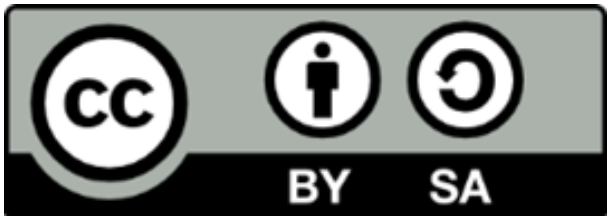
ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

Ενότητα 8α: Ταξινόμηση-Σύγκριση αλγορίθμων ταξινόμησης

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.

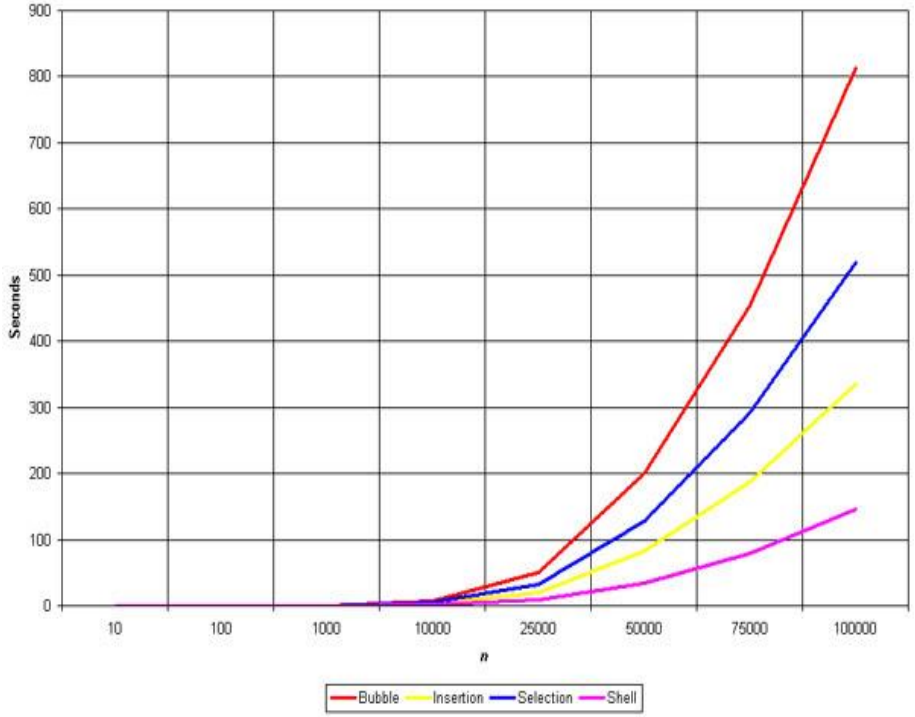
Ανάλυση Αλγορίθμων Ταξινόμησης

- Όπως είπαμε η πολυπλοκότητα είναι ένα χαρακτηριστικό στοιχείο ενός αλγόριθμου. Έχουμε χρονική πολυπλοκότητα (time complexity) και πολυπλοκότητα ανάγκης σε μνήμη (space complexity)
- Η κατηγοριοποίηση της πολυπλοκότητας (χρονικής ή απαραίτητης μνήμης) σε απλές μορφές (από την καλύτερη προς τη χειρότερη) είναι:
  - Θ(1): Σταθερή (**constant**) χρονική (ή μνήμης) πολυπλοκότητα. Ο αλγόριθμος παίρνει τον ίδιο χρόνο (ή μνήμη) να παράγει αποτέλεσμα ανεξάρτητα από τον όγκο των δεδομένων που επεξεργάζεται
  - Θ(log(*n*)): Λογαριθμική (logarithmic) χρονική (ή μνήμης) πολυπλοκότητα. Ο χρόνος εκτέλεσης (ή η ανάγκες σε μνήμη) έχει λογαριθμική σχέση με τον όγκο των δεδομένων
  - $\Theta(\log^2(n))$: Λογαριθμική (logarithmic) χρονική (ή μνήμης) πολυπλοκότητα. Ο χρόνος εκτέλεσης (ή η ανάγκες σε μνήμη) έχει τετραγωνική λογαριθμική σχέση με τον όγκο των δεδομένων
  - Θ(n): Γραμμική (linear) χρονική (ή μνήμης) πολυπλοκότητα. Ο χρόνος εκτέλεσης (ή η ανάγκες σε μνήμη) έχει γραμμική σχέση με τον όγκο των δεδομένων
  - Θ(n log(*n*)): Γραμμικά-λογαριθμική (n log(n)) χρονική (ή μνήμης) πολυπλοκότητα. Ο χρόνος εκτέλεσης (ή η ανάγκες σε μνήμη) έχει σχέση με τον όγκο των δεδομένων που δίδεται από τη συνάρτηση n log(n)
  - $\Theta(n^2)$: Τετραγωνική (quadratic) χρονική (ή μνήμης) πολυπλοκότητα. Ο χρόνος εκτέλεσης (ή η ανάγκες σε μνήμη) έχει τετραγωνική σχέση με τον όγκο των δεδομένων
  - $\Theta(n^3)$: Κυβική (cubic) χρονική (ή μνήμης) πολυπλοκότητα. Ο χρόνος εκτέλεσης (ή η ανάγκες σε μνήμη) έχει κυβική σχέση με τον όγκο των δεδομένων
  - $\Theta(2^n)$: Εκθετική (exponential) χρονική (ή μνήμης) πολυπλοκότητα. Ο χρόνος εκτέλεσης (ή η ανάγκες σε μνήμη) έχει εκθετική σχέση με τον όγκο των δεδομένων

