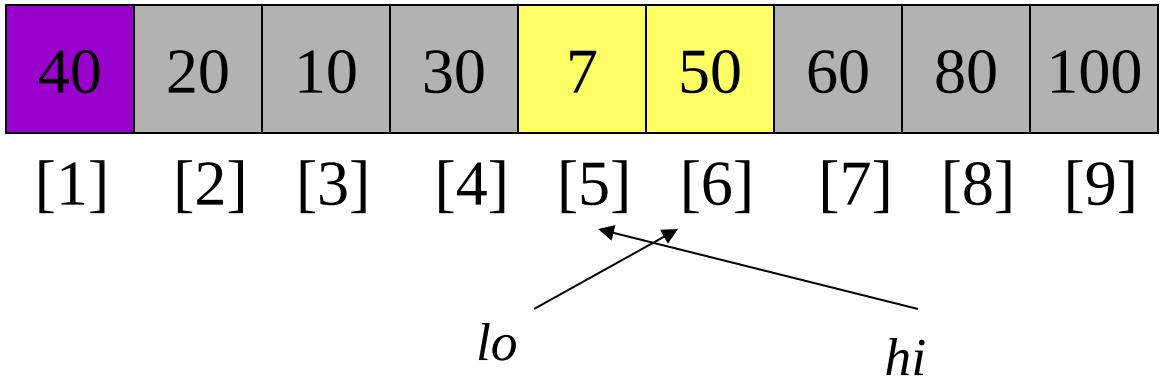
   4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

   5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

   6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

7. while *hi* >= *lo;*

pivot index = 1



Ένα παράδειγμα

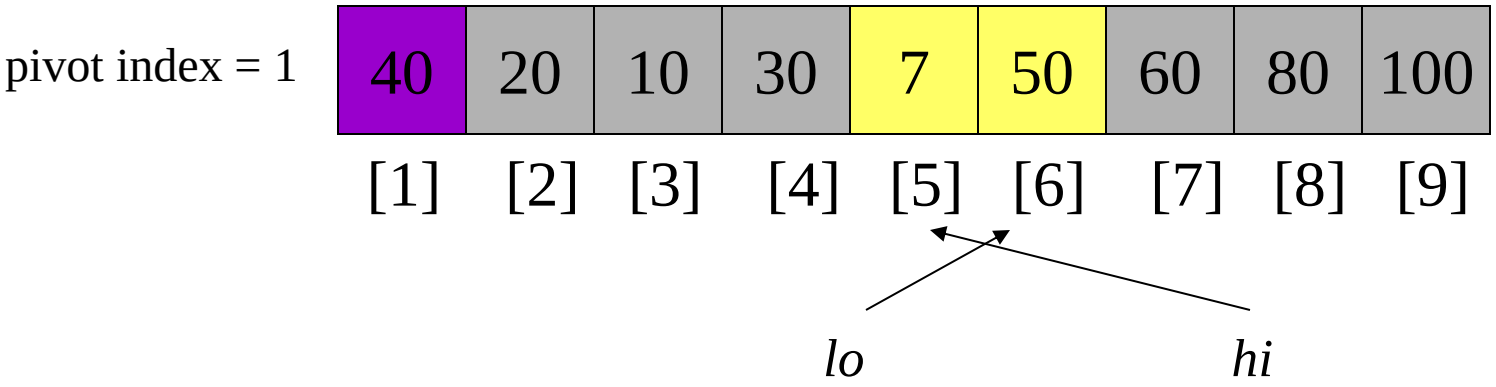
3. do

   4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

   5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

   6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

→ 7. while *hi* >= *lo;*



Ένα παράδειγμα

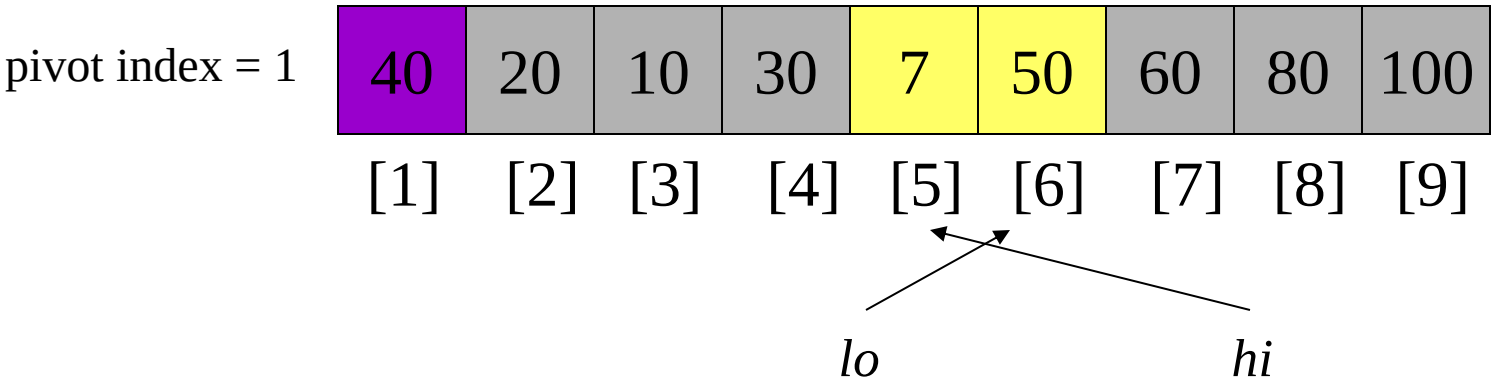
3. do

   4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

   5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

   6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

→ 7. while *hi* >= *lo;*



Ένα παράδειγμα

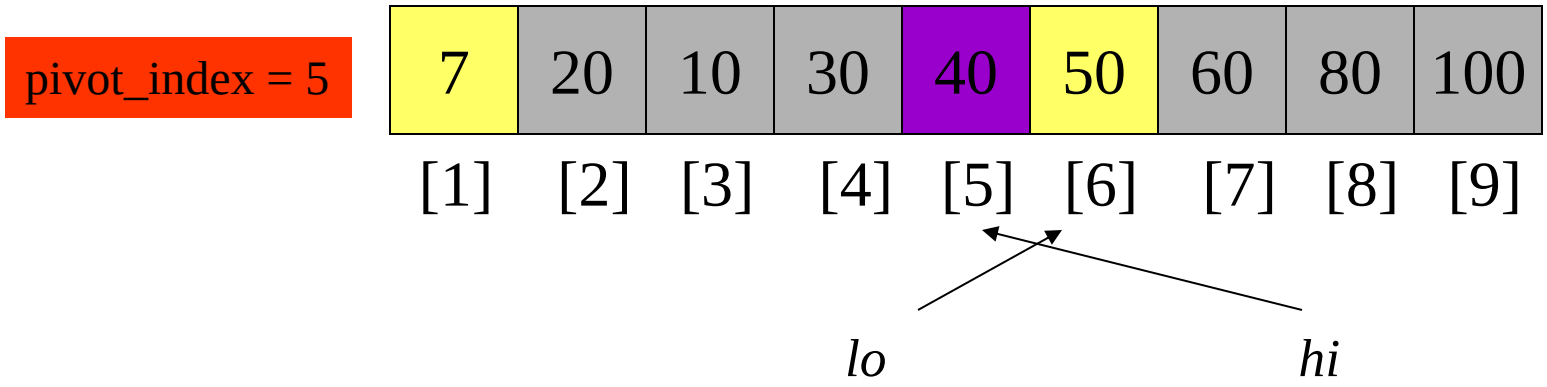
3. do

   4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

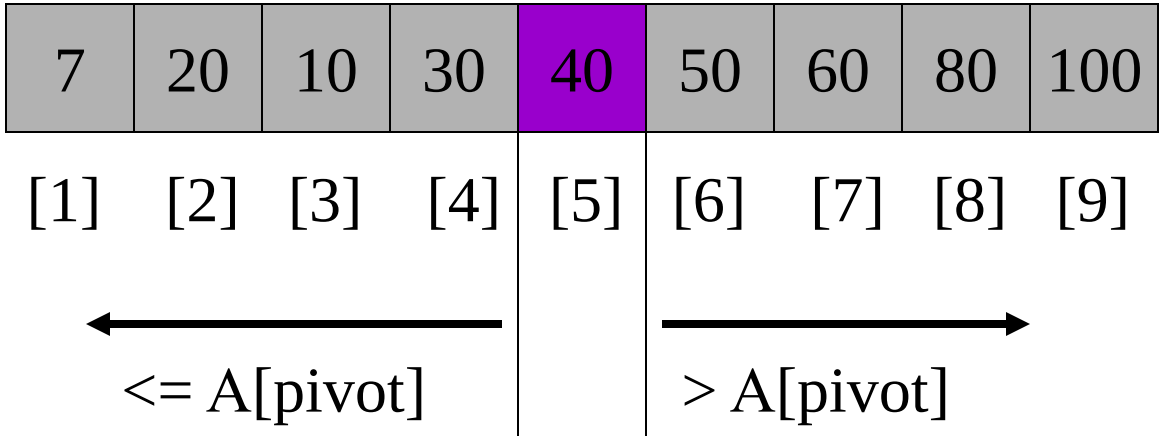
   5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

   6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

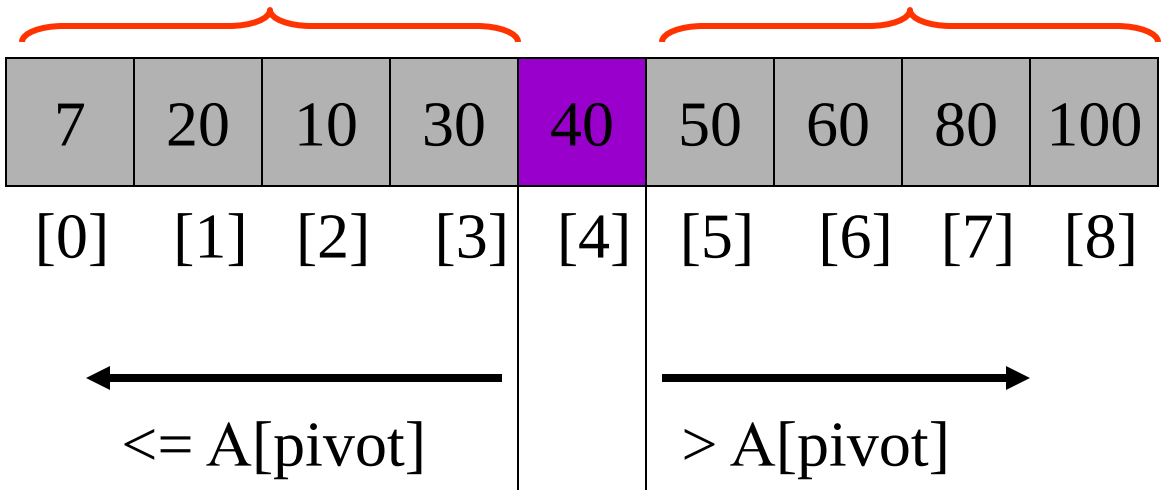
7. while *hi* >= *lo;*
8. swap(A[left], A[hi]);



