Αλγόριθμοι Ταξινόμησης – Μέρος 4

Μανόλης Κουμπαράκης

Μέθοδοι Ταξινόμησης Βασισμένοι σε Συγκρίσεις Κλειδιών

- Οι αλγόριθμοι ταξινόμησης που είδαμε μέχρι τώρα αποφασίζουν πώς να ταξινομήσουν μια δοσμένη ακολουθία στοιχείων βασισμένοι σε **συγκρίσεις κλειδιών**.
- Μια ερώτηση η οποία προκύπτει από την μελέτη αυτών των αλγορίθμων είναι η εξής: Μπορούμε να ταξινομήσουμε μια ακολουθία n στοιχείων σε χρόνο καλύτερο από $O(n \log n)$;

Μέθοδοι Ταξινόμησης Βασισμένοι σε Συγκρίσεις Κλειδιών

- Αν η βασική πράξη που κάνουν οι αλγόριθμοι μας είναι η σύγκριση δύο κλειδιών τότε η πολυπλοκότητα χρόνου $O(n \log n)$ είναι η καλύτερη που μπορούμε να πετύχουμε.

Ένα Κάτω Φράγμα για τους Αλγόριθμους Ταξινόμησης

- Υποθέστε ότι θέλουμε να ταξινομήσουμε μια ακολουθία $S = (x_0, x_1, \dots, x_n)$.
- Υποθέτουμε ότι όλα τα στοιχεία της S είναι διαφορετικά μεταξύ τους (αυτό δεν είναι περιορισμός επειδή αποδεικνύουμε ένα κάτω φράγμα).
- Δεν μας ενδιαφέρει αν η S υλοποιείται από ένα πίνακα ή μια συνδεδεμένη λίστα επειδή, για την απόδειξη του κάτω φράγματος, μετράμε μόνο συγκρίσεις κλειδιών.

