Quick Sort

Quick Sort (1/30)

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 (2/30)

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 (3/30)

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 (4/30)

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

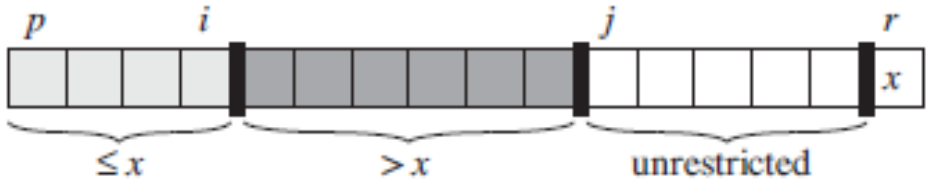
Ο αλγόριθμος διαχωρισμού έχει ως εξής:

```
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 (5/30)

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)

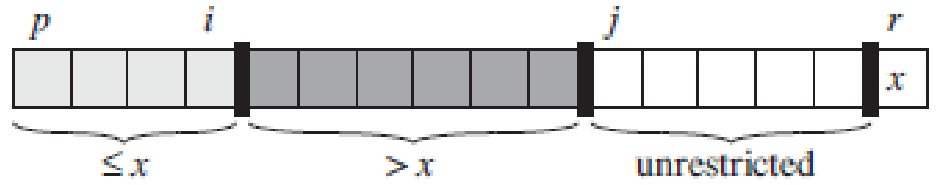
1. If $p \le k \le i$, then $A[k] \le x$.
2. If $i + 1 \le k \le j - 1$, then $A[k] > x$.
3. If $k = r$, then $A[k] = x$.

Quick Sort (6/30)

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
```



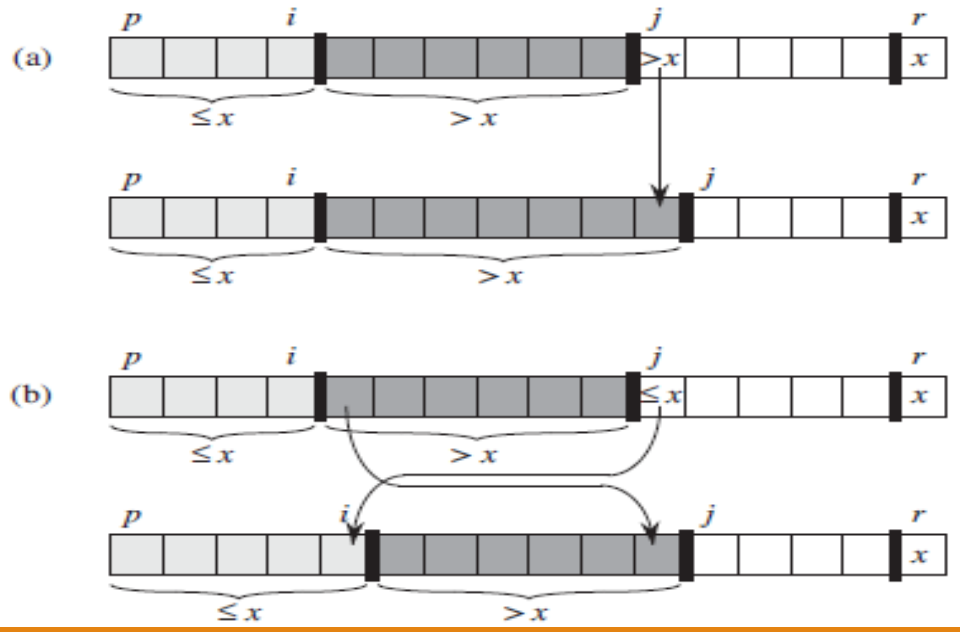
1. If $p \le k \le i$, then $A[k] \le x$.
2. If $i + 1 \le k \le j - 1$, then $A[k] > x$.
3. If $k = r$, then $A[k] = x$.

Quick Sort (7/30)

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 (8/30)