Αποτέλεσμα διαμερισμού

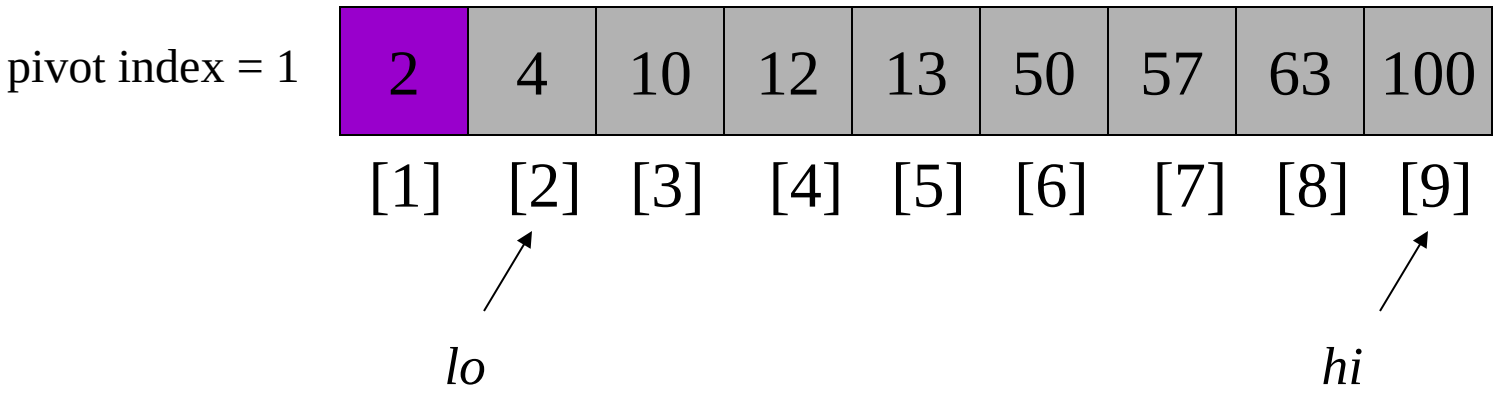


Αναδρομή: Quicksort Sub-arrays



Quicksort: Χειρότερη περίπτωση

- Υποθέτουμε ότι το πρώτο στοιχείο επιλέγεται ως pivot.
- Υποθέτουμε ότι ο πίνακας είναι ήδη ταξινομημένος:



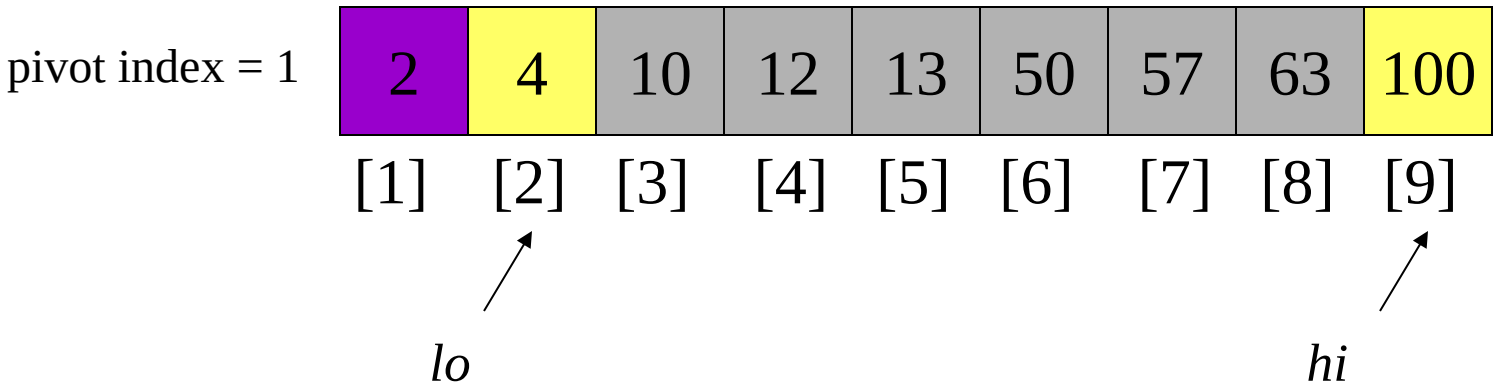
3. do

4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

7. while *hi* >= *lo;*



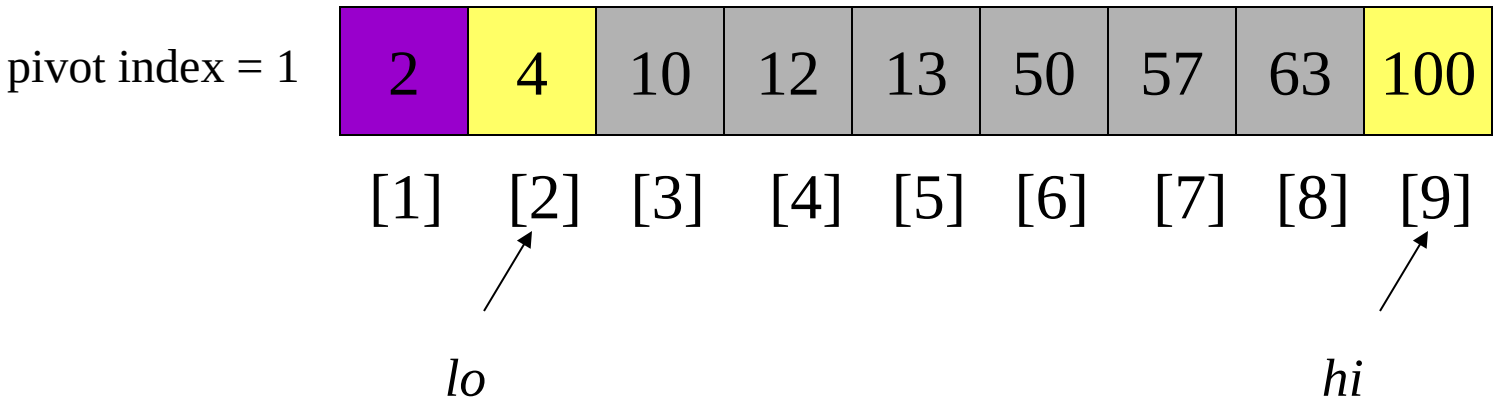
3. do

4. do $lo \leftarrow lo + 1$ while $A[lo] <= pivot$

5. do $hi \leftarrow hi - 1$ while $A[hi] >= pivot$

6. if $lo < hi$ swap($A[lo]$, $A[hi]$)

7. while $hi >= lo$;



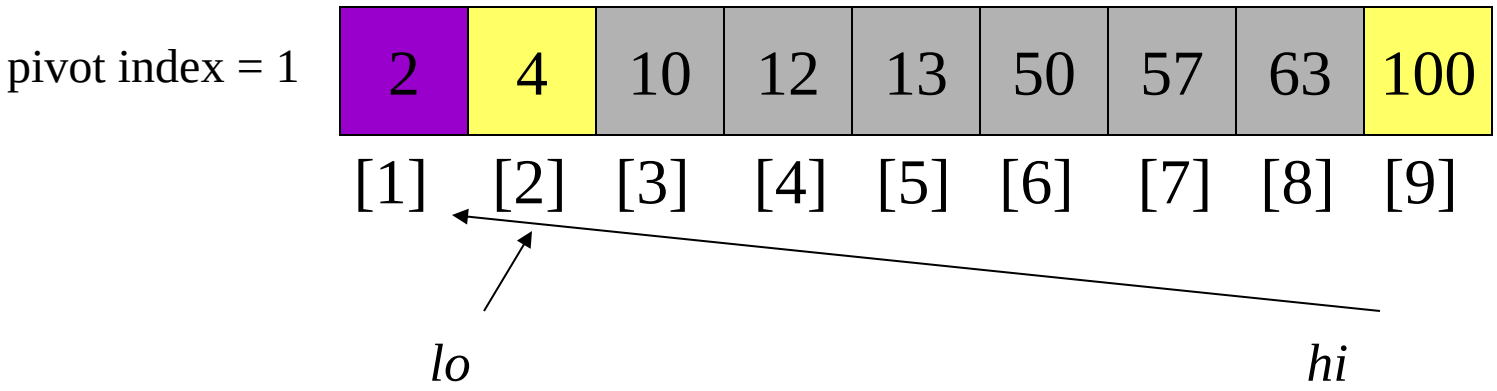
3. do

4. do $lo \leftarrow lo + 1$ while $A[lo] <= pivot$

5. do $hi \leftarrow hi - 1$ while $A[hi] >= pivot$

6. if $lo < hi$ swap($A[lo]$, $A[hi]$)

7. while $hi >= lo$;



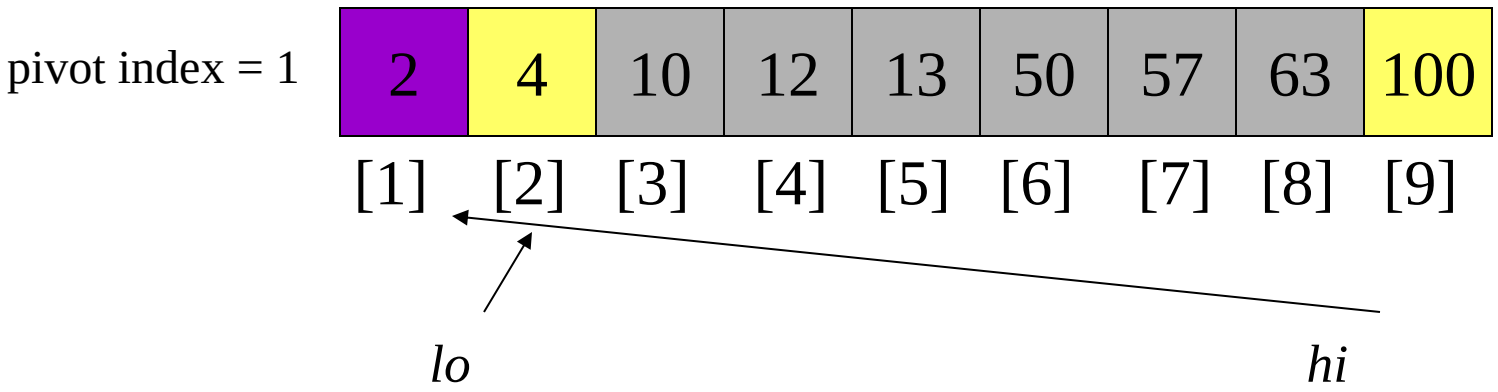
3. do

4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

7. while *hi* >= *lo;*



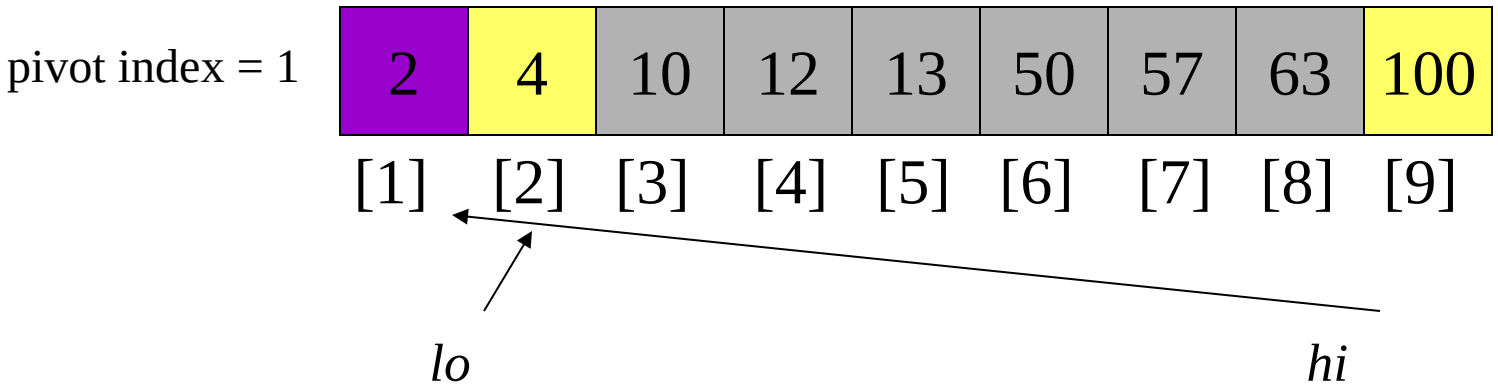
3. do

    4. do $lo \leftarrow lo + 1$ while $A[lo] <= pivot$

    5. do $hi \leftarrow hi - 1$ while $A[hi] >= pivot$

    6. if $lo < hi$ swap($A[lo]$, $A[hi]$)

→ 7. while $hi >= lo$;