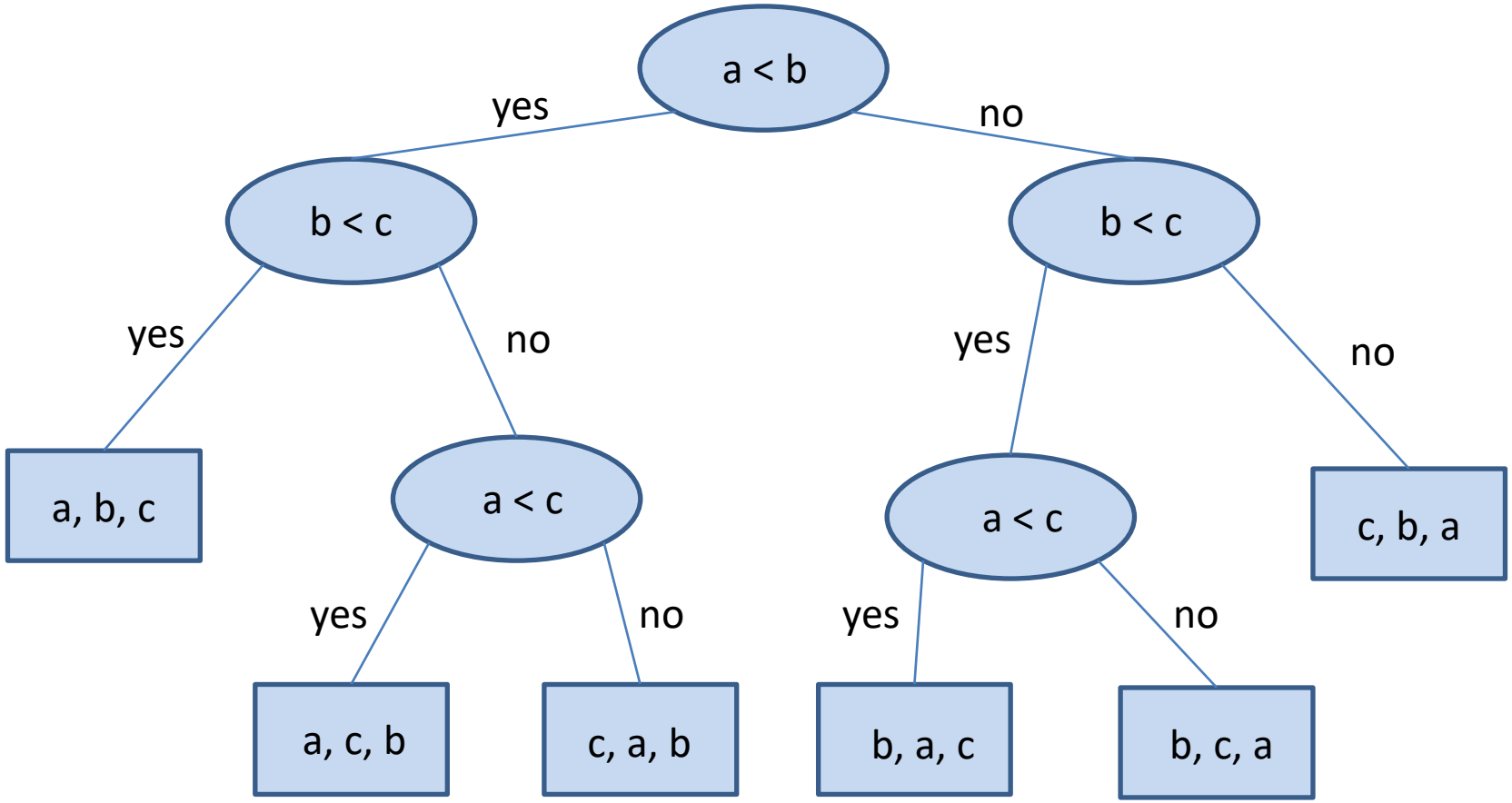
Δένδρα Αποφάσεων

- Κάθε φορά που ένας αλγόριθμος ταξινόμησης συγκρίνει δύο κλειδιά x_i και x_j, δηλαδή ρωτάει «Είναι το x_i μικρότερο του x_j;», υπάρχουν δύο πιθανές απαντήσεις: «ναι» και «όχι».
- Βασιζόμενος στον αποτέλεσμα αυτής της σύγκρισης, ο αλγόριθμος ταξινόμησης μπορεί να εκτελέσει κάποιους εσωτερικούς υπολογισμούς (τους οποίους δεν λαμβάνουμε υπόψη μας εδώ) και μετά να εκτελέσει άλλη μία σύγκριση κλειδιών που έχει πάλι δύο απαντήσεις.
- Άρα μπορούμε να παραστήσουμε ένα αλγόριθμο ταξινόμησης βασισμένο σε συγκρίσεις κλειδιών με ένα **δένδρο αποφάσεων (decision tree).**

Δένδρα Αποφάσεων

- Ένα **δένδρο αποφάσεων** T είναι ένα δυαδικό δένδρο το οποίο σε κάθε εσωτερικό του κόμβο περιέχει μια σύγκριση ανάμεσα σε δύο κλειδιά, και οι ακμές από ένα εσωτερικό κόμβο v στα παιδιά του αντιστοιχούν στους υπολογισμούς που προκύπτουν από τις απαντήσεις «ναι» και «όχι».

Παράδειγμα Δένδρου Αποφάσεων



Δένδρα Αποφάσεων

- Κάθε πιθανή αρχική **αντιμετάθεση (permutation)** των στοιχείων της S κάνει τον υποθετικό αλγόριθμο ταξινόμησης να εκτελέσει μια ακολουθία συγκρίσεων που αντιστοιχεί σε ένα μονοπάτι στο T, από τη ρίζα σε κάποιο εξωτερικό κόμβο.
- Κάθε εξωτερικός κόμβος του T αντιστοιχεί στην ακολουθία συγκρίσεων για το πολύ μια αντιμετάθεση των στοιχείων της S (αλλιώς ο αλγόριθμος δεν είναι σωστός).
- Ο χειρότερος χρόνος εκτέλεσης του αλγόριθμου αντιστοιχεί στο **ύψος** του δένδρου.
- Το **μέγεθος** και το **σχήμα** του δένδρου είναι συνάρτηση του αλγόριθμου και του πλήθους των στοιχείων που ταξινομούνται.