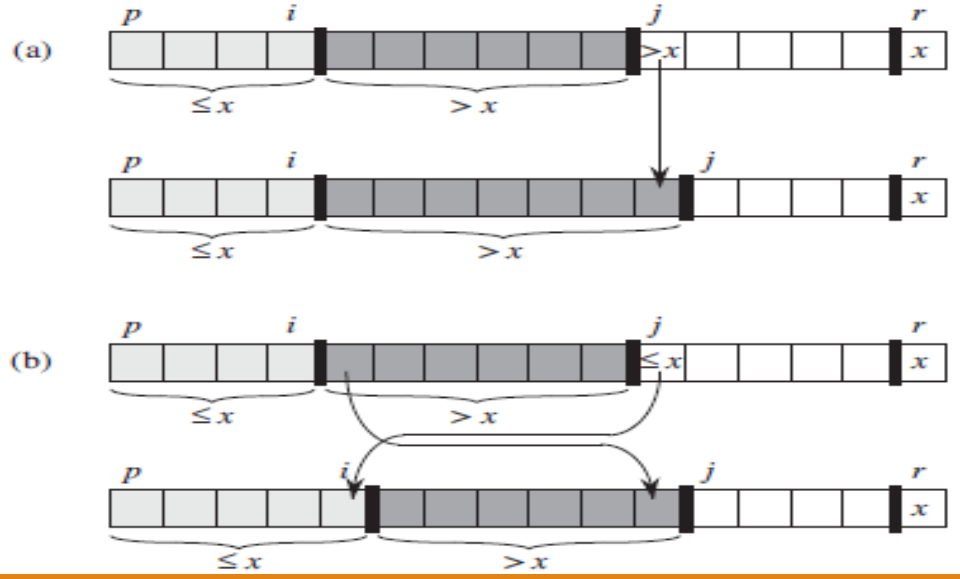
Ο τερματισμός του αλγορίθμου

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
```



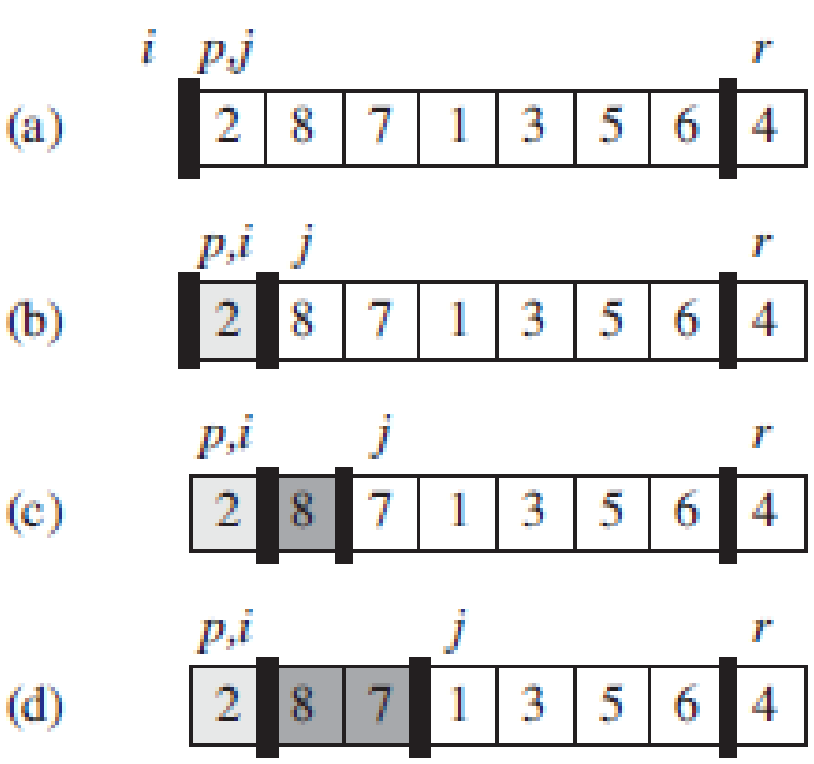
Quick Sort (9/30)

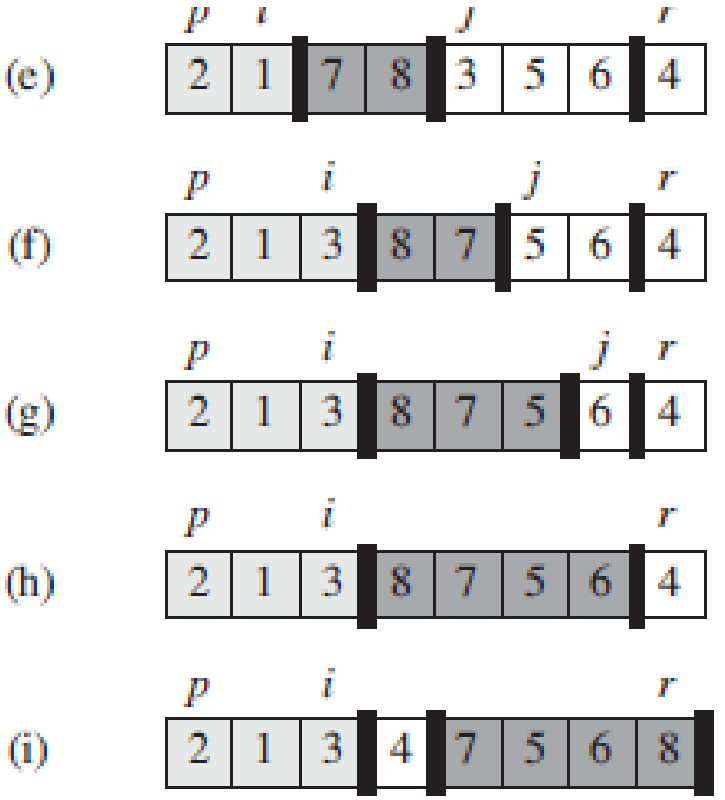
Ο χρόνος εκτέλεσης του τμήματος του διαχωρισμού έχει πολυπλοκότητα Θ(n)

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

Quick Sort (10/30)

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 (11/30)

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 (12/30)

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 (13/30)

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 (14/30)

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 (15/30)

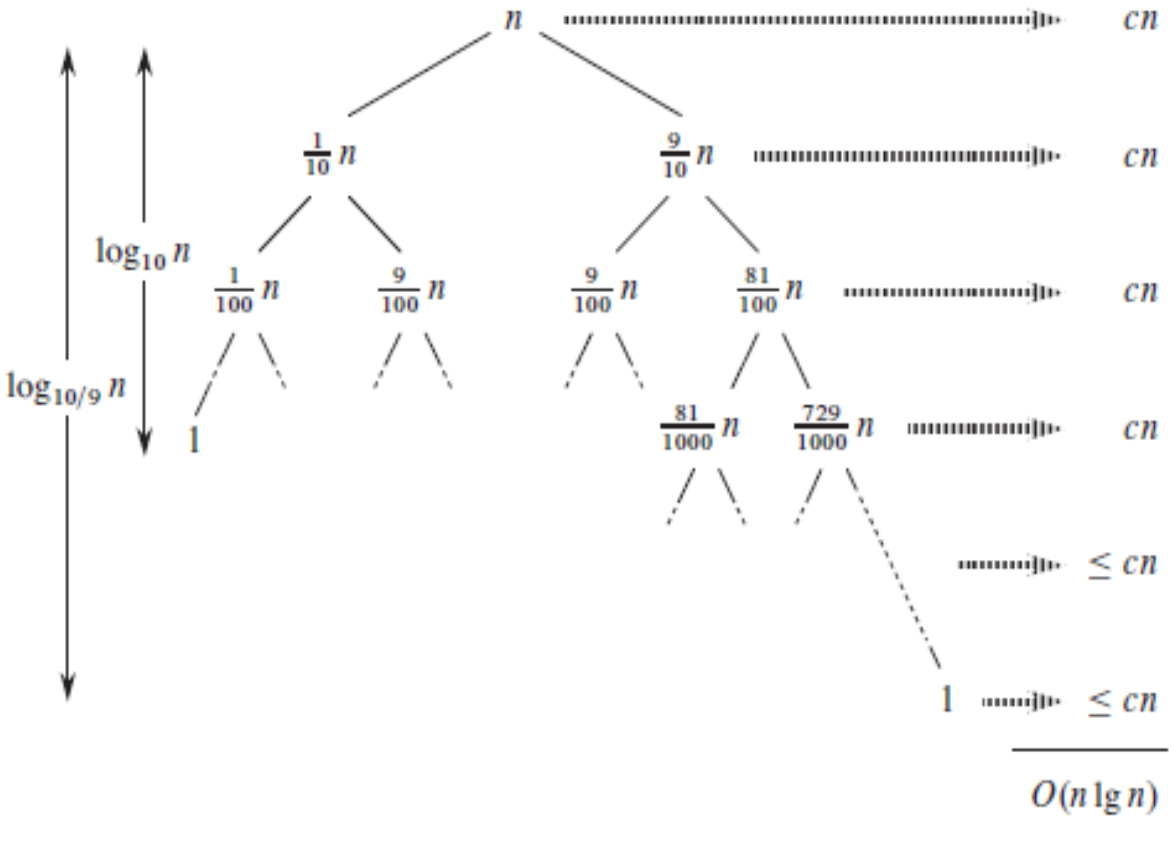
Η μέση περίπτωση είναι πιο κοντά στην καλύτερη περίπτωση παρά στη χειρότερη

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 $\log_{10}n=\Theta(\log n)$

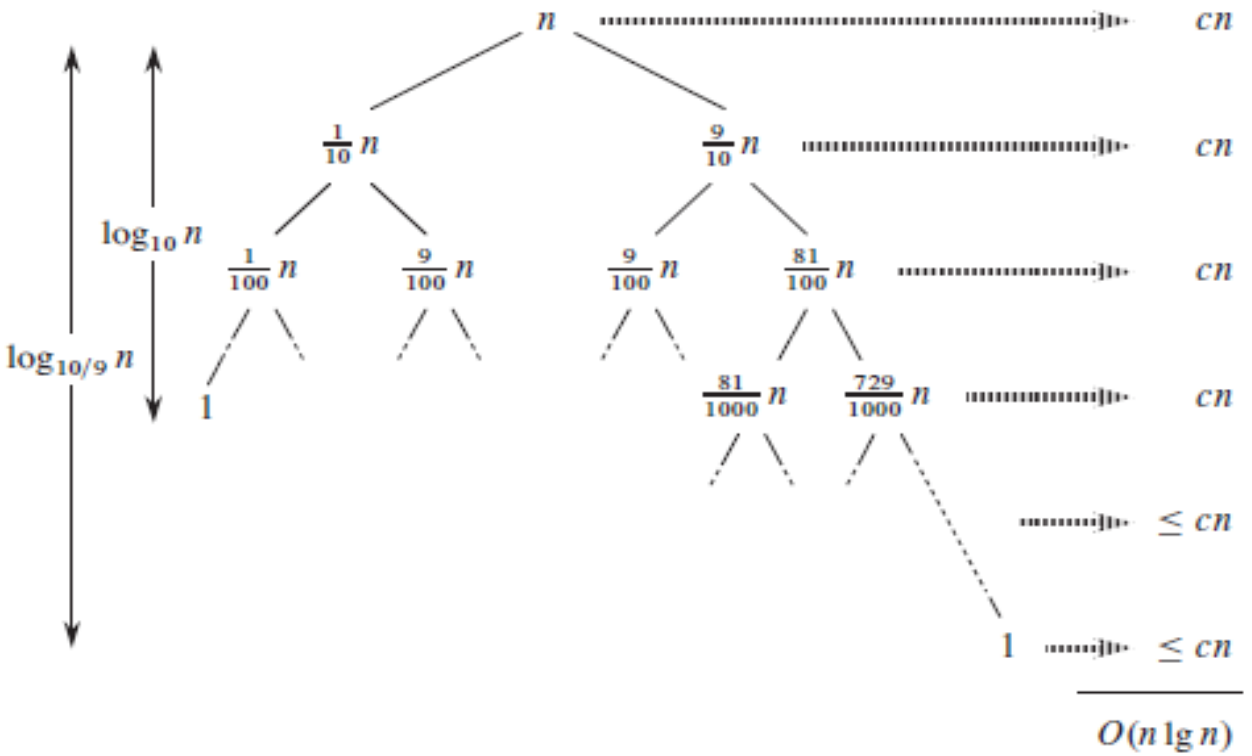


Quick Sort (16/30)

