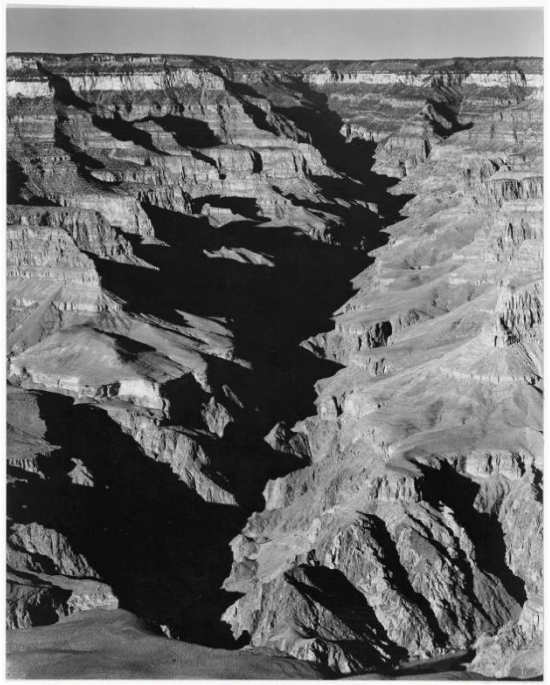
Παρουσίαση για χρήση με το σύγγραμμα, Αλγόριθμοι Σχεδίαση και Εφαρμογές, των M. T. Goodrich and R. Tamassia, Wiley, 2015 (στα ελληνικά από εκδόσεις Μ. Γκιούρδας)

Διαίρει και βασίλευε



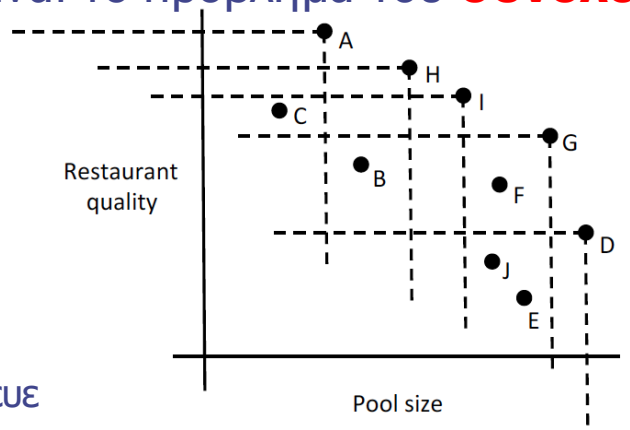
Grand Canyon from South Rim, 1941. Ansel Adams. U.S. government image. U.S. National Archives and Records Administration.

Εφαρμογή: Σύνολο μεγίστων

- Ας υποθέσουμε ότι εξετάζουμε ξενοδοχεία και ότι τα δύο μοναδικά κριτήρια που μας ενδιαφέρουν είναι το **μέγεθος της πισίνας**, και το **σκορ ποιότητας του εστιατορίου** όπως έχει αξιολογηθεί από έναν οδηγό εστιατορίων.
- Συνεπώς, ένα σύνολο ξενοδοχείων μπορεί να αναπαρασταθεί, αποτυπώνοντας κάθε ξενοδοχείο ως ένα σημείο, (x, y), όπου x είναι το μέγεθος πισίνας και y είναι το σκορ ποιότητας του εστιατορίου.
- Λέμε ότι ένα τέτοιο σημείο είναι **μέγιστο (ή μη κυριαρχούμενο)** σ' ένα σύνολο αν δεν υπάρχει άλλο σημείο, (x', y'), στο συγκεκριμένο σύνολο έτσι ώστε $x \leq x'$ και $y \leq y'$.
- Τα μη κυριαρχούμενα σημεία είναι οι πιθανές καλύτερες επιλογές βάσης αυτών των δύο διαστάσεων και η εύρεση τους είναι το πρόβλημα του **συνόλου μεγίστων (ή pareto layer)**.