Πρόταση

- Ο χρόνος εκτέλεσης οποιουδήποτε αλγόριθμου ταξινόμησης που βασίζεται σε συγκρίσεις κλειδιών είναι $\Omega(n \log n)$ στη χειρότερη περίπτωση.
- Απόδειξη;

Ο Συμβολισμός $\Omega(.)$

- Όπως ο συμβολισμός $O(.)$ μας παρέχει ένα τρόπο για να πούμε ότι μια συνάρτηση είναι μικρότερη ή ίση με κάποια άλλη, ο παρακάτω συμβολισμός $\Omega(.)$ μας παρέχει ένα τρόπο για να πούμε ότι μια συνάρτηση είναι **μεγαλύτερη ή ίση** με κάποια άλλη.
- **Ορισμός**. Λέμε ότι η συνάρτηση $f(n)$ είναι $\Omega\big(g(n)\big)$ αν η $g(n)$ είναι $O(f(n))$, ή αλλιώς, αν υπάρχει μια πραγματική σταθερά $c > 0$ και μια ακέραια σταθερά $n_0 \geq 1$ που είναι τέτοιες ώστε $f(n) \geq cg(n)$, για $n \geq n_0$.

Παράδειγμα

- Η συνάρτηση $3n \log n + 2n$ είναι $\Omega(n \log n)$.
- Αυτό είναι εύκολο να αποδειχθεί επειδή $3n \log n + 2n \geq 3n \log n$ για κάθε $n \geq 2$.

Απόδειξη του Κάτω Φράγματος

- Στη χειρότερη περίπτωση, ο χρόνος εκτέλεσης ενός αλγόριθμου ταξινόμησης βασισμένου σε συγκρίσεις κλειδιών είναι μεγαλύτερος ή ίσος με το ύψος του δένδρου απόφασης T που αντιστοιχεί στο αλγόριθμο αυτό όπως είπαμε παραπάνω.
- Κάθε εξωτερικός κόμβος του T αντιστοιχεί σε μια αντιμετάθεση των στοιχείων της S. Επιπλέον, κάθε αντιμετάθεση των στοιχείων της S αντιστοιχεί σε ένα εξωτερικό κόμβο του T.
- Ο αριθμός των αντιμεταθέσεων n στοιχείων είναι $n! = n(n-1)(n-2)\cdots 2 \cdot 1$.
- Άρα το δένδρο T πρέπει να έχει $n!$ εξωτερικούς κόμβους.

Απόδειξη του Κάτω Φράγματος

- Από τις ιδιότητες των δυαδικών δένδρων που έχουμε αποδείξει στο παρελθόν προκύπτει ότι το ύψος του T είναι τουλάχιστον $\log(n!)$.
- Επειδή στο γινόμενο $n!$ υπάρχουν τουλάχιστον $n/2$ όροι που είναι μεγαλύτεροι ή ίσοι με $n/2$ έχουμε

$$\log(n!) \geq \log\left(\frac{n}{2}\right)^{\frac{n}{2}} = \frac{n}{2}\log\frac{n}{2} = \frac{n}{2}\log n - \frac{n}{2} \geq \frac{n}{2}\log n - \frac{n}{4}\log n = \frac{1}{4}n\log n \text{ για } n \geq n_0.$$

- Δηλαδή η $\log(n!)$ είναι $\Omega(n\log n)$.

Ταξινόμηση σε Γραμμικό Χρόνο

- Έχουμε δείξει ότι χρειαζόμαστε $\Omega(n \log n)$ χρόνο στη χειρότερη περίπτωση για να ταξινομήσουμε ένα πίνακα μεγέθους n χρησιμοποιώντας αλγορίθμους που βασίζονται στη σύγκριση κλειδιών.
- Έχουμε παρουσιάσει αλγόριθμους που είναι **βέλτιστοι** δηλαδή έχουν πολυπλοκότητα $O(n \log n)$ στη χειρότερη περίπτωση (π.χ. τον heapsort ή τον mergesort).
- **Ερώτηση**: Υπάρχουν καλύτεροι αλγόριθμοι πού δεν βασίζονται στη σύγκριση κλειδιών;

Ταξινόμηση σε Γραμμικό Χρόνο

- Υπάρχουν αλγόριθμοι ταξινόμησης που τρέχουν σε **γραμμικό χρόνο** αλλά είναι κατάλληλοι μόνο για πίνακες με συγκεκριμένους τύπους στοιχείων.

- Θα παρουσιάσουμε τους αλγόριθμους bucket-sort και radix-sort.