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 (17/30)

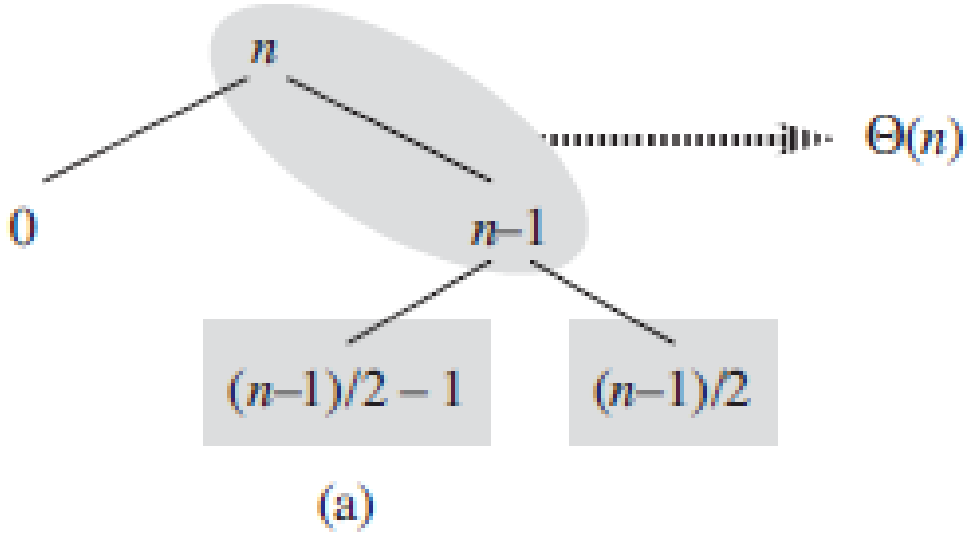
Ο αλγόριθμος εξαρτάται από την κατανομή των στοιχείων στον πίνακα και όχι από το ποια στοιχεία είναι αυτά

Αν υποθέσουμε ότι οι αναδιατάξεις είναι ισοπίθανες και τρέξουμε τον αλγόριθμο για τυχαίους πίνακες, τότε θα πάρουμε περιπτώσεις όπου οι δύο υπο-πίνακες θα είναι ισοκατανεμημένοι και άλλες όπου δεν θα έχουμε ισοκατανομή

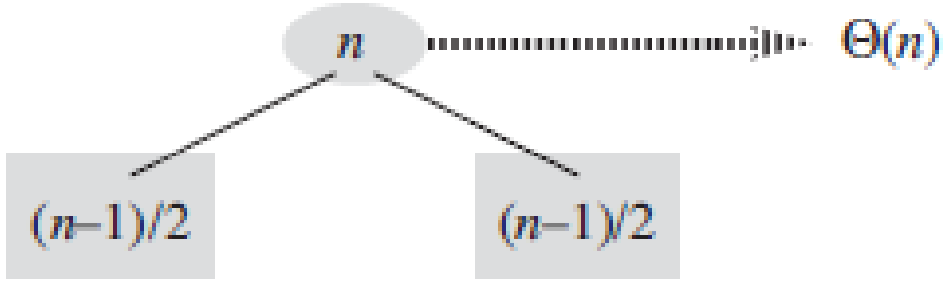
Στη μέση περίπτωση, η κατανομή των ‘καλών’ και ‘κακών’ διαχωρισμών θα είναι τυχαία ‘τοποθετημένες’ στο δένδρο διαχωρισμού

Quick Sort (18/30)

Υποθέτουμε ότι οι ‘καλοί’ διαχωρισμοί ανήκουν στην καλύτερη περίπτωση και οι ‘κακοί’ στην χειρότερη περίπτωση



(a)



(b)

Quick Sort (19/30)

Αν έχουμε συνδυασμό ‘καλών’ και ‘κακών’ διαχωρισμών π.χ. ένας κακός διαχωρισμός ακολουθείται από ένα καλό τότε θα έχουμε για τα δύο βήματα, υπο-πίνακες μεγέθους **0, (n-1)/2-1, (n-1)/2**