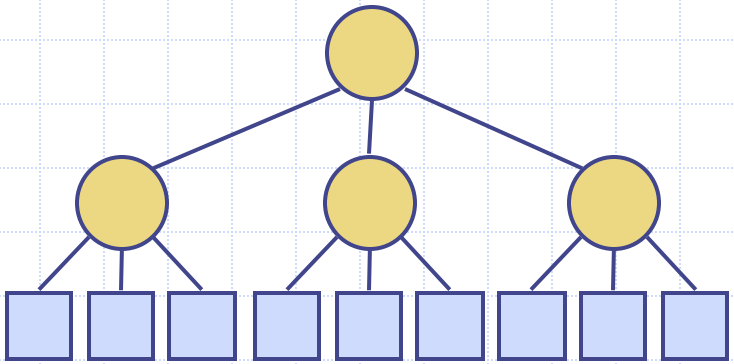
Μπορούμε να βρούμε αποδοτικά όλα τα μέγιστα σημεία με διαίρει και βασίλευε.

Εδώ το σύνολο είναι το: {A,H,I,G,D}.



Διαίρει και βασίλευε

- Το διαίρει και βασίλευε είναι ένα γενικό παράδειγμα αλγοριθμικής σχεδίασης:
  - Διαίρει: διαχωρισμός των δεδομένων εισόδου S σε δύο ή περισσότερα διακριτά υποσύνολα S_1, S_2, …
  - Βασίλευε: επίλυση των υπό-προβλημάτων αναδρομικά.
  - Συγχώνευσε: συγχώνευση των λύσεων S_1, S_2, …, σε μία λύση για το S.
- Η βασική περίπτωση για την αναδρομή είναι υπό-προβλήματα σταθερού μεγέθους.
- Η ανάλυση των αλγορίθμων αυτών μπορεί να γίνει με αναδρομικές εξισώσεις.



Ταξινόμηση με συγχώνευση (επανάληψη)

- Η ταξινόμηση με συγχώνευση (merge-sort) σε ακολουθία εισόδου S με n στοιχεία:
  - Διαίρει: κατάτμηση του S σε δύο ακολουθίες S_1 και S_2 περίπου $n/2$ στοιχείων η κάθε μια.
  - Βασίλευε: αναδρομική ταξινόμηση των S_1 και S_2.
  - Συγχώνευση: συγχώνευση των S_1 και S_2 σε μία μοναδική ταξινομημένη ακολουθία.

Algorithm *mergeSort*(S)
  Input sequence S with n elements
  Output sequence S sorted according to C
  if $S.size() > 1$
    $(S_1, S_2) \leftarrow partition(S, n/2)$
    mergeSort(S_1)
    mergeSort(S_2)
    $S \leftarrow merge(S_1, S_2)$

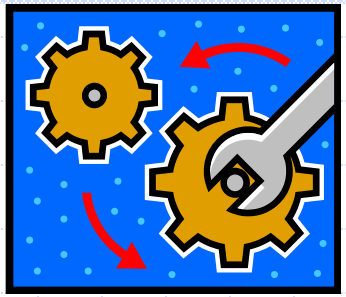
Ανάλυση της αναδρομικής εξίσωσης

- Το βήμα «συγχώνευση» της merge-sort περιλαμβάνει τη συνένωση δύο ταξινομημένων ακολουθιών, με $n/2$ στοιχεία η καθεμία, σε μια ταξινομημένη ακολουθία και χρειάζεται τουλάχιστον bn βήματα, όπου b είναι μια σταθερά.
- Ομοίως, η βασική περίπτωση ($\text{όταν } n < 2$) χρειάζεται το πολύ b βήματα.
- Έτσι, αν $T(n)$ είναι ο χρόνος του merge-sort:

$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(n/2) + bn & \text{if } n \geq 2 \end{cases}$$

- Μπορούμε έτσι να αναλύσουμε το χρόνο της merge-sort βρίσκοντας μία εξίσωση κλειστής μορφής για την παραπάνω εξίσωση (δηλαδή μία εξίσωσης που το $T(n)$ βρίσκεται μόνο στην αριστερή μεριά).

Επαναληπτική αντικατάσταση

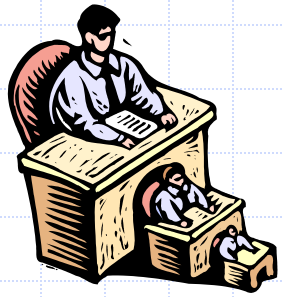


- Στην μέθοδο της επαναληπτικής αντικατάστασης (iterative substitution), ή αλλιώς μέθοδο “plug-and-chug”, εφαρμόζουμε επαναληπτικά την αναδρομική εξίσωση στον εαυτό της και προσπαθούμε να βρούμε ένα μοτίβο (pattern):

$$\begin{aligned} T(n) &= 2T(n/2) + bn \\ &= 2(2T(n/2^2)) + b(n/2)) + bn \\ &= 2^2 T(n/2^2) + 2bn \\ &= 2^3 T(n/2^3) + 3bn \\ &= 2^4 T(n/2^4) + 4bn \\ &= \ldots \\ &= 2^i T(n/2^i) + ibn \end{aligned}$$

- Σημειώστε ότι η βασική περίπτωση, T(n)=b, συμβαίνει όταν $2^i=n$. Δηλαδή, i = log n.
- Έτσι, $T(n) = bn + bn \log n$
- Οπότε, η T(n) είναι O(n log n).

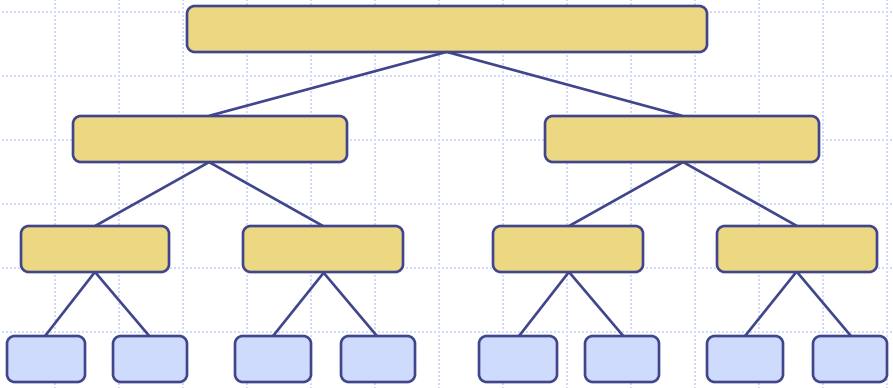
Το δένδρο αναδρομής



- Σχεδιάστε το δένδρο αναδρομής για την αναδρομική εξίσωση και αναζήτηση ενός μοτίβου:

$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(n/2) + bn & \text{if } n \geq 2 \end{cases}$$

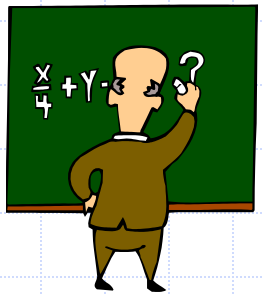
Βάθος	Τ's	Μέγεθος	Χρόνος
0	1	n	bn
1	2	$n/2$	bn
i	2^i	$n/2^i$	bn
…	…	…	…



Συνολικός χρόνος = $bn + bn \log n$

(τελευταίο επίπεδο συν όλα τα προηγούμενα επίπεδα)

Μέθοδος εικασίας και ελέγχου



- Στην μέθοδο εικασίας και ελέγχου (guess and test), κάνουμε μία εικασία για τη λύση κλειστής μορφής και μετά προσπαθούμε να την αποδείξουμε με επαγωγή:

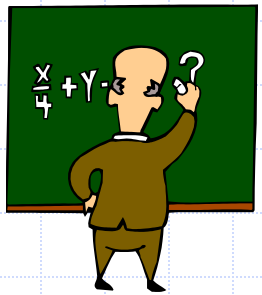
$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(n/2) + bn\log n & \text{if } n \geq 2 \end{cases}$$

- Εικασία: T(n) < cn log n.

$$\begin{aligned} T(n) &= 2T(n/2) + bn\log n \\ &= 2(c(n/2)\log(n/2)) + bn\log n \\ &= cn(\log n - \log 2) + bn\log n \\ &= cn\log n - cn + bn\log n \end{aligned}$$

- Λάθος: δεν μπορούμε να κάνουμε την τελευταία γραμμή να είναι λιγότερο από cn log n

Μέθοδος εικασίας και ελέγχου (συν.)



- Θυμηθείτε την αναδρομική εξίσωση:

$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(n/2) + bn\log n & \text{if } n \geq 2 \end{cases}$$

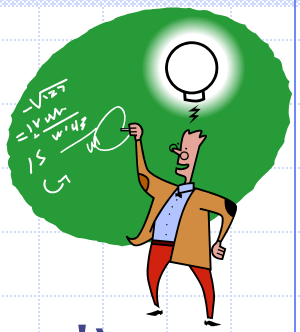
- Εικασία #2: $T(n) < cn\log^2 n$.

$$\begin{aligned} T(n) &= 2T(n/2) + bn\log n \\ &= 2(c(n/2)\log^2(n/2)) + bn\log n \\ &= cn(\log n - \log 2)^2 + bn\log n \\ &= cn\log^2 n - 2cn\log n + cn + bn\log n \\ &\leq cn\log^2 n \end{aligned}$$

  - Αν $c > b$.

- Έτσι, η $T(n)$ είναι $O(n\log^2 n)$.
- Γενικά, για να χρησιμοποιήσετε την μέθοδο, χρειάζεται να κάνετε μια καλή εικασία και πρέπει να είστε καλοί στις αποδείξεις με επαγωγή.

Η Μάστερ μέθοδος



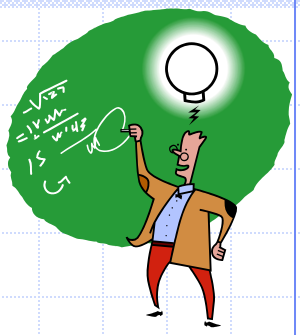
- Πολλές αναδρομικές εξισώσεις του διαίρει και βασίλευε έχουν την μορφή:

$$T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$$

- Το Μάστερ θεώρημα:

1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$
3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$,
   provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.

Μάστερ μέθοδος, Παράδειγμα 1

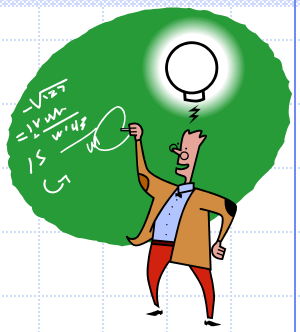


- Η μορφή: $$T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$$
- Το Μάστερ θεώρημα:
  1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
  2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$
  3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$, provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.
- Παράδειγμα:

$$T(n) = 4T(n/2) + n$$

Λύση : $\log_b a = 2$, η περίπτωση 1 λέει ότι η T(n) είναι $O(n^2)$.

Μάστερ μέθοδος, Παράδειγμα 2

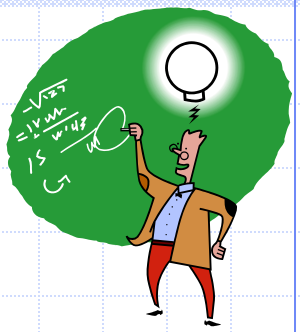


- Η μορφή: $T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$
- Το Μάστερ θεώρημα:
  1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
  2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$
  3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$, provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.
- Παράδειγμα:

$$T(n) = 2T(n/2) + n \log n$$

Λύση: $\log_b a = 1$, η περίπτωση 2 λέει ότι η T(n) είναι $O(n \log^2 n)$.

Μάστερ μέθοδος, Παράδειγμα 3

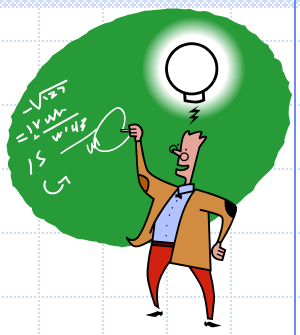


- Η μορφή: $$T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$$
- Το Μάστερ θεώρημα:
  1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
  2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$
  3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$, provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.
- Παράδειγμα:

$$T(n) = T(n/3) + n \log n$$

Λύση: $\log_b a = 0$, η περίπτωση 3 λέει ότι η T(n) είναι O(n log n).

Μάστερ μέθοδος, Παράδειγμα 4

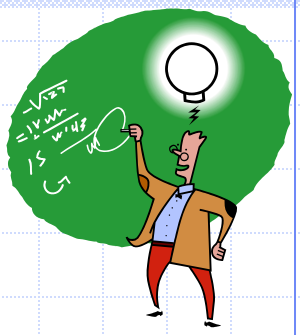


- Η μορφή: $$T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$$
- Το Μάστερ θεώρημα:
  1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
  2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$
  3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$, provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.
- Παράδειγμα:

$$T(n) = 8T(n/2) + n^2$$

Λύση: $\log_b a = 3$, η περίπτωση 1 λέει ότι η T(n) είναι $O(n^3)$.

Μάστερ μέθοδος, Παράδειγμα 5

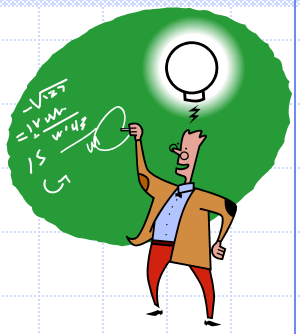


- Η μορφή: $T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$
- Το Μάστερ θεώρημα:
  1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
  2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$
  3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$, provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.
- Παράδειγμα:

$$T(n) = 9T(n/3) + n^3$$

Λύση: $\log_b a = 2$, η περίπτωση 3 λέει ότι η T(n) είναι $O(n^3)$.

Μάστερ μέθοδος, Παράδειγμα 6

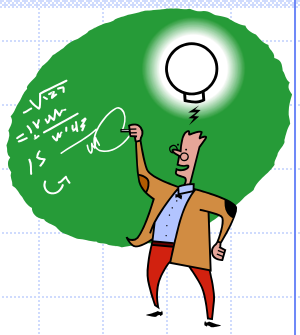


- Η μορφή: $T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$
- Το Μάστερ θεώρημα:
  1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
  2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$
  3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$, provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.
- Παράδειγμα:

$$T(n) = T(n/2) + 1$$ (δυαδική αναζήτηση)

Λύση: $\log_b a = 0$, η περίπτωση 2 λέει ότι η T(n) είναι O(log n).

Μάστερ μέθοδος, Παράδειγμα 7

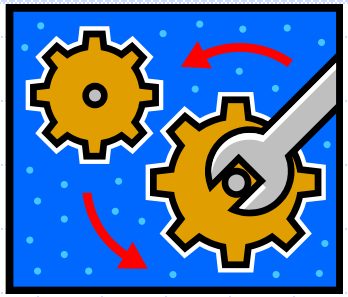


- Η μορφή: $$T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$$
- Το Μάστερ θεώρημα:
  1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
  2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$
  3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$, provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.
- Παράδειγμα:

$$T(n) = 2T(n/2) + \log n$$ (κατασκευή σωρού)

Λύση: $\log_b a = 1$, η περίπτωση 1 λέει ότι η T(n) είναι O(n).

Αιτιολόγηση υψηλού επιπέδου του Μάστερ θεωρήματος

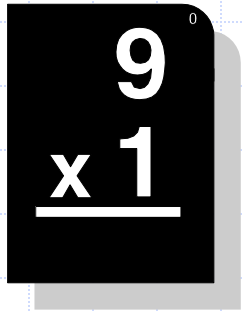


- Εφαρμόζουμε τη μέθοδο επαναληπτικής αντικατάστασης, ας δούμε αν μπορούμε να βρούμε κάποιο μοτίβο

$$
\begin{aligned}
T(n) &= aT(n/b) + f(n) \\
&= a(aT(n/b^2)) + f(n/b)) + bn \\
&= a^2T(n/b^2) + af(n/b) + f(n) \\
&= a^3T(n/b^3) + a^2 f(n/b^2) + af(n/b) + f(n) \\
&= \ldots \\
&= a^{\log_b n}T(1) + \sum_{i=0}^{(\log_b n)-1} a^i f(n/b^i) \\
&= n^{\log_b a}T(1) + \sum_{i=0}^{(\log_b n)-1} a^i f(n/b^i)
\end{aligned}
$$

- Προκύπτουν τρείς περιπτώσεις
  - Ο πρώτος όρος κυριαρχεί
  - Κάθε όρος του αθροίσματος είναι εξίσου κυρίαρχος
  - Το άθροισμα είναι μία γεωμετρική σειρά

Πολλαπλασιασμός ακεραίων



- Αλγόριθμος: Πολλαπλασιασμός δύο n-bit ακεραίων, των I και J.
  - Βήμα διαίρεσης: διαχώρισε τα I και J σε high-order bits και low-order bits

$$I = I_h 2^{n/2} + I_l$$

$$J = J_h 2^{n/2} + J_l$$

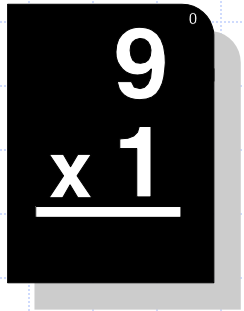
  - Μπορούμε να ορίσουμε το I * J πολλαπλασιάζοντας τα μέρη και προσθέτοντας:

$$I * J = (I_h 2^{n/2} + I_l) * (J_h 2^{n/2} + J_l)$$

$$= I_h J_h 2^n + I_h J_l 2^{n/2} + I_l J_h 2^{n/2} + I_l J_l$$

  - Έτσι, **T(n) = 4T(n/2) + n**, που υπονοεί ότι η T(n) είναι $O(n^2)$.
  - Αλλά αυτό δεν είναι καλύτερο από τον αλγόριθμο που μάθαμε στο δημοτικό.

Ένας βελτιωμένος αλγόριθμος πολλαπλασιασμού ακεραίων



- Αλγόριθμος: Πολλαπλασιασμός δύο ακεραίων n-bits, I και J.
  - Βήμα διαίρει: Διαχωρισμός των I και J σε bits υψηλής και χαμηλής τάξης

$$I = I_h 2^{n/2} + I_l$$

$$J = J_h 2^{n/2} + J_l$$

μόνο 3 διαφορετικοί πολλαπλασιασμοί πινάκων

  - Παρατηρήστε ότι υπάρχει ένας διαφορετικός τρόπος για να πολλαπλασιάσουμε τα μέρη:

$$\begin{aligned} I * J &= I_h J_h 2^n + [(I_h - I_l)(J_l - J_h) + I_h J_h + I_l J_l]2^{n/2} + I_l J_l \\ &= I_h J_h 2^n + [(I_h J_l - I_l J_l - I_h J_h + I_l J_h) + I_h J_h + I_l J_l]2^{n/2} + I_l J_l \\ &= I_h J_h 2^n + (I_h J_l + I_l J_h)2^{n/2} + I_l J_l \end{aligned}$$

  - Έτσι, **T(n) = 3T(n/2) + n**, που υπονοεί ότι η T(n) είναι $O(n^{\log_2 3})$, λόγω του Μάστερ θεωρήματος.
  - Έτσι, η T(n) είναι $O(n^{1.585})$.

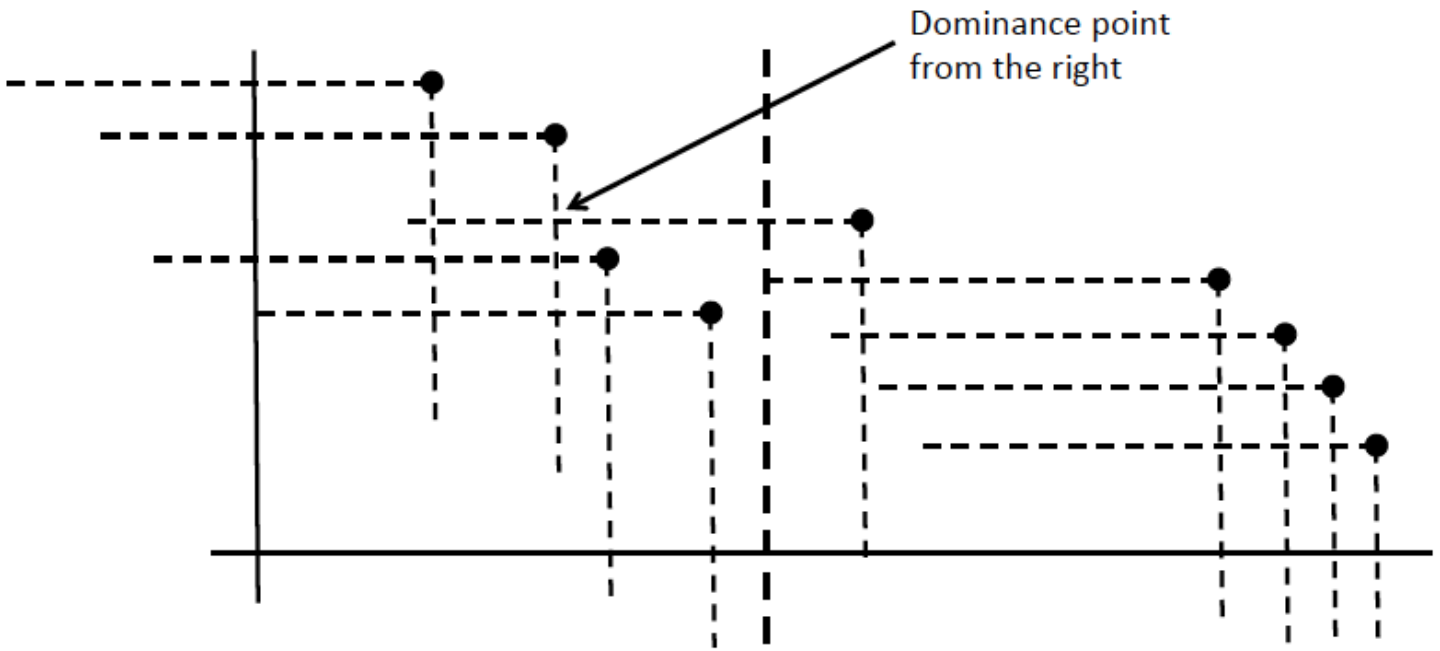
Επιλύοντας το πρόβλημα του Μέγιστου συνόλου

- Ας επιστρέψουμε στο πρόβλημα της εύρεσης του **μέγιστου συνόλου** για ένα σύνολο, S, με n σημεία στο επίπεδο.
- Η αποδοτική επίλυση του προβλήματος παρουσιάζει ενδιαφέρον για την πολυκριτηριακή βελτιστοποίηση στην οποία επιδιώκεται η βελτιστοποίηση η οποία στηρίζεται σε πολλές μεταβλητές.
- Για παράδειγμα, στην εισαγωγή χρησιμοποιήσαμε το παράδειγμα της επιλογής ξενοδοχείων με βάση δύο μεταβλητές, το μέγεθος της πισίνας και το σκορ ποιότητας του εστιατορίου.
- Ένα σημείο είναι **μέγιστο σημείο** στο S αν δεν υπάρχει άλλο σημείο, (x', y'), στο S έτσι ώστε $x \leq x'$ και $y \leq y'$.

Διαίρει και βασίλευε επίλυση

- Για ένα σύνολο σημείων, **S**, με **n** σημεία στο επίπεδο, υπάρχει ένας απλός αλγόριθμος διαίρει και βασίλευε για την κατασκευή του **μέγιστου συνόλου** του **S**.
- Αν **n ≤ 1**, το σύνολο μεγίστων είναι απλώς το ίδιο το **S**.
- Αλλιώς, έστω ότι το **p** είναι το διάμεσο σημείο διατάσσοντας λεξικογραφικά τα σημεία του **S**, διατάσσοντας πρώτα κατά τις x συντεταγμένες και έπειτα κατά τις y συντεταγμένες αν υπάρχουν ισοβαθμίες.
- Στη συνέχεια λύνουμε αναδρομικά το **πρόβλημα μεγίστου συνόλου** για το σύνολο σημείων αριστερά αυτής της γραμμής καθώς και επίσης για τα σημεία στα δεξιά της γραμμής.
- Δεδομένων αυτών των λύσεων, το **σύνολο μεγίστων** σημείων στα δεξιά είναι επίσης **μέγιστα σημεία** για το **S**.
- Αλλά κάποια απ' αυτά τα **μέγιστα σημεία** για το αριστερό σύνολο μπορεί να υστερούν από ένα σημείο στα δεξιά, όπως το σημείο, **q**, που είναι πλέον αριστερά.
- Μπορούμε τότε να ελέγξουμε το **αριστερό σύνολο των μέγιστων** και να αφαιρέσουμε τα σημεία που κυριαρχούνται απ' το **q**, μέχρι να φτάσουμε στο σημείο όπου επεκτείνεται η κυριαρχία του **q**.
- Η ένωση του υπόλοιπου **συνόλου μεγίστων** από τα αριστερά και του **συνόλου μέγιστων** από τα δεξιά είναι το **σύνολο των μέγιστων** για το **S**.

Παράδειγμα του βήματος συγχώνευσης



Ψευδό-κώδικας

Algorithm MaximaSet(S):

Input: A set, S, of n points in the plane
Output: The set, M, of maxima points in S

if $n \leq 1$ **then**
    return S
Let p be the median point in S, by lexicographic (x, y)-coordinates
Let L be the set of points lexicographically less than p in S
Let G be the set of points lexicographically greater than or equal to p in S
$M_1 \leftarrow$ MaximaSet(L)
$M_2 \leftarrow$ MaximaSet(G)
Let q be the lexicographically smallest point in M_2
for each point, r, in M_1 **do**
    if $x(r) \leq x(q)$ **and** $y(r) \leq y(q)$ **then**
        Remove r from M_1
return $M_1 \cup M_2$

Μια μικρή λεπτομέρεια υλοποίησης

- Για να αναλύσουμε τον αλγόριθμο διαίρει και βασίλευε μέγιστου συνόλου, υπάρχει μια λεπτομέρεια υλοποίησης που πρέπει να αποσαφηνίσουμε.
- Υπάρχει το πρόβλημα της αποτελεσματικής εύρεσης του σημείου, p, που είναι το διάμεσο σημείο σε μια λεξικογραφική διάταξη των σημείων του S σύμφωνα με τις συντεταγμένες τους (x, y).
- Υπάρχουν δύο άμεσες πιθανότητες:
- Μία επιλογή είναι να χρησιμοποιήσουμε έναν αλγόριθμο εύρεσης διαμέσου τιμής γραμμικού χρόνου, όπως αυτός που περιγράφεται στην Ενότητα 9.2. Το αποτέλεσμα είναι ένας καλός ασυμπτωτικός χρόνος εκτέλεσης, αλλά αυξάνεται η πολυπλοκότητα της υλοποίησης.
- Μια άλλη επιλογή είναι να ταξινομήσουμε τα σημεία στο S λεξικογραφικά κατά τις συντεταγμένες τους (x, y) ως ένα βήμα προ-επεξεργασίας, πριν καλέσουμε τον αλγόριθμο MaximaSet στο S. Δεδομένου αυτού του βήματος προ-επεξεργασίας, το διάμεσο σημείο είναι απλώς εκείνο το σημείο, που βρίσκεται στο μέσο της λίστας.

Ανάλυση

- Σε κάθε περίπτωση, τα υπόλοιπα μη αναδρομικά βήματα μπορούν να εκτελεστούν σε χρόνο O(n), και συνεπώς ο χρόνος εκτέλεσης για τον αλγόριθμο διαίρει και βασίλευε μέγιστου συνόλου δίνεται από τον ακόλουθο τύπο (όπου το b είναι σταθερά):

$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(n/2) + bn & \text{if } n \geq 2 \end{cases}$$

- Έτσι, ακολουθώντας το Μάστερ θεώρημα, ο αλγόριθμος είναι O(n log n).