Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Αλγόριθμοι και Πολυπλοκότητα

1η Σειρά Αναλυτικών Ασκήσεων στο Μάθημα "Αλγόριθμοι και Πολυπλοκότητα"

Άσκηση 1:

α)

$$\sum_{k=1}^{n} k2^{-k}=\Theta(1)<2^{((loglogn)^4)}<\log(\binom{2n}{n})\approx n2^{(2^{\log n})}=\Theta(n)<\frac{\log(n!)}{(loglogn)^5}<(\sqrt{(n)})^{(loglogn)}<n\sum_{k=1}^{n}\binom{n}{k}\approx\sum_{k=1}^{n}k2^k=\Theta(n2^n)$$

β)Για όλες τις παρακάτω περιπτώσεις, όπως έχει ειπωθεί και στο μάθημα θεωρούμε ότι ισχύει:

f(n) > af(n/b), όπου f η συνάρτηση κόστους σε κάθε αναδρομή

Επομένως, προκύπτουν τα παρακάτω αποτελέσματα:

1. $T(n)=6T(n/3)+n^2logn$

Από Μ.Τ:

a=6 , b = 3 → #φύλλα = $nn^{(\log_3 2)}$

$f(n)=n^2logn \rightarrow f(n)>\varphi\acute{\upsilon}\lambda\lambda\alpha \rightarrow T(n)=\Theta(n^2logn)$

2. $T(n)=9T(n/3)+n^2logn$

$\varphi\acute{\upsilon}\lambda\lambda\alpha=n^2$

$f(n)=n^2logn \rightarrow f(n)>\varphi\acute{\upsilon}\lambda\lambda\alpha \rightarrow T(n)=\Theta(n^2logn)$

3. $T(n)=11T(n/3)+n^2 logn$

$\varphi\upsilon\lambda\lambda\alpha=n^{(\log_3 11)}(1)$
$11<12\rightarrow \log_3 11<\log_3 12=\log_3 4+1\rightarrow n^{(\log_3 11)}<nn^{(\log_3 4)}<n^2 logn\rightarrow T(n)=\Theta(n^2 logn)$

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

$\gamma 1=1/4, \gamma 2=1/2\rightarrow \gamma 1+\gamma 2=3/4<1-\varepsilon, \varepsilon>0$

Επομένως, έχουμε :

T(n) = n = Θ(n)

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

γ1 = ¼, γ2 = ¼ , γ3 = ½ → γ1+γ2+γ3 = 1 → T(n) = Θ(nlogn)

6. $T(n)=T(n^{(2/3)})+\Theta(logn)$

Κάνοντας το δέντρο της αναδρομής προκύπτει ότι πρόκειται για μια φθίνουσα γεωμετρική σειρά.

Άρα:

$$T(n)=\sum_{k=0}^{loglogn} \log(n^{((2/3)^k)})=\sum_{k=0}^{loglogn}(2/3)^k logn=logn\sum_{k=0}^{loglogn}(2/3)^k=logn\left(\frac{1}{1-\frac{2}{3}}\right)=3logn\rightarrow T(n)=\Theta(logn)$$

Παρατήρηση:

Μπορεί κανείς να προκύψει στο ίδιο αποτέλεσμα, αν θεωρήσει, ότι το δέντρο συνεχώς “εξειδικευεται”, μείωνοντας την είσοδο. Επομένως, αφού το δέντρο “στενεύει”, τα φύλλα μικραίνουν, οπότε απο ΜΤ έχουμε το παραπάνω αποτέλεσμα.

7. $T(n)=T(n/3)+\sqrt{(n)}$

$n^{(\log_3 1)}=1$

$f(n)=\sqrt{(n)}$

Επομένως, $T(n)=\Theta(\sqrt{(n)})$

Έστω, ε > 0, ώστε $n^{(\log_3 1+\varepsilon)}=n^{\varepsilon}$

$\sqrt{(n)} > n^{\varepsilon} \leftrightarrow 1/2 > \varepsilon \leftrightarrow \varepsilon \in (0,1/2)$

Επομένως, για τυχαίο ε στο (0,1/2) η f(n) υπερισχύει πολυωνυμικά, τότε από master theory καταλήγουμε στο παραπάνω αποτέλεσμα.

