Αλγόριθμοι Ταξινόμησης

Εισαγωγή (1/3)

Η έρευνα για αποδοτικούς αλγόριθμους ταξινόμησης έχει τις ρίζες της από τις αρχές της εμφάνισης της Επιστήμης Η/Υ

Μια συλλογή αντικειμένων για τα οποία μπορεί να υλοποιηθεί σύγκριση πρόκειται να ταξινομηθούν

Η πρόθεση είναι να βάλουμε τα αντικείμενα σε τέτοια σειρά ώστε $A[i] <= A[j]$ για $i<j$

Αν υπάρχουν διπλά αντικείμενα τότε αυτά πρέπει να είναι συνεχόμενα

Η ταξινομημένη λίστα πρέπει να είναι μια αναδιάταξη (permutation) των αντικειμένων

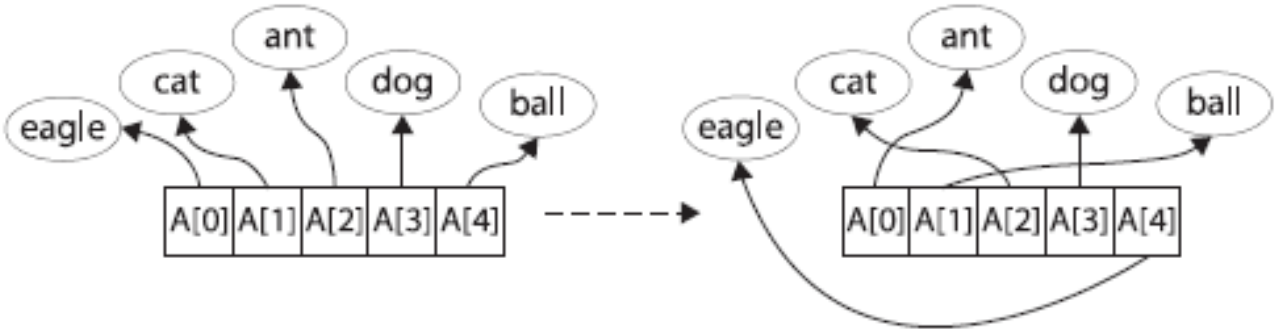
Εισαγωγή (2/3)

Τα στοιχεία / αντικείμενα μπορεί να είναι στη μνήμη ή στο σύστημα αρχείων

Η αποθηκευμένες πληροφορίες στη μνήμη είναι είτε pointer-based ή value-based

Pointer based

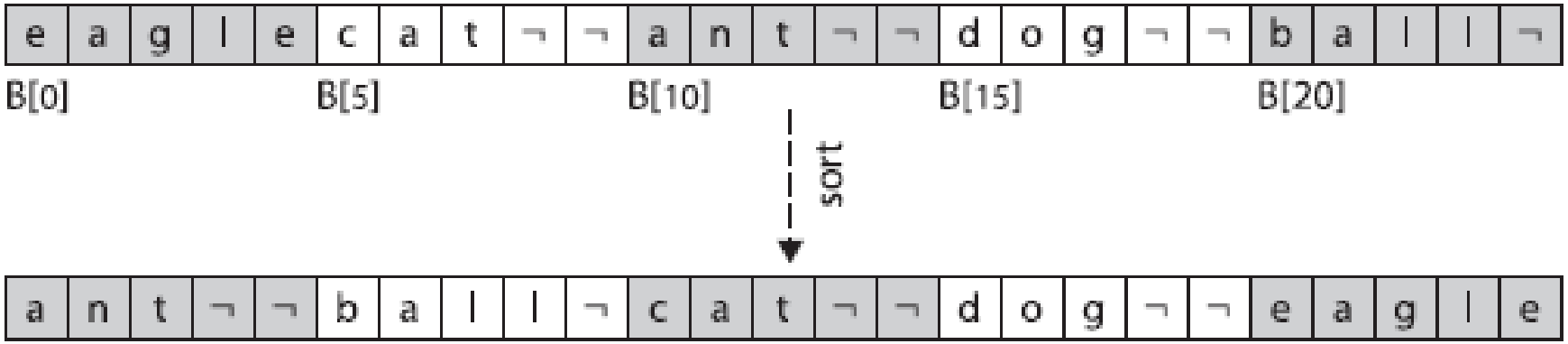
- Αποθηκεύουμε τους δείκτες προς την πληροφορία



Εισαγωγή (3/3)

Value based

- Αποθηκεύουμε τα στοιχεία σε record blocks



Insertion Sort

Εισαγωγή

Αποδοτικός αλγόριθμος για ταξινόμηση μικρού πλήθους στοιχείων

Ανήκει στην κατηγορία των αλγορίθμων **Decrease and Conquer**

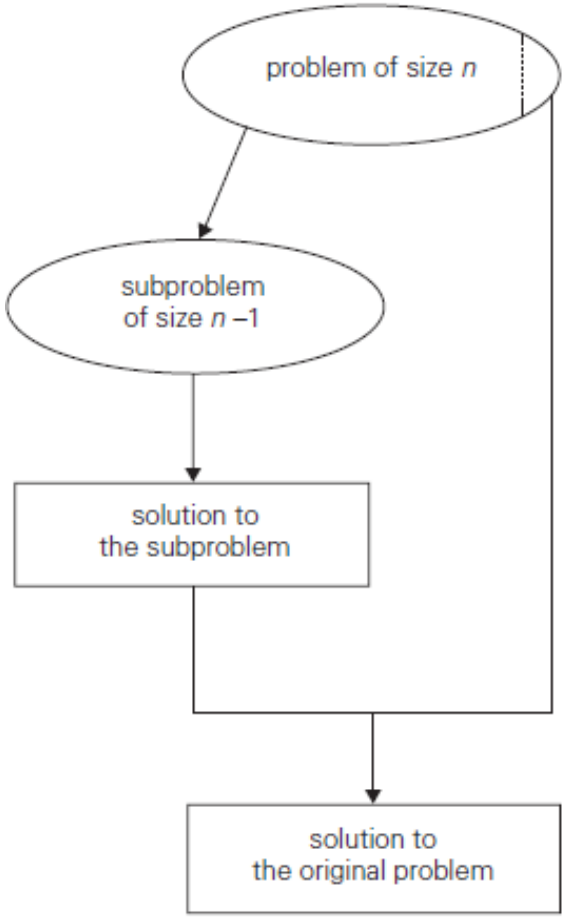
- Αποτιμά τη σχέση μιας λύσης με τη λύση του προβλήματος σε μικρότερο μέγεθος του προβλήματος
- Αν η σχέση οριστεί, τότε η λύση έρχεται είτε ακολουθώντας μια bottom-up ή μια top-down προσέγγιση
- Η top-down οδηγεί σε μια αναδρομική σχέση (υπάρχουν περιπτώσεις που λύνονται με μη αναδρομικές σχέσεις)
- Η bottom-up υλοποιείται επαναληπτικά ξεκινώντας από τη λύση στο μικρότερο μέγεθος και ακολουθώντας μια **incremental** προσέγγιση

Decrease and Conquer (1/3)

Τρεις παραλλαγές:

- Μείωση κατά μια σταθερά
  - Σε κάθε επανάληψη μειώνουμε κατά τη σταθερά
  - Υπολογισμός του α^{ν}
  - $\alpha^{\nu} = \alpha^{\nu-1}\,\alpha$
  - Οπότε

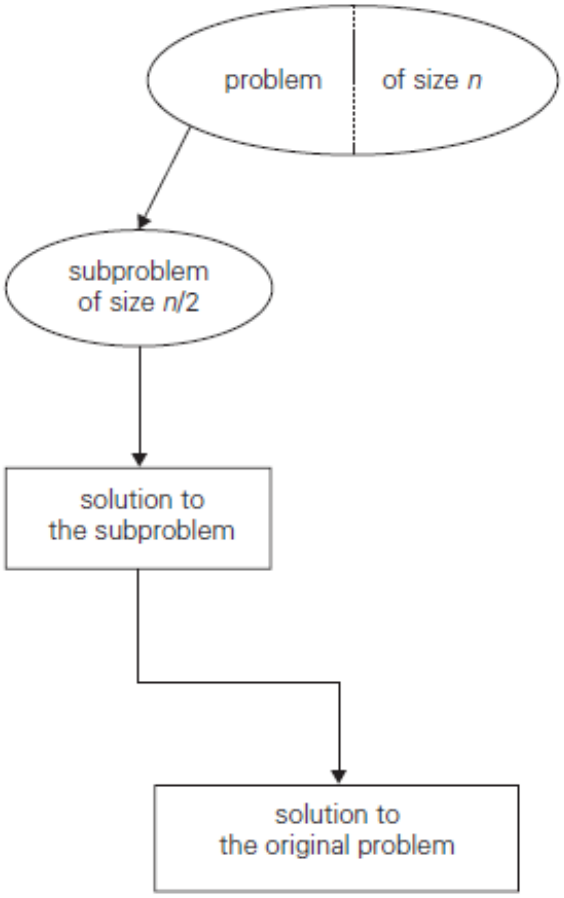
$$f(n) = \begin{cases} f(n-1) \cdot a & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$$