Σε αυτή την περίπτωση, το κόστος του διαχωρισμού θα είναι: **Θ(n) + Θ(n-1) = Θ(n) – περίπτωση (a)**

Στην περίπτωση (b), για τους δύο υπο-πίνακες μεγέθους (n-1)/2 το κόστος διαχωρισμού είναι πάλι **Θ(n)**

Quick Sort (20/30)

Randomized version

- Αντί να επιλέξουμε για pivot το A[r], επιλέγουμε ένα τυχαίο στοιχείο του A[p,r]
- Ανταλλάσσουμε το A[r] με ένα τυχαίο στοιχείο
- Με αυτό τον τρόπο θεωρούμε ισοπίθανα το A[r] με τα υπόλοιπα r-p+1 στοιχεία
- Οι αλλαγές στους αλγορίθμους έχουν ως εξής:

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

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

Quick Sort (21/30)

Γενική Ανάλυση του αλγορίθμου:

- Χειρότερη περίπτωση
  - Αναδρομική σχέση

$$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n)$$

  - Όπου το q κινείται από το 0 μέχρι το n-1
  - Υποθέτουμε ότι T(n)<= cn^2 για κάποιο c
  - Με αντικατάσταση παίρνουμε ότι

$$\begin{aligned} T(n) &\le \max_{0 \le q \le n-1} (cq^2 + c(n-q-1)^2) + \Theta(n) \\ &= c \cdot \max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) + \Theta(n) \end{aligned}$$