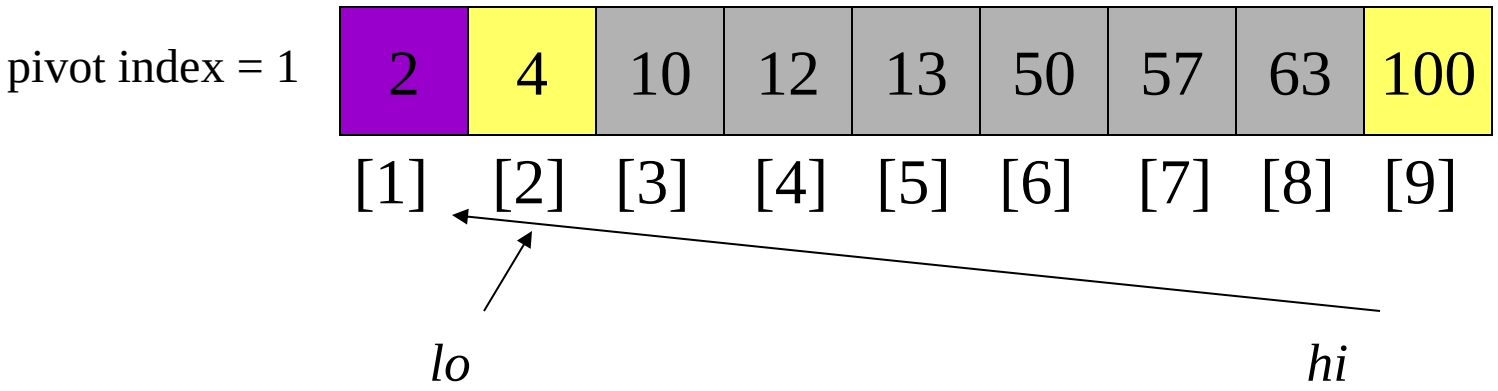
3. do

    4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

    5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

    6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

7. while *hi* >= *lo;*
8. swap(A[left], A[hi]);



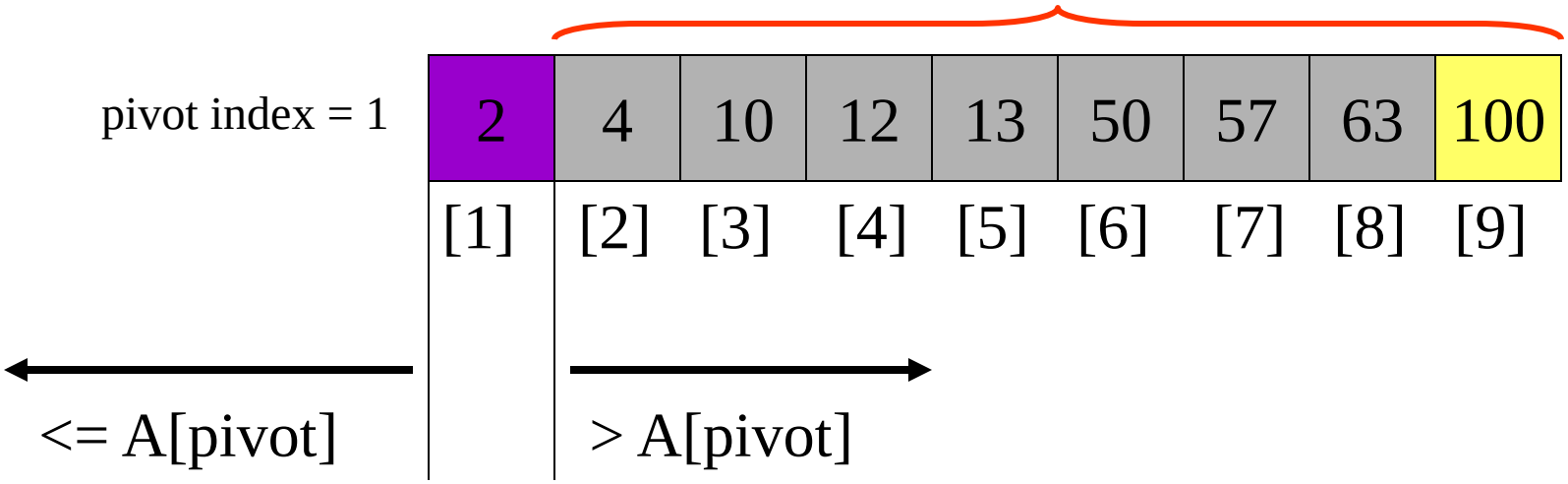
3. do

    4. do *lo* ← *lo* + 1 while *A*[*lo*] <= *pivot*

    5. do *hi* ← *hi* - 1 while *A*[*hi*] >= pivot

    6. if *lo* < *hi* swap(*A*[*lo*], *A*[*hi*])

7. while *hi* >= *lo;*
8. swap(A[left], A[hi]);



Πρόταση 1.

Η πολυπλοκότητα της γρήγορης ταξινόμησης είναι $\Theta(n^2)$ στη χειρότερη περίπτωση.

Απόδειξη

Σε πίνακα μεγέθους *n,* το επιλεγόμενο στοιχείο ως pivot στη χειρότερη περίπτωση είναι το μικρότερο ή το μεγαλύτερο (ταξινομημένος πίνακας σε αύξουσα ή φθίνουσα διάταξη), και τα μεγέθη των δύο υποπινάκων που θα προκύψουν να είναι 0 και *n* - 1 αντίστοιχα (αφού η μία θέση του πίνακα θα καταληφθεί από τον ίδιο τον pivot). Έτσι προκύπτει η ακόλουθη αναδρομική εξίσωση:

$$T(n) = T(n-1) + n \quad (1)$$

Ο όρος *n* του δεξιού σκέλους αντιστοιχεί στις συγκρίσεις που θα εκτελεσθούν κατά τη σάρωση του πίνακα (εντολές 4-5) πριν επιτευχθεί μια διαμέριση.

Για την αναδρομική εξίσωση διαδοχικά ισχύει:

$$T(n) = T(n-1) + n$$
$$T(n-1) = T(n-2) + (n-1)$$
$$T(n-2) = T(n-3) + (n-2)$$
$$\ldots\ldots$$
$$T(2) = T(1) + 2$$
$$T(1) = T(0) + 1$$

Με συνεχείς αντικαταστάσεις προς τα πίσω στην (1) και *T(0)* = 0 προκύπτει ότι:

$$T(n) = 1 + 2 + 3 + \ldots + n =$$

$$= \sum_{i=1}^{n} i = n(n+1)/2 = \Theta(n^2)$$

Πρόταση 2.

Η πολυπλοκότητα της γρήγορης ταξινόμησης είναι $\Theta(n\log_2 n)$ στην καλύτερη περίπτωση.

Απόδειξη

Η καλύτερη περίπτωση συμβαίνει όταν κάθε φορά ως pivot επιλέγεται ένα στοιχείο που λαμβάνοντας την τελική του θέση υποδιαιρεί τον πίνακα σε δύο υποπίνακες ίσου μεγέθους. Επομένως ισχύει η αναδρομική εξίσωση:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$$

με αρχικές συνθήκες $T(0) = 0$. Για την ευκολία της ανάλυσης υποθέτουμε ότι $n = 2^k$, Οπότε:

$$T(n) = 2T(n/2) + n \Rightarrow \frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1$$

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Ή η απόδειξη όπως το παράδειγμα 1, σελ. 31, 04_Recurrence Relations.pdf

$$\frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1 \quad (1)$$

Η αναδρομική εξίσωση (1) είναι πλέον εύκολη υπόθεση καθώς ισχύει διαδοχικά:

$$\begin{cases} \dfrac{T(n/2)}{n/2} = \dfrac{T(n/4)}{n/4} + 1 \\ \ldots\ldots \\ \dfrac{T(2)}{2} = \dfrac{T(1)}{1} + 1 \end{cases}$$

Αθροίζοντας τα αριστερά και δεξιά σκέλη αυτών των εξισώσεων προκύπτει ότι:

$$\frac{T(n)}{n} = \frac{T(1)}{1} + \underbrace{1+1+\ldots+1}_{\text{k φορές}}$$

$$\frac{T(n)}{n} = \frac{T(1)}{1} + k \qquad (n = 2^k \Rightarrow \log_2 n = k)$$

Εχουμε:

$$\frac{T(n)}{n} = \frac{T(1)}{1} + \log_2 n = 1 + \log_2 n$$

$$\frac{T(n)}{n} = \log_2 n + 1 \Rightarrow T(n) = n\log_2 n + n \in \Theta(n\log_2 n)$$ *(Απόδειξη με κανόνα ορίου)*

Πρόταση 3.

Η πολυπλοκότητα της γρήγορης ταξινόμησης είναι Θ(n $\log_2 n$) στη μέση περίπτωση.

Απόδειξη

Για να εξετάσουμε τη μέση περίπτωση θα υποθέσουμε ότι για *n* στοιχεία, κάθε διάταξη των στοιχείων αυτών (όπου *n!* είναι το πλήθος των διατάξεων) είναι ισοπίθανη. Επομένως, το κάθε φορά επιλεγόμενο pivot τελικά επιλέγεται με τυχαίο τρόπο και άρα όλα τα στοιχεία *n* έχουν την ίδια πιθανότητα 1/n να αποτελέσουν στοιχείο διαχωρισμού (pivot).

Ετσι καταλήγουμε σε τυχαίες διαμερίσεις του πίνακα, σε υποπίνακες μεγέθους *k* και *n*-*k*-1, όπου *k*=0, 1, …, *n*-1.

Θα χειρισθούμε τη μέση περίπτωση με τη βοήθεια της επόμενης αναδρομικής εξίσωσης για n ≥ 2:

$$T(n) = n + \frac{1}{n}\sum_{k=0}^{n-1}\left(T(k) + T(n-k-1)\right)$$

με αρχικές συνθήκες *T(0)* = 0.

$$T(n) = n + \frac{1}{n}\sum_{k=0}^{n-1}\left(T(k) + T(n-k-1)\right)$$

με αρχικές συνθήκες

$T(0) = 0$.

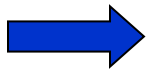
Λόγω ισομετρίας και των αρχικών συνθηκών, η προηγούμενη αναδρομική εξίσωση μετασχηματίζεται σε:

$$\sum_{k=0}^{n-1}\left(T(k)+T(n-k-1)\right) = \left[T(0)+T(1)+\ldots+T(n-1)\right]+\left[T(n-0-1)+T(n-1-1)+\ldots+T(n-(n-1)-1)\right] =$$

$$= \left[0+1+\ldots+T(n-1)\right]+\left[T(n-0-1)+T(n-1-1)+\ldots+T(1)+T(0)\right] =$$

$$= \left[T(1)+\ldots+T(n-1)\right]+\left[T(n-1)+T(n-2)+\ldots+T(1)\right] =$$

$$= \sum_{k=1}^{n-1}T(k)+\sum_{k=1}^{n-1}T(k) = 2\sum_{k=1}^{n-1}T(k) \qquad \text{άρα} \qquad T(n) = n + \frac{2}{n}\sum_{k=1}^{n-1}\left(T(k)\right)$$



