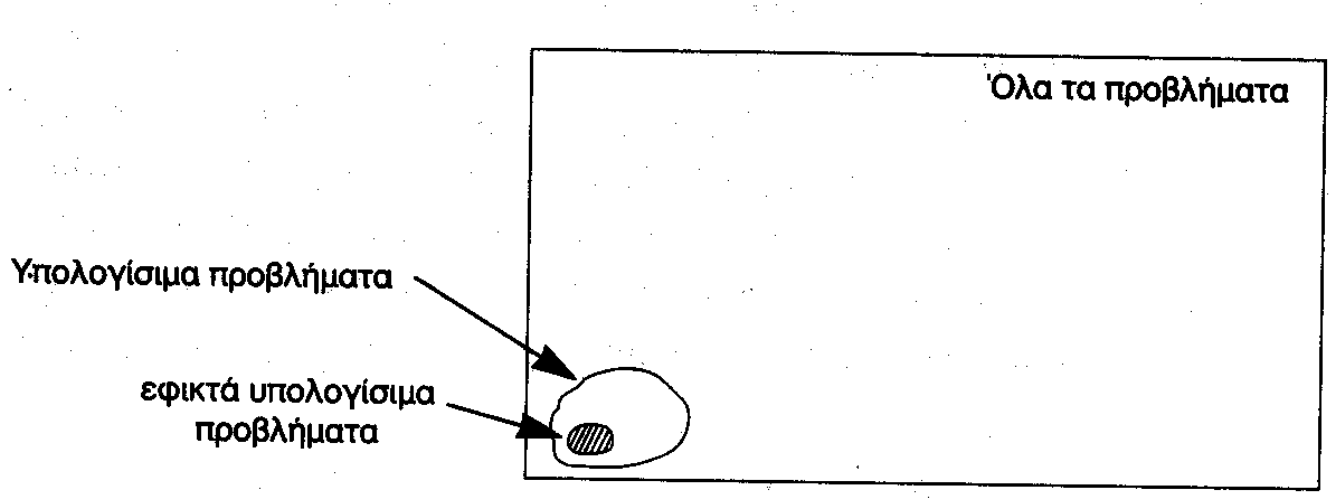
Θεωρία αλγορίθμων

θεωρία αλγορίθμων

1. Πολυπλοκότητα (Complexity)
   - Υπολογιστικοί πόροι
   - Διαίρει και Βασίλευε
   - Η θέση του σειριακού υπολογισμού
   - Μη εφικτά προβλήματα
   - NP -Πληρότητα
   - Παράλληλοι υπολογιστές
2. Ορθότητα (Correctness)
   - Σφάλματα (bugs)
   - Επαγωγή (Induction)
   - Ισχυρισμοί (Assertions)
   - Τερματισμός

Πολυπλοκότητα

- Θεωρία υπολογισιμότητας: ποια προβλήματα επιδέχονται αλγοριθμική λύση και ποια όχι
- Θεωρία πολυπλοκότητας: ερωτήματα γύρω από τη χρήση υπολογιστικών πόρων (ποιο μέρος των πόρων χρειάζεται για την επίλυση τους)



Πολυπλοκότητα

- Δε σχετίζεται με την πολυπλοκότητα σχεδιασμού ή κατανόησης του αλγόριθμου από ένα μηχανικό
- Σχετίζεται για παράδειγμα με το χρόνο εκτέλεσης του αλγορίθμου (χρόνος εκτέλεσης εντολών ≠ αριθμό εντολών που εμφανίζονται στο πρόγραμμα)
- Χρονική πολυπλοκότητα (time complexity)π.χ., αλγόριθμος με εισαγωγή $(1/2) * (N^2 – N) \rightarrow N^2$
- Χωρική πολυπλοκότητα (space complexity): π.χ., αλγόριθμος με εισαγωγή → $n+1 \cong n$

Υπολογιστικοί πόροι

- Βασικοί υπολογιστικοί πόροι: χρόνος, μνήμη, και υλικό (hardware)
  - Χρόνος: η περίοδος από την αρχή ως το τέλος της εκτέλεσης του αλγορίθμου
  - Μνήμη: το ποσό του μέσου αποθήκευσης που απαιτείται από τον αλγόριθμο (αποθήκευση επιμέρους αποτελεσμάτων)
  - Υλικό: ποσό φυσικού εξοπλισμού που απαιτείται (π.χ., αριθμός επεξεργαστών)

Παράδειγμα: αλγόριθμος πολλαπλασιασμού

```
      1984
X     6713
    -------
     5952
    1984
   13888
11904
-------------
13318592
```

Είσοδος 2 n-ψήφιους αριθμους
Άθροισμα n σειρών με n ή n+1 στοιχεία
Υπολογισμός σειράς σε n βήματα
Υπολογισμών n σειρών σε nxn=n^2 βήματα
Άθροιση σειρών: χρόνος ανάλογος του n^2
Χρόνος εκτέλεσης του αλγορίθμου ανάλογος του n^2

Παράδειγμα: αλγόριθμος πολλαπλασιασμού

Εναλλακτικός αλγόριθμος πολλαπλασιασμού

Τεχνική Διαίρει και βασίλευε

$$A\Gamma = 19 \times 67 = 1273$$

$$A\Delta + \Gamma B = 19 \times 13 + 67 \times 84 = 247 + 5628 = 5875$$

$$B\Delta = 84 \times 13 = 1092$$

13318592

Χρόνος εκτέλεσης $n^{\log 3} = n^{1.59}$

Ταχύτερος χρόνος εκτέλεσης σε σειριακή μηχανή nlogn log logn

Ταχύτερος χρόνος σε παράλληλη μηχανή logn

Χρήση υπολογιστικών πόρων

- Το ίδιο πρόβλημα μπορεί να λυθεί με διαφορετικούς αλγόριθμους και με χρήση διαφορετικών πόρων
  - Πρόβλημα η εύρεση του καλύτερου αλγόριθμου = λιγότεροι πόροι
  - Πολλές φορές η μείωση χρήσης ενός πόρου αυξάνει τη χρήση κάποιου άλλου
  - Ανάγκη εύρεσης της κατάλληλης ισορροπίας (ή trade-off) που εξυπηρετεί τις ανάγκες μας
- Το ποσό των πόρων που χρησιμοποιείται εξαρτάται από το μέγεθος των δεδομένων εισόδου
  - Για n χαρακτήρες διάφορες εκφράσεις π.χ., $n+7$, $3n^2+5n$, $\log n+17$
  - Ασυμπτωτική συμπεριφορά = ο όρος που υπερισχύει (π.χ., $3n^2+5n$)

Χρήση υπολογιστικών πόρων

Μόνο σε παράλληλους ΗΥ

Πολύ ικανοποιητικό για σειριακούς ΗΥ

ΠΙΝΑΚΑΣ 2.1

ΧΡΟΝΟΣ ΕΚΤΕΛΕΣΗΣ ΤΕΣΣΑΡΩΝ ΑΛΓΟΡΙΘΜΩΝ

Μέγεθος n των δεδομένων εισόδου	$\log_2 n$ μικροδευτερόλεπτα	n μικροδευτερόλεπτα	n^2 μικροδευτερόλεπτα	2^n μικροδευτερόλεπτα
10	0.000003 δευτερόλεπτα	0.00001 δευτερόλεπτα	0.0001 δευτερόλεπτα	0.01 δευτερόλεπτα
100	0.000007 δευτερόλεπτα	0.0001 δευτερόλεπτα	0.01 δευτερόλεπτα	10^{14} αιώνες
1000	0.00001 δευτερόλεπτα	0.001 δευτερόλεπτα	1 δευτερόλεπτο	αστρονομικός
10 000	0.000013 δευτερόλεπτα	0.01 δευτερόλεπτα	1.7 λεπτά	αστρονομικός
100 000	0.000017 δευτερόλεπτα	0.1 δευτερόλεπτα	2.8 ώρες	αστρονομικός

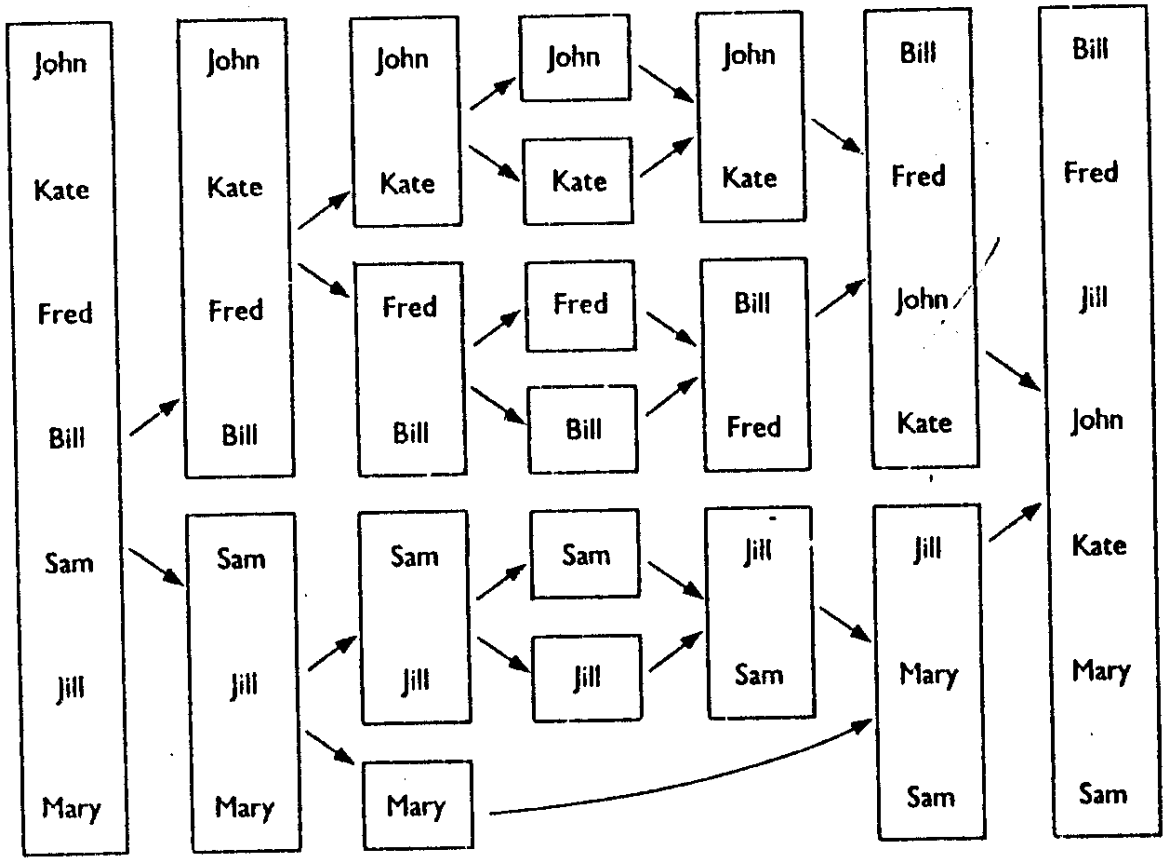
Χρήση υπολογιστικών πόρων

- Εκθετικοί αλγόριθμοι : ασυμπτωτική συμπεριφορά 2^n ή c^n
- Πολυωνυμικοί αλγόριθμοι: ασυμπτωτική συμπεριφορά n, n^2, n^c
- Αρκετές φορές ένας αλγόριθμος για διαφορετικά δεδομένα εισόδου (ακόμα και του ίδιου μήκους) να χρησιμοποιεί διαφορετικό πλήθος πόρων
  - Πολυπλοκότητα χειρότερης περίπτωσης (worst case complexity)
  - Πολυπλοκότητα μέσης περίπτωσης (average case complexity)
  - Τυπική απόκλιση (standard deviation)

Πολυπλοκότητα προβλημάτων

- Πολυπλοκότητα ενός προβλήματος: η πολυπλοκότητα του καλύτερου αλγόριθμου που επιλύει το πρόβλημα
- Άνω φράγμα πολυπλοκότητας (upper bound): η πολυπλοκότητα του καλύτερου αλγόριθμου που αντιμετωπίζει το πρόβλημα
- Κάτω φράγμα (lower bound): επίλυση ενός προβλήματος με τη χρήση τουλάχιστον κάποιου πλήθους πόρων

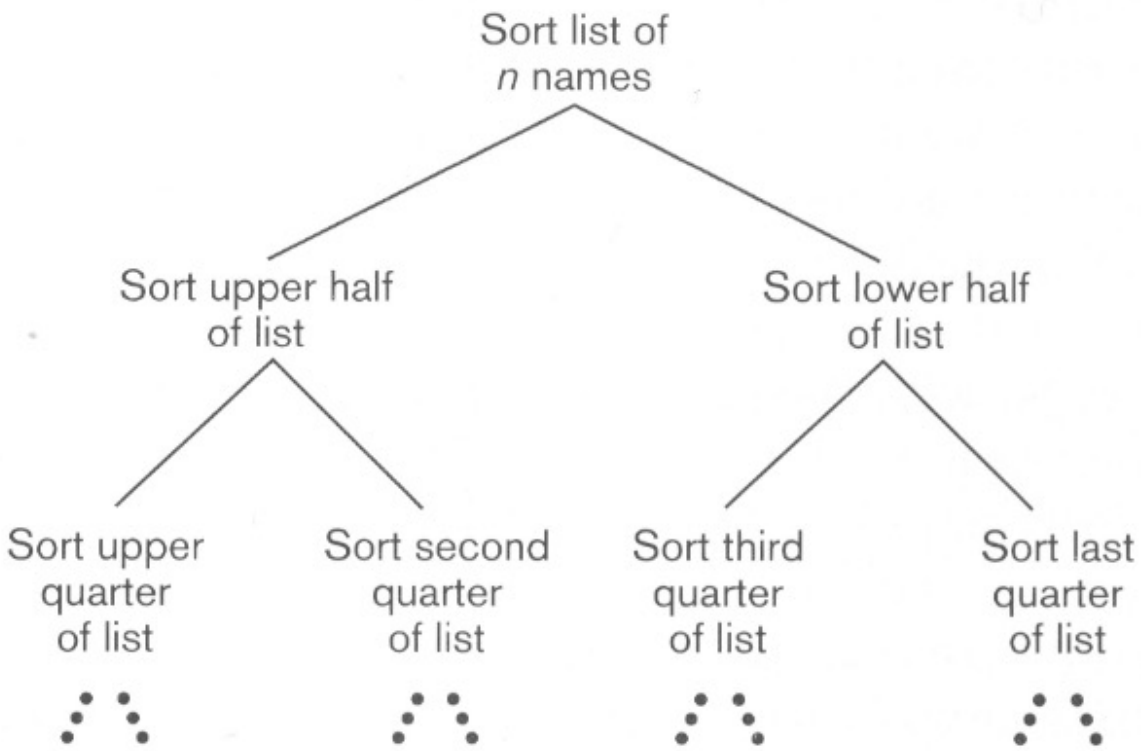
Διαίρει και βασίλευε: Παράδειγμα merge - sort



Παράδειγμα merge - sort

```
procedure MergeSort (List)
if (List has more than one entry)
 then(Apply the procedure MergeSort to sort the first half of List;
      Apply the procedure MergeSort to sort the second half of List;
      Apply the procedure MergeLists to merge the first and second
        halves of List to produce a sorted version of List
      )
```

Παράδειγμα merge - sort



Παράδειγμα merge - sort

```
procedure MergeLists (InputListA, InputListB, OutputList)
if (both input lists are empty) then (Stop, with OutputList empty)
if (InputListA is empty)
   then (Declare it to be exhausted)
   else (Declare its first entry to be its current entry)
if (InputListB is empty)
   then (Declare it to be exhausted)
   else (Declare its first entry to be its current entry)
while (neither input list is exhausted) do
      (Put the "smaller" current entry in OutputList;
      if (that current entry is the last entry in its corresponding input list)
         then (Declare that input list to be exhausted)
      (  else (Declare the next entry in that input list to be the list's current entry
      )
Starting with the current entry in the input list that is not exhausted,
      copy the remaining entries to OutputList.
```

Συγχώνευση δύο λιστών με μήκος α και β δεν περιέχει περισσότερες από α+β συγκρίσεις

Παράδειγμα merge - sort

- Το αρχικό πρόβλημα μεγέθους n μετατρέπεται σε 2 άλλα μεγέθους περίπου n/2
- Τα δύο προβλήματα μετατρέπονται σε 4 προβλήματα μεγέθους περίπου n/4
- Αριθμός συγκρίσεων σε κάθε επίπεδο όχι μεγαλύτερο από το συνολικό άθροισμα των στοιχείων των υπο-λιστών
- Ο αριθμός των επιπέδων στο δέντρο είναι της τάξης του logn

Διαίρει και βασίλευε (divide and conquer)

- Merge – Sort

  $T(n) = 2T(n/2) + cn$

  $T(1) = k$

Αποδεικνύεται ότι $T(n) = cn\log n + kn$

(ασυμπτωτικός χρόνος εκτέλεσης ανάλογος του nlogn)

- Ταχύτερος πολλαπλασιασμός

(Α+Β)(Γ+Δ), ΑΓ, ΒΔ

$T(n) = 3T(n/2) + cn$, $T(1) = k$

Αποδεικνύεται ότι $T(n) = (2c+k)n^{\log 3} - 2cn$

(ασυμπτωτικός χρόνος εκτέλεσης ανάλογος του $n^{\log 3}$ ή $n^{1.59}$)

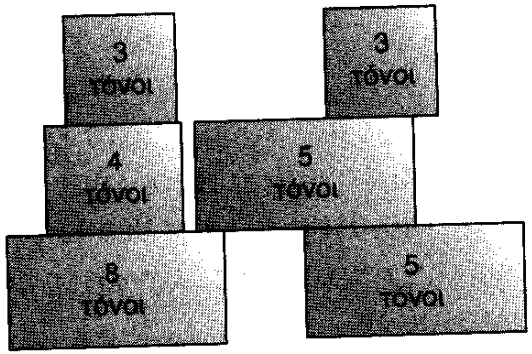
Η θέση του σειριακού υπολογισμού

- Διαχωρισμός εφικτών και ανέφικτων αλγορίθμων: Οι αλγόριθμοι που εκτελούνται σε **πολυωνυμική χρονική** διάρκεια είναι **εφικτοί** ενώ οι άλλοι δεν είναι
- Αν ένας αλγόριθμος κατασκευάζεται από 2 εφικτούς αλγόριθμους τότε είναι εφικτός (ιδιότητα κλειστότητας)
- Η εφικτότητα είναι ιδιότητα ανεξάρτητη από συγκεκριμένο υπολογιστή
- Ότι εκτελείται σε πολυωνυμικό χρόνο σε ένα υπολογιστή εκτελείται σε πολυωνυμικό χρόνο σε άλλο υπολογιστή
- Θέση σειριακού υπολογισμού: όλοι οι σειριακοί υπολογιστές έχουν πολυωνυμικούς χρόνους εκτέλεσης που σχετίζονται μεταξύ τους
- Όχι μόνο τα υπολογίσιμα αλλά και τα εφικτά υπολογίσιμα προβλήματα είναι τα ίδια για όλους τους υπολογιστές
- Θεωρία πολυπλοκότητας ανεξάρτητη υπολογιστή

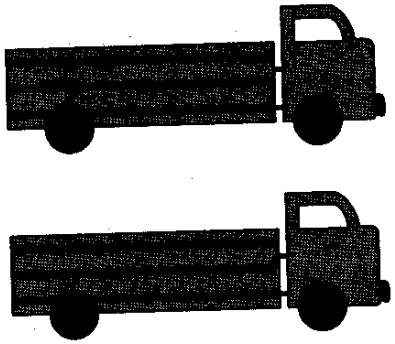
Μη εφικτά προβλήματα

- Προβλήματα μη εφικτά σε πολυωνυμικό χρόνο → δύσκολη απόδειξη (κανένας αλγόριθμος που τρέχει σε πολυωνυμικό χρόνο δεν μπορεί να λύσει το πρόβλημα)
  - Π.χ., υπολογισμός νικηφόρας στρατηγικής από μια δεδομένη θέση για σκακιέρα nxn
- Προβλήματα για τα οποία δεν έχει αποδειχθεί αν είναι εφικτά ή όχι
  - Π.χ., Μεταφορά N κιβωτίων με διαφορετικά βάρη σε T φορτηγά που μπορούν να μεταφέρουν ένα φορτίο βάρους W. (πρόβλημα συσκευασίας)

Μη εφικτά προβλήματα



N = 6 κιβώτια

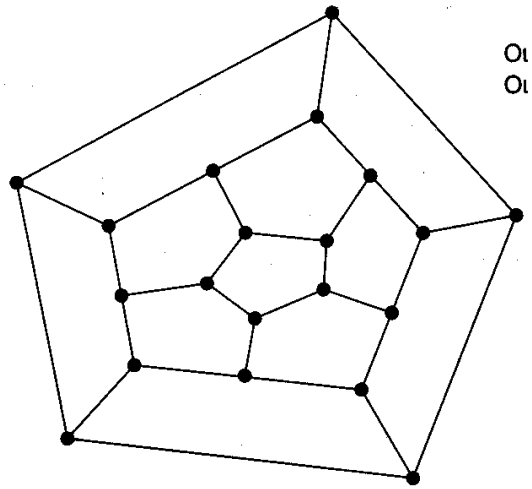


T = 2 φορτηγά
μέγιστο φορτίο W = 14 τόνοι

- Κάθε γνωστός αλγόριθμος για το πρόβλημα αυτό είναι εκθετικού χρόνου
- Λύση? Να φορτώνουμε συνέχεια τα μεγαλύτερα κιβώτια στο ένα φορτηγό μέχρι να μη χωράει άλλα και μετά να φορτώσουμε τα υπόλοιπα?

Μη εφικτά προβλήματα

- Το πρόβλημα του πλανόδιου πωλητή (traveling salesperson problem)
  - Με δεδομένο ένα χάρτη δρόμων με Ν πόλεις, είναι δυνατόν ένα πωλητής να ολοκληρώσει ένα ταξίδι χωρίς να ξεπεράσει ένα αριθμό μιλίων, με την προϋπόθεση ότι θα περάσει από κάθε πόλη μία μονό φορά?



Οι τελείες αναπαριστούν πόλεις.
Οι γραμμές αναπαριστούν δρόμους

Μη εφικτά προβλήματα

- Για το πρόβλημα αυτό δεν έχει αποδειχθεί αλγόριθμος πολυωνυμικού χρόνου αλλά ούτε έχει αποδειχθεί η μη ύπαρξη του
- Σχετικό πρόβλημα (κύκλος του Hamilton): υπάρχει διαδρομή στο χάρτη που να περνάει από κάθε πόλη ακριβώς μια φορά?
- Το πρόβλημα του ωρολογίου προγράμματος: λίστα μαθημάτων, μαθητών και χρονικών διαστημάτων που είναι διαθέσιμα και πρέπει να φτιαχτεί έτσι το πρόγραμμα ώστε κανείς μαθητής να μην έχει χρονική σύγκρουση στα μαθήματα που παρακολουθεί
  - Δεν υπάρχει αλγόριθμος πολυωνυμικού χρόνου

Πώς αντιμετωπίζουμε μη εφικτά προβλήματα?

- Αναζήτηση προσεγγιστικής λύσης αντί για μία ακριβή λύση
  - λίγες χρονικές συγκρούσεις στο πρόβλημα του ωρολογίου προγράμματος είναι ικανοποιητική λύση
  - ένας σύντομος αλγόριθμος μιας λογικά σύντομης διαδρομής είναι ικανοποιητική λύση στο πρόβλημα του πλανόδιου πωλητή
- Εφαρμογή αλγορίθμων με ικανοποιητική απόκριση σε ένα μέσο πλήθος δεδομένων εισόδου αλλά με εκθετική συμπεριφορά στις χειρότερες περιπτώσεις
  - Προσοχή σε προγράμματα που απαιτούν εγγυημένη χρονική απόκριση
- Χαλαρή αντιμετώπιση της συνθήκης ότι ο αλγόριθμος πρέπει να είναι σωστός

Πώς αντιμετωπίζουμε μη εφικτά προβλήματα?

- Π.χ., ιδιότητα πρώτων αριθμών
- Τυπικός αλγόριθμος είναι εκθετικός
- Εναλλακτική προσέγγιση ισχύει: Αν χ πρώτος τότε $\forall$ y, $1 \leq y \leq x-1$ ισχύει

  $ΜΚΔ(χ,y)=1$ και $y^{(x-1)/2} \bmod x = J(y,x)$

- Αποδεικνύεται ότι αν χ όχι πρώτος τότε τουλάχιστον οι μισές τιμές του y δεν ικανοποιούν τη συνθήκη

Πώς αντιμετωπίζουμε μη εφικτά προβλήματα?

Module prime(x)

{Ελέγχει αν είναι πρώτος οποιοσδήποτε περιττός αριθμός. Αν ο χ είναι πρώτος τότε η έξοδος είναι 'Πρώτος'. Αν ο χ δεν είναι πρώτος τότε 'Όχι πρώτος' πιθανόν να είναι η έξοδος, αλλά υπάρχει πολύ μικρή πιθανότητα ο αλγόριθμος να τυπώσει λάθος την ένδειξη 'Πρώτος'}

repeat e φορές

διάλεξε τυχαία ένα αριθμό y ανάμεσα στους 1 και χ-1

if ΜΚΔ(χ,y) ≠ 1

then τύπωσε 'Όχι πρώτος' και σταμάτα

αν $y^{(x-1)/2} \bmod x \neq J(y,x)$

then τύπωσε 'Όχι πρώτος' και σταμάτα

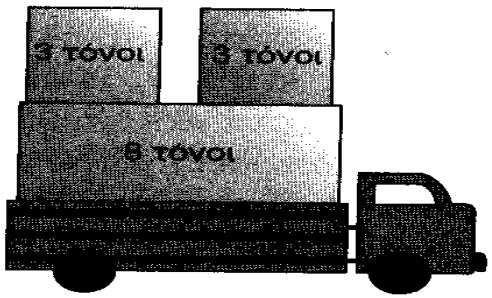
τύπωσε 'Πρώτος'

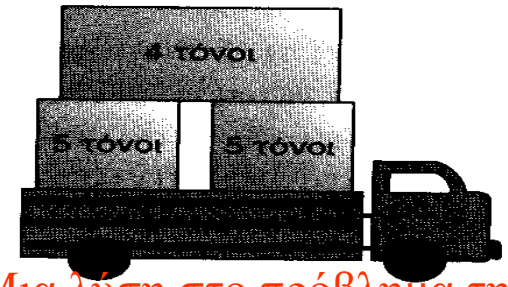
Πώς αντιμετωπίζουμε μη εφικτά προβλήματα?

- Η σταθερά e παίρνει τιμές της τάξης του 20 ή 30
- Αν χ πρώτος τότε το πρόγραμμα δουλεύει σωστά
- Αν χ δεν είναι πρώτος τότε σε κάθε επανάληψη 0,5 η πιθανότητα η τιμή του y να μην ικανοποιεί τη συνθήκη
- Μετά από 20 επαναλήψεις η πιθανότητα λάθους είναι 2 μικρότερη από 1 στο εκατομμύριο
- Πιθανολογικός αλγόριθμος

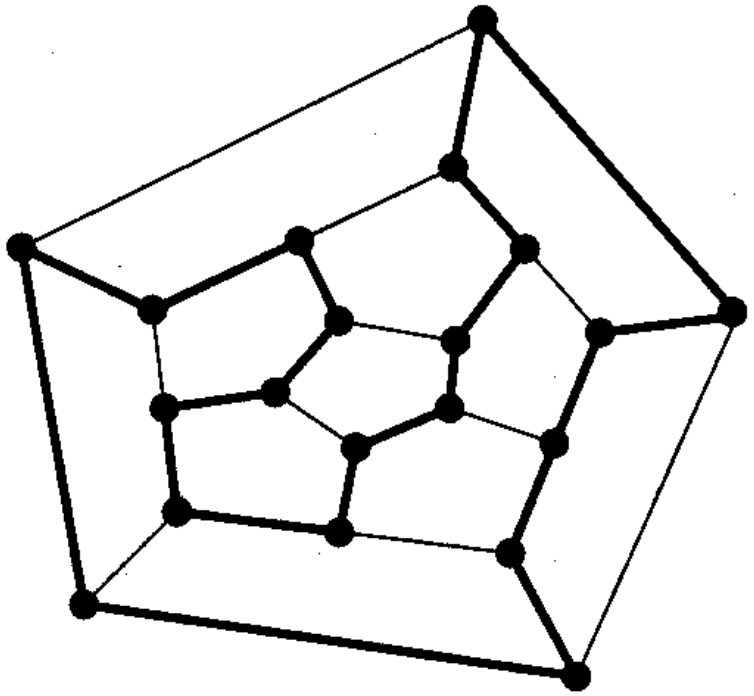
NP Πληρότητα

- Τα προβλήματα που περιγράψαμε προηγουμένως δεν επιλύονται σε πολυωνυμικό χρόνο
- Κοινή ιδιότητα: αν βρεθεί η σωστή λύση επαληθεύεται σε πολυωνυμικό χρόνο





Μια λύση στο πρόβλημα της συσκευασίας



Μια λύση στο πρόβλημα του κύκλου του Hamilton

NP Πληρότητα

- Το σύνολο των προβλημάτων που έχουν γρήγορο αλγόριθμο επαλήθευσης λέγονται NP
- Όλα τα εφικτά προβλήματα ανήκουν στα NP
- 1971 ο S. Cook έδειξε ότι ένας αριθμός ανοικτών προβλημάτων ήταν μεταξύ των δυσκολότερων NP προβλημάτων
- NP-πλήρες (NP complete): αν για κάποιο από αυτά ανακαλυφθεί αλγόριθμος που να τρέχει σε πολυωνυμικό χρόνο, τότε θα υπάρχει αλγόριθμος πολυωνυμικού χρόνου για κάθε NP- πρόβλημα

NP Πληρότητα

- Πρόβλημα συσκευασίας, κύκλος του Hamilton, του πλανόδιου πωλητή είναι NP πλήρες
- Όλα τα NP πλήρη προβλήματα είναι ισοδύναμα
- Αν αποδειχτεί ότι κάποιο από αυτά είναι ανέφικτο τότε όλα αυτά θα είναι ανέφικτα
- Μέχρι τώρα όλες οι προσπάθειες για να αποδείξουν αν είναι εφικτά ή όχι έχουν αποτύχει
- Πρακτικό όφελος: Αν ένα πρόβλημα είναι ισοδύναμο με ένα NP πλήρες πρόβλημα τότε είναι πολύ πιθανό ότι δεν θα βρούμε μια εφικτή λύση

θεωρία αλγορίθμων

Ορθότητα (Correctness)

- Σφάλματα (bugs)
- Επαγωγή (Induction)
- Ισχυρισμοί (Assertions)
- Τερματισμός

Σφάλματα - Bugs

- Τα περισσότερα προγράμματα περιέχουν λάθη
- Η επιτυχής εκτέλεση ενός προγράμματος για χρόνια δεν εξασφαλίζει τη μη ύπαρξη λαθών
- Η διόρθωση τους λέγεται αποσφαλματοποίηση (debugging)
- Μέθοδοι κατασκευής προγραμμάτων: με δοκιμή (testing) ή με απόδειξη (proving)
- Δοκιμή ενός προγράμματος: εκτέλεση με ένα συγκεκριμένο σύνολο δεδομένων (δεδομένα δοκιμής – test data)
  - Εκτέλεση με υπολογιστή ή με χειρογραφική ιχνηλάτηση (tracing)
  - Το αποτέλεσμα σχετίζεται με το σύνολο δεδομένων εισόδου που επιλέξαμε

• Η απόδειξη ότι είναι σωστό σημαίνει την πιστοποίηση της ορθότητας για όλα τα δεδομένα εισόδου

Σφάλματα - Bugs

- Η δοκιμή προγραμμάτων ως τεχνική αποσφαλματοποίησης έχει υπερισχύσει έναντι των αποδείξεων
- Η ορθή εκτέλεση των προγραμμάτων για κάποια δεδομένα εισόδου δεν εξασφαλίζει την ορθότητα για όλα τα δεδομένα εισόδου
- Η κατασκευή αποδείξεων είναι μια δύσκολη διεργασία
- Άτυπες αποδείξεις ορθότητας αποτελούν ουσιαστικά την αναπαράσταση της καλής κατανόησης του προγράμματος
- Η σχεδίαση και ορθότητα ενός αλγορίθμου θα πρέπει να συμβαδίζουν

Σφάλματα - Bugs

- Τεκμηρίωση (documentation): ύπαρξη γραπτών στοιχείων για το πώς δουλεύει ένα πρόγραμμα *(κατά μία έννοια άτυπη απόδειξη ορθότητας)*
- Επιτραπέζιος έλεγχος (desk checking): Προσεκτική ανάγνωση και κατανόηση ενός προγράμματος, με σκοπό τον έλεγχο ορθότητας του *(κατά μία έννοια άτυπη απόδειξη ορθότητας)*
- Ένας αλγόριθμος είναι ορθός σε σχέση με τις προδιαγραφές (specifications) του, αν ο αλγόριθμος παράγει τα προσδιοριζόμενα αποτελέσματα για εκείνες τις τιμές εισόδου που ορίζονται από τις προδιαγραφές
- Η πιστοποίηση της ορθότητας σε σχέση με τις προδιαγραφές είναι μη υπολογίσιμο πρόβλημα
- Μερική ορθότητα (partial correctness): Οποτεδήποτε ένας αλγόριθμος τερματίζει, το παραγόμενο αποτέλεσμα είναι το αναμενόμενο
- Ιδιαίτερα σημαντικό για την ορθότητα ενός προγράμματος είναι και η απόδειξη τερματισμού του
- Πλήρως ορθό (totally correct): ένα πρόγραμμα που πάντα τερματίζει και το οποίο είναι μερικώς ορθό

Επαγωγή

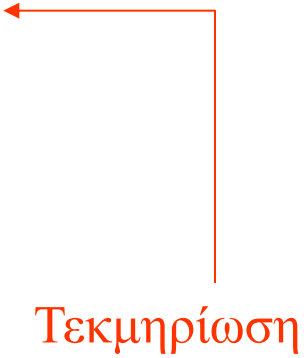
```
module ύψωσε-σε-δύναμη(x)
{Εκτυπώνει το 2^x υποθέτοντας ότι το x είναι μη αρνητικός ακέραιος}
θέσε το ΑΘΡΟΙΣΜΑ ίσο με 1
repeat x φορές
   θέσε το ΑΘΡΟΙΣΜΑ ίσο με ΑΘΡΟΙΣΜΑ +ΑΘΡΟΙΣΜΑ
   ***
εκτύπωσε το ΑΘΡΟΙΣΜΑ
```

Προδιαγραφή

Απόδειξη ορθότητας = κατανόηση ότι στο n πέρασμα άθροισμα = 2^n
Απόδειξη μερικής ορθότητας
Απόδειξη πλήρους ορθότητας ?

Επαγωγή

module *υψωσε-σε-δυναμη(x)*
{Εκτυπώνει το 2^x υποθέτοντας ότι το *x* είναι μη αρνητικός ακέραιος}
θέσε το ΑΘΡΟΙΣΜΑ ίσο με 1
repeat *x φορές*
  θέσε το ΑΘΡΟΙΣΜΑ ίσο με ΑΘΡΟΙΣΜΑ +ΑΘΡΟΙΣΜΑ
  {Τη *n* -οστή φορά που φτάνουμε σ' αυτό το σημείο,
   το *ΑΘΡΟΙΣΜΑ* θα ισούται με 2^n}
εκτύπωσε το ΑΘΡΟΙΣΜΑ



Επαγωγή

Επαγωγική υπόθεση: Την *n*-οστή φορά που φτάνουμε στο ***, *ΑΘΡΟΙΣΜΑ* = 2^n.

Βάση: Όταν n = 1, το *ΑΘΡΟΙΣΜΑ* παίρνει την τιμή 1+1 που ισούται με 2^1.

Επαγωγικό βήμα: Υποθέστε ότι η επαγωγική υπόθεση είναι αληθής για κάποια συγκεκριμένη τιμή του n. Την (*n* +1)-οστή φορά που φτάνουμε στο *** , το *ΑΘΡΟΙΣΜΑ* τίθεται ίσο με την προηγούμενη τιμή του προστιθέμενη στον εαυτό της. Λόγω της επαγωγικής υπόθεσης, η προηγούμενη τιμή που έχει το *ΑΘΡΟΙΣΜΑ* είναι 2^n. Συνεπώς η νέα τιμή είναι $2^n + 2^n = 2^{n+1}$. Συνεπώς η επαγωγική υπόθεση ισχύει επίσης για *n* +1.

Συμπέρασμα: Από τη βάση και το επαγωγικό βήμα προκύπτει ότι η επαγωγική υπόθεση ισχύει για κάθε τιμή του $n \geq 1$.

Επίσης

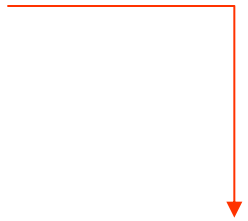
Για n = 0: δεν εισέρχεται στο βρόχο και άθροισμα =1

Άρα σε κάθε περίπτωση που φτάνουμε στον τερματισμό τυπώνεται το 2^{χ}

Επαγωγή

Προδιαγραφές: όταν x,y είναι μη αρνητικοί ακέραιοι ο αλγόριθμος τυπώνει το ΜΚΔ

```
module Ευκλειδης(x,y)
{Εκτυπώνει τον Μέγιστο Κοινό Διαιρέτη των μη αρνητικών ακεραίων x και y}
while y ≠ 0 do
    υπολόγισε το υπόλοιπο του x/y
    θέσε το x ίσο με το y
    θέσε το y ίσο με το υπόλοιπο
    ***
εκτύπωσε το x
```



Ο ΜΚΔ των x΄,y΄ στο σημείο αυτό ισούται με τον ΜΚΔ των αρχικών x,y

Επαγωγή

Απόδειξη της μερικής ορθότητας

Επαγωγική υπόθεση: όταν φτάνουμε στο ***, GCD(x,y)=g

Βάση: x=gx' και y = gy' όπου x', y' δεν έχουν κοινούς παράγοντες.
Επίσης x=my+r. Άρα, gx'=mgy'+ r ή r = g(x'-my') →το r έχει παράγοντα το g.
Επίσης, δεδομένου ότι gx'=mgy'+ r, τότε r/g και y' δεν έχουν κοινό παράγοντα γιατί τότε x' και y' θα είχαν κοινό παράγοντα. Άρα r και y έχουν ΜΚΔ το g.
Άρα την πρώτη φορά που φτάνουμε στο *** ισχύει

GCD(x,y) = GCD(y, mod x/y)=GCD(y,r)=g

Επαγωγικό Βήμα: Η βάση αποδεικνύει επίσης ότι τη (n+1) φορά που φτάνουμε στο ***, GCD(x,y) ισούται με το GCD(x,y) τη n-οστή φορά που φτάνουμε στο *** και ότι η τιμή του είναι το g

Συμπέρασμα: Η επαγωγική υπόθεση ισχύει για κάθε τιμή n≥1, δηλαδή οποτεδήποτε φτάνουμε στο ***

Επαγωγή

module *Ευκλειδης* (*x*,*y*)
{Εκτυπώνει τον Μέγιστο Κοινό Διαιρέτη των μη αρνητικών ακεραίων x και y}
while $y \neq 0$ **do**
    υπολόγισε το υπόλοιπο του x/y
    θέσε το x ίσο με το y
    θέσε το y ίσο με το υπόλοιπο
    {σ' αυτό το σημείο το *GCD* (*x*,*y*) ισούται με το GCD των αρχικών εισόδων
     επειδή ...}
εκτύπωσε το x

Τεκμηρίωση

Επαγωγή

Επαγωγή σε αναδρομικούς αλγορίθμους

```
module εκτύπωσε-δένδρο (T)
{Εκτυπώνει όλους τους κόμβους του δυαδικού δένδρου T
 από αριστερά προς τα δεξιά}
if το T δεν είναι κενό
   then εκτύπωσε-δένδρο (αριστερό υποδένδρο του T)
        εκτύπωσε τη ρίζα του T
        εκτύπωσε-δένδρο (δεξιό υποδένδρο του T)
```

Επαγωγή

Επαγωγική υπόθεση: Ο αλγόριθμος εκτυπώνει όλους τους κόμβους του δένδρου T από αριστερά προς τα δεξιά.

Βάση: Αν το T δεν περιέχει κόμβους, τότε είναι κενό και ο αλγόριθμος δεν κάνει τίποτα. Μπορούμε να πούμε ότι όλοι οι κόμβοι του δένδρου εκτυπώνονται με τη σωστή σειρά, αφού δεν υπάρχουν κόμβοι.

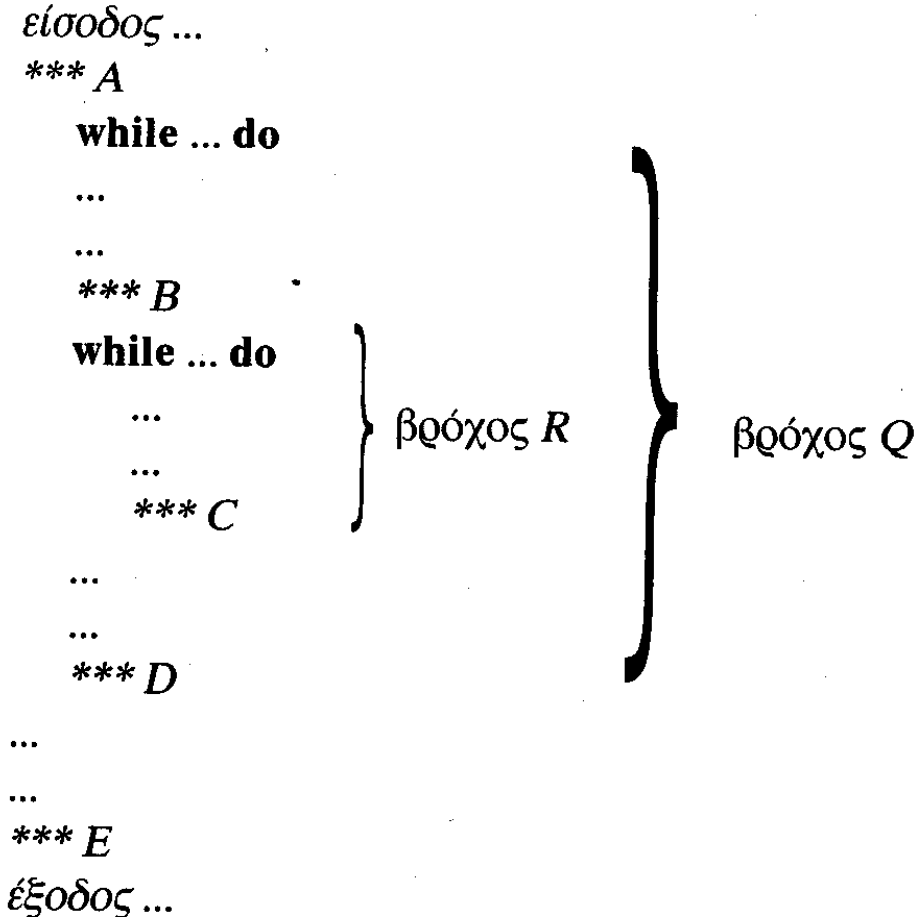
Επαγωγικό βήμα: Υποθέστε ότι το T έχει n κόμβους και ότι η επαγωγική υπόθεση ισχύει για όλα τα δένδρα με λιγότερους από n κόμβους. Το αριστερό υποδένδρο είναι ένα δένδρο με λιγότερους από n κόμβους (αφού δεν περιλαμβάνει τη ρίζα), και έτσι από την επαγωγική υπόθεση όλοι οι κόμβοι του εκτυπώνονται από τα αριστερά προς τα δεξιά. Στη συνέχεια ο αλγόριθμος εκτυπώνει τη ρίζα. Τελικά, εφαρμόζοντας πάλι την επαγωγική υπόθεση, όλοι οι κόμβοι του δεξιού υποδένδρου εκτυπώνονται από αριστερά προς τα δεξιά. Συνεπώς όλοι οι κόμβοι του T εκτυπώνονται με τη σωστή σειρά, αποδεικνύοντας ότι η επαγωγική υπόθεση ισχύει επίσης για δένδρα με n κόμβους.

Συμπέρασμα: Η επαγωγική υπόθεση ισχύει για κάθε δυαδικό δένδρο.

Ισχυρισμοί

- Οι προδιαγραφές κάθε τμήματος αποτελούνται από δύο μέρη: εύρος των τιμών εισόδου και το επιθυμητό αποτέλεσμα
- Τα βασικά αυτά μέρη είναι γνωστά ως προϋποθέσεις (preconditions) και συνέπειες (post-conditions)
- Οι προϋποθέσεις και οι συνέπειες είναι γνωστές και ως ισχυρισμοί (assertions) αφού δηλώνουν την αλήθεια κάποιας συνθήκης
- Οι ισχυρισμοί τοποθετούνται σε κατάλληλα σημεία στους αλγορίθμους (π.χ., στα σημεία ***) , και είναι περισσότερα στους πολύπλοκους αλγορίθμους
- Η τοποθέτηση τους πρέπει να γίνει με τέτοιο τρόπο ώστε να είναι εύκολο να γίνει κατανοητή η ροή του ελέγχου από ένα ισχυρισμό σε ένα άλλο.
- Τοποθέτηση ενός τουλάχιστον ισχυρισμού σε κάθε βρόχο

Ισχυρισμοί



Είναι αναγκαίο να αποδειχθεί ότι αν A αληθής και εκτελούνται οι εντολές ανάμεσα στα A, B τότε ο B είναι αληθής. Και όμοια B→C, C→C, C→D, D→B, D→E, A→E (αν ο βρόχος Q δεν εκτελεστεί ποτέ), B→ D (αν ο βρόχος R δεν εκτελεστεί ποτέ).