Ο Αλγόριθμος bucket-sort

- Θεωρείστε ένα πίνακα S μεγέθους n του οποίου τα κλειδιά είναι ακέραιοι στο διάστημα $[0, N-1]$ για κάποιο ακέραιο $N \geq 2$.
- Η βασική ιδέα του αλγόριθμου **bucket-sort** είναι να χρησιμοποιήσουμε τα κλειδιά σαν δείκτες σε ένα **πίνακα κάδων (buckets)** B που έχει στοιχεία $B[0], \dots, B[N-1]$.
- Ο αλγόριθμος λειτουργεί ως εξής.
- Στην αρχή διασχίζουμε τον πίνακα S και τοποθετούμε τα στοιχεία του στον πίνακα κάδων B. Το στοιχείο του πίνακα S με κλειδί k τοποθετείται στον κάδο $B[k]$ πού είναι μια ακολουθία από στοιχεία του S που έχουν κλειδί k.
- Στη συνέχεια βάζουμε τα στοιχεία του B πίσω στον S σε ταξινομημένη σειρά, απαριθμώντας τα περιεχόμενα των κάδων $B[0], \dots, B[N-1]$.

Ο Αλγόριθμος bucket-sort

- Είναι εύκολο να δούμε ότι ο αλγόριθμος bucket-sort τρέχει σε $O(n + N)$ χρόνο και χρειάζεται $O(n + N)$ χώρο.
- Επομένως ο αλγόριθμος bucket-sort είναι αποδοτικός όταν η παράμετρος N είναι μικρή σε σχέση με το n π.χ. $N = O(n)$ ή $N = O(n \log n)$. Η απόδοση του αλγόριθμου χειροτερεύει όταν το N μεγαλώνει σε σχέση με το n.

Ο Αλγόριθμος bucket-sort

- Μια σημαντική ιδιότητα του αλγόριθμου bucket-sort είναι ότι λειτουργεί σωστά ακόμα και αν υπάρχουν πολλά στοιχεία με την ίδια τιμή κλειδιού.
- Μπορούμε να κρατήσουμε ένα μετρητή για κάθε τιμή του κλειδιού ή να υλοποιήσουμε τους κάδους σαν συνδεδεμένες λίστες κλειδιών.

Ευσταθείς Μέθοδοι Ταξινόμησης

- Μια μέθοδος ταξινόμησης λέγεται ευσταθής (stable) αν διατηρεί τη σχετική σειρά των στοιχείων μιας ακολουθίας με διπλά κλειδιά.
- **Ορισμός**. Έστω $S = ((x_0, k_0), \dots, (x_{n-1}, k_{n-1}))$ μια ακολουθία ζευγαριών προς ταξινόμηση. Θα λέμε ότι ένας αλγόριθμος ταξινόμησης είναι **ευσταθής** αν, για κάθε δύο στοιχεία (x_i, k_i) και (x_j, k_j) της S τέτοια ώστε $k_i = k_j$ και το (x_i, k_i) προηγείται του (x_j, k_j) στην S πριν την ταξινόμηση (δηλαδή, $i < j$), τότε το (x_i, k_i) προηγείται του (x_j, k_j) στην S και μετά την ταξινόμηση.

Παράδειγμα

- Θεωρήστε τις παρακάτω εγγραφές που περιέχουν επώνυμα φοιτητών και έτος φοίτησης στο πανεπιστήμιο. Υποθέστε ότι αρχικά ταξινομούνται ως προς το επώνυμο:

Επώνυμο	Έτος φοίτησης
Adams	1
Black	2
Brown	4
Jackson	2
Jones	4
Smith	1
Thompson	4
Washington	2
White	3
Wilson	3

Παράδειγμα

- Μια **ασταθής ταξινόμηση** ως προς το έτος φοίτησης είναι η παρακάτω:

Επώνυμο	Έτος φοίτησης
Adams	1
Smith	1
Washington	2
Jackson	2
Black	2
White	3
Wilson	3
Thompson	4
Brown	4
Jones	4

Παράδειγμα

- Μια **ευσταθής ταξινόμηση** ως προς το έτος φοίτησης είναι η παρακάτω:

Επώνυμο	Έτος φοίτησης
Adams	1
Smith	1
Black	2
Jackson	2
Washington	2
White	3
Wilson	3
Brown	4
Jones	4
Thompson	4

Ερωτήσεις

- Ποιοι από τους αλγόριθμους ταξινόμησης που έχουμε παρουσιάσει είναι ευσταθείς;
- Είναι ο αλγόριθμος bucket-sort ευσταθής;

Bucket-sort

- Ο αλγόριθμος bucket-sort είναι ευσταθής αν κάθε φορά που βάζουμε ένα στοιχείο στην S ή σε ένα κάδο $B[i]$, το βάζουμε στο τέλος της ακολουθίας, και κάθε φορά που βγάζουμε ένα στοιχείο αυτό είναι το πρώτο στοιχείο της ακολουθίας.