Decrease and Conquer (2/3)

- Μείωση κατά ένα σταθερό παράγοντα
  - Στις περισσότερες εφαρμογές ο παράγοντας είναι ίσος με 2
  - Υπολογισμός του α^{ν}
  - $\alpha^{\nu} = (\alpha^{\nu/2})^2$
  - Αν το ν είναι περιττός, πρέπει να υπολογίσουμε το $\alpha^{\nu-1}$ και μετά να πολλαπλασιάσουμε με το α
  - Οπότε

$$a^n = \begin{cases} (a^{n/2})^2 & \text{if } n \text{ is even and positive} \\ (a^{(n-1)/2})^2 \cdot a & \text{if } n \text{ is odd} \\ 1 & \text{if } n = 0 \end{cases}$$



Decrease and Conquer (3/3)

- Μεταβλητό μέγεθος μείωσης
  - Το pattern μείωσης είναι μεταβλητό σε κάθε επανάληψη
  - Παράδειγμα: Μέγιστος κοινός διαιρέτης

$$\gcd(m, n) = \gcd(n, m \bmod n)$$

Insertion Sort (1/9)

Εφαρμόζει μείωση κατά ένα

Υποθέτουμε ότι το μικρότερο πρόβλημα της ταξινόμησης των n-2 στοιχείων έχει ήδη επιλυθεί για την ταξινόμηση των n-1 στοιχείων

Βρίσκουμε την κατάλληλη θέση για ένα στοιχείο μέσα στα n-1 και το τοποθετούμε εκεί

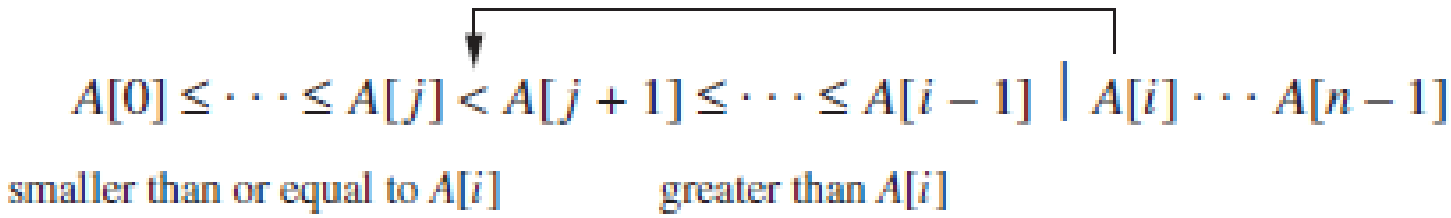
Το στοιχείο A[i] μπαίνει στην κατάλληλη θέση

Η προσπέλαση στα στοιχεία γίνεται ξεκινώντας από αριστερά προς τα δεξιά

Insertion Sort (2/9)

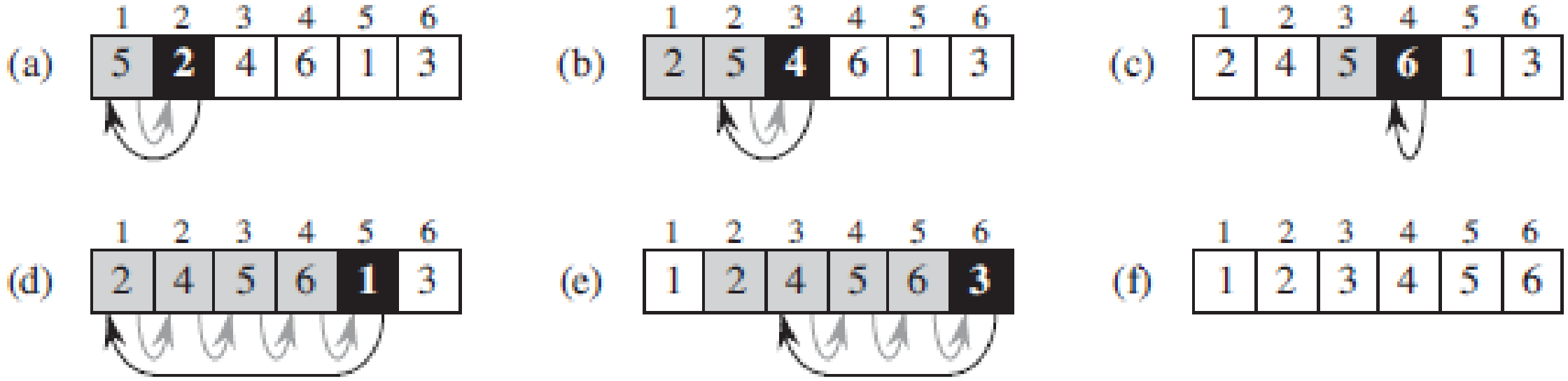
ALGORITHM *InsertionSort*$(A[0..n-1])$

//Sorts a given array by insertion sort
//Input: An array $A[0..n-1]$ of n orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
for $i \leftarrow 1$ **to** $n-1$ **do**
    $v \leftarrow A[i]$
    $j \leftarrow i-1$
    while $j \geq 0$ **and** $A[j] > v$ **do**
        $A[j+1] \leftarrow A[j]$
        $j \leftarrow j-1$
    $A[j+1] \leftarrow v$



Insertion Sort (3/9)

$$A = \langle 5, 2, 4, 6, 1, 3 \rangle$$



Insertion Sort (4/9)

Βασική λειτουργία

- Σύγκριση A[j] > υ

Στη χειρότερη περίπτωση η σύγκριση εκτελείται για κάθε τιμή του j

Η χειρότερη περίπτωση είναι ένας πίνακας με στοιχεία σε φθίνουσα σειρά

Το πλήθος των συγκρίσεων στη χειρότερη περίπτωση είναι:

$$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \in \Theta(n^2)$$

Insertion Sort (5/9)

Στην καλύτερη περίπτωση, η σύγκριση θα εκτελεστεί μόνο μια φορά για κάθε επανάληψη του εξωτερικού loop

Στην καλύτερη περίπτωση ο αρχικός πίνακας είναι ήδη ταξινομημένος

Πλήθος συγκρίσεων

$$C_{best}(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n)$$

Στην ουσία, δεν είναι χρήσιμη προσέγγιση

Insertion Sort (6/9)

Στη μέση περίπτωση ο αλγόριθμος εκτελεί περίπου τις μισές συγκρίσεις από ότι στη χειρότερη περίπτωση

$$C_{avg}(n) \approx \frac{n^2}{4} \in \Theta(n^2)$$

Insertion Sort (7/9)

Εναλλακτική προσέγγιση στην ανάλυση του αλγορίθμου:

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $A[i+1] = key$	c_8	$n-1$

Insertion Sort (8/9)

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1)$$

$$+ c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Καλύτερη περίπτωση (γραμμική συνάρτηση)

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \end{aligned}$$

Insertion Sort (9/9)

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1)$$

$$+ c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Χειρότερη περίπτωση

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \left(\frac{n(n+1)}{2} - 1\right)$$

$$+ c_6 \left(\frac{n(n-1)}{2}\right) + c_7 \left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n$$

$$- (c_2 + c_4 + c_5 + c_8)$$

Insertion Sort (10/10)

Demo

http://visualgo.net/sorting.html#

http://www.sorting-algorithms.com/insertion-sort

https://www.bluffton.edu/~nesterd/java/SortingDemo.html

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ

ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ

ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

ΔΙΑΛΕΞΗ 5

Selection Sort

Selection Sort (1/6)

Αναζητά στη λίστα των αντικειμένων το μικρότερο στοιχείο

Τοποθετεί το μικρότερο στοιχείο στη θέση του

Αρχικά ξεκινά από την πρώτη θέση της λίστας

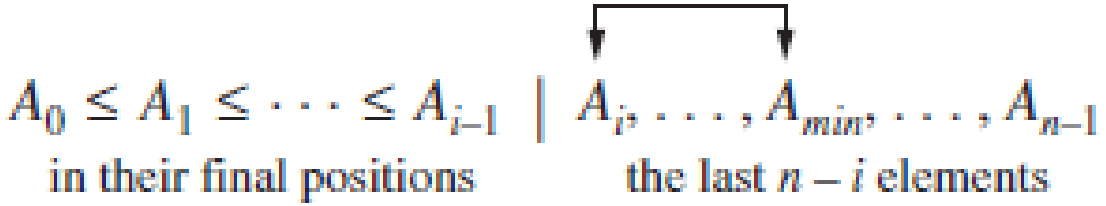
Έπειτα προχωρά στη δεύτερη, την τρίτη, κ.ο.κ.