Ισχυρισμοί

module σφαιρίδια
{Υπολογίζει τον μέγιστο αριθμό σφαιριδίων που έχουν το ίδιο χρώμα}
θέσε τον μέγιστο ίσο με το 0
repeat *για κάθε χρώμα*
  θέσε το πλήθος ίσο με το 0
  repeat *για κάθε σφαιρίδιο*
    if *το σφαιρίδιο έχει αυτό το χρώμα*
      then *πρόσθεσε* 1 *στο πλήθος*
  if *πλήθος* > *μέγιστο*
    then *θέσε τον μέγιστο ίσο με το πλήθος*
    εκτύπωσε τον μέγιστο

βρόχος R

βρόχος Q

Ισχυρισμοί

```
module σφαιρίδια
{Υπολογίζει τον μέγιστο αριθμό σφαιριδίων που έχουν το ίδιο χρώμα}
{(A) Ένας αριθμός πολύχρωμων σφαιριδίων δίνεται στην είσοδο}
θέσε τον μέγιστο ίσο με το 0
repeat για κάθε χρώμα
   θέσε το πλήθος ίσο με το 0
   repeat για κάθε σφαιρίδιο
      if το σφαιρίδιο έχει αυτό το χρώμα
         then πρόσθεσε 1 στο πληθος
      {(B) Όταν περάσουμε από αυτό το σημείο n διαδοχικές φορές, το πλήθος θα ισούται
       με τον αριθμό των σφαιριδίων, από τα n πρώτα, που έχουν το δεδομένο χρώμα}
   {(C) Το πλήθος τώρα ισούται με τον συνολικό αριθμό των σφαιριδίων
    με το δεδομένο χρώμα}
   if πλήθος > μέγιστο
      then θέσε τον μέγιστο ίσο με το πληθος
   {(D) Όταν περάσουμε από αυτό το σημείο m φορές, το μέγιστο θα ισούται
    με τον μέγιστο αριθμό των σφαιριδίων που έχουν το ίδιο χρώμα,
    από τα πρώτα m χρώματα}
{(E) Το μέγιστο τώρα ισούται με τον μέγιστο αριθμό των σφαιριδίων
 που έχουν το ίδιο χρώμα}
εκτύπωσε τον μέγιστο
```

Τερματισμός

- Η απόδειξη μερικής ορθότητας δεν εγγυάται την εξαγωγή αποτελέσματος
- Μια τέτοια εγγύηση χρειάζεται απόδειξη ολικής ορθότητας (δηλαδή τερματίζει ή φτάνει στην εντολή εξόδου του)

```
module Fermat(n).
{Δοκιμάζει το τελευταίο θεώρημα του Fermat για δεδόμενο εισόδου n}
repeat για a = 1,2,3, . . . {για πάντα}
  repeat για b = 1, 2, 3, . . . , a
    repeat για c = 2, 3, 4, . . ., a + b
      if a^n + b^n = c^n
        then τύπωσε τα a, b, c και n και σταμάτα.
```

- Η απόδειξη μερικής ορθότητας είναι εύκολη !
- Η απόδειξη ολικής ορθότητας είναι σαφώς πολύ πιο δύσκολη εργασία

Τερματισμός

- Μηχανική μέθοδος για την απόδειξη τερματισμού?
- Μια τεχνική για να καταλαβαίνουμε γιατί ένας βρόχος πρέπει να τερματίζει → σημαντική ελάττωση ή αύξηση της τιμής της μεταβλητής σε κάθε εκτέλεση του βρόχου

module *Ευκλείδης* (x, y)
{τυπώνει τον Μέγιστο Κοινό Διαιρέτη δύο μη αρνητικών ακεραίων x και y}
while y ≠ 0 **do**
υπολόγισε το υπόλοιπο x/y
Θέσε το x ίσο με y
Θέσε το y ίσο με το υπόλοιπο
{Σ' αυτό το σημείο ο GCD (x,y) είναι ίσος με το GCD των αρχικών δεδομένων}.
τύπωσε το x

module *funnyΕυκλείδης* (x,y).
{Δεν κάνει τίποτα και δεν τερματίζει ποτέ}
while y ≠ 0 **do**
μήν κάνεις τίποτα
{σ' αυτό το σημείο ο GCD (x,y) είναι ίσος με το GCD των αρχικών δεδομένων}
τύπωσε το x

Τερματισμός

- Συνάρτηση του Ackerman

```
module A (x,y)
{Η συνάρτηση του Ackermann A(x,y) έχει δύο δεδομένα x και y που πρέπει
 να είναι μη αρνητικοί ακέραιοι}
if x = 0
   then η απάντηση είναι y + 1
      else if y = 0
              then η απάντηση είναι A (x-1, 1)
              else η απάντηση είναι A (x-1, A (x, y - 1))
```

- Δοκιμάστε για A(2,2) ή A(3,1)...
- Και οι τρεις αναδρομικές κλήσεις μειώνουν άμεσα ή έμμεσα το χ

Τερματισμός

```
module τύπωσε_δένδρο(T)
{τυπώνει όλους τους κόμβους ενός δυαδικού δένδρου Τ με σειρά από αριστερά
 προς τα δεξιά}
if το Τ δεν είναι κενό
   then τύπωσε_δένδρο (το αριστερό υποδένδρο του Τ)
        τύπωσε τη ρίζα του Τ
        τύπωσε_δένδρο (το δεξιό υποδένδρο του Τ)
```

Μείωση του μεγέθους του δέντρου τουλάχιστον κατά 1 σε κάθε κλήση

Τερματισμός

```
module tricky(x)
{Αυτός ο αλγόριθμος περιμένει να εισαγάγουμε έναν x που είναι θετικός ακέραιος}
while x > 1 do
    if x είναι άρτιος
        then θέσε x ίσο με x/2
        else θέσε x ίσο με 3x + 1
```

- Τερματίζει για αρκετές τιμές του χ
- Κανείς δεν έχει αποδείξει αν τελειώνει ή δεν τελειώνει σε κάθε περίπτωση