Άσκηση 2:

α) Ο αλγόριθμος για το 1ο ζήτημα αποτελείται από τα εξής βήματα:

- Βρες το μεγαλύτερο στοιχείο στον πίνακα των i-στοιχείων, $i \in [1, n]$
- Με μια προθεματική περιστροφή k-θέσεων, όσο και η θέση του, τοποθετησέ το στην αρχή, $k \in [1, i]$
- Με μια προθεματική περιστροφή i-θέσεων, τοποθετησέ το στο τέλος
- Στην επόμενη ανακύκλωση μείωσε τον αριθμό των στοιχείων κατά 1, και επανάλαβε την διαδικασία στον υποπίνακα i-1 στοιχείων, από την 1η θέση του πίνακα μέχρι την i-1 θέση .

Η παραπάνω διαδικασία στην χειρότερη πρίπτωση απαιτεί:

- 2 περιστροφές για τα n-2 στοιχεία του αρχικού πίνακα
- 1 περιστροφή για την ταξινόμηση ενός πίνακα 2 στοιχείων, ο οποίος προφανώς ταξινομείται προφανώς με το πολύ 1 προθεματική περιστροφή

Για να γίνει πι κατανοητό, δίνεται ένα παράδειγμα. Έστω, ότι έχουν ταξινομηθεί τα n-2 στοιχεία και έχουμε προκύψει στον πίνακα, με στοιχεία τα 1,2.

Τότε η χειρότερη περίπτωση είναι, ο τελευταίος υποπίνακας να έχει την παρακάτω μορφή:

- [2 , 1]
- →
- [1 , 2]

Με μια περιστροφή ταξινομήθηκε. Επομένως, καταλήγουμε ότι, για την συνάρτηση κόστους ισχύει:

$$f(n) = 2(n - 2) + 1 = 2n - 3 = O(2n-3) = O(n)$$

Στη συγκεκριμένη περίπτωση, μας ενδιαφέρουν οι σταθερές οπότε έχουμε:

$f(n) = O(2n-3)$

β) Ο αλγόριθμος για το 2ο ζήτημα περιγράφεται από τα εξής βήματα:

- Βρίσκουμε το στοιχείο με την μεγαλύτερη απόλυτη τιμή
- Το τοποθετούμε στην αρχή με 1 προσημασμένη προθεματική περιστροφή
- Αν είναι θετικό, απαιτούνται 2 περιστροφές για να τοποθετηθεί στο τέλος
- Αν είναι αρνητικό, απαιτείται 1 περιστροφή για να τοποθετηθεί στο τέλος
- Το πλήθος του πίνακα μειώνεται κατά 1, και επαναλαμβάνουμε την διαδικασία για τα n-2 στοιχεία στους αντίστοιχους υποπίνακες

- Ο πίνακας , με τα στοιχεία 1,2 , ταξινομείται με το πολύ 4 περιστροφές

Στην χειρότερη περίπτωση, ο παραπάνω αλγόριθμος απαιτεί 3n-3 περιστροφές, όπως υποστηρίζει η παρακάτω ανάλυση:

- Στην πρώτη ανακύκλωση απαιτούνται το πολύ 2 περιστροφές, καθώς όλα τα στοιχεία είναι θετικά
- Για τα επόμενα n-3 στοιχεία, απαιτούνται το πολύ 3 περιστροφές
- Για τα 2 τελευταία το πολύ 4 περιστροφές

Επομένως, έχουμε:

$$f(n) = 3(n-3) + 2 + 4 = 3n-3 \rightarrow$$

$$\mathbf{f(n) = O(3n-3)}$$

Η χειρότερη περίπτωση συναντάται σε πίνακες της μορφής:

[n n-1 … 4 3 2 1] →
[-n n-1 … 4 3 2 1]→
[-1 -2 -3 -4 …-(n-1) n]→
[-1 -2 -3 -4 …-(n-1)]→(μείωση μεγέθους πίνακα)
[n-1…. 4 3 2 1]→
[-(n-1) … 4 3 2 1]→
[-1 -2 -3 -4 …. n-1]→
[-1 -2 -3 -4 …. -(n-2)]→(μείωση μεγέθους πίνακα)
.
.
.
.
.
[-1 -2] →
[2 1] →

[-2 1] →
[-1 2] →
[1 2] Done!!