Τη σχέση $T(n)=n+\frac{2}{n}\sum_{k=1}^{n-1}(T(k))$ πολλαπλασιάζουμε επί *n* και προκύπτει

$$nT(n)=n^2+2\sum_{k=1}^{n-1}(T(k))$$

Επίσης στην ίδια σχέση αντικαθιστούμε το *n* με *n*-1 και πολλαπλασιάζουμε με *n*-1, οπότε προκύπτει:

$$(n-1)T(n-1)=(n-1)^2+2\sum_{k=1}^{n-2}(T(k))$$

Αφαιρώντας τα αντίστοιχα σκέλη των δυο τελευταίων εκφράσεων:

$$nT(n)-(n-1)T(n-1)=n^2+2\sum_{k=1}^{n-1}(T(k))-n^2+2n-1-2\sum_{k=1}^{n-2}(T(k))$$

$$=2\left[\sum_{k=1}^{n-2}(T(k))+T(n-1)\right]+2n-1-2\sum_{k=1}^{n-2}(T(k))$$

$$=2\sum_{k=1}^{n-2}(T(k))+2T(n-1)+2n-1-2\sum_{k=1}^{n-2}(T(k))=2T(n-1)+2n-1$$

προκύπτει $nT(n)-(n-1)T(n-1)=2T(n-1)+2n-1$

$$nT(n)-(n-1)T(n-1)=2T(n-1)+2n-1 \quad \Rightarrow$$

$$nT(n)=(2+n-1)T(n-1)+2n-1 \quad \Rightarrow \quad nT(n)=(n+1)T(n-1)+2n-1$$

$$\frac{T(n)}{n+1}=\frac{T(n-1)}{n}+\frac{2n-1}{n(n+1)}$$

Με διαδοχικές αντικαταστάσεις του *n* με *n*-1 (με *n* ≥2) προκύπτει:

$$\frac{T(n)}{n+1}=\frac{T(n-1)}{n}+\frac{2n-1}{n(n+1)}$$

$$\frac{T(n-1)}{n}=\frac{T(n-2)}{n-1}+\frac{2n-3}{(n-1)n}$$

$$\frac{T(n-2)}{n-1}=\frac{T(n-3)}{n-2}+\frac{2n-5}{(n-2)(n-1)}$$

$$\frac{T(2)}{3}=\frac{T(1)}{2}+\frac{3}{2\mathrm{X}3}$$

Αθροίζοντας αντίστοιχα τα αριστερά και δεξιά σκέλη και απλοποιώντας προκύπτει :

$$\frac{T(n)}{n+1}=\frac{2n-1}{n(n+1)}+\frac{2n-3}{(n-1)n}+\ldots+\frac{3}{2\mathrm{X}3}=\sum_{i=2}^{n}\frac{2i-1}{i(i+1)}=\sum_{i=2}^{n}\frac{3i-i-1}{i(i+1)}=\sum_{i=2}^{n}\left[\frac{3i}{i(i+1)}-\frac{i+1}{i(i+1)}\right]$$

Επομένως πρέπει να υπολογίσουμε το άθροισμα του δεξιού σκέλους:

$$\sum_{i=2}^{n}\left[\frac{3i}{i(i+1)}-\frac{i+1}{i(i+1)}\right]=\sum_{i=2}^{n}\left[\frac{3}{i+1}-\frac{1}{i}\right]$$

$$\sum_{i=2}^{n}\left[\frac{3i}{i(i+1)}-\frac{i+1}{i(i+1)}\right]=\sum_{i=2}^{n}\left[\frac{3}{i+1}-\frac{1}{i}\right]$$

$$H_n=\sum_{i=1}^{n}\frac{1}{i}=1+\sum_{i=2}^{n}\frac{1}{i}\Rightarrow\quad\sum_{i=2}^{n}\frac{1}{i}=H_n-1$$

$$H_{n+1}=\sum_{i=1}^{n+1}\frac{1}{i}=1+\frac{1}{2}+\frac{1}{3}\ldots+\frac{1}{n+1}=1+\frac{1}{2}+\sum_{i=2}^{n}\frac{1}{i+1}\Rightarrow\quad\sum_{i=2}^{n}\frac{1}{i+1}=H_{n+1}-1-\frac{1}{2}$$

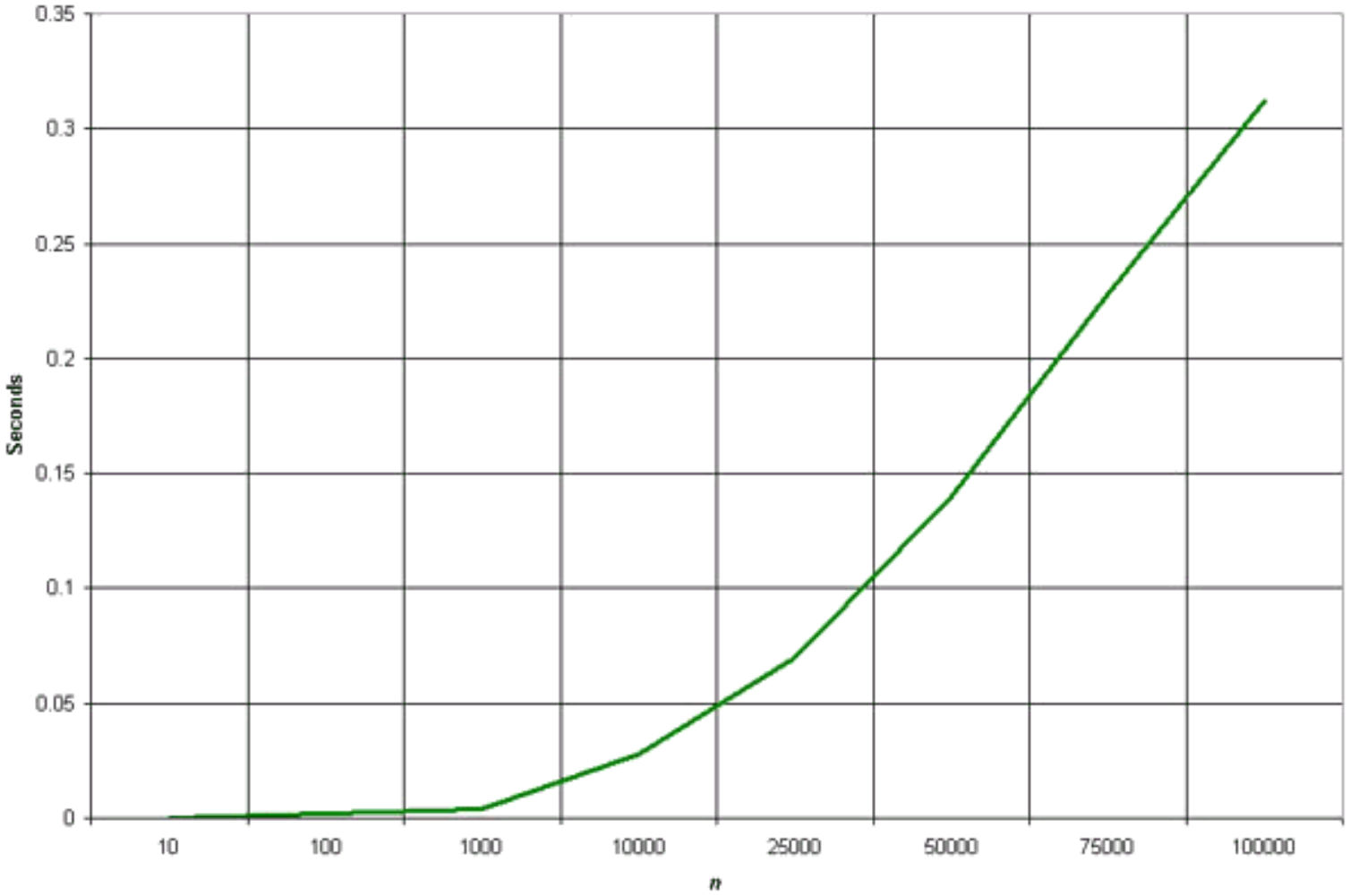
$$\sum_{i=2}^{n}\left[\frac{3}{i+1}-\frac{1}{i}\right]=3\left[H_{n+1}-1-\frac{1}{2}\right]-\left[H_n-1\right]=3\left[H_n+\frac{1}{n+1}-1-\frac{1}{2}\right]-\left[H_n-1\right]$$

$$=3H_n+\frac{3}{n+1}-3-\frac{3}{2}-H_n+1=2H_n-\frac{7}{2}+\frac{3}{n+1}$$

$$\frac{T(n)}{n+1}=2H_n-\frac{7}{2}+\frac{3}{n+1}\Rightarrow T(n)=2(n+1)H_n-\frac{7}{2}(n+1)+3$$

$$T(n)=2(n+1)H_n-\frac{7}{2}(n+1)+3=(n+1)\left(2H_n-\frac{7}{2}\right)+3$$

$$H_n\in\Theta(\ln n)=\Theta(\log_2 n)\qquad T(n)\in(n+1)\Theta(\log_2 n)=\Theta(n\log_2 n)$$


0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
Seconds
10
100
1000
10000
25000
50000
75000
100000
n

Quicksort: χώρος αποθήκευσης

Η γρήγορη ταξινόμηση δεν είναι επιτόπιος αλγόριθμος, δηλαδή δεν αρκείται στο χώρο του συγκεκριμένου πίνακα αλλά χρειάζεται επιπλέον χώρο.

Ο χώρος αυτός δεν φαίνεται καθαρά στον ανωτέρω κώδικα αλλά απαιτείται από το μεταγλωττιστή κατά την αναδρομή.

Εύλογα προκύπτει η ερώτηση σχετικά με το μέγεθος αυτού του απαιτούμενου χώρου.

Στη χειρότερη περίπτωση το μέγιστο βάθος της αναδρομής θα είναι n-1 κλήσεις, αριθμός υπερβολικός.

Για το λόγο αυτό έχει σχεδιάσει μια επαναληπτική εκδοχή της γρήγορης ταξινόμησης, δηλαδή μια εκδοχή που υλοποιεί την απαραίτητη στοίβα, όπου τοποθετούνται τα όρια του μικρότερου υποπίνακα, όπως φαίνεται στην επόμενη διαδικασία iterative_quicksort.

Αλγόριθμος iterative_quicksort(left,right);

```
1.      do
2.          while (left < right) do
3.                  lo ← left ; hi ← right+1; pivot ← A[left]; p=left; q=right;
4.          do
5.                  do lo ← i+1 while A[lo]<=pivot;
6.                  do hi ← hi-1 while A[hi]>=pivot;
7.                  if lo<hi then swap(A[lo], A[hi]);
8.          while hi>=lo;
9.          swap(A[left], A[hi]);
10.         if ((hi - p) < (q - hi)) then
11.                 push(hi+1); push(q); q ← hi-1;
12.         else
13.                 push(p); push(hi-1); p ← hi+1;
14.                 if stack is empry then return;
15.                 pop(q); pop(p);
16.     while (true)
```

Ταξινόμηση με Συγχώνευση

- Με τον όρο συγχώνευση (merging) εννοούμε την πράξη της δημιουργίας ενός νέου ταξινομημένου πίνακα που περιέχει όλα τα στοιχεία δύο (ή και περισσοτέρων) πινάκων, που είναι ήδη ταξινομημένοι.
- Η διαδικασία Merge που ακολουθεί υλοποιεί τη συγχώνευση δύο ταξινομημένων πινάκων A και B (με *n* και m ακεραίους αντίστοιχα) στον πίνακα C.
- Θεωρείται ότι οι δύο πίνακες έχουν και μία επιπλέον θέση που χρησιμεύει για την αποθήκευση ενός φρουρού (sentinel).

Αλγόριθμος merge;

1. $i \leftarrow 1$; $j \leftarrow 1$;
2. $A[n+1] \leftarrow maxint$; $B[m+1] \leftarrow maxint$;
3. **for** $k \leftarrow 1$ **to** $n+m$ do
4. **if** $A[i] < B[j]$ then
5. $C[k] \leftarrow A[i]$; $i \leftarrow i+1$
6. **else**
7. $C[k] \leftarrow B[j]$; j $\leftarrow j+1$

Από το βρόχο της εντολής 3, είναι προφανές ότι η πολυπλοκότητα της συγχώνευσης είναι $\Theta(n+m)$, θεωρώντας φραγμένο από επάνω το κόστος της ερώτησης της εντολής 4-6.