Κάθε φορά το μικρότερο στοιχείο τοποθετείται στη θέση του ($1^η$, $2^η$, $3^η$, …)

Ανήκει στην κατηγορία των Brute Force αλγορίθμων

Selection Sort (2/6)

Γενικά ο αλγόριθμος αναζητά το i^{th} μικρότερο στοιχείο στα n-1 στοιχεία και το τοποθετεί στην i θέση



Μετά από n-1 περάσματα η λίστα έχει ταξινομηθεί

Selection Sort (3/6)

Αλγόριθμος

ALGORITHM *SelectionSort*$(A[0..n-1])$

//Sorts a given array by selection sort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
for $i \leftarrow 0$ **to** $n-2$ **do**
    $min \leftarrow i$
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[j] < A[min]$ $min \leftarrow j$
    swap $A[i]$ and $A[min]$

Selection Sort (4/6)

Παράδειγμα

- Ταξινόμηση των 89, 45, 68, 90, 29, 34, 17

89	45	68	90	29	34	**17**
17 |	45	68	90	**29**	34	89
17	29 |	68	90	45	**34**	89
17	29	34 |	90	**45**	68	89
17	29	34	45 |	90	**68**	89
17	29	34	45	68 |	90	**89**
17	29	34	45	68	89 |	90

Selection Sort (5/6)

Ανάλυση πολυπλοκότητας

- Μέγεθος εισόδου: n
- Βασική λειτουργία: A[j] < A[min]
- Πλήθος εκτελέσεων

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i)$$

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2}$$

- Πολυπλοκότητα: $\Theta(n^2)$
- Πολυπλοκότητα των αντιμεταθέσεων: $\Theta(n)$

Selection Sort (6/6)

Demo

https://visualgo.net/en/sorting

http://www.sorting-algorithms.com/selection-sort

Bubble Sort

Bubble Sort (1/7)

Ανήκει στην κατηγορία των Brute Force αλγορίθμων

Συγκρίνει γειτονικά στοιχεία και τα ανταλλάσσει αν είναι εκτός σειράς

Οι συνεχόμενες αντιμεταθέσεις προκαλούν τη μετακίνηση και τοποθέτηση του κάθε στοιχείου στη θέση του (προσομοίωση φυσαλίδας)

Στο $1^{ο}$ πέρασμα, το $1^{ο}$ στοιχείο μπαίνει στη θέση του, στο $2^{ο}$ πέρασμα, το $2^{ο}$ στοιχείο μπαίνει στη θέση του, κ.ο.κ.

Bubble Sort (2/7)

Στο πέρασμα i (0<= i <= n-2) η εκτέλεση του αλγορίθμου έχει ως εξής:

$$A_0, \ldots, A_j \overset{?}{\leftrightarrow} A_{j+1}, \ldots, A_{n-i-1} \mid \underbrace{A_{n-i} \leq \cdots \leq A_{n-1}}_{\text{in their final positions}}$$

Bubble Sort (3/7)

Αλγόριθμος

ALGORITHM *BubbleSort*$(A[0..n-1])$

//Sorts a given array by bubble sort

//Input: An array $A[0..n-1]$ of orderable elements

//Output: Array $A[0..n-1]$ sorted in nondecreasing order

for $i \leftarrow 0$ **to** $n-2$ **do**

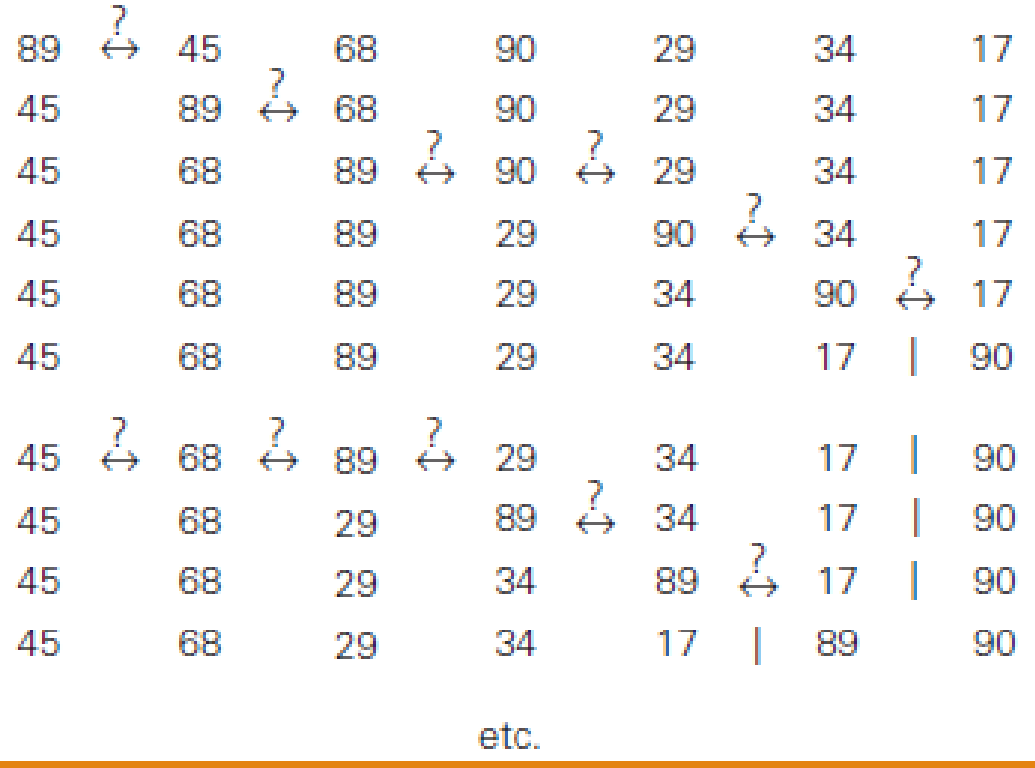
  for $j \leftarrow 0$ **to** $n-2-i$ **do**

    if $A[j+1] < A[j]$ swap $A[j]$ and $A[j+1]$

Bubble Sort (4/7)

Παράδειγμα

- Ταξινόμηση των 89, 45, 68, 90, 29, 34, 17



Bubble Sort (5/7)

Ανάλυση πολυπλοκότητας

- Μέγεθος εισόδου: n
- Βασική λειτουργία: A[j+1] < A[j]
- Πλήθος εκτελέσεων

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \sum_{i=0}^{n-2} [(n-2-i) - 0 + 1]$$

$$= \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2} \in \Theta(n^2)$$

- Πολυπλοκότητα: $\Theta(n^2)$
- Πολυπλοκότητα των αντιμεταθέσεων: $\Theta(n^2)$

$$S_{worst}(n) = C(n) = \frac{(n-1)n}{2} \in \Theta(n^2)$$

Bubble Sort (6/7)

Βελτιωμένη έκδοση

```
for(int i = 0 ; i &lt; a.length ; i++){
        boolean swap = false;
        for(int j = 0 ; j&lt; a.length - i-1 ; j++){
                if(a[j] &gt; a[j+1]){
                        temp = a[j];
                        a[j] = a[j+1];
                        a[j+1]=temp;
                        swap=true;
                }
        }
        if(!swap)
                break;
}
```

Bubble Sort (7/7)

Demo

https://visualgo.net/en/sorting

http://www.sorting-algorithms.com/bubble-sort

https://www.bluffton.edu/~nesterd/java/SortingDemo.html

Merge Sort

Merge Sort (1/8)