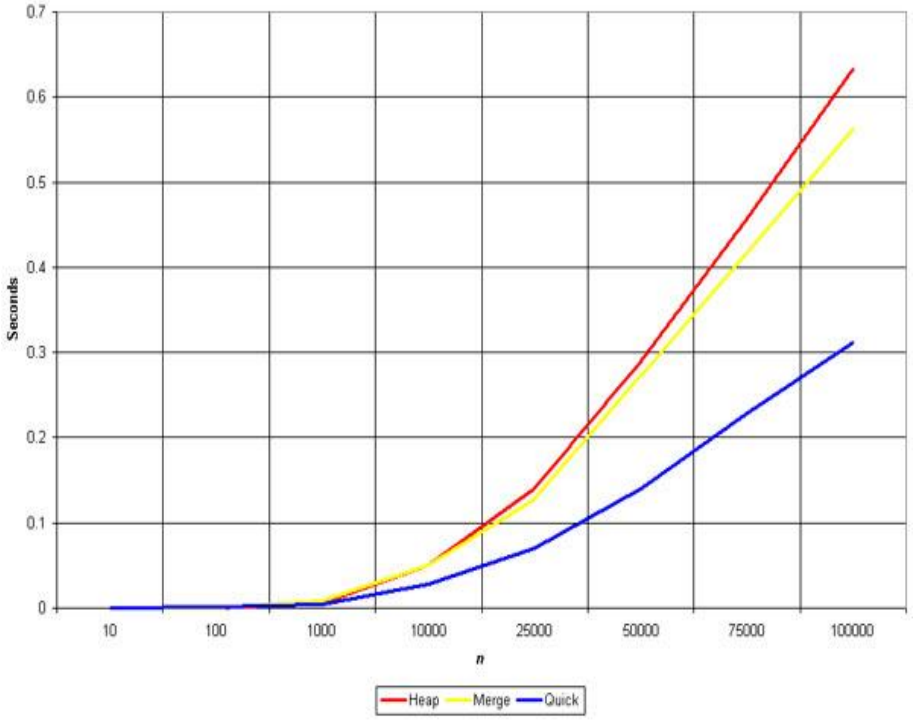
Ανάλυση Αλγορίθμων Ταξινόμησης

- Για να δούμε τη διαφορά μεταξύ των διαφόρων κατηγοριών πολυπλοκότητας ας θεωρήσουμε ότι έχουμε να ταξινομήσουμε 100, και 1000 στοιχεία
- Ένας αλγόριθμος γραμμικής σταθερής πολυπλοκότητας θα πάρει περίπου 1 χρονική μονάδα να παράγει αποτέλεσμα και για τα 100 και για τα 1000 στοιχεία
- Ένας αλγόριθμος λογαριθμικής χρονικής πολυπλοκότητας θα πάρει περίπου 2 χρονικές μονάδες να παράγει αποτέλεσμα για τα 100 στοιχεία, και 3 χρονικές μονάδες να παράγει αποτέλεσμα για τα 1000 στοιχεία
- Ένας αλγόριθμος γραμμικής χρονικής πολυπλοκότητας θα πάρει περίπου 100 χρονικές μονάδες να παράγει αποτέλεσμα για τα 100 στοιχεία και 1000 μονάδες για τα 1000 στοιχεία
- Ένας αλγόριθμος τετραγωνικής χρονικής πολυπλοκότητας θα πάρει περίπου 10.000 χρονικές μονάδες να παράγει αποτέλεσμα για τα 100 στοιχεία και 1.000.000 για τα 1000 στοιχεία