Quick Sort (22/30)

Η παράσταση $q^2 + (n - q - 1)^2$ έχει δεύτερη παράγωγο θετική και έτσι μεγιστοποιείται στο [0, n-1] και παίρνουμε ότι

$$\max_{0 \le q \le n-1}(q^2 + (n - q - 1)^2) \le (n - 1)^2 = n^2 - 2n + 1$$

και

$$\begin{aligned} T(n) &\le cn^2 - c(2n - 1) + \Theta(n) \\ &\le cn^2 \end{aligned}$$

Αν επιλέξουμε ένα c αρκετά μεγάλο τέτοιο ώστε ο όρος c(2n-1) υπερισχύει του Θ(n).

Έτσι: $T(n)=O(n^2)$ – κάποιες περιπτώσεις έχουν $\Omega(n^2)$ και άρα $T(n)=\Theta(n^2)$

Quick Sort (23/30)

Ο quicksort και ο randomized quicksort διαφέρουν μόνο στην επιλογή του pivot

Ο χρόνος εκτέλεσης εξαρτάται από το χρόνο που δαπανάται στο διαχωρισμό

Το πολύ n κλήσεις μπορεί να γίνουν στο τμήμα του διαχωρισμού

Κάθε κλήση απαιτεί O(1) για την ανάθεση συν το χρόνο για το loop

Σε κάθε iteration γίνεται μια σύγκριση

Πρέπει να μετρήσουμε το πλήθος των εκτελέσεων της γραμμής 4