Merge sort

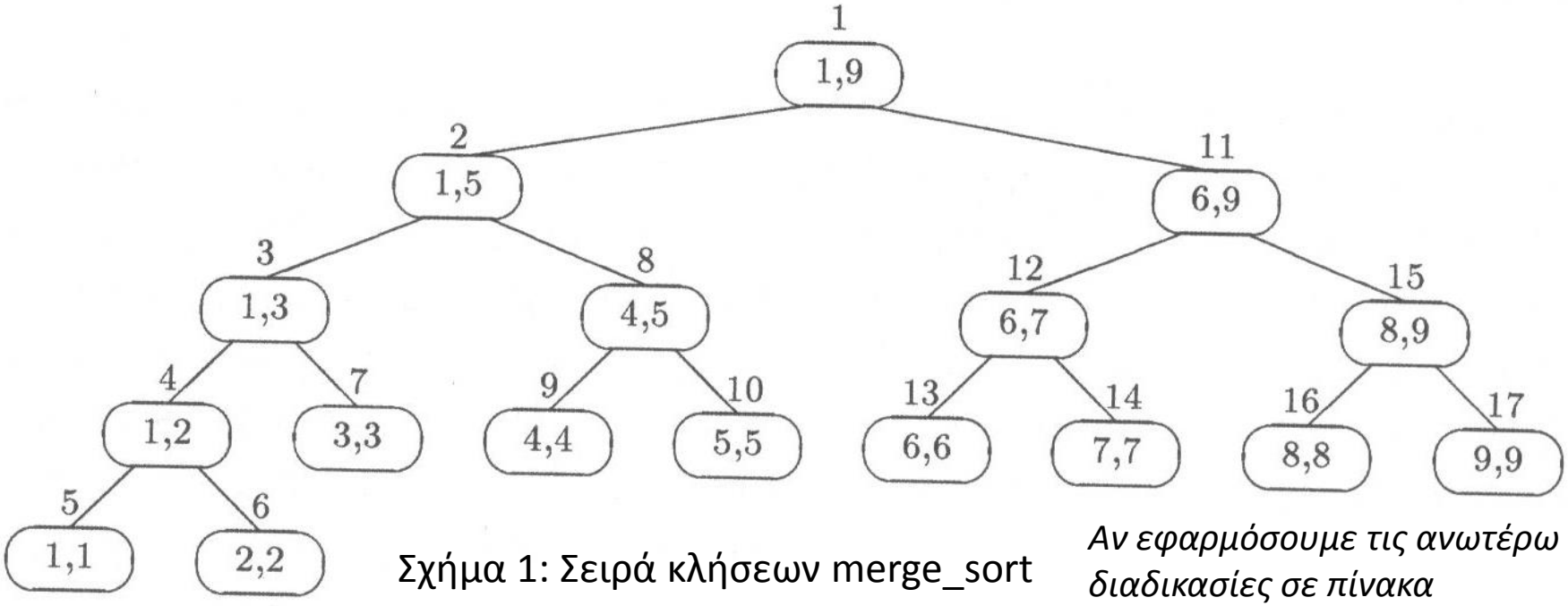
- Η ταξινόμηση με **ευθεία συγχώνευση** (*straight merge sort*) κατά καιρούς έχει υλοποιηθεί με πλήθος τρόπων, όπως επαναληπτικά ή αναδρομικά, για λίστες ή πίνακες κοκ.
- Στη συνέχεια ακολουθεί μία εκδοχή της για πίνακες, η οποία αποτελείται από δύο τρήματα: την αναδρομική merge_sort και τη merge.
- Στις εντολές 16-18 της merge_sort βρίσκεται το μεσαίο στοιχείο του πίνακα και εκτελούνται 2 κλήσεις στους δύο υποπίνακες που προκύπτουν. Από εδώ προκύπτει ότι η μέθοδος ανήκει στην οικογένεια των αλγορίθμων ***Διαίρει και Βασίλευε***.
- Η διαδικασία merge που καλείται όταν προκύψουν στοιχειώδεις υποπίνακες αναλαμβάνει τις συγκρίσεις των στοιχείων και τα τακτοποιεί με τη βοήθεια ενός βοηθητικού πίνακα B

Αλγόριθμος merge(*A*, *left*, *middle*, *right*);

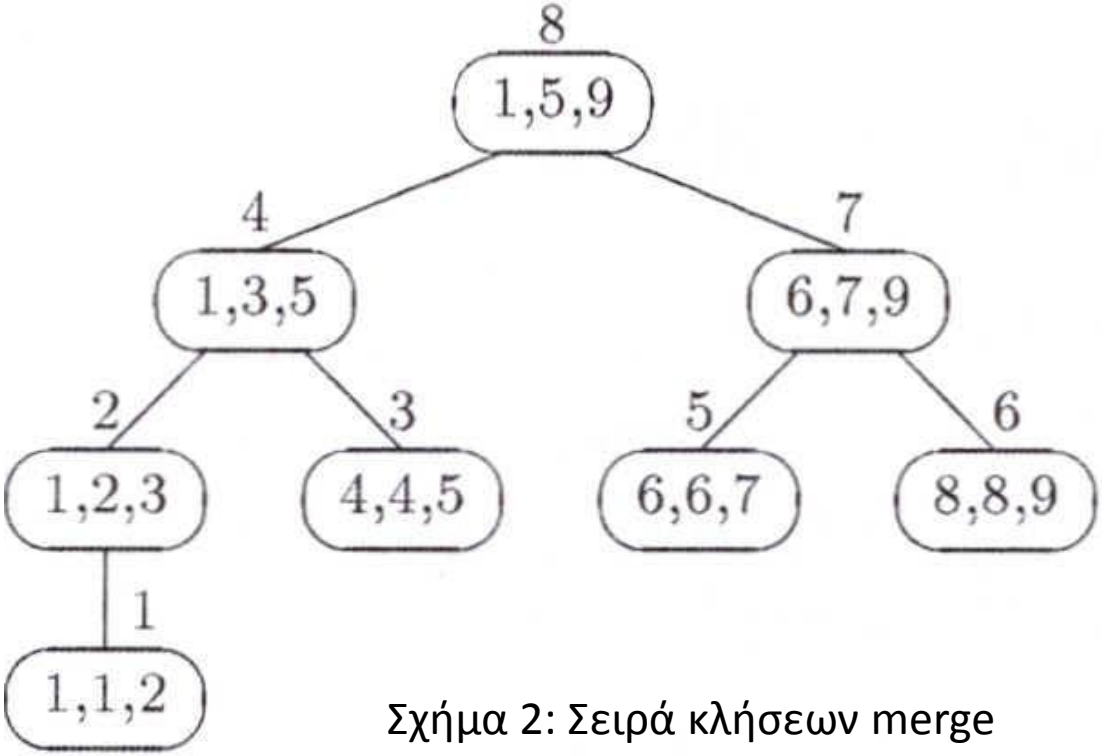
```
1.   first ← left; second ← middle +1; temp ← left;
2.    while (first <= middle) and (second <= right) do
3.       if A[first] <= A[second] then
4.                B[temp] ← A[first]; first ← first +1;
5.       else
6.                B[temp] ← A[second]; second ← second + 1;
7.       temp ← temp+1
8.   if first<= middle then
9.       for k ← first to middle do
10.               B[temp] ← A[k]; temp ← temp + 1
11.  else
12.      for k ← second to right do
13.               B[temp] ← A[k]; temp ← temp + 1
14.  for k ← left to right do A[k] ← B[k]
```

Αλγόριθμος merge_sort(*A*, *left*, *right*);

15. **if** *left* < *right* **then**
16. *middle* ← $\lfloor (left + right)/2 \rfloor$; *//divide step*
17. merge_sort(*A*, *left*, *middle*); *//conquer step*
18. merge_sort(*A*, *middle*+1,*right*); *//conquer step*
19. merge(*A*, *left*, *middle*, *right*) *//combine step*