Χαρακτηριστικό παράδειγμα της μεθόδου διαίρει και βασίλευε

Διαιρεί τη λίστα στη μέση και ταξινομεί τα τμήματα αναδρομικά

ALGORITHM *Mergesort*$(A[0..n-1])$

//Sorts array $A[0..n-1]$ by recursive mergesort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
if $n > 1$
copy $A[0..\lfloor n/2 \rfloor - 1]$ to $B[0..\lfloor n/2 \rfloor - 1]$
copy $A[\lfloor n/2 \rfloor..n-1]$ to $C[0..\lceil n/2 \rceil - 1]$
Mergesort$(B[0..\lfloor n/2 \rfloor - 1])$
Mergesort$(C[0..\lceil n/2 \rceil - 1])$
Merge(B, C, A)

Merge Sort (2/8)

Διαδικασία συγχώνευσης

- Δύο δείκτες υιοθετούνται ώστε να δείχνουν στο πρώτο στοιχείο των λιστών που πρόκειται να συγχωνευτούν
- Τα δύο στοιχεία ελέγχονται και το μικρότερο μπαίνει στη νέα λίστα που δημιουργείται
- Ο δείκτης του μικρότερου στοιχείου ‘προχωρά’ στο επόμενο στοιχείο
- Η διαδικασία επαναλαμβάνεται μέχρι να εξαντληθούν οι πίνακες
- Αν εξαντληθεί ο ένας πίνακας τότε τα στοιχεία του άλλου αντιγράφονται στη νέα δομή

Merge Sort (3/8)

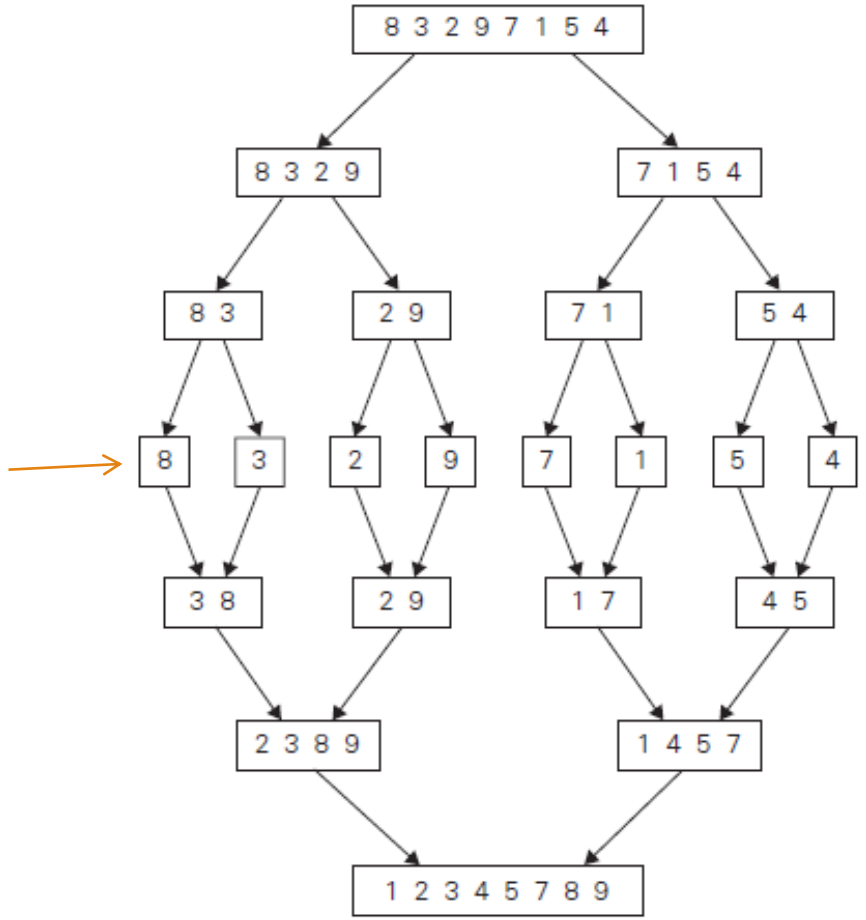
Αλγόριθμος συγχώνευσης

ALGORITHM $Merge(B[0..p-1], C[0..q-1], A[0..p+q-1])$

//Merges two sorted arrays into one sorted array
//Input: Arrays $B[0..p-1]$ and $C[0..q-1]$ both sorted
//Output: Sorted array $A[0..p+q-1]$ of the elements of B and C
$i \leftarrow 0;\ j \leftarrow 0;\ k \leftarrow 0$
while $i < p$ **and** $j < q$ **do**
    if $B[i] \leq C[j]$
        $A[k] \leftarrow B[i];\ i \leftarrow i+1$
    else $A[k] \leftarrow C[j];\ j \leftarrow j+1$
    $k \leftarrow k+1$
if $i = p$
    copy $C[j..q-1]$ to $A[k..p+q-1]$
else copy $B[i..p-1]$ to $A[k..p+q-1]$

Merge Sort (4/8)

Παράδειγμα: ταξινόμηση των 8, 3, 2, 9, 7, 1, 5, 4



Merge Sort (5/8)

Ανάλυση

- Πλήθος εκτελέσεων

$$C(n) = 2C(n/2) + C_{merge}(n) \quad \text{for } n > 1, \quad C(1) = 0$$

- Συγχώνευση
  - Σε κάθε βήμα, μια σύγκριση γίνεται και έπειτα ο συνολικός αριθμός των στοιχείων προς επεξεργασία μειώνεται κατά 1
  - Στη χειρότερη περίπτωση καμία από τις δύο υπο-λίστες δεν ‘αδειάζει’ πριν από την άλλη
  - Άρα $C_{merge}(n) = n - 1$

$$C_{worst}(n) = 2C_{worst}(n/2) + n - 1 \quad \text{for } n > 1, \quad C_{worst}(1) = 0$$

Merge Sort (6/8)

Εφαρμόζουμε το master θεώρημα και έχουμε:

$$C_{worst}(n) \in \Theta(n \log n)$$

Merge Sort (7/8)

Το πλήθος των συγκρίσεων στη χειρότερη περίπτωση είναι πολύ κοντά στο θεωρητικό ελάχιστο

Για μεγάλο n το πλήθος των συγκρίσεων στη μέση περίπτωση είναι 0.25n

Το πλεονέκτημα του merge sort σε σύγκριση με τους heap and quick sort είναι η σταθερότητα που επιδεικνύει

Το μειονέκτημα του αλγορίθμου είναι η γραμμική ποσότητα επιπλέον χώρου που απαιτεί

Merge Sort (8/8)

Demo

https://visualgo.net/en/sorting

http://www.sorting-algorithms.com/merge-sort

https://www.bluffton.edu/~nesterd/java/SortingDemo.html

Quick Sort

Quick Sort (1/17)

Ανήκει στην κατηγορία των διαίρει και βασίλευε αλγορίθμων

Παρά το γεγονός ότι στη χειρότερη περίπτωση έχει πολυπλοκότητα $\Theta(n^2)$ υιοθετείται πρακτικά αφού έχει πολύ καλή επίδοση στη μέση περίπτωση

Έχει επίσης πολύπλοκη λογική εκτέλεσης

Quick Sort (2/17)

Ταξινόμηση ενός πίνακα A[p,r]

- Χωρισμός του πίνακα σε δύο (πιθανώς άδειους) υπο-πίνακες A[p,q-1], A[q+1,r] τέτοιοι ώστε κάθε στοιχείο του A[p,q-1] να είναι μικρότερο ή ίσο του A[q] και κάθε στοιχείο του A[q+1,r] να είναι μεγαλύτερο ή ίσο του A[q] – Υπολογισμός / εύρεση του A[q]
- Ταξινόμηση των δύο υπο-πινάκων A[p,q-1], A[q+1,r] μέσω αναδρομικών κλήσεων του αλγορίθμου quicksort
- Μια και οι δύο υπο-πίνακες είναι ταξινομημένοι, δεν απαιτείται προσπάθεια για συγχώνευση τους – Ο πίνακας A είναι ήδη ταξινομημένος

Quick Sort (3/17)

Η αρχική κλήση του αλγορίθμου είναι: QUICKSORT(A, 1, A.length)

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

Quick Sort (4/17)

Το σημαντικότερο τμήμα του αλγορίθμου είναι η εύρεση του στοιχείου A[q] και ο διαχωρισμός του αρχικού πίνακα

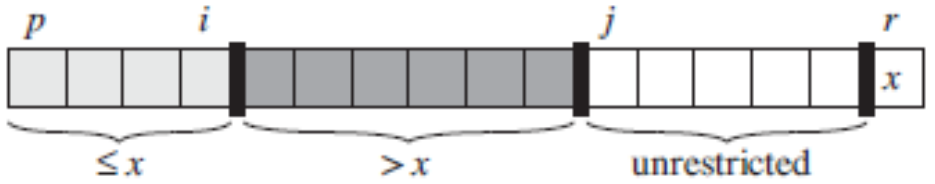
Ο αλγόριθμος διαχωρισμού έχει ως εξής:

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4       if A[j] ≤ x
5            i = i + 1
6            exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Quick Sort (5/17)

Επιλογή του στοιχείου A[r] ως pivot γύρω από το οποίο θα γίνει ο διαχωρισμός του A[p,r]

Ο πίνακας χωρίζεται σε 4 περιοχές (πιθανώς άδειες)




```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Στην αρχή κάθε επανάληψης (γραμμές 3-6), οι περιοχές ικανοποιούν τις ακόλουθες ιδιότητες (για κάθε k)

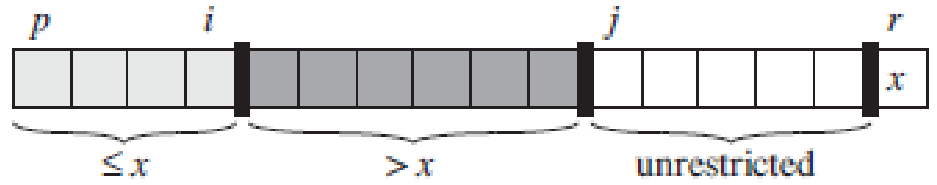
1. If $p \le k \le i$, then $A[k] \le x$.
2. If $i + 1 \le k \le j - 1$, then $A[k] > x$.
3. If $k = r$, then $A[k] = x$.

Quick Sort (6/17)

Οι δείκτες j και r-1 ορίζουν μια περιοχή στην οποία τα στοιχεία δεν έχουν κάποια ιδιαίτερη σχέση με το pivot x

Πριν από την 1η επανάληψη, θέτουμε i=p-1, j=p. Αφού δεν υπάρχουν τιμές στην περιοχή που ορίζεται από τα i+1, j-1, οι πρώτες δύο συνθήκες ικανοποιούνται

```
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```



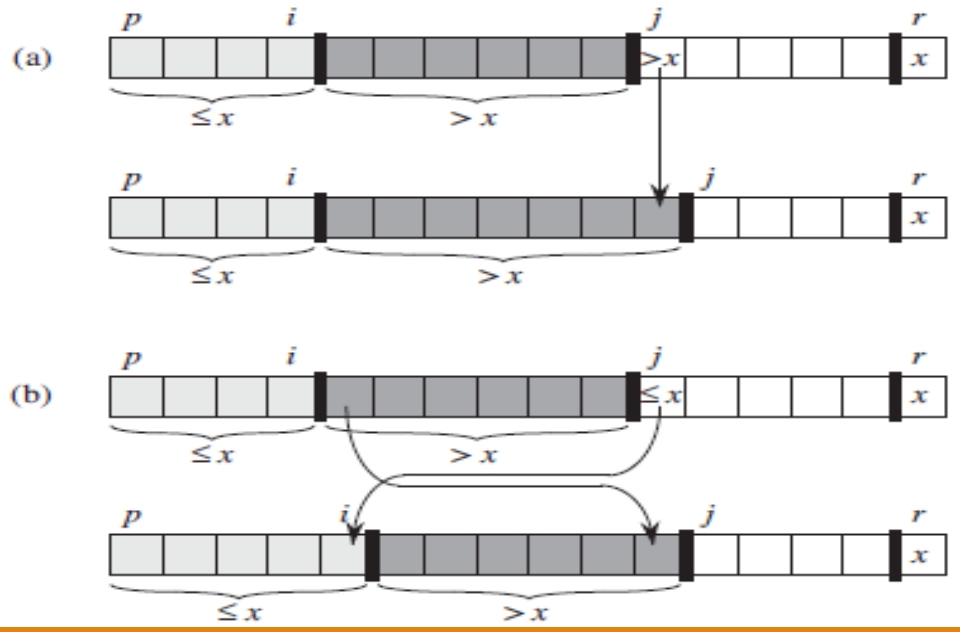
1. If $p \le k \le i$, then $A[k] \le x$.
2. If $i + 1 \le k \le j - 1$, then $A[k] > x$.
3. If $k = r$, then $A[k] = x$.

Quick Sort (7/17)

Όπως φαίνεται στην εικόνα, με βάση τη συνθήκη στη γραμμή 4, αν A[j]>x τότε αυξάνεται η τιμή του j κατά 1. Μόνο το condition 2 ικανοποιείται και όλα τα άλλα στοιχεία παραμένουν όπως έχουν

Όταν A[j]<=x, το i αυξάνει κατά 1 και γίνεται ανταλλαγή των A[i], A[j] και έπειτα αυξάνει το j κατά 1 - Με την ανταλλαγή έχουμε A[i]<=x και ικανοποιούμε την 1η συνθήκη – Επίσης έχουμε ότι A[j-1]>x

```
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```



Quick Sort (8/17)

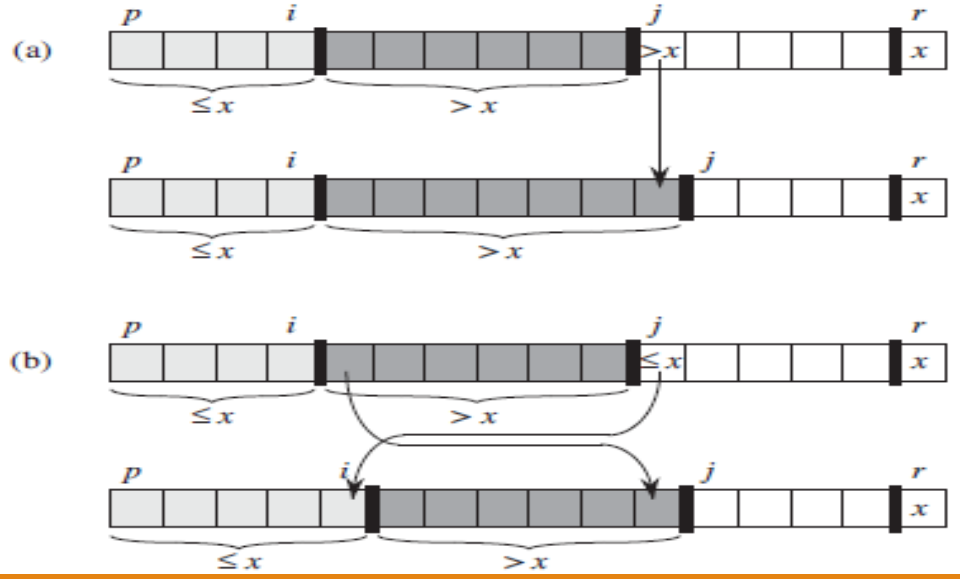
Ο τερματισμός του αλγορίθμου

Με βάση τα προηγούμενα κάθε στοιχείο του πίνακα έχει μπει σε ένα από τρία σύνολα: μικρότερα ή ίσα του x, μεγαλύτερα του x και ένα μονοσύνολο, το x

Οι τελευταίες δύο γραμμές του αλγορίθμου ανταλλάσσουν το x με το πιο αριστερά στοιχείο της περιοχής με τα στοιχεία που είναι μεγαλύτερα του x

Στη συνέχεια ο αλγόριθμος επιστρέφει το νέο δείκτη του pivot