```
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 (24/30)

Αν X είναι το πλήθος των συγκρίσεων (γραμμή 4) σε ολόκληρη την εκτέλεση του quicksort ο χρόνος εκτέλεσης θα είναι **O(n+X)**

Πρέπει να υπολογίσουμε το X βγάζοντας ένα όριο για το συνολικό αριθμό των συγκρίσεων

Έστω ότι τα στοιχεία του πίνακα είναι τα $z_1, z_2, \ldots, z_n$

και z_i είναι το i μικρότερο στοιχείο

Παίρνουμε το σύνολο $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$

των στοιχείων ανάμεσα στα z_i and z_j

Quick Sort (25/30)

Τα στοιχεία συγκρίνονται μόνο με το pivot και μετά το πέρας του τμήματος διαχωρισμού το pivot δεν ξαναελέγχεται

Υιοθετούμε την indicator function

$$X_{ij} = \mathrm{I}\{z_i \text{ is compared to } z_j\}$$

που δείχνει αν μια σύγκριση λαμβάνει χώρα στον αλγόριθμο (όχι μόνο στο τμήμα διαχωρισμού)

Αφού κάθε ζεύγος συγκρίνεται μια φορά το πλήθος των συγκρίσεων θα είναι: $X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$

Quick Sort (26/30)

Παίρνοντας την αναμενόμενη τιμή έχουμε:

$$\mathrm{E}[X] = \mathrm{E}\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathrm{E}[X_{ij}]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

Μένει να υπολογίσουμε την πιθανότητα σύγκρισης ενός ζεύγους

Quick Sort (27/30)

Αν έχουμε ως pivot το x με $z_i < x < z_j$

γνωρίζουμε ότι τα δύο στοιχεία z_i, z_j δεν πρόκειται να συγκριθούν σε επόμενες φάσεις του αλγορίθμου

Αν το z_i επιλεγεί ως pivot πριν από τα υπόλοιπα στοιχεία του Z_{ij} τότε το z_i θα συγκριθεί με τα στοιχεία του Z_{ij} εκτός από τον εαυτό του

Αν το z_j επιλεγεί ως pivot πριν από τα υπόλοιπα στοιχεία του Z_{ij} τότε το z_j θα συγκριθεί με τα στοιχεία του Z_{ij} εκτός από τον εαυτό του

Οπότε, αφού το Z_{ij} έχει j-i+1 στοιχεία και κάθε ένα από αυτά έχει την ίδια πιθανότητα να επιλεγεί ως pivot, η πιθανότητα επιλογής είναι **1/(j-i+1)**

Quick Sort (28/30)

Έτσι έχουμε:

$$
\begin{aligned}
\Pr\{z_i \text{ is compared to } z_j\} &= \Pr\{z_i \text{ or } z_j \text{ is first pivot chosen from } Z_{ij}\} \\
&= \Pr\{z_i \text{ is first pivot chosen from } Z_{ij}\} \\
&\quad + \Pr\{z_j \text{ is first pivot chosen from } Z_{ij}\} \\
&= \frac{1}{j-i+1} + \frac{1}{j-i+1} \\
&= \frac{2}{j-i+1}
\end{aligned}
$$

Quick Sort (29/30)

Άρα:

$$
\begin{aligned}
\mathrm{E}[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} && k = j - i \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\lg n) \\
&= O(n \lg n)
\end{aligned}
$$

Quick Sort (30/30)

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)

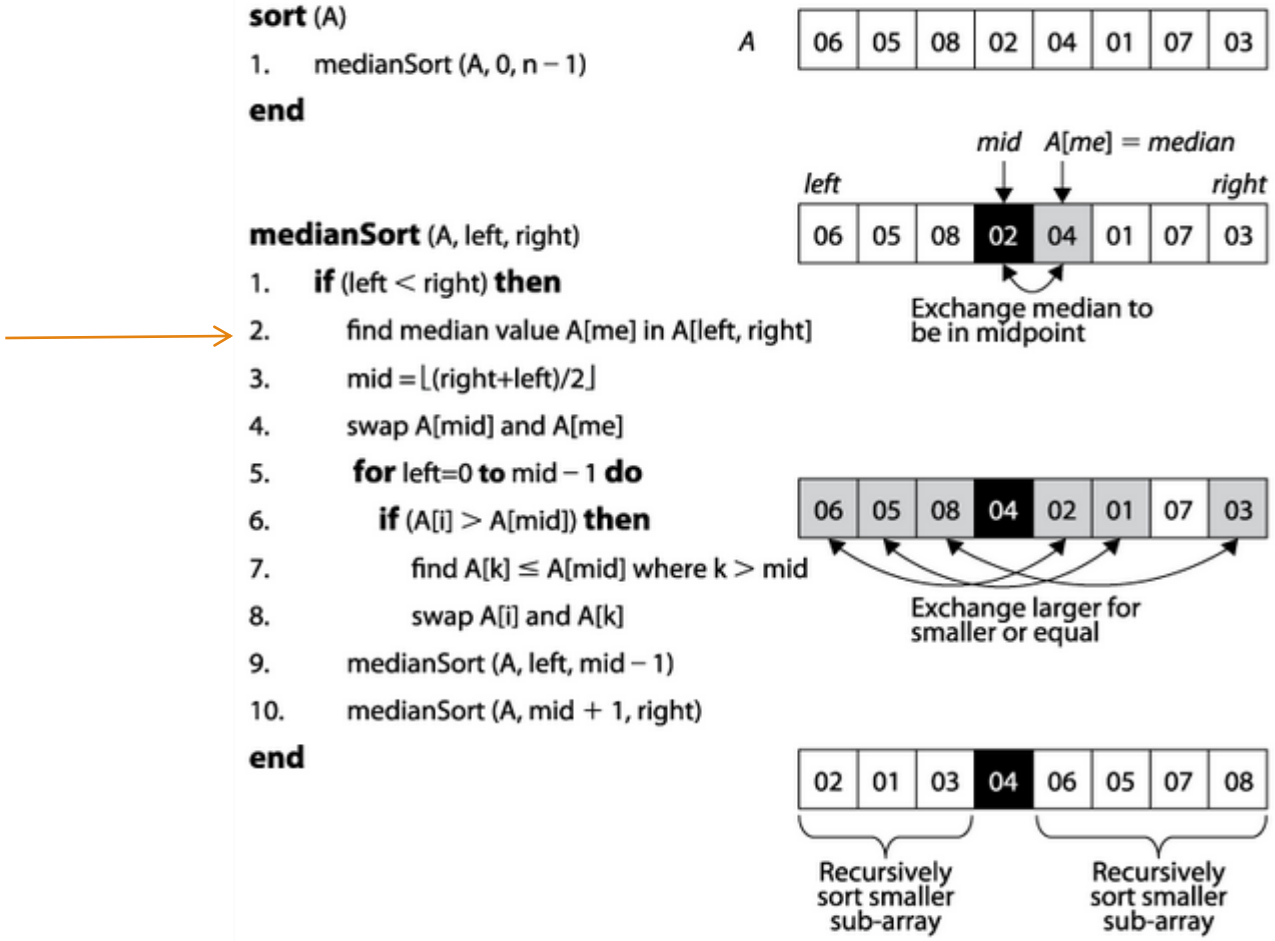
Ο αλγόριθμος ανταλλάσσει τα στοιχεία που είναι στο αριστερό μέρος και είναι μεγαλύτερα από το στοιχείο στη μέση με στοιχεία που είναι στη δεξιά πλευρά και είναι μικρότερα από το στοιχείο στη μέση