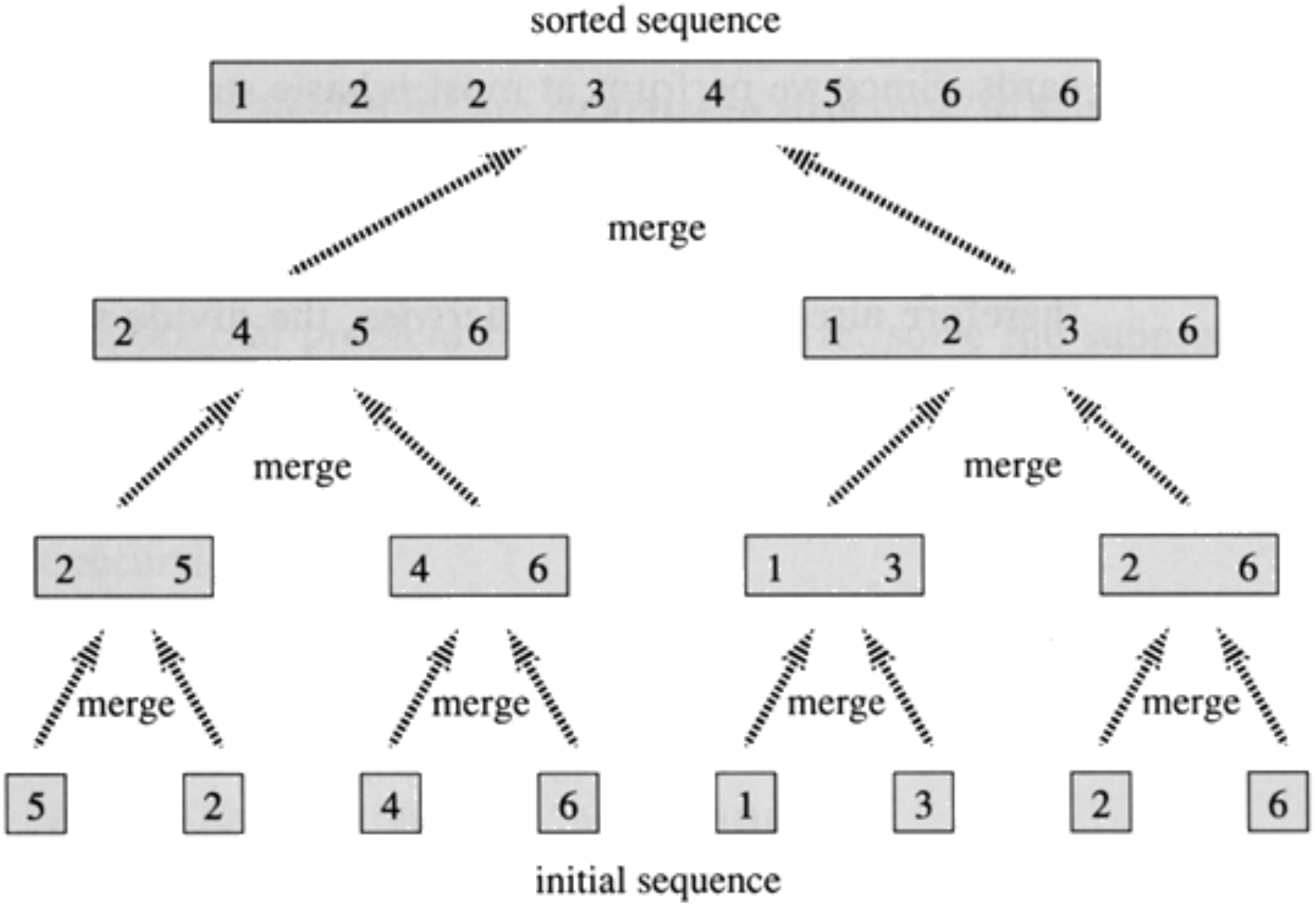
Σχήμα 1: Σειρά κλήσεων merge_sort



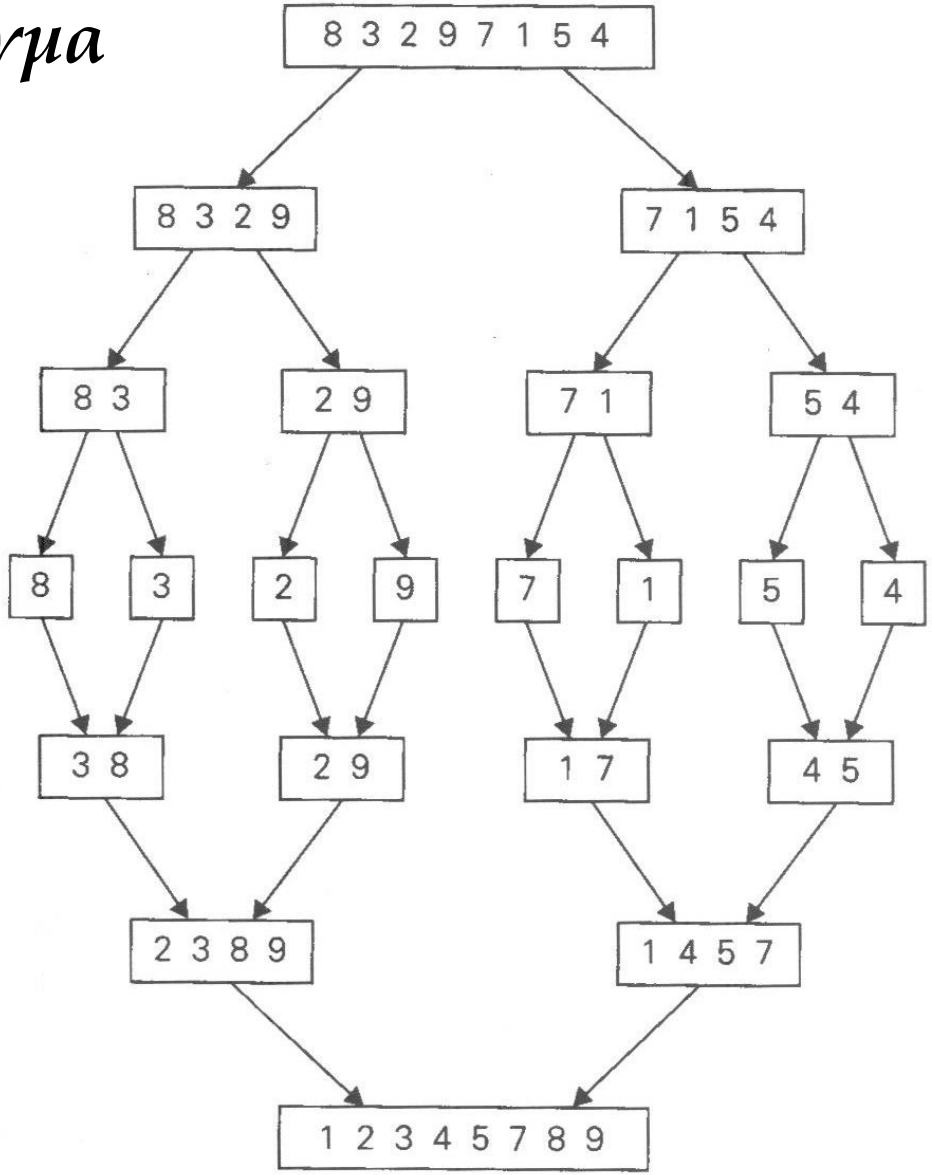
Σχήμα 2: Σειρά κλήσεων merge

Αν εφαρμόσουμε τις ανωτέρω διαδικασίες σε πίνακα A=[52, 12, 71, 56, 5, 10, 19, 90, 45} με τα 9 κλειδιά, τότε η mergesort θα κληθεί 17 φορές με τη σειρά που απεικονίζεται επάνω από κάθε κόμβο του σχήματος 1, ενώ μέσα σε κάθε κόμβο φαίνονται τα ορίσματα της αντίστοιχος κλήσης. Στο Σχήμα 2 φαίνονται 8 κλήσεις της merge, με την αντίστοιχη τάξη εκτός του κόμβου και τα ορίσματα εντός του κόμβου. Και στις δύο περιπτώσεις η διάσχιση του δένδρου ακλουθεί τη λογική της αναζήτησης με προτεραιότητα βάθους (dfs).

Ένα παράδειγμα



Ένα παράδειγμα



Πρόταση 5.

Η πολυπλοκότητα της merge_sort είναι Θ(n $\log_2 n$) στη χειρότερη περίπτωση.

Απόδειξη

Από τον αλγόριθμο merge_sort εξάγουμε την επόμενη αναδρομική εξίσωση: $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$

με αρχική συνθήκη *T(1)* = 1, ενώ οι τρεις όροι του δεξιού σκέλους αντιστοιχούν στις εντολές 17-19.

Συγκεκριμένα το κόστος σε συγκρίσεις της εντολής 19 προκύπτει από όσα αναφέρθησαν προηγουμένως για τη διαδικασία συγχώνευσης 2 πινάκων.

Για την ευκολία της ανάλυσης υποθέτουμε ότι $\boldsymbol{n = 2^k}$, έτσι ώστε το κάθε βήμα διαμερισμού οδηγεί σε 2 υποπροβλήματα ακριβώς μεγέθους n/2.

Από την αναδρομική εξίσωση ισχύει διαδοχικά:

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$= 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n = 2^2 T\left(\frac{n}{4}\right) + 2\frac{n}{2} + n = 2^2 T\left(\frac{n}{2^2}\right) + 2n$$

$$= 2^2\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n = 8T\left(\frac{n}{8}\right) + 4\frac{n}{4} + 2n = 2^3 T\left(\frac{n}{2^3}\right) + 3n$$

$n = 2^k \Rightarrow \log_2 n = k$

...

$$= 2^k T\left(\frac{n}{2^k}\right) + kn = nT\left(\frac{n}{n}\right) + kn = nT(1) + kn$$

$$= n + n\log_2 n \in \Theta(n\log_2 n)$$

(Απόδειξη με κανόνα ορίου)

Η μέθοδος ταξινόμησης με ευθεία συγχώνευση αξίζει την προσοχή μας γιατί είναι πολύ **ευσταθής** (*stable*) μέθοδος, δηλαδή ανεξάρτητα από τη φύση των δεδομένων θα εκτελέσει τον ίδιο αριθμό συγκρίσεων στη μέση και τη χειρότερη περίπτωση.

Το χαρακτηριστικό αυτό έρχεται σε αντίθεση με το μειονέκτημα της γρήγορης ταξινόμησης, που διακρίνεται από πολυπλοκότητα $\Theta(n^2)$ για τη χειρότερη περίπτωση.

Ωστόσο, το συμπέρασμα αυτό προκύπτει γιατί στην ανάλυση λάβαμε υπόψη μας μόνο το πλήθος των εκτελούμενων συγκρίσεων. Σε μία πραγματική υλοποίηση, η χρήση του βοηθητικού πίνακα Β και οι εκτελούμενες καταχωρήσεις είναι επίσης πράξεις μη αμελητέου χρονικού κόστους, οι οποίες επιβαρύνουν τη συνολική επίδοση.

Τελικώς, αν και η ταξινόμηση με συγχώνευση δεν είναι ίσως η καλύτερη μέθοδος ταξινόμησης στην κύρια μνήμη, εν τούτοις αποτελεί τη βάση για μεθόδους εξωτερικής ταξινόμησης.

Ο αλγόριθμος merge μπορεί να μπορεί να εκφραστεί εναλλακτικά με τη χρήση 2 βοηθητικών πινάκων *L* και *R* όπου αντιγράφονται τα στοιχεία του πίνακα *A*, συγκεκριμένα τα

A[*left* . . *middle*] αντιγράφονται στον *L*[1 . . *n*1] και

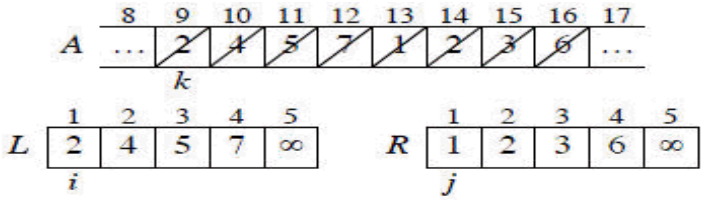
A[*middle* + 1 . . *right*] αντιγράφονται στον *R*[1 . . *n*2]

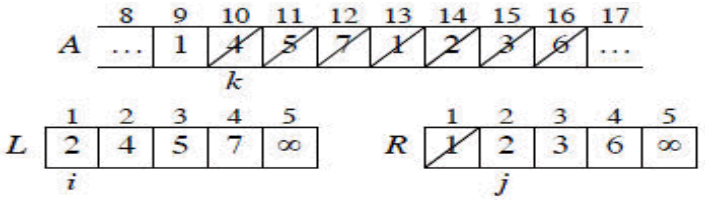
όπου *n*1 = *middle* − *left* + 1 και *n*2 = *right* − *middle*.

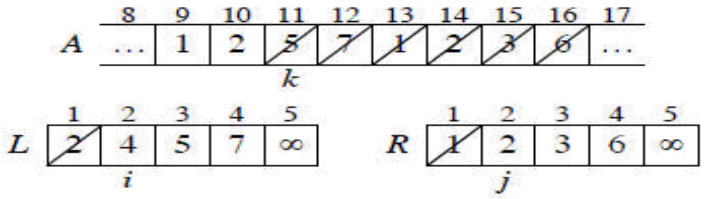
Χρησιμοποιούμε μια τιμή φρουρό (την τιμή ∞) και την τοποθετούμε στη θέση *L*[*n*1+1] και *R*[*n*2+1], έτσι δε χρειάζεται να ελέγξουμε αν κάποιος από τους 2 πίνακες είναι κενός.

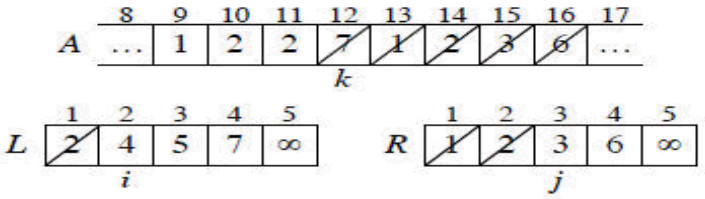
merge (*A*, *left*, *middle*, *right*)

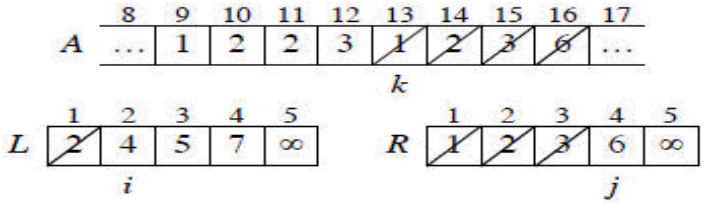
```
1.   n1 ← middle − left + 1
2.   n2 ← right − middle
3.   Create arrays L[1 . . n1 + 1] and R[1 . . n2 + 1]
4.   for i ← 1 to n1 do
5.        L[i] ← A[left + i − 1]
6.   for j ← 1 to n2 do
7.        R[j] ← A[right + j ]
8.   L[n1 + 1] ← ∞
9.   R[n2 + 1] ← ∞
10.  i ← 1
11.  j ← 1
12.  for k ← left to right do
13.       if L[i ] ≤ R[ j] then
14.                 A[k] ← L[i]
15.                 i ← i + 1
16.            else A[k] ← R[j]
17.                 j ← j + 1
```

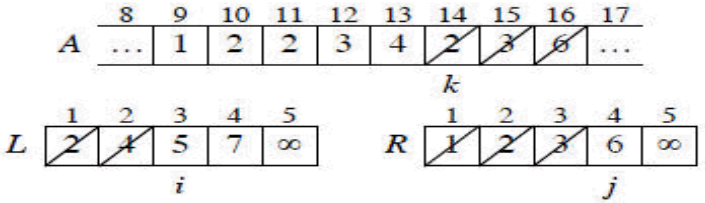

8 9 10 11 12 13 14 15 16 17
A ... 2 4 5 7 1 2 3 6 ...
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

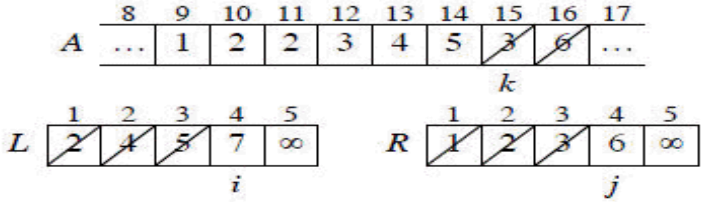

8 9 10 11 12 13 14 15 16 17
A ... 1 4 5 7 1 2 3 6 ...
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