- Ένας αλγόριθμος κυβικής χρονικής πολυπλοκότητας θα πάρει περίπου 1.000.000 χρονικές μονάδες να παράγει αποτέλεσμα για τα 100 στοιχεία και 1.000.000.000 για τα 1000 στοιχεία
- Ένας αλγόριθμος εκθετικής πολυπλοκότητας θα πάρει περίπου 1.267.650.600.228.229.401.496.703.205.376 χρονικές μονάδες να παράγει αποτέλεσμα για τα 100 στοιχεία και 1.07150860718626732094842504906e+301 για 1000 στοιχεία
- Εάν ένας αλγόριθμος παίρνει πάνω από ένα τύπο δεδομένων σαν είσοδο μπορούμε να έχουμε εκφράσεις πολυπλοκότητας της μορφής $\Theta(n\ m)$ ή $\Theta(n \log m)$ όπου n, m είναι το πλήθος των δύο τύπων των δεδομένων αντίστοιχα

ΕΜΠΕΙΡΙΚΑ ΔΕΔΟΜΕΝΑ



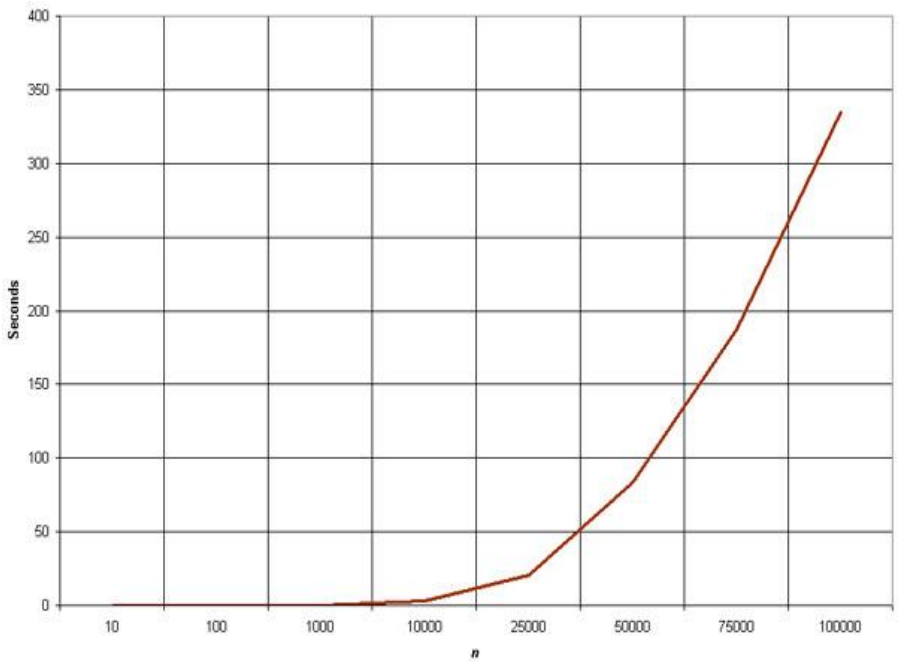
Αλγόριθμοι Ταξινόμησης με τετραγωνική Πολυπλοκότητα ($\Theta\ n^2$)



Αλγόριθμοι Ταξινόμησης με γραμμικά λογαριθμική Πολυπλοκότητα $\Theta(n \log(n))$

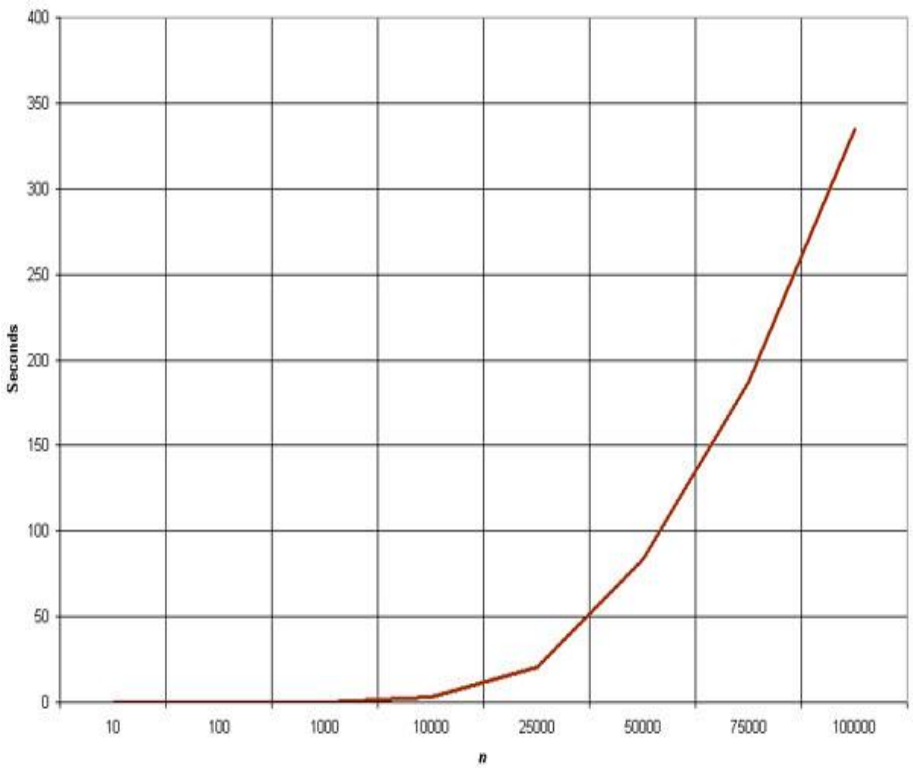
Insertion Sort

- Γενικά: Ο αλγόριθμος βάζει σε κάθε βήμα ένα στοιχείο από την αρχική (μη ταξινομημένη) λίστα στη σωστή του θέση στη τελική ταξινομημένη λίστα
- Υπέρ: Εύκολος στην υλοποίηση
- Κατά: Αργός για μεγάλες λίστες