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

Median Sort (2/9)

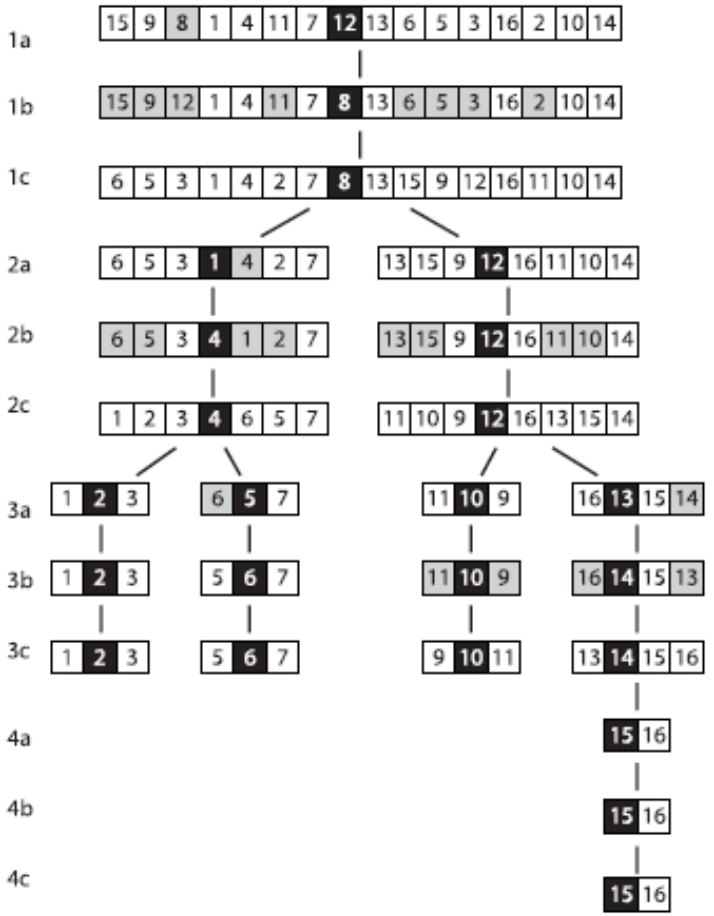
```
sort (A)
1.    medianSort (A, 0, n − 1)
end

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

Counting Sort

Counting Sort (1/6)

Ο αλγόριθμος υποθέτει ότι το κάθε ένα στοιχείο από τα n είναι ένας ακέραιος στο διάστημα [0,k] για κάποιο k

Όταν k=O(n), ο αλγόριθμος έχει πολυπλοκότητα Θ(n)

Για κάθε είσοδο x, καθορίζει το πλήθος των στοιχείων που είναι μικρότερα του x

Χρησιμοποιεί αυτή την πληροφορία για να τοποθετήσει το x απ' ευθείας στη θέση του

Για παράδειγμα, αν 15 στοιχεία είναι μικρότερα από το x, τότε το x πρέπει να μπει στη $16^{η}$ θέση

Για ένα πίνακα A[1,n], ο αλγόριθμος απαιτεί ένα πίνακα B[1,n] για το τελικό αποτέλεσμα και ένα πίνακα C[0,k] για προσωρινή αποθήκευση

Counting Sort (2/6)

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i
 7  for i = 1 to k
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

Counting Sort (3/6)

Ο πίνακας C περιέχει το πλήθος των στοιχείων που είναι ίσα με το δείκτη της κάθε θέσης