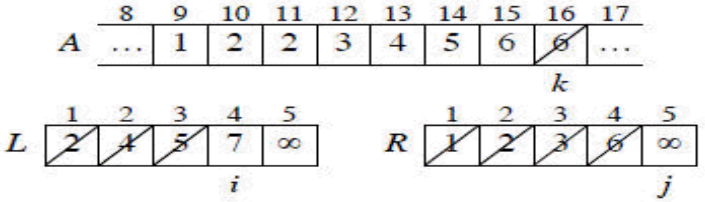

8 9 10 11 12 13 14 15 16 17
A ... 1 2 5 7 1 2 3 6 ...
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

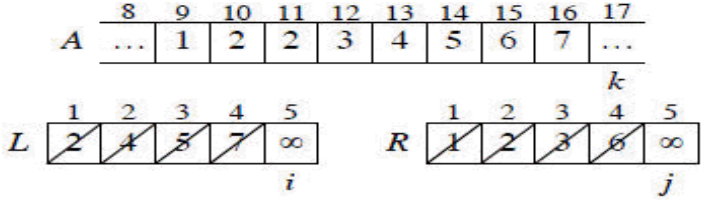

8 9 10 11 12 13 14 15 16 17
A ... 1 2 2 7 1 2 3 6 ...
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j


8 9 10 11 12 13 14 15 16 17
A ... 1 2 2 3 1 2 3 6 ...
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j


8 9 10 11 12 13 14 15 16 17
A ... 1 2 2 3 4 2 3 6 ...
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j


8 9 10 11 12 13 14 15 16 17
A ... 1 2 2 3 4 5 3 6 ...
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j


8 9 10 11 12 13 14 15 16 17
A ... 1 2 2 3 4 5 6 6 ...
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j


8 9 10 11 12 13 14 15 16 17
A ... 1 2 2 3 4 5 6 7 ...
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

Ανάλυση πολυπλοκότητας Merge sort με Κύριο θεώρημα

Τα 2 πρώτα for loops (γραμμή 4 και 6) χρειάζονται $\Theta(n1 + n2) = \Theta(n)$ χρόνο.

Το τελευταίο for loop (γραμμή 12) εκτελεί n επαναλήψεις, κάθε μια απαιτεί σταθερό χρόνο. Συνεπώς ο συνολικός χρόνος είναι $\Theta(n)$.

Για απλότητα θεωρούμε το n ως δύναμη του 2 έτσι ώστε το βήμα διαμοιρασμού να οδηγεί σε 2 υποπροβλήματα ιδίου μεγέθους $n/2$.

Η βασική περίπτωση συμβαίνει για $n = 1$.

Για $n \geq 2$, ο χρόνος για την εκτέλεση των βημάτων του merge sort:

1. **Διαίρει (Divide)**: Υπολογισμός του *middle* ως μέσος όρος των *left* και *right* που απαιτεί σταθερό χρόνο $\Theta(1)$.
2. **Κυρίευε (Conquer)**: Αναδρομική επίλυση των 2 υποπροβλημάτων, καθένα μεγέθους $n/2$, δηλαδή $2T(n/2)$.
3. **Συνδύασε (Combine)**: Συγχώνευση σε ένα πίνακα με n-στοιχεία απαιτεί $\Theta(n)$ χρόνο.

Αθροίζοντας τα παραπάνω (1) & (3) δίνουν μια συνάρτηση που είναι γραμμικού χρόνου n, δηλαδή $\Theta(n)$.

Προσθέτοντας τον όρο $2T(n/2)$ από το βήμα «κυρίευε» (2) καταλήγουμε στην παρακάτω αναδρομική σχέση

Αρα η αναδρομική σχέση για το merge sort είναι:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\,, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1\,. \end{cases}$$

Ανάλυση πολυπλοκότητας Merge sort με Κύριο θεώρημα

Η αναδρομική σχέση για το merge sort είναι: $T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$

Με βάση το κεντρικό θεώρημα έχουμε:

Εστω $a \geq 1$, και $b > 1$ σταθερές, f(n) μια συνάρτηση και $T(n)$ μια συνάρτηση που ορίζεται επί των μη αρνητικών ακεραίων από την αναδρομική σχέση

$$T(n) = aT(n/b) + f(n)$$

Όπου n/b εννοείται είτε $\lfloor n/b \rfloor$ είτε $\lceil n/b \rceil$! Στην περίπτωση αυτή το $T(n)$ μπορεί να φραγεί ασυμπτωτικά ως εξής:

$f(n) = O\left(n^{\log_b a - \varepsilon}\right)$ για κάποια σταθερά ε > 0, τότε $T(n) = \Theta\left(n^{\log_b a}\right)$

$f(n) = \Theta\left(n^{\log_b a}\right)$ τότε $T(n) = \Theta\left(n^{\log_b a} \log_2 n\right)$

$f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ για κάποια σταθερά ε > 0, και αν $af(n/b) \leq cf(n)$

για κάποια σταθερά ε < 1 και για όλα τα n από κάποια τιμή και πάνω, τότε $T(n) = \Theta(f(n))$

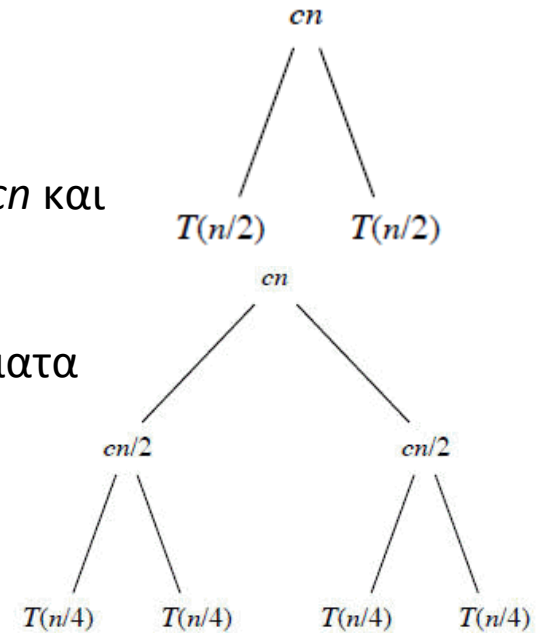
Με βάση το **Κύριο θεώρημα** (Master Theorem) έχουμε $a = 2$, $b = 2$, $f(n) = \Theta(n)$, τότε

$$T(n) = \Theta\left(n^{\log_2 2} \log_2 n\right) = \Theta(n \log_2 n)$$

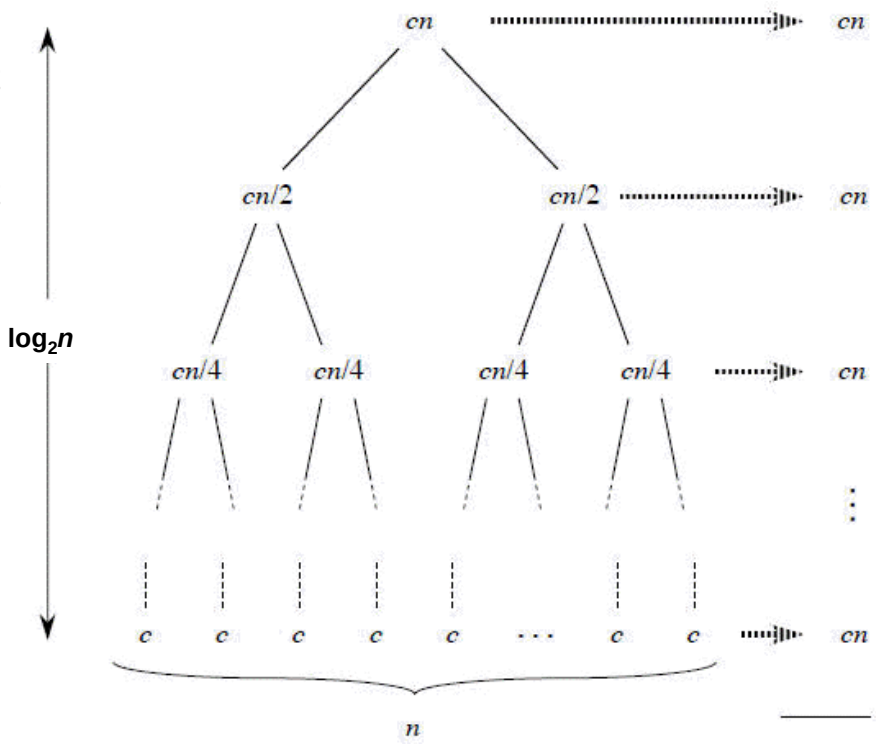
Η 1η εικόνα δείχνει ότι για το αρχικό πρόβλημα έχουμε κόστος *cn* και επιπλέον δυο υποπροβλήματα που το καθένα κοστίζει T (*n*/2).

Η 2η εικόνα δείχνει ότι για καθένα από τα υποπροβλήματα μεγέθους-*n*/2, έχουμε κόστος *cn*/2, και επιπλέον 2 υποπροβλήματα που το καθένα κοστίζει T (*n*/4).

Και συνεχίζουμε με την επέκταση του δένδρου μέχρις ότου η διάσταση του κάθε υποπροβλήματος να γίνει 1.
Στο δένδρο αναδρομής κάθε επίπεδο έχει κόστος *cn*.



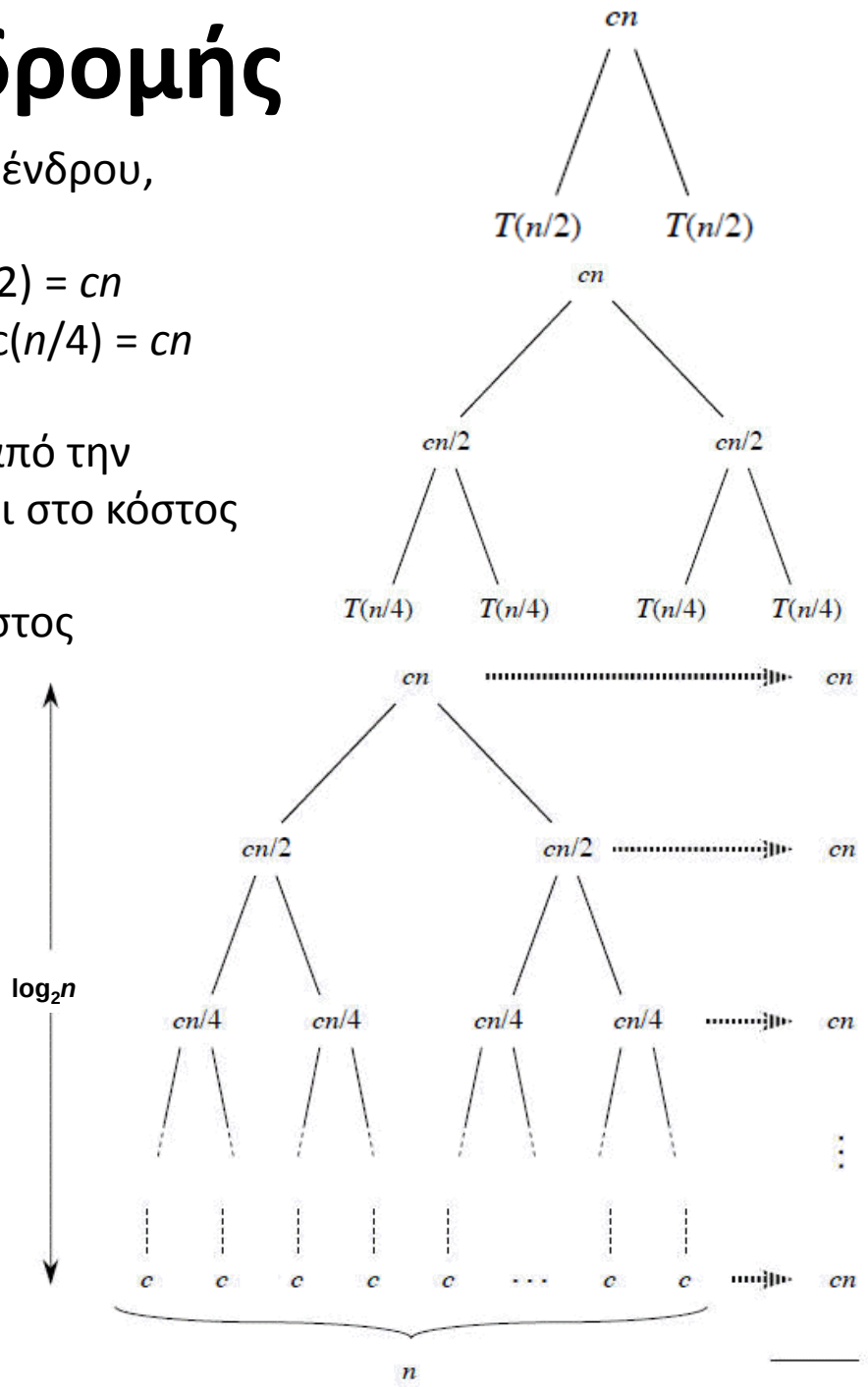
Το 1ο επίπεδο έχει κόστος *cn*.
Το επόμενο επίπεδο έχει 2 υποπροβλήματα με κόστος το καθένα *cn*/2.
Το επόμενο επίπεδο έχει 4 υποπροβλήματα με κόστος το καθένα *cn*/4.
Κάθε φορά που κατεβαίνουμε 1 επίπεδο το πλήθος των υποπροβλημάτων διπλασιάζεται αλλά το κόστος κάθε υποπροβλήματος μειώνεται στο μισό.

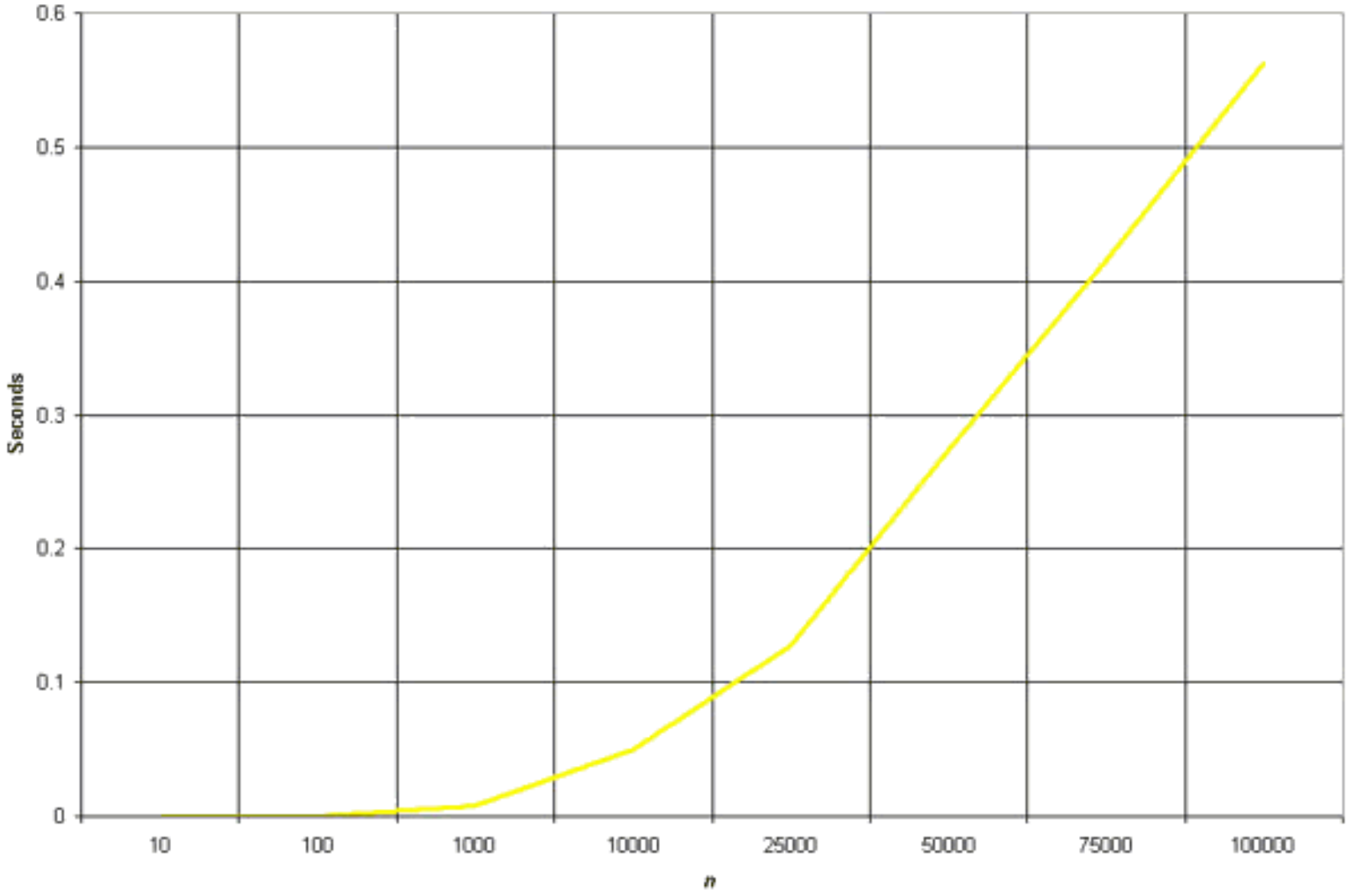


Δένδρο αναδρομής

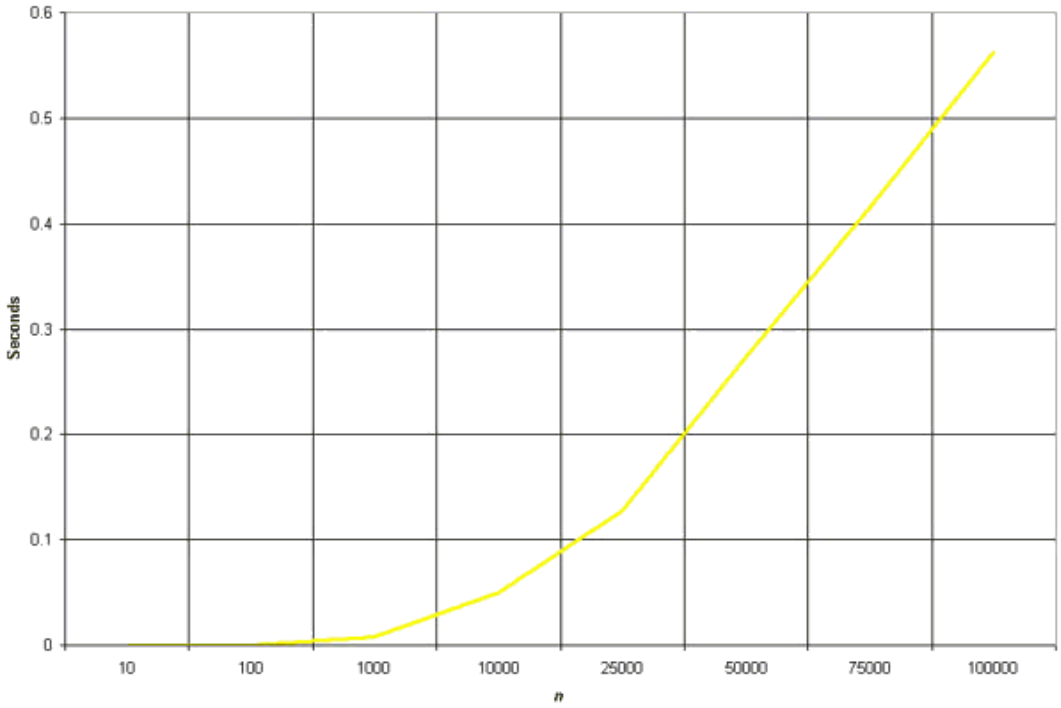
Αθροίζοντας τα κόστη σε κάθε επίπεδο του δένδρου, έχουμε:
Στο υψηλότερο επίπεδο έχουμε $c(n/2) + c(n/2) = cn$
Στο επόμενο αυτού : $c(n/4) + c(n/4)\ c(n/4) + c(n/4) = cn$
…..
Γενικά στο *i* επίπεδο που απέχει *i* βαθμίδες από την κορυφή έχει 2^i κόμβους καθένας συνεισφέρει στο κόστος μια ποσότητα $c(n/2^i)$.
Επομένως το επίπεδο αυτό έχει συνολικό κόστος $2^i\ c(n/2^i) = cn$.

Στο χαμηλότερο επίπεδο υπάρχουν *n* κόμβοι, καθένας από τους οποίους συνεισφέρει κόστος c, οπότε το συνολικό κόστος του επιπέδου αυτού είναι *cn*
Συνεπώς το κόστος σε κάθε επίπεδο παραμένει το ίδιο.
Το ύψος του δένδρου αναδρομής είναι $\log_2 n$ και έχει $\log_2 n + 1$ επίπεδα.

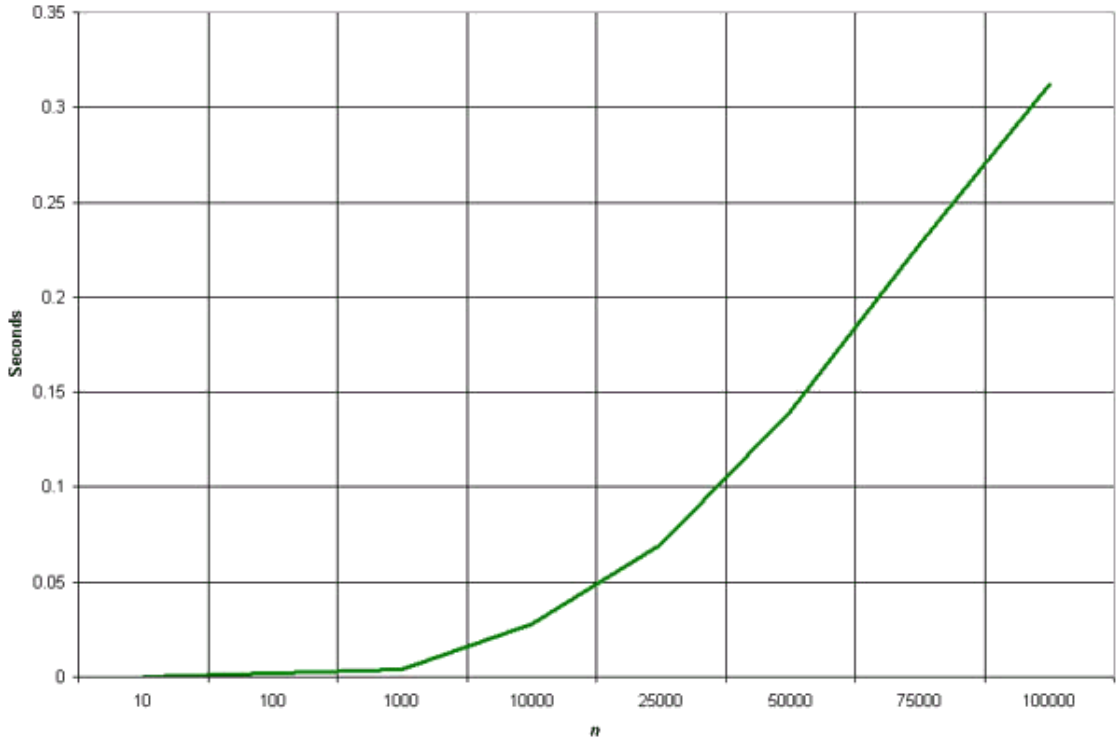



0.6
0.5
0.4
0.3
0.2
0.1
0
Seconds
10
100
1000
10000
25000
50000
75000
100000
n

Mergesort



Quicksort



Τέλος Ενότητας