γ)

1) Σύμφωνα με την εκφώνηση, κάποιο συμβατό ζεύγος ακεραίων περιγράφεται από τα παρακάτω χαρακτηριστικά:

1) Οι απόλυτες τιμές των 2 αριθμών να διαφέρουν κατά 1(είναι διαδοχικοί)
2) Οι αριθμοί να είναι ταξινομημένοι και ομόσημοι στον πίνακα

Με τις παραπάνω θεωρήσεις προκύπτουν οι εξής 2 περιπτώσεις:

1) *"Τουλάχιστον ένας αριθμός είναι θετικός. Έστω p ο μεγαλύτερος από τους θετικούς, τότε:"*

   Ο -(p+1) θα βρίσκεται :
   i)είτε πριν το p στον πίνακα
   ii)είτε μετά το p στον πίνακα
   iii) είτε p=n(max)

i. <u>Το -(p+1) βρίσκεται πριν το p στον πίνακα:</u>
   - Με 1 περιστροφή το p τοποθετείται στην αρχή
   - Με 1 περιστροφή το -p τοποθετείται μια θέση πριν το p+1 με το σωστό πρόσημο

   - Για παράδειγμα:
     [….-(p+1)….p....]→
     [-p….(p+1)….]→
     [….p (p+1)….] Done!

ii. <u>Το -(p+1) βρίσκεται μετά το p στον πίνακα:</u>
   - Με μια περιστροφή το -(p+1) τοποθετείται στην αρχή
   - Με μια περιστροφή το (p+1) τοποθετείται μια θέση πριν το -p με το σωστό πρόσημο

   - Για παράδειγμα:
     [….p….-(p+1)….]→
     [(p+1)….-p….]→
     [….-(p+1) p….] Done!

iii. <u>Το p=n:</u>

- Με 2 περιστροφές τοποθετείται στο τέλος

- Για παράδειγμα:
  - [….n….]→
  - [-n….]→
  - [….n] Done!

2)"Όλοι οι αριθμοί είναι αρνητικοί."

i. *Έστω -p ένας αριθμός με την ιδιότητα, ο -(p+1) να είναι πριν από τον p στον πίνακα.*

- Με 1 περιστροφή τοποθετείται στην αρχή
- Με μια 1 περιστροφή τοποθετείται στην προηγούμενη θέση του πίνακα, όπου ανήνει ο -p

- Για παρέδειγμα:

  [...-(p+1)….-p….]→
  [(p+1)….-p….]→
  [….-(p+1) -p….] Done!

ii. *Για όλα τα -p το -(p+1) είναι μετά το -p. Δηλαδή θα έχουμε έναν πίνακα της μορφής:*

[-1 -2 -3….-n]

Επομένως, αποδείχτηκε ότι για όλες τις μορφές των Ατ(ενδιάμεσων πινάκων) εκτός του πίνακα [-1 -2 -3….-n], σε το πολύ 2 περιστροφές μπορεί να δημιουργηθεί ένα συμβατό ζεύγος.

2)Χρησιμοποιώντας το παραπάνω λήμμα, μπορούμε να βρούμε κατάλληλο αλγόριθμο ταξινόμησης του πίνακα με πολυπλοκότητα O(2n).