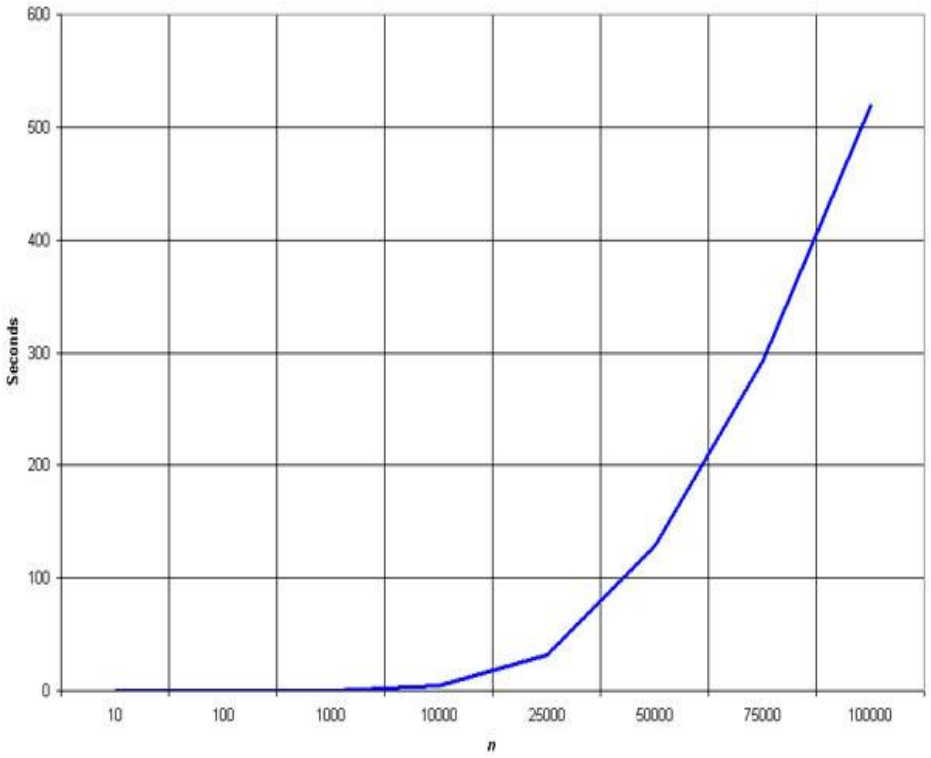
Insertion Sort

- Χρήση: Είναι ένας αλγόριθμος «μέσης-λύσης» που είναι καλός για ταξινόμηση μερικών χιλιάδων δεδομένων. Ο insertion sort είναι περίπου δύο φορές πιο γρήγορος από τον bubble sort και περίπου 40% πιο γρήγορος από τον selection sort. Ο insertion sort δεν ενδείκνυται να χρησιμοποιείται για την ταξινόμηση περισσότερων από δύο χιλιάδες στοιχεία ή για την διαδοχική και επαναληπτική ταξινόμηση λιστών με περισσότερα από μερικές εκατοντάδες στοιχείων



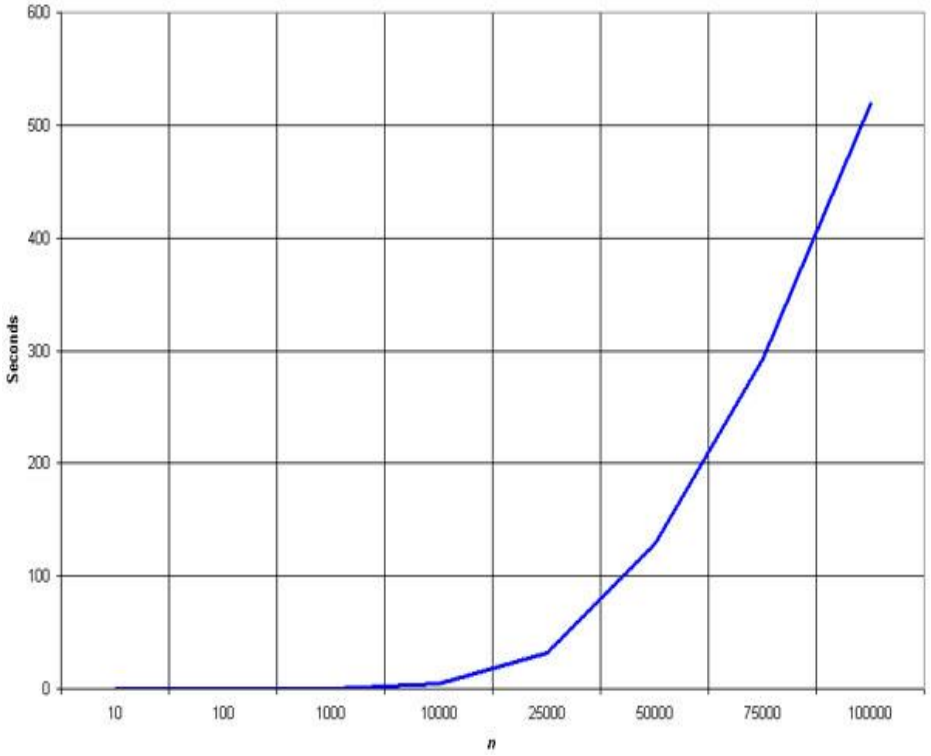
Selection Sort

- Γενικά: Ο αλγόριθμος σε κάθε βήμα βρίσκει το μικρότερο (ή μεγαλύτερο) στοιχείο που έχει μείνει στην αρχική λίστα, και το βάζει σαν επόμενο στοιχείο στην τελική λίστα
- Υπέρ: Εύκολος στην υλοποίηση
- Κατά: Αργός για μεγάλες λίστες



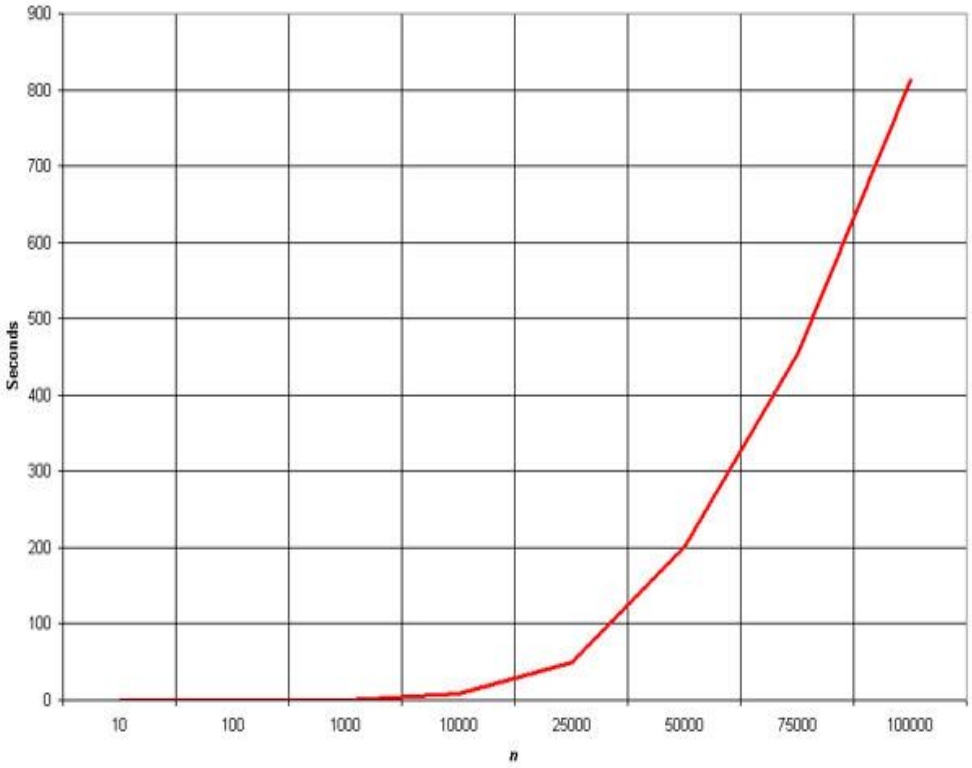
Selection Sort

- Χρήση: Είναι ένας αλγόριθμος πολύ παρόμοιος με τον insertion sort, αλλά πιο αργός οπότε ο insertion sort είναι καλύτερη επιλογή. Ο selection sort είναι περίπου 60% πιο γρήγορος από τον bubble sort (όμως θυμηθείτε ότι ο insertion sort είναι 100% πιο γρήγορος από τον bubble sort). Γενικά δεν υπάρχει καλός λόγος να προτιμήσουμε τον selection sort αλλά εάν θελήσουμε να τον χρησιμοποιήσουμε δεν πρέπει να τον χρησιμοποιήσουμε για ταξινόμηση περισσότερων από 1000 στοιχεία ή για την επαναληπτική ταξονόμηση λιστών με περισσότερα από 200 στοιχεία.



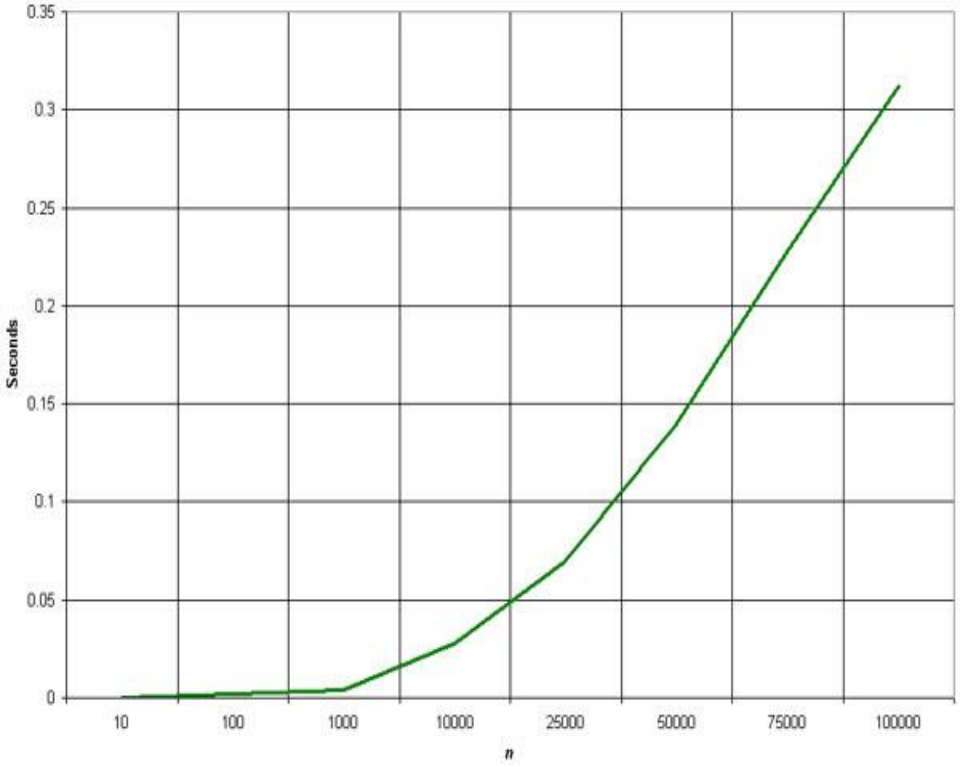
Bubble Sort

- Γενικά: Ο αλγόριθμος βάζει σε κάθε βήμα βρίσκει ελέγχει εάν το επόμενό του στοιχείο είναι μικρότερο (ή μεγαλύτερο) και εναλλάσσονται εάν χρειάζεται
- Υπέρ: Εύκολος στην υλοποίηση
- Κατά: Πολύ αργός για μεσαίες και μεγάλες λίστες.
- Χρήση: Είναι ένας αλγόριθμος πολύ αργός εκτός και εάν η λίστα δεν είναι πολύ μεγάλη και ήδη σχεδόν ή κατα το πλείστον ταξινομημένη Θ(n)!. Δεν πρέπει να τον χρησιμοποιήσουμε για ταξινόμηση περισσότερων από 100 στοιχεία ή για την επαναληπτική ταξινόμηση λιστών με περισσότερα από μερικές δεκάδες στοιχείων.