Λεξικογραφική Διάταξη

- Ας υποθέσουμε ότι θέλουμε να ταξινομήσουμε ακολουθίες στοιχείων με κλειδιά που είναι ζευγάρια (k, l), όπου k και l είναι ακέραιοι στο διάστημα $[0, N-1]$ για κάποιο ακέραιο $N \geq 2$.
- Η **λεξικογραφική διάταξη** αυτών των κλειδιών ορίζεται ως εξής: $(k_1, l_1) < (k_2, l_2)$ αν $k_1 < k_2$ ή $k_1 = k_2$ και $l_1 < l_2$.
- Η έννοια της λεξικογραφικής διάταξης γενικεύεται εύκολα για κλειδιά που είναι πλειάδες d στοιχείων για $d > 2$.

Ο Αλγόριθμος Ταξινόμησης Βάσης

- Ο αλγόριθμος **ταξινόμησης βάσης (radix-sort)** ταξινομεί ακολουθίες στοιχείων με κλειδιά που είναι ζευγάρια (ή γενικότερα πλειάδες).
- Ο αλγόριθμος ταξινομεί μια δοσμένη ακολουθία S σε **δύο περάσματα** χρησιμοποιώντας την ευσταθή έκδοση του bucket-sort: Πρώτα την ταξινομεί ως προς τη δεύτερη συνιστώσα του κλειδιού και μετά ως προς τη πρώτη.
- Είναι εύκολο να δούμε ότι αν κάνουμε τις δύο ταξινομήσεις με αντίστροφη σειρά, το αποτέλεσμα δεν θα είναι σωστό.

Παράδειγμα

- Θεωρήστε την παρακάτω ακολουθία (δείχνουμε μόνο τα κλειδιά):
$$S = ((3,3), (1,5), (2,5), (1,2), (2,3), (1,7), (3,2), (2,2))$$

- Αν ταξινομήσουμε την S με την ευσταθή έκδοση του bucket-sort ως προς την πρώτη συνιστώσα του κλειδιού, τότε παίρνουμε την ακολουθία
$$S_1 = \big((1,5), (1,2), (1,7), (2,5), (2,3), (2,2), (3,3), (3,2)\big).$$

- Αν ταξινομήσουμε την S_1 με την ευσταθή έκδοση του bucket-sort ως προς την δεύτερη συνιστώσα του κλειδιού, τότε παίρνουμε την ακολουθία
$$\mathrm{S}_{1,2} = \big((1,2), (2,2), (3,2), (2,3), (3,3), (1,5), (2,5), (1,7)\big).$$

- Παρατηρήστε ότι αυτή η ακολουθία δεν είναι ταξινομημένη λεξικογραφικά.

Παράδειγμα

- Αν ταξινομήσουμε την S πρώτα ως προς την δεύτερη συνιστώσα του κλειδιού παίρνουμε την παρακάτω ακολουθία:

  $S_2 = \big((1,2), (3,2), (2,2), (3,3), (2,3), (1,5), (2,5), (1,7)\big)$

- Αν τώρα ταξινομήσουμε την S_2 ως προς την πρώτη συνιστώσα του κλειδιού παίρνουμε την παρακάτω ακολουθία:

  $S_{2,1} = ((1,2), (1,5), (1,7), (2,2), (2,3), (2,5), (3,2), (3,3))$

- Παρατηρήστε ότι η παραπάνω ακολουθία είναι λεξικογραφικά ταξινομημένη.

Ο Αλγόριθμος Ταξινόμησης Βάσης

- Η μέθοδος αυτή μπορεί να επεκταθεί και στην περίπτωση που έχουμε d-πλειάδες.
- Ο αλγόριθμος ταξινόμησης βάσης είναι χρήσιμος αν κάθε στοιχείο της ακολουθίας που ταξινομούμε μπορεί να θεωρηθεί ότι αποτελείται από μια πλειάδα ψηφίων (ή συμβόλων ή γραμμάτων).
- **Παράδειγμα**: Μπορούμε να αναπαραστήσουμε κάθε ακέραιο αριθμό ανάμεσα στο 0 και το 99 ως μια πλειάδα δεκαδικών ψηφίων. Σ' αυτή την περίπτωση ο αλγόριθμος ταξινόμησης βάσης ταξινομεί πρώτα ως προς το **λιγότερο σημαντικό** στοιχείο και μετά ως προς το περισσότερο σημαντικό.