```
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```



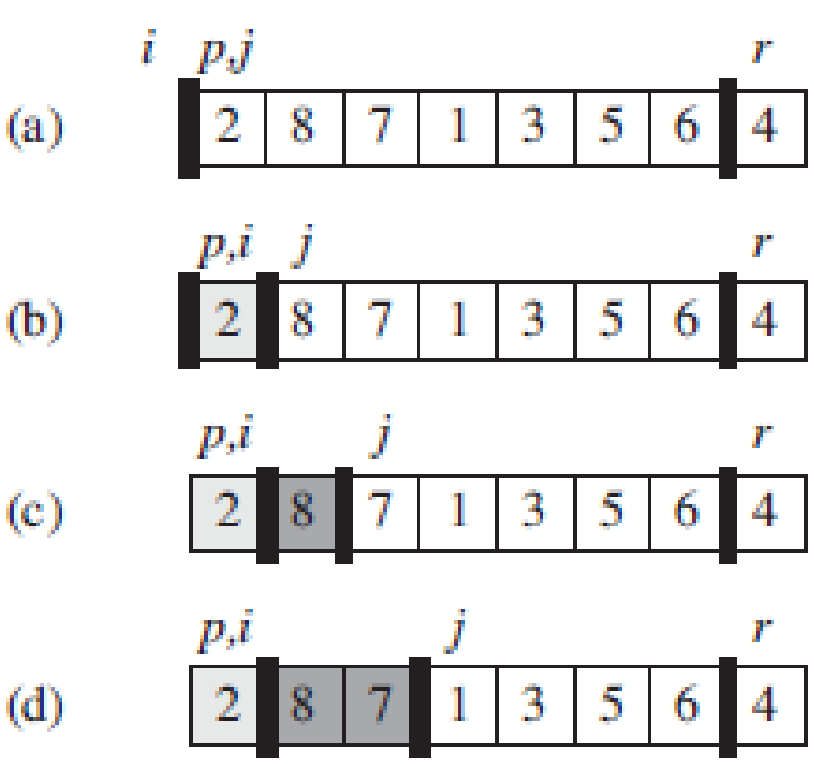
Quick Sort (9/17)

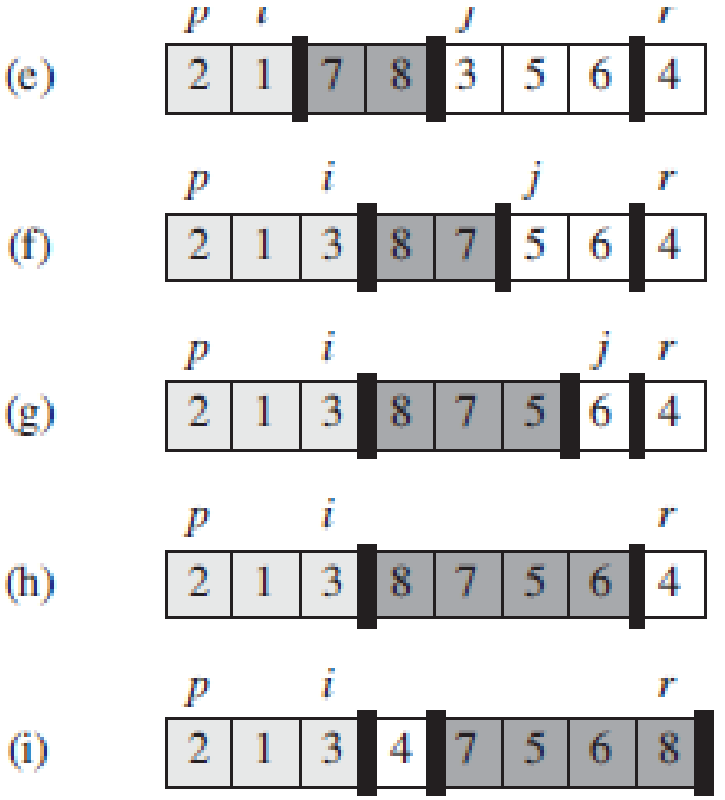
Ο χρόνος εκτέλεσης του τμήματος του διαχωρισμού έχει πολυπλοκότητα Θ(n)

Για τον πίνακα A[p,r] έχουμε ότι n=r-p+1

Quick Sort (10/17)