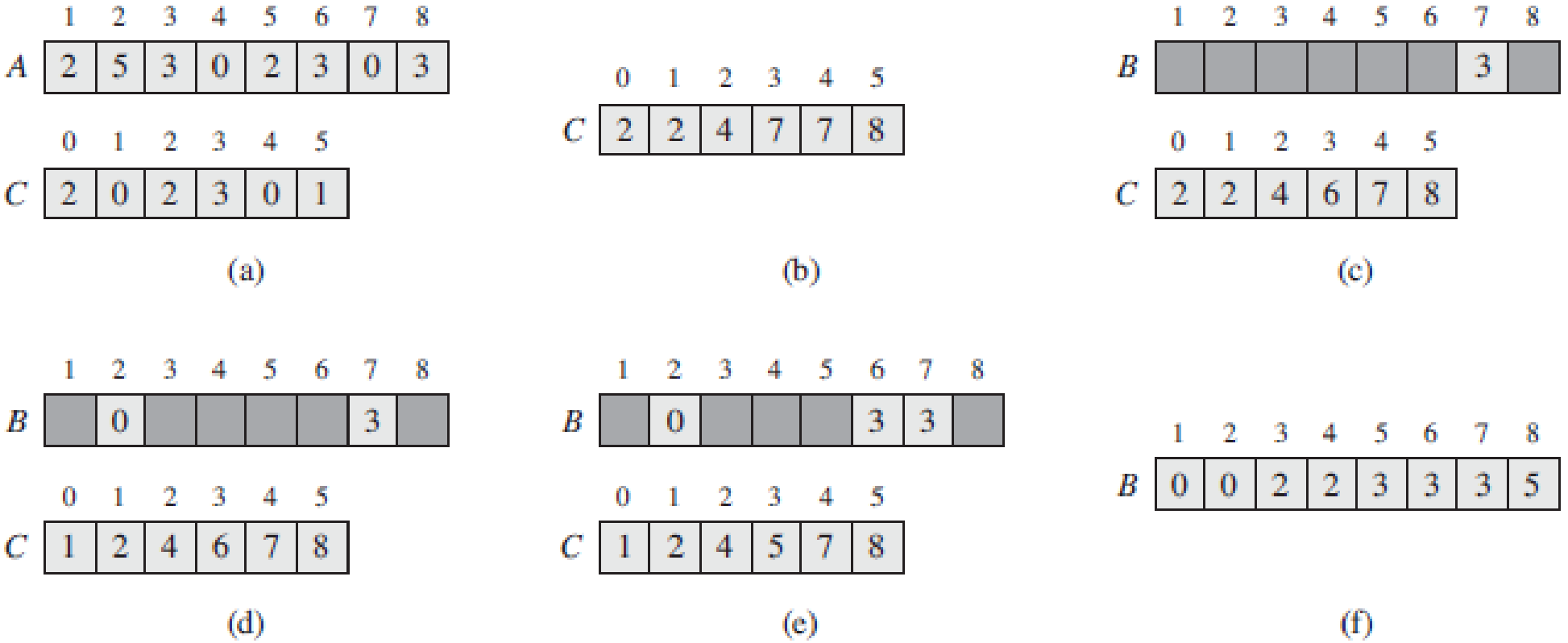
Παράδειγμα: το C[i] έχει το πλήθος των στοιχείων που είναι ίσα με i

Στις γραμμές 7-8, ο αλγόριθμος μετράει πόσα στοιχεία είναι μικρότερα ή ίσα από το i

Οι τελευταίες γραμμές του αλγορίθμου βάζουν το κάθε στοιχείο στη σωστή θέση

Counting Sort (4/6)

Παράδειγμα



Counting Sort (5/6)

Ανάλυση

- Οι γραμμές 2-3 έχουν πολυπλοκότητα Θ(k)
- Οι γραμμές 4-5 έχουν πολυπλοκότητα Θ(n)
- Οι γραμμές 7-8 έχουν πολυπλοκότητα Θ(k)
- Οι γραμμές 10-12 έχουν πολυπλοκότητα Θ(n)
- Συνολική πολυπλοκότητα: Θ(k+n)
- Αν k=O(n), τότε ο αλγόριθμος έχει πολυπλοκότητα Θ(n)
- Το μικρότερο όριο είναι το Ω(n logn)
- Ο αλγόριθμος δεν κάνει συγκρίσεις!
- Πρόκειται για ένα σταθερό αλγόριθμο αφού οι είσοδοι που είναι ίδιες εμφανίζονται με την ίδια σειρά στο τελικό αποτέλεσμα

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i
 7  for i = 1 to k
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

Counting Sort (6/6)

Demo

https://visualgo.net/en/sorting?slide=1