Πρόταση

- Έστω S μία ακολουθία από n ζευγάρια κλειδί-τιμή, κάθε ένα από τα οποία έχει κλειδί της μορφής $(k_1, k_2, \dots, k_d)$, όπου k_i είναι ένας ακέραιος στο διάστημα $[0, N-1]$ για κάποιο $N \geq 2$. Μπορούμε να ταξινομήσουμε την S λεξικογραφικά σε $O\big(d(n+N)\big)$ χρόνο χρησιμοποιώντας τον αλγόριθμο ταξινόμησης βάσης.

Αλγόριθμοι Ταξινόμησης: Σύνοψη

- Αν θέλουμε να ταξινομήσουμε μικρούς πίνακες (π.χ., με 100 στοιχεία) οι **στοιχειώδεις αλγόριθμοι** που παρουσιάσαμε είναι κατάλληλοι.
- Αν ο δοσμένος πίνακας είναι σχεδόν ταξινομημένος, η ταξινόμηση με εισαγωγή είναι η προτιμητέα.
- Αν τα στοιχεία του πίνακα είναι μεγάλα και η αντιμετάθεση τους κοστίζει, τότε η ταξινόμηση με επιλογή είναι η κατάλληλη.

Αλγόριθμοι Ταξινόμησης: Σύνοψη

- Ο αλγόριθμος ταξινόμησης με συγχώνευση είναι κατάλληλος για περιπτώσεις που τα δεδομένα μας δεν χωράνε στην κύρια μνήμη και είναι αποθηκευμένα στο δίσκο.
- Για δεδομένα που χωράνε στην κύρια μνήμη, ο αλγόριθμοι `quicksort` και `heapsort` είναι προτιμότεροι του `mergesort` επειδή ταξινομούν τα δεδομένα επιτόπου.
- Μεταξύ των `quicksort` και `heapsort`, πειραματικές μελέτες έχουν δείξει ότι υπερτερεί ο `quicksort` τις περισσότερες φορές. Οπότε ο `quicksort` είναι ένας εξαιρετικός αλγόριθμος γενικής χρήσης για δεδομένα που βρίσκονται στην κύρια μνήμη. Γι αυτό χρησιμοποιείται από την συνάρτηση βιβλιοθήκης `qsort` της C.

Αλγόριθμοι Ταξινόμησης: Σύνοψη

- Όμως η $O(n^2)$ χρονική πολυπλοκότητα χειρίστης περίπτωσης του `quicksort` είναι απαγορευτική για εφαρμογές πραγματικού χρόνου όπου πρέπει να εγγυηθούμε τον χρόνο ολοκλήρωσης μιας ταξινόμησης.
- Σ' αυτές τις περιπτώσεις καταλληλότερος αλγόριθμος είναι ο `heapsort` επειδή έχει χρονική πολυπλοκότητα χειρίστης περίπτωσης $O(n \log n)$ και ταξινομεί επιτόπου.

Αλγόριθμοι Ταξινόμησης: Σύνοψη

- Αν θέλουμε να ταξινομήσουμε πίνακες με μικρά ακέραια κλειδιά ή πλειάδες κλειδιών, τότε οι αλγόριθμοι `bucket-sort` και `radix-sort` είναι οι πιο κατάλληλοι.
- Μάλιστα αν η παράσταση $d(n+N)$ είναι σημαντικά μικρότερη της $n \log n$ τότε οι αλγόριθμοι αυτοί είναι προτιμότεροι του `quicksort` ή του `heapsort`.

Μελέτη

- M.T. Goodrich, R. Tamassia and D. Mount. *Data Structures and Algorithms in C++*. 2nd edition, 2011.
  - Sections 11.3.1, 11.3.2 and 11.3.3
- M.T. Goodrich, R. Tamassia. *Δομές Δεδομένων και Αλγόριθμοι σε Java*. 5η έκδοση. Εκδόσεις Δίαυλος.
  - Κεφ. 11.3.1, 11.3.2 και 11.3.3