Παράδειγμα εκτέλεσης






```
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Quick Sort (11/17)

Ο χρόνος εκτέλεσης εξαρτάται από το αν το πλήθος των στοιχείων στους υπο-πίνακες είναι ισορροπημένο

Εξαρτάται από τα στοιχεία που χρησιμοποιούνται για το διαχωρισμό

Αν τα στοιχεία είναι ισοκατανεμημένα ο αλγόριθμος είναι ασυμπτωτικά τόσο γρήγορος όσο το merge sort

Αν τα στοιχεία δεν είναι ισοκατανεμημένα, ο αλγόριθμος είναι ασυμπτωτικά τόσο αργός όσο το insertion sort

Quick Sort (12/17)

Η χειρότερη περίπτωση είναι όταν ο αλγόριθμος παράγει ένα υπο-πρόβλημα που περιλαμβάνει δύο υπο-πίνακες, ένα με n-1 στοιχεία και ένα με 0 στοιχεία (άδειος υπο-πίνακας)

Για τη χειρότερη περίπτωση, υποθέτουμε ότι ο παραπάνω διαχωρισμός ισχύει για όλες τις αναδρομικές κλήσεις

Ο διαχωρισμός κοστίζει Θ(n) και η αναδρομική κλήση σε ένα πίνακα μεγέθους 0 έχει κόστος T(0) = Θ(1) (μόνο το return εκτελείται)

Συνεπώς:

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) \end{aligned}$$

Quick Sort (13/17)

Με τη μέθοδο της αντικατάστασης μπορούμε να αποδείξουμε ότι $T(n)=\Theta(n^2)$

Στη χειρότερη περίπτωση, ο αλγόριθμος έχει ίδια πολυπλοκότητα με τον insertion sort

Η χειρότερη πολυπλοκότητα συναντάται επίσης όταν ο πίνακας είναι ήδη ταξινομημένος – ο insertion sort σε αυτή την περίπτωση έχει πολυπλοκότητα $O(n)$

Quick Sort (14/17)

Στην καλύτερη περίπτωση, το τμήμα διαχωρισμού παράγει δύο υπο-πίνακες μεγέθους περίπου n/2 (floor(n/2), ceiling(n/2)-1)

Η αναδρομική σχέση είναι: **$T(n)=2T(n/2)+\Theta(n)$**

Με βάση το master θεώρημα (περίπτωση 2) έχουμε: **$T(n) = \Theta(n \log n)$**

Quick Sort (15/17)

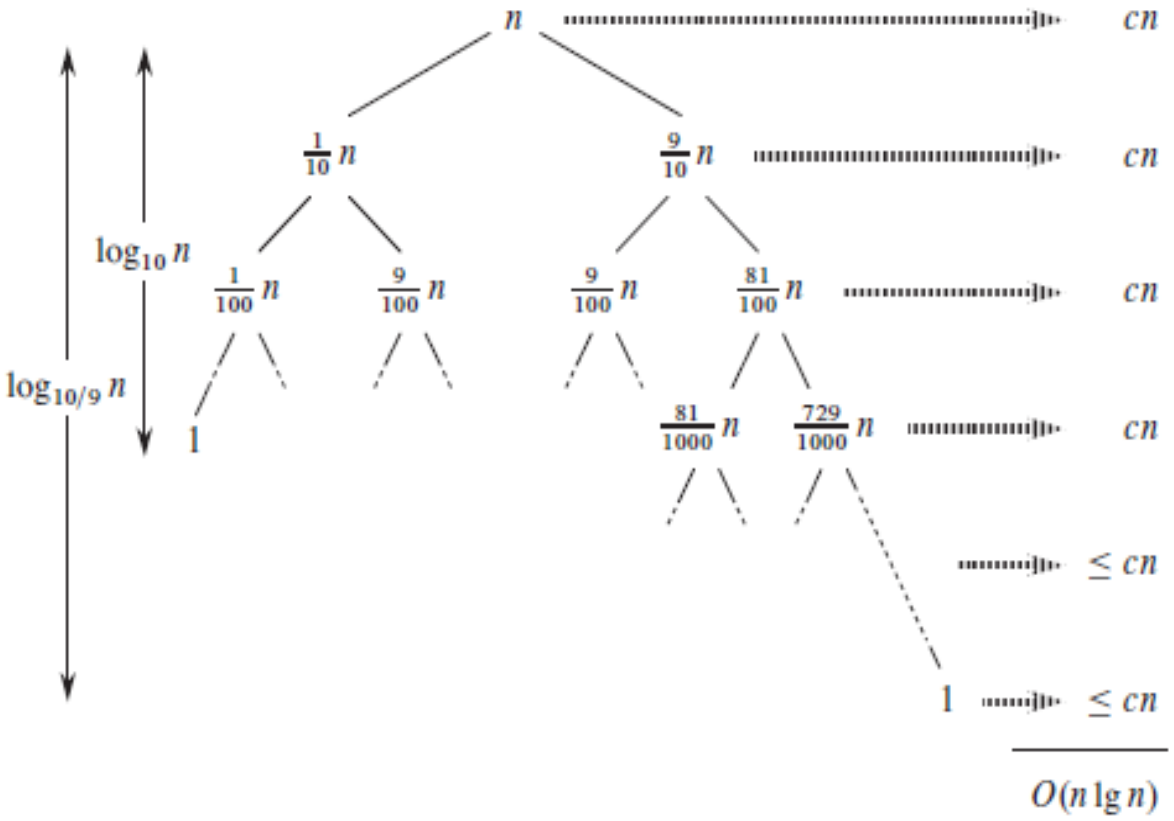
Η μέση περίπτωση είναι πιο κοντά στην καλύτερη περίπτωση παρά στη χειρότερη

Ας υποθέσουμε ότι ο διαχωρισμός παράγει 9-προς-1 αναλογία

Παίρνουμε την εξής αναδρομική σχέση

T(n)=T(9n/10)+T(n/10)+cn

Το βάθος του δένδρου είναι $\log_{10}n=\Theta(\log n)$

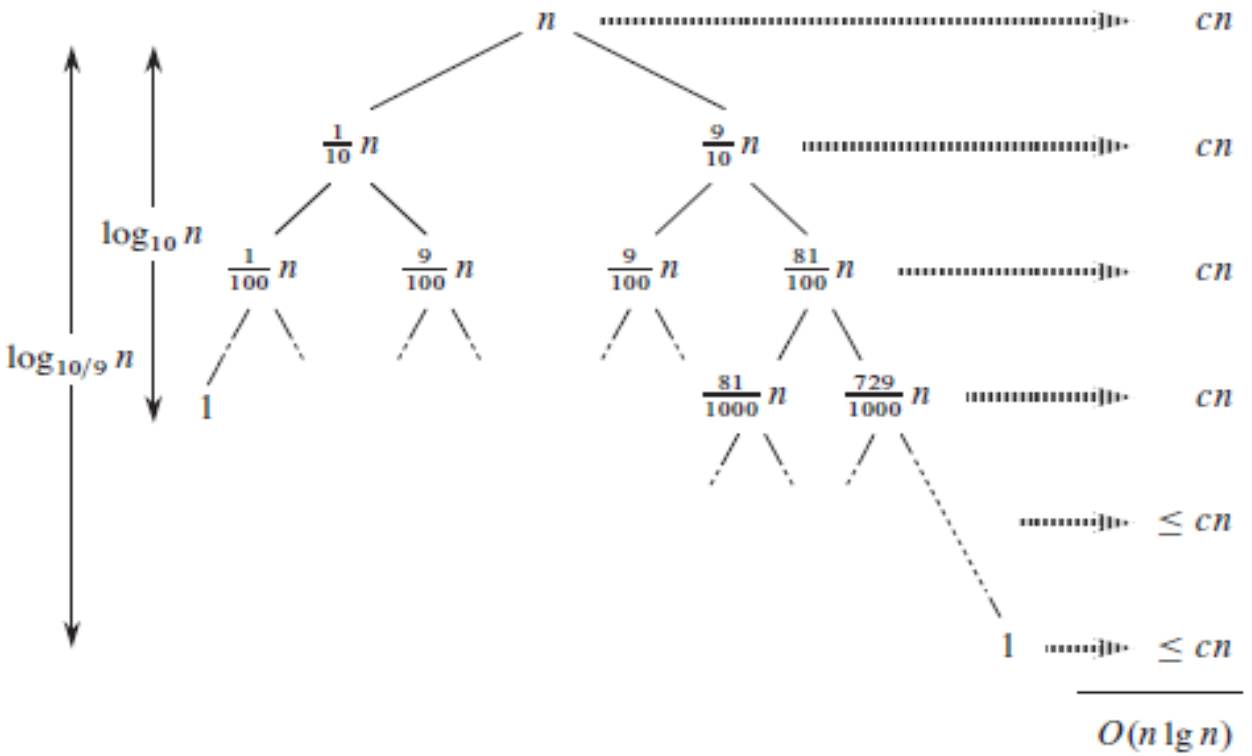


Quick Sort (16/17)

Η αναδρομή τερματίζει στο $\log_{10/9} n = \Theta(\log n)$

Συνολικό κόστος: O(n logn)

Βλέπουμε πως παρά το γεγονός ότι ο αρχικός διαχωρισμός δεν είναι ισοκατανεμημένος, ο αλγόριθμος έχει πολυπλοκότητα O(n logn)



Quick Sort (17/17)

Demo

https://visualgo.net/en/sorting

http://www.sorting-algorithms.com/quick-sort

https://www.bluffton.edu/~nesterd/java/SortingDemo.html

Median Sort

Median Sort (1/9)

Ανήκει στην κατηγορία Διαίρει και Βασίλευε

Ανταλλάσσει το μεσαίο (median) στοιχείο με το στοιχείο στο μέσο του πίνακα (middle element)

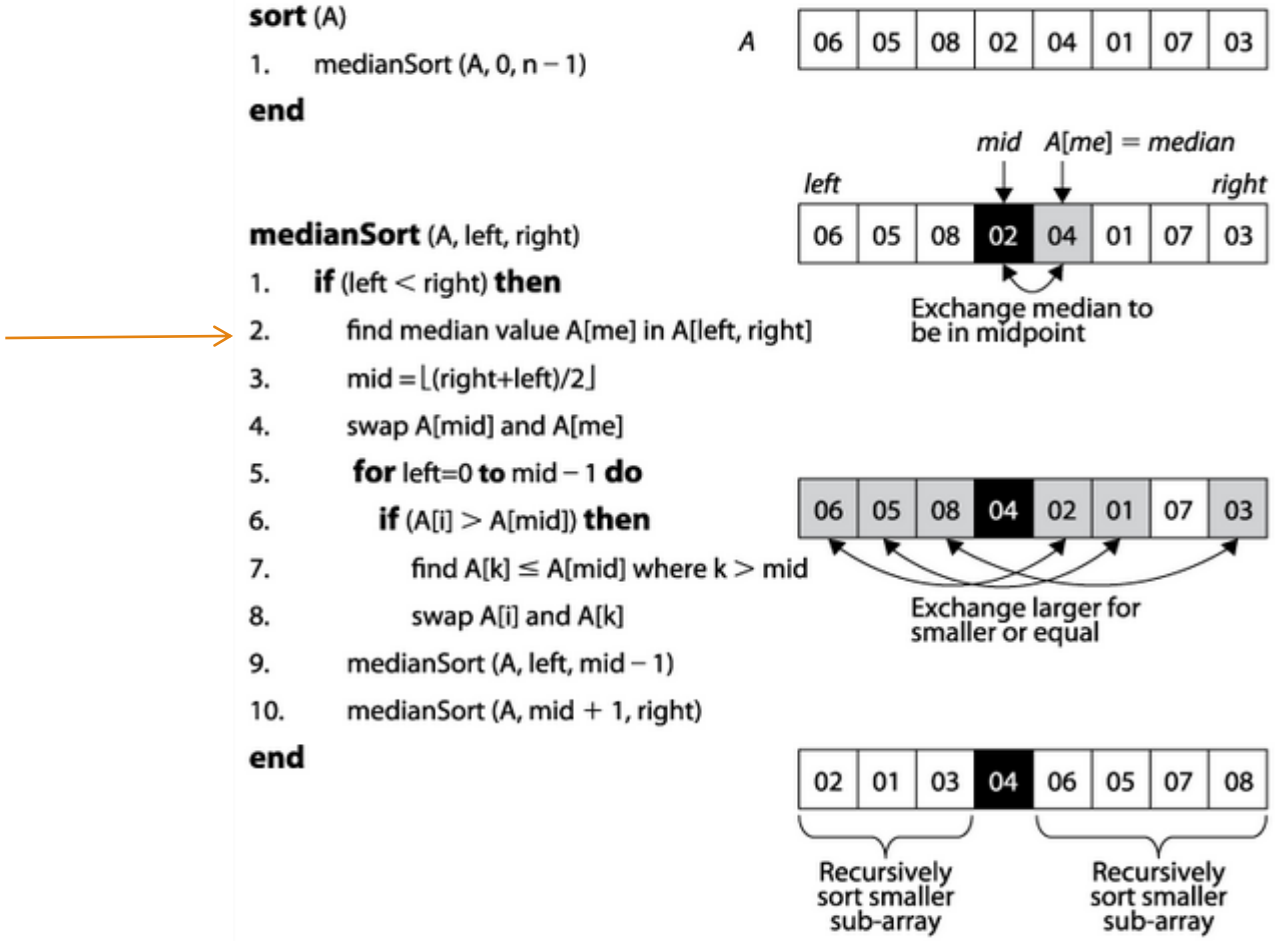
Ο αλγόριθμος ανταλλάσσει τα στοιχεία που είναι στο αριστερό μέρος και είναι μεγαλύτερα από το στοιχείο στη μέση με στοιχεία που είναι στη δεξιά πλευρά και είναι μικρότερα από το στοιχείο στη μέση

Αυτό χωρίζει τον πίνακα σε δύο υπο-πίνακες που περιλαμβάνουν περίπου τα μισά στοιχεία

Median Sort (2/9)

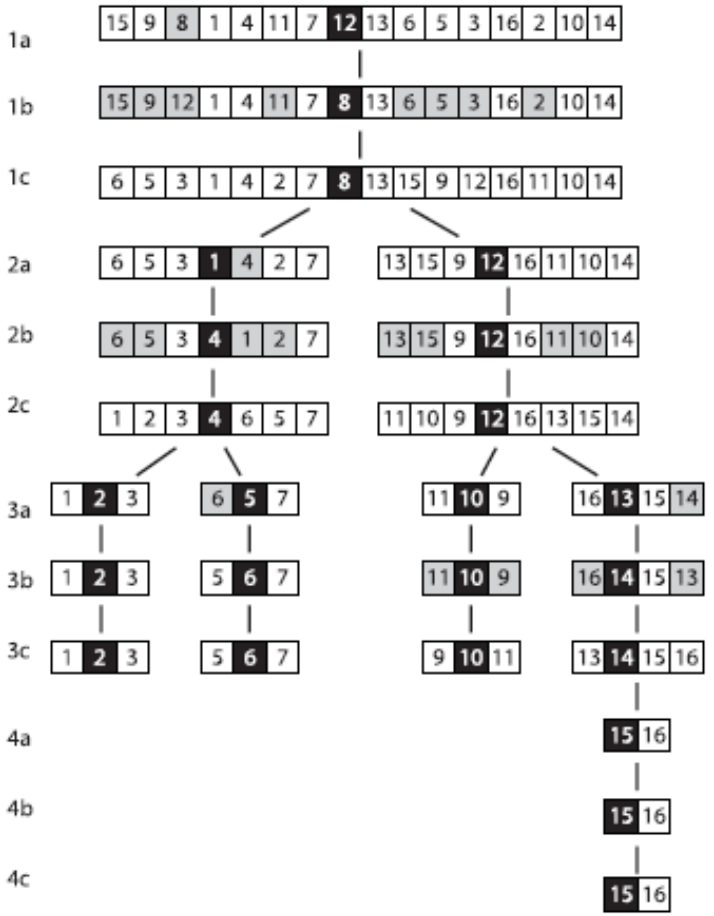
```
sort (A)
1.    medianSort (A, 0, n − 1)
end

