Μηχανικών Πληροφορικής &
Ηλεκτρονικών Συστημάτων

ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

Αντικειμενοστρεφής Προγραμματισμός

Πολυπλοκότητα – Αλγόριθμοι αναζήτησης

Ασδρέ Κατερίνα
asdre@ihu.gr

Πολυπλοκότητα -Αλγόριθμος

πρόγραμμα για την επίλυση ενός προβλήματος = ***μέθοδος*** (ανεξάρτητη από γλώσσα προγραμματισμού).

Εκτελώντας τα βήματά της οδηγούμαστε στη λύση του προβλήματος.

***αλγόριθμος* (algorithm)**: μια πεπερασμένη, αιτιοκρατική και αποτελεσματική μέθοδος επίλυσης του προβλήματος, κατάλληλη για υλοποίηση σε ένα πρόγραμμα Η/Υ.

- έχουν διατυπωθεί αλγόριθμοι για πολλά προβλήματα εδώ και 2.300 χρόνια. Κλασικό παράδειγμα ο αλγόριθμος του Ευκλείδη για υπολογισμό του μέγιστου κοινού διαιρέτη δύο ακεραίων αριθμών

Πολυπλοκότητα -Αλγόριθμος

- ✓ Για κάθε (επιλύσιμο) πρόβλημα Π υπάρχει ένα σύνολο αλγορίθμων $\{A1, A2 \dots, A\kappa\}$ που το επιλύουν, k ≥ 1.
- ✓ Για το πρόβλημα Π υπάρχει αλγόριθμος Ai, 1 ≤ i ≤ k, τέτοιος ώστε για κάθε είσοδο I του Π ο αλγόριθμος Ai δίνει σωστό αποτέλεσμα.
- ❑ Ερώτημα: Από τους Ai, 1 ≤ i ≤ k, αλγορίθμους, ποιον να επιλέξω; Ο αλγόριθμος που θα επιλέξω είναι πάντα καλύτερος από τους άλλους, ό,τι είσοδο του Π κι αν έχω?

Πολυπλοκότητα -Αλγόριθμος

- ✓ Οι κύριες παράμετροι απόδοσης ενός αλγόριθμου:
  - Χρόνος εκτέλεσης
  - Απαιτούμενοι πόροι, π.χ. μνήμη, επικοινωνία (π.χ. σε κατανεμημένα συστήματα)
  - Βαθμός δυσκολίας υλοποίησης
- ✓ Ένας αλγόριθμος θα λέγεται αποδοτικός ή αποτελεσματικός, εάν ελαχιστοποιεί τις παραπάνω παραμέτρους.

Πολυπλοκότητα -Αλγόριθμος

- ✓ Εστιάζοντας στον χρόνο αναδιατυπώνουμε το προηγούμενο ερώτημα:
- ❑ ποιος από όλους τους *k* αλγορίθμους που επιλύουν το πρόβλημα είναι ο πιο αποτελεσματικός, δηλαδή ποιος αλγόριθμος έχει τη μικρότερη πολυπλοκότητα χρόνου (=μικρό χρόνο εκτέλεσης);

Πολυπλοκότητα -Αλγόριθμος

✓ Εστιάζοντας στον χρόνο αναδιατυπώνουμε το προηγούμενο ερώτημα:

❑ ποιος από όλους τους *k* αλγορίθμους που επιλύουν το πρόβλημα είναι ο πιο αποτελεσματικός, δηλαδή ποιος αλγόριθμος έχει τη μικρότερη πολυπλοκότητα χρόνου (=μικρό χρόνο εκτέλεσης);

- ❖ Εμπειρική προσέγγιση
- ❖ Θεωρητική προσέγγιση

Πολυπλοκότητα -Αλγόριθμος

❖ Εμπειρική πολυπλοκότητα

- κωδικοποιούμε τον αλγόριθμο σε μια γλώσσα προγραμματισμού
- τον εκτελούμε σε έναν Η/Υ με πολλά και διάφορα μεγέθη εισόδων και
- μετρούμε το χρόνο εκτέλεσης του (χρόνος μηχανής)

➢ Το αποτέλεσμα εξαρτάται από τον Η/Υ, την γλώσσα και τις ικανότητες του προγραμματιστή

Πολυπλοκότητα -Αλγόριθμος

❖ Θεωρητική πολυπλοκότητα

- Έστω n είναι το μέγεθος της εισόδου. Εάν $T1(n)$ και $T2(n)$ είναι οι χρόνοι εκτέλεσης δύο διαφορετικών εφαρμογών του αλγόριθμου Α τότε υπάρχει πάντα σταθερά c, τέτοια ώστε $T1(n) = c \cdot T2(n)$, με n πολύ μεγάλο (από την αρχή της σταθερότητας)

➢ Η παραπάνω αρχή ΔΕΝ εξαρτάται από τον Η/Υ, την γλώσσα και τις ικανότητες του προγραμματιστή!

ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

Πολυπλοκότητα -Αλγόριθμος

❖ Θεωρητική πολυπλοκότητα

- θα χρειαστούμε τον **αριθμό των βασικών πράξεων** που εκτελούνται από τον αλγόριθμο και **όχι τον ακριβή χρόνο** που απαιτούν
- το πλήθος των βασικών πράξεων ενός αλγόριθμου εξαρτάται από το μέγεθος της εισόδου του.

Πολυπλοκότητα -Αλγόριθμος

❖ Θεωρητική πολυπλοκότητα

Βασικές πράξεις:

- Ανάθεση μιας τιμής σε κάποια μεταβλητή
- Σύγκριση δύο τιμών
- Βασικές αριθμητικές πράξεις (π.χ. πρόσθεση, πολλαπλασιασμός, κλπ.)
- Εύρεση της τιμής ενός συγκεκριμένου στοιχείου σε έναν πίνακα

Πολυπλοκότητα -Αλγόριθμος

❖ Θεωρητική πολυπλοκότητα

Εκφράζουμε την πολυπλοκότητα χρόνου $T(n)$ ενός αλγόριθμου ως συνάρτηση του μεγέθους της εισόδου του, δηλαδή:

$T(n) = f(|I|)$, όπου $|I|$ το μέγεθος της εισόδου.

Αναλύοντας την πολυπλοκότητα χρόνου ενός αλγόριθμου, διακρίνουμε τις παρακάτω δύο περιπτώσεις:

- **Ανάλυση Χειρότερης Περίπτωσης** (worst-case analysis)
- **Ανάλυση Αναμενόμενης Περίπτωσης** (average-case analysis)

Πολυπλοκότητα -Αλγόριθμος

❖ Θεωρητική πολυπλοκότητα

Έστω ότι έχουμε $T(n)=8n + 6$.

Τότε $T(n)=8n$ ή $T(n) = n$

Άρα: η ασυμπτωτική συμπεριφορά της $T(n) = 8n + 6$ περιγράφεται από τη συνάρτηση $T(n) = n$ (όσο το n μεγαλώνει). Τότε λέμε ότι ο αλγόριθμός μας έχει χρονική πολυπλοκότητα $Θ(T(n))$ ή $Θ(n)$.

Πολυπλοκότητα -Αλγόριθμος

- Θεωρητική πολυπλοκότητα

Θέτουμε άνω όριο (χειρότερη περίπτωση) και εισάγεται ο λεγόμενος συμβολισμός Ο (Ο – notation ή big-O), από την αγγλική λέξη order.

Πολυπλοκότητα -Αλγόριθμος

❖ Θεωρητική πολυπλοκότητα

Κλάσεις Πολυπλοκότητας

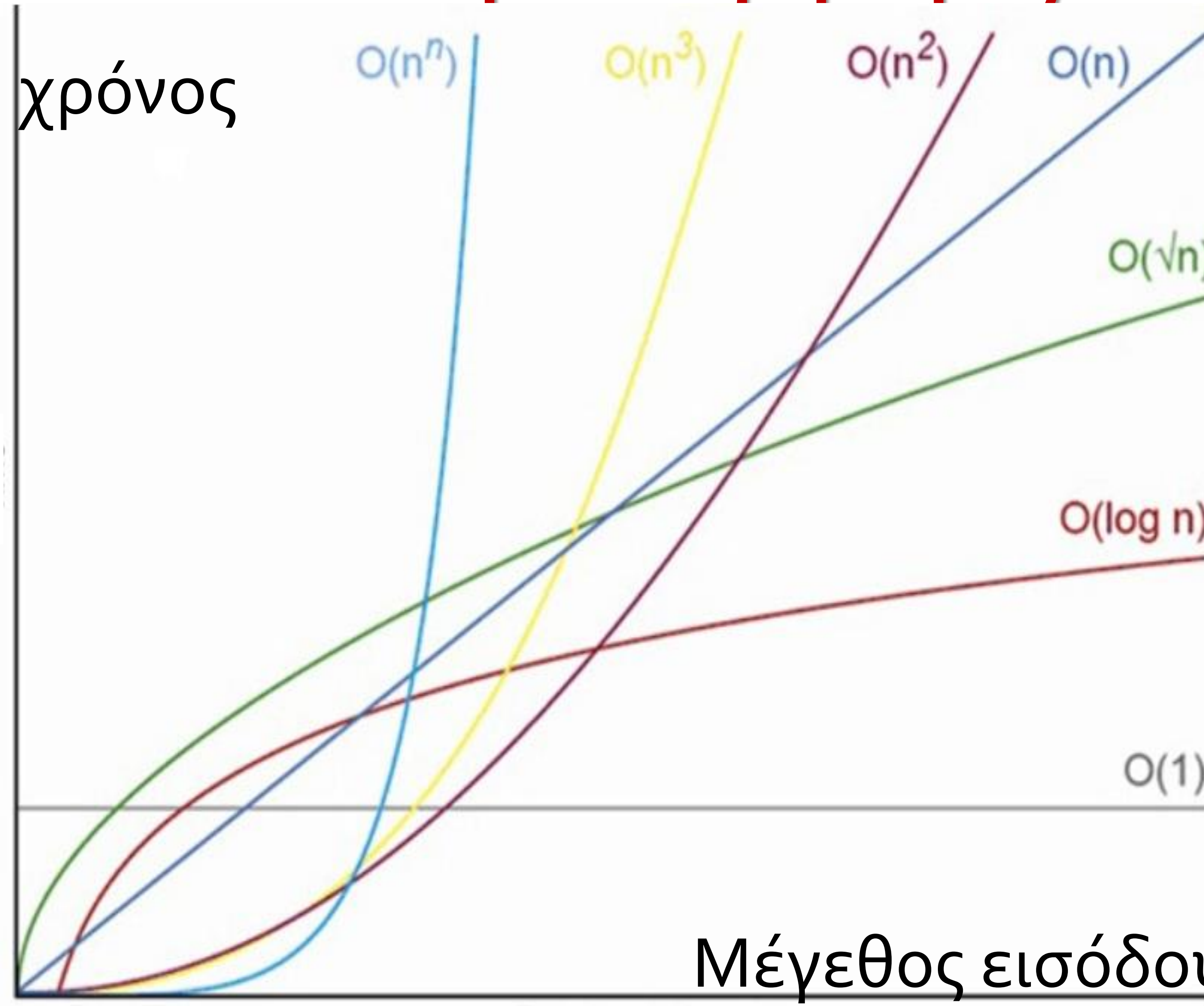
- **Λογαριθμικοί Αλγόριθμοι.** Έχουν χρόνο εκτέλεσης $T(n) = O(\log^k n)$, όπου k είναι μια θετική σταθερά
- **Γραμμικοί Αλγόριθμοι.** Έχουν χρόνο εκτέλεσης $T(n) = O(n)$
- **Πολυωνυμικοί Αλγόριθμοι.** Έχουν χρόνο εκτέλεσης $T(n) = O(n^k)$, όπου k είναι μια θετική σταθερά.
- **Εκθετικοί Αλγόριθμοι.** Έχουν χρόνο εκτέλεσης $T(n) = O(c^n)$, όπου $c > 1$.

$\log n$
n
$n \log n$
n^2
n^3
2^n
$n!$

ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

Πολυπλοκότητα -Αλγόριθμος

χρόνος



Μέγεθος εισόδου n

$\log n$

n

$n \log n$

n^2

n^3

2^n

$n!$

Σειριακή ή γραμμική αναζήτηση

- ✓ Συγκρίνουμε το στοιχείο που αναζητούμε διαδοχικά με όλα τα στοιχεία του πίνακα, ξεκινώντας από το πρώτο στοιχείο.
- ✓ Η αναζήτηση σταματά όταν βρεθεί το στοιχείο που αναζητάμε ή όταν φτάσουμε στο τελευταίο στοιχείο του πίνακα.

- ➢ Σε ταξινομημένους πίνακες σταματάμε μόλις βρούμε στοιχείο με μεγαλύτερη τιμή από το αυτό που αναζητούμε.

Σειριακή ή γραμμική αναζήτηση

```
public class LinearSearch {
  public static int LSearch(int[] array, int key){
          for(int i=0;i<array.length;i++){
                if(array[i] == key)  return i;
          }
          return -1;
  }
```

Σειριακή ή γραμμική αναζήτηση

```
//public class LinearSearch {
    public static void main(String args[]){
      int[] a= {11,9,3,24,8,2,56,1};
      int key = 2;
      System.out.println("number "+key+" is at
                              position: "+LSearch(a,key));
    }
}
```

Έξοδος:

number 2 is at position: 5

Σειριακή ή γραμμική αναζήτηση

```
public static int LSearch(int[] array, int key){
        for(int i=0;i<array.length;i++){
            if(array[i] == key) return i;
        }
        return -1;
}
```

- Ποια είναι η πολυπλοκότητα στη χειρότερη περίπτωση?
- Ποια είναι η πολυπλοκότητα στην καλύτερη περίπτωση?

Δυαδική αναζήτηση

- Εφαρμόζεται μόνο σε ταξινομημένα στοιχεία.
- Η ιδέα:
- Βρίσκουμε το μεσαίο στοιχείο του ταξινομημένου πίνακα και το συγκρίνουμε με το αναζητούμενο στοιχείο.
  - Αν το βρήκαμε τότε σταματάμε την αναζήτηση.
  - Αν δεν βρέθηκε, τότε ελέγχουμε αν το αναζητούμενο στοιχείο είναι μικρότερο ή μεγαλύτερο από το μεσαίο αυτό στοιχείο.
    - Αν είναι μικρότερο, τότε περιορίζουμε την αναζήτηση στο πρώτο μισό του πίνακα (αύξουσα τάξη τιμών), ενώ αν είναι μεγαλύτερο, περιορίζουμε την αναζήτηση στο δεύτερο μισό.
    - Η διαδικασία συνεχίζεται για το κατάλληλο πρώτο ή δεύτερο μισό του πίνακα, μετά για το ¼ του πίνακα, κλπ., μέχρι να βρεθεί το στοιχείο ή να μην είναι δυνατόν να χωριστεί περαιτέρω ο πίνακας σε δύο μέρη.

Δυαδική αναζήτηση

- Εφαρμόζεται μόνο σε ταξινομημένα στοιχεία.
- Η ιδέα:
- Βρίσκουμε το μεσαίο στοιχείο του ταξινομημένου πίνακα και το συγκρίνουμε με το αναζητούμενο στοιχείο.
  - Αν το βρήκαμε τότε σταματάμε την αναζήτηση.
  - Αν δεν βρέθηκε, τότε ελέγχουμε αν το αναζητούμενο στοιχείο είναι μικρότερο ή μεγαλύτερο από το μεσαίο αυτό στοιχείο.
    - Αν είναι μικρότερο, τότε περιορίζουμε την αναζήτηση στο πρώτο μισό του πίνακα (αύξουσα τάξη τιμών), ενώ αν είναι μεγαλύτερο, περιορίζουμε την αναζήτηση στο δεύτερο μισό.
    - Η διαδικασία συνεχίζεται για το κατάλληλο πρώτο ή δεύτερο μισό του πίνακα, μετά για το ¼ του πίνακα, κλπ., μέχρι να βρεθεί το στοιχείο ή να μην είναι δυνατόν να χωριστεί περαιτέρω ο πίνακας σε δύο μέρη.

Δυαδική αναζήτηση

```
class MyBinarySearch1 {
    public static void main (String[] args) {
        int nums[]={1,4,15,27,39,40,57,63};
        int key=40;
        System.out.println(“number "+ key + " is
found at position: "+ binarysearch(nums,key));
    }
```

ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

Δυαδική αναζήτηση

```
class MyBinarySearch1 {
    public static int binarysearch(int[] a,int k){
        int mid, left=0, right=a.length-1;
        int found = -1;
        while (found == -1 && left <= right) {
            mid = (left + right) / 2;
            if (k < a[mid])   // to k sto 1o miso
                right = mid-1;
            else if (k>a[mid])// to k sto 2o miso
                left = mid + 1;
            else found = mid;
        }
        return found;
    }
}
```

```
run:
number 40 is found at position: 5
```

Δυαδική αναζήτηση

- Ποια είναι η πολυπλοκότητα στη χειρότερη περίπτωση?
- Ποια είναι η πολυπλοκότητα στην καλύτερη περίπτωση?

Δυαδική αναζήτηση

- Ποια είναι η πολυπλοκότητα στη χειρότερη περίπτωση?
- Ποια είναι η πολυπλοκότητα στην καλύτερη περίπτωση?

- ✓ Όταν ο αλγόριθμος διαιρεί στη μέση την είσοδο (**n** τιμές) ο αριθμός των συγκρίσεων είναι της τάξης O(log n)

Δυαδική αναζήτηση με αναδρομή

```
class RecBinSearch {
      public static void main (String[] args) {
        int a[]={1,4,15,27,39,40,57,63};
        int key=63;
        int found = recBS(a,key,0,a.length-1);
        if (found > -1){
            System.out.println("number: "+key+"
                    is found at position: "+found);
        }
        else
            System.out.println("not found");
      }
```

Δυαδική αναζήτηση με αναδρομή

```
class RecBinSearch {
  public static int recBS(int[] a,int key,int left,
                                    int right) {
      int mid;
      if (right < left) return -1;
      mid = (left + right) / 2;
      if (a[mid] < key)
           return recBS(a,key,mid+1,right);
      else if (a[mid] > key)
            return recBS(a,key,left,mid-1);
      else return mid;
  }
}
```

```
run:
number: 63 is found at position: 7
```

Η δυαδική αναζήτηση είναι έτοιμη

```
import java.util.Arrays;
class RecBinSearch {
      public static void main (String[] args) {
        int a[]={1,4,15,27,39,40,57,63};
        int key=63;
        int found = Arrays.binarySearch(a,key);
        if (found > -1){
            System.out.println("number: "+key+"
                    is found at position: "+found);
        }
        else
            System.out.println("not found");
      }
```