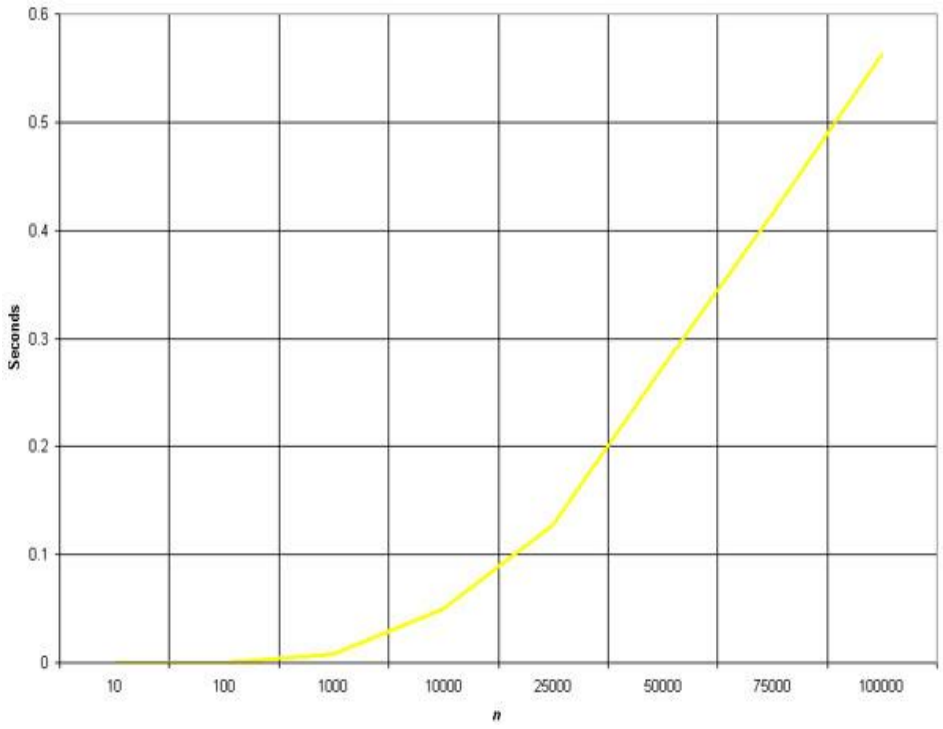
Quick Sort

- Υπέρ: Πολύ γρήγορος
- Κατά: Σχετικά μεγάλη κατανάλωση μνήμης λόγω αναδρομής. Κακή σχετικά ταχύτητα όταν η αρχική λίστα είναι ήδη ταξινομημένη
- Χρήση: Είναι γενικά ένας πολύ καλός αλγόριθμος από άποψη ταχύτητας. Καλός να χρησιμοποιείται για μέτρια και μεγάλα δεδομένα τα οποία δεν είναι ήδη ταξινομημένα. Δεν έχει κάποιο πλεονέκτημα εάν χρησιμοποιηθεί για πολύ μικρές λίστες (μέχρι 100 στοιχεία). Ο quick sort είναι λίγο-πολύ η πρώτη μας επιλογή για μεγάλα δεδομένα τα οποία δεν είναι ήδη ταξινομημένα και εάν δεν θέλουμε να χρησιμοποιήσουμε άλλους αλγόριθμους π.χ. heap sort



Merge Sort

- Γενικά: Ο αλγόριθμος χωρίζει τη μη ταξινομημένη λίστα σε δύο ίσες υπο-λίστες και ταξινομεί αναδρομικά την κάθε μια. Στο τέλος συνδυάζει τις δύο ταξινομημένες υπο-λίστες σε μία.
- Υπέρ: Σχετικά γρήγορος
- Κατά: Σχετικά μεγάλη κατανάλωση μνήμης λόγω αναδρομής
- Χρήση: Είναι σχετικά πιο γρήγορος για μεγάλα δεδομένα από το heap sort αλλά απαιτεί διπλάσια μνήμη. Ο heap sort είναι καλύτερος όμως από τον merge sort για *πολύ* μεγάλα δεδομένα. Γενικά ο merge sort είναι χειρότερος από τον quick sort και είναι κακή επιλογή όταν η μνήμη είναι περιορισμένη



Τέλος Ενότητας