medianSort (A, left, right)
1.    if (left < right) then
2.        find median value A[me] in A[left, right]
3.        mid = ⌊(right+left)/2⌋
4.        swap A[mid] and A[me]
5.        for left=0 to mid − 1 do
6.            if (A[i] > A[mid]) then
7.                find A[k] ≤ A[mid] where k > mid
8.                swap A[i] and A[k]
9.        medianSort (A, left, mid − 1)
10.       medianSort (A, mid + 1, right)
end
```



Median Sort (3/9)

Παράδειγμα



Median Sort (4/9)

Η επίδοση του αλγορίθμου εξαρτάται από την αποδοτική επιλογή του median σε ένα μη ταξινομημένο πίνακα

Ας υποθέσουμε ότι έχουμε στη διάθεσή μας μια συνάρτηση partition(left, right, pivotIndex)

Η συνάρτηση επιλέγει το στοιχείο A[pivotIndex] να είναι το στοιχείο που χωρίζει τον πίνακα A σε δύο τμήματα: το πρώτο περιλαμβάνει στοιχεία μικρότερα ή ίσα του pivot και το δεύτερο που περιέχει στοιχεία που είναι μεγαλύτερα ή ίσα του pivot

Ισχύει πως left <= pivotIndex <= right

Η υλοποίηση μπορεί να αναζητηθεί στο Algorithms in a Nutshell

Median Sort (5/9)

Η συνάρτηση διαχωρισμού δεν ταξινομεί τα στοιχεία, απλά επιστρέφει το index του pivot

Το pivot υιοθετείται για την εύρεση του k^{th} στοιχείου αναδρομικά στο A[left, right] για οποιοδήποτε 1<=k<=right-left+1

Αν k=pivotIndex+1

- Το pivot είναι το k στοιχείο

Αν k<pivotIndex+1

- Το k στοιχείο είναι το k στοιχείο του A[left,pivotIndex]

Αν k>pivotIndex+1

- Το k στοιχείο είναι το k-pivotIndex στοιχείο του A[pivotIndex+1, right]

Median Sort (6/9)

Για την επιλογή του k υιοθετούνται διάφορες στρατηγικές:

- Επιλογή της $1^{ης}$ ή της τελευταίας θέσης
- Επιλογή μιας τυχαίας θέσης στον A[left, right]

Αν η επιλογή του pivot δεν είναι αποδοτική, η επιλογή του k θα έχει επίδοση $O(n^2)$

Η επίδοση της καλύτερης και της μέσης περίπτωσης είναι O(n)

Median Sort (7/9)

Ανάλυση

- Στη μέση περίπτωση, ο αλγόριθμος έχει πολυπλοκότητα O(n logn)
- Στη χειρότερη περίπτωση, ο αλγόριθμος έχει πολυπλοκότητα $O(n^2)$
- Το τμήμα διαχωρισμού είναι αυτό που επιβαρύνει περισσότερο

Median Sort (8/9)

Αλγόριθμος Blum-Floyd-Pratt-Rivest-Rarjan (BFPRT)

- Επιλογή του pivot
- Ομαδοποιεί τα στοιχεία σε n/4 ομάδες των 4 στοιχείων (αγνοεί μέχρι 3 στοιχεία που δεν ταιριάζουν με τις ομάδες των τεσσάρων)
- Εντοπίζει το median σε κάθε μια από τις ομάδες – απαιτούνται πέντε συγκρίσεις
- Πολυπλοκότητα (n/4)5=1.25n ~ O(n)
- Το median είναι το τρίτο στοιχείο σε κάθε ομάδα
- Όλα τα median στοιχεία αποτελούν ένα νέο σύνολο M
- Εύρεση του median στο M
- Αναδρομική κλήση στο M

Median Sort (9/9)

Παραδείγματα εκτέλεσης

Χειρότερη Περίπτωση

n	Randomized pivot selection	Leftmost pivot selection	Blum-Floyd-Pratt-Rivest-Tarjan pivot selection
256	0.000088	0.000444	0.00017
512	0.000213	0.0024	0.000436
1,024	0.000543	0.0105	0.0011
2,048	0.0012	0.0414	0.0029
4,096	0.0032	0.19	0.0072
8,192	0.0065	0.716	0.0156
16,384	0.0069	1.882	0.0354
32,768	0.0187	9.0479	0.0388
65,536	0.0743	47.3768	0.1065
131,072	0.0981	236.629	0.361

Καλύτερη Περίπτωση

n	Randomized pivot selection	Leftmost pivot selection	Blum-Floyd-Pratt-Rivest-Tarjan pivot selection
256	0.00009	0.000116	0.000245
512	0.000197	0.000299	0.000557
1,024	0.000445	0.0012	0.0019
2,048	0.0013	0.0035	0.0041
4,096	0.0031	0.0103	0.0128
8,192	0.0082	0.0294	0.0256
16,384	0.018	0.0744	0.0547
32,768	0.0439	0.2213	0.4084
65,536	0.071	0.459	0.5186
131,072	0.149	1.8131	3.9691