Ο αλγόριθμος αποτελείται από τα παρακάτω βήματα:

- Δημιουργούμε ένα συμβατό ζεύγος με τους παραπάνω κανόνες(κάθε ζεύγος απαιτεί το πολύ 2 περιστροφές)
- Συμπτίσουμε το ζεύγος σε έναν από τους δύο αριθμούς(διότι δεν το ξανά επεξεργαζόμαστε)
- Μειώνουμε τον αριθμό των στοιχείων του πίνακα κατά 1
- Στον νέο πίνακα επαναλαμβάνουμε την διαδικασία
- Μετά από n επαναλήψεις, ο πίνακας είναι ταξινομημένος

Η αναλλοίωτη, που εγγυάται πάντα την ορθότητα του αλγορίθμου, έιναι ότι:
- **Δεν πειράζουμε** συμβατά ζέυγη
- Ο αρχικός πίνακας περιέχει θετικά στοιχεία, οπότε μετά από το πολύ 2n περιστροφές τα στοιχεία θα ναι σίγουρα όλα θετικά

Επίσης, κατά τη διάρκεια του αλγορίθμου ίσως, προκύψει πίνακας της μορφής:

[-(i+1) -(i+2) -(i+3) …. -(n-1) -n...1 2 3... i-1 i] (1)

Παρατηρούμε ότι με τα παρακάτω βήματα ταξινομείται κάθε φορά 1 στοιχείο:

- Flip n
- Flip n-1
- Για παράδειγμα ο πίνακας (1) γίνεται:

[-(i+1) -(i+2) -(i+3) …. -(n-1) -n...1 2 3... i-1 i] →
[-i -(i-1) ….-3 -2 -1 ….n (n-1)….(i+3) (i+2) (i+1)]→
[-(i+2) -(i+3) …. -(n-1) -n...1 2 3 ...i-1 i (i+1)]

Επομένως, εκτελώντας n φορές τα παραπάνω βήματα ο πίνακας ταξινομείται.

Έστω , ότι προκύπτει η μορφή του (1) στα ενδιάμεσα στάδια, τότε θα ‘χουμε:

- 2i το πολύ περιστροφές για τα πρώτα i στοιχεία του πίνακα
- 2(n-i) περιστροφές ακριβώς για τα n-i στοιχεία του πίνακα

f(n) = 2i + 2(n-i)= 2n

Άσκηση 3:

1) Μια λογική έως απλή λύση είναι να σκεφτεί κανείς ότι, εφόσον ζητείται ο μικρότερος αριθμός στιβών, πρέπει να φροντίζω η κάθε κάρτα που επιλέγεται να έχει περισσότερες πιθανότητες να τοποθετηθεί σε μια στίβα από το να δημιουργήσει μια νέα.

Αυτό επιτυγχάνεται, αν από κάθε σύνολο κορυφών S = {c' | c' > c}, επιλέγεται η στίβα με την μικρότερη κορυφή $c', c' \in S$, για να τοποθετηθεί η νέα κάρτα.

Η ορθότητα του αλγορίθμου αποδεικνύεται με την παραπάνω πρόταση.

Μια trivial υλοποίηση αυτής της ιδέας είναι ο παρακάτω αλγόριθμος:

- Τραβάω μια κάρτα
- Συγκρίνω το νούμερο με όλες τις κορυφές
- Επιλέγω την ελάχιστη κορυφή, από αυτές που μπορεί να τοποθετηθεί η κάρτα

Αυτός ο αλγόριθμος έχει προφανώς στη χειρότερη περίπτωση πολυπλοκότητα

$$f(n)=O(n^2)$$

Αφού:

$$f(n)=1+\sum_{i=2}^{n} i/2=\frac{1}{2}+\frac{1}{2}\sum_{i=1}^{n} i=\frac{1}{2}+\frac{1}{2}n\frac{(n+1)}{2}\rightarrow O(n^2)$$

Παρατηρώντας λίγο καλύτερα το πρόβλημα, κανείς μπορεί να προσέξει ότι, οι κορυφές των στιβών είναι πάντα ταξινομημένες σε αύξουσα σειρά(αυτό οφείλεται στη συνθήκη δημιουργίας νέας στίβας και στον αλγόριθμο που ακολουθούμε για να τοποθετούμε νέες κάρτες).

Επομένως, εφαρμόζοντας Binary Search μείωνεται η πολυπλοκότητα του προβήματος σε **O(nlogn).**

2) Λόγω των κανόνων του παιχνιδιού είναι προφανές ότι, κάθε στίβα θα ναι ταξινομημένη επομένως με μια απλή merge, μπορούμε να κάνουμε τη σύνθεση.

Άρα, εφαρμόζοντας πρώτα τον αλγόριθμο της 1 και μετά την merge καταλήγουμε:

f(n) = O(nlogn) + O(nlogn) = O(nlogn)

3) Με είσοδο την ακολουθία 3 2 4 7 8 1 5 6, ο αλγόριθμος της (1) εκτελεί τα παρακάτω βήματα:

3
---
3 2
---
3 2
4
---
3 2
4
7
---
3 2
4
7
8
---
3 2 1
4
7
8
---
3 2 1
4
7 5
8
---
3 2 1
4
7 5
8 6

Με βάση τον αλγόριθμο της (2) θα γίνει ταξινόμηση της τράπουλας, συνθέτοντας ανά δύο τις στίβες. Έχοντας, τις παραπάνω στίβες προκύπτει, σε χρόνο Θ(n)(αφού δεν είναι η χειρότερη περίπτωση):

1 2 3 4
5 6 7 8

---
1 2 3 4 5 6 7 8

Ο αριθμός των στιβών και το μήκος της μέγιστης αύξουσας υπακολουθίας εισόδου είναι 4.

4) Μπορούμε να κάνουμε την παρακάτω ανάλυση για να αποφανθούμε του ζητουμένου:

Έστω ότι βρισκόμαστε στον i-οστο γύρο, με λ_i στίβες, όπου $i \in [1, n], \lambda_i \in [0, i-1]$.

Στον i γύρο καταφτάνει το 1ο στοιχείο της μέγιστης αύξουσας υπακολουθίας, η οποία αποτελείται από τα α1 α2 α3 ….αk, $k \in [1, n]$ και μ_k η στίβα, στην οποία μπορεί να τοποθετηθεί το κάθε στοιχείο της υπακολουθίας, $\mu_k \in [1, \lambda_i + 1]$ (λόγω της δυνατότητας δημιουργίας).

Διακρίνουμε 2 περιπτώσεις:

α) Όλα τα στοιχεία της υπακολουθίας να ‘ρθουν διαδοχικά
β) Να υπάρχουν ενδιάμεσα στοιχεία από έναν όρο της ακολουθίας μέχρι τον άλλον

Και στις 2 περιπτώσεις ισχύει το παρακάτω:

Για το 1ο στοιχείο της υπακολουθίας:

$$\mu_1 \in [1, \lambda_i + 1] (1)$$

Για όλα τα υπόλοιπα είτε υπάρχουν ή όχι ενδιάμεσα ισχύει:

$$\mu_2 \geq \mu_1 + 1$$
$$\mu_3 \geq \mu_2 + 1$$
$$\vdots$$
$$\mu_\kappa - 1 \geq \mu_\kappa - 2 + 1$$
$$\mu_\kappa \geq \mu_\kappa - 1 + 1$$

Αθροίζοντας, προκύπτει:

$$\mu_\kappa \geq \kappa - 1 + \mu_1$$

Από (1):

$$\kappa \leq \mu_\kappa \leq \lambda_i$$

Άρα, η θέση του τελευταίου όρου της υπακολουθίας θα είναι τουλάχιστον k, άρα θα δημιουργηθούν τουλάχιστον k στίβες.]

5) Σύμφωνα με γνωστό αλγόριθμο(3), όπου βρίσκει το μήκος της μέγιστης αύξουσας υπακολουθίας σε χρόνο O(nlogn), μπορούμε να σημειώσουμε κάποιες ομοιότητες με το πρόβλημα.

Αρχικά, σε κάθε γύρο η κορυφή της κάθε στίβας, αποτελεί το "κεφάλι" της κάθε λίστας, που αποτελεί υποψήφια για την μέγιστη αύξουσα υπακολουθία στον αλγόριθμο(3).

Ο αλγόριθμος (3) χαρακτηρίζεται από τα 3 παρακάτω βήματα:

1. Αν το A[i] είναι μικρότερο από όλα τα κεφάλια, τότε δημιουργούμε μια νέα λίστα.
2. Αν το A[i] είναι μεγαλύτερο απ' όλα, τότε αντιγράφουμε την μεγαλύτερη κι της προσθέτουμε το A[i](αυξάνεται το μήκος κατά 1)
3. Αν είναι ανάμεσα, τότε βρίσκουμε τη λίστα της οπόιας το κεφάλι είναι το ceil value του A[i], την αντιγράφουμε και της προσθέτουμε το A[i], διαγράφουμε κάθε άλλη λίστα με το ίδιο μέγεθος

Μπορεί να δει κανείς πλέον, τις ομοιότητες:

1. Αν ένα στοιχείο είναι μικρότερο απ' όλες τις κορυφές, τότε το τοποθετώ στην πρώτη στίβα(και δεν με επηρεάζει στο μήκος της υπακολουθίας)
2. Αν είναι μεγαλύτερο απ' όλα, τότε δημιούγησε νέα(αύξησε το μήκος υπακολουθίας/αριθμό στιβών κατά 1)
3. Αν είναι ανάμεσα, τοποθέτησε το στοιχείο στην στίβα με το μικρότερο κεφάλι από τις στίβες που χωράει(ουσιαστικά σβήνει το προηγούμενο κεφάλι της στίβας κι αντιγραφει την ίδια στίβα με διαφορετικό κεφάλι, ώστε να είναι διαθέσιμο και για τιμές μικρότερες του προηγούμενου κεφαλιού)

Λόγω αυτής της αντιστοιχίας, ο αλγόριθμος του (1) μπορεί κι λύνει το πρόβλημα της υπακολουθίας.

Είναι αξιόλογο να σημειωθεί ότι, κατά τη διάρκεια του αλγορίθμου (1) βρίσκουμε την μέγιστη φθίνουσα υπακολουθία. Είναι λογικό, καθώς η συνθήκη τοποθέτησης σε συνδυασμό με τον κανόνα του (1), προσπαθούν να τοποθετήσουν όσα περισσότερα στοιχεία σε μια στίβα γίνεται, τα οποία είναι σε φθίνουσα σειρά.

Επομένως, το μήκος της μέγιστης αύξουσας υπακολουθίας ισούται με το μήκος της μεγαύτερης στίβας.

6)Σε μια τράπουλα nm+1 στοιχείων, το 1 στοιχείο μπορεί να θεωρηθεί το 1ο στοιχείο που δημιουργήσει την πρώτη στίβα.

Επομένως, μια τράπουλα nm στοιχείων μπορούμε να φανταστούμε την τράπουλα ως ένα πίνακα A από στοιχεία διάστασης n x m. Ο πίνακας καταλαμβάνει συγκεκριμένο όγκο στο χώρο. Επομένως, στην καλύτερη περίπτωση, που ο πίνακας να τοποθετηθεί αυτούσιος, επειδή έχουμε ήδη μια στίβα θα ισχύει σίγουρα 1 από τις 2 ισότητες(αν δεν τοποθετηθει αυτούσιος είναι πορφανές ότι θα ισχύει το μεγαλύτερο).

Πιο αναλυτικά(στην καλύτερη περίπτωση):

Έστω A_{nxm} :

```
        1   2    .    .    .    m
A =  1 [ a11   a12   .    .   a1n ]
     2 |  .     .    .    .    .|
     . |  .     .    .    .    .|
     . |  .     .    .    .    .|   +                  =>
     . |  .     .    .    .    .|          | c |
     n [ am1    .    .    .   anm ]        |___|
```

$A_{((n+1)xm)}$ ή $A_{(nx(m+1))}$

Το πρόβλημα ανάγεται στην **αρχή του περιστερώνα**, καθώς κάθε γραμμή περιέχει m στοιχεία. Επομένως, μόλις τελειώσουν ή θα χρησιμοποιήσουμε n+1 γραμμές ή 1 γραμμή θα χει m+1 στοιχεία.

Ένα τέτοιο παράδειγμα είναι:

5 4 3 2 1 10 9 8 7 6 15 14 13 12 11 20 19 18 17 16 25 24 23 22 21

<u>Άσκηση 4:</u>

α)
Έστω X_a_i(k), X_b_i(k) οι θέσεις των σωματιδίων a_i, b_i αντίστοιχα.
Αρχικά διακρίνουμε τις δυο ακραίες περιπτώσεις:

1. Τα σωματίδια a και τα σωματίδια β κινούνται με ταχύτητα V_{min}

Η εξίσωση που περιγράφει την κινησή τους σ'αυτή την περίπτωση είναι:

- $X_{(a_i(k))} = k V_{min}$
- $X_{(b_i(k))} = L - kV_{min}$

  Η σύγκρουση γίνεται στο σημείο:

  $X_{(a_i(k))} = X_{(b_i(k))} \rightarrow k_{min} = L/2 V_{min}$

2. Τα σωματίδια a και τα σωματίδια β κινούνται με ταχύτητα V_{max}

Η εξίσωση που περιγράφει την κινησή τους σ'αυτή την περίπτωση είναι:

- $X_{(a_i(k))} = k V_{max}$
- $X_{(b_i(k))} = L - kV_{max}$

  Η σύγκρουση γίνεται στο σημείο:

  $X_{(a_i(k))} = X_{(b_i(k))} \rightarrow k_{max} = L/2 V_{max}$

Επομένως, η σύγκρουση θα γίνει τη χρονική στιγμή $k \in [L/2V_{max}, L/2V_{min}]$

Ο αλγόριθμος επίλυσης του προβληματος είναι ο εξής:

1. Εκτελούμε Binary Search στο διάστημα, που ανήκει η σύγκρουση
2. Για κάθε χρονική στιγμή καλούμε τον YY1 για τα 2n στοιχεία
3. Βρίσκουμε το σωματίδιο με την μεγαλύτερη θέση από τα a, έστω X_i
4. Βρίσκουμε το σωματίδιο με τη μικρότερη θέση από τα b, έστω X_j
5. Συγκρίνουμε τα X_i, X_j
6. Αν X_i > X_j, τότε η σύγκρουση έχει γίνει σε προηγούμενη χρονική στιγμή, άρα κρατάμε μόνο το αριστερά μισό
7. Αν X_i < X_j, τότε η σύγκρουση έχει γίνει σε μεταγενέστερη χρονική στιγμή, άρα κρατάμε μόνο το δεξιά μισό
8. Αν X_i = X_j, τότε είναι η στιγμή σύγκρουσης

Η χειρότερη περίπτωση προφανλως του προβλήματος είναι να χουμε φτάσει στα φύλλα, δηλαδή να 'χουμε κάνει logn διαμερίσεις του χρόνου και να 'χει μείνει μία χρονική στιγμή. Επομένως, η πολυπλοκότητα του αλγορίθμου είναι:

$$T(n,L,V_{max},V_{min}) = T(n,\frac{1}{2V_{min}} - \frac{1}{2V_{max}}) = 2n + 2O(n) + T(n,\frac{(\frac{1}{2V_{min}} - \frac{1}{2V_{max}})}{2})$$

$$\rightarrow T(n,L,V_{max},V_{min}) = O(nlog(\frac{1}{2V_{min}} - \frac{1}{2V_{max}}))$$

β) Η ιδέα για τον παρακάτω αλγόριθμο είναι η εξής:

1. Επιλέγουμε ένα τυχαίο σωματίδιο a, έστω a_i
2. Βρίσκουμε την στιγμή συγκρουσής του με όλα τα b, και με σε O(n) βρίσκουμε τον μικρότερο χρόνο και το αντίστοιχο σωματίδιο, έστω b_j

- Κάνουμε τώρα την εξής υπόθεση ότι, δεν υπάρχουν άλλα σωματίδια b αριστερότερα της σύγκρουσης, η οποία επαληθεύεται καθώς θα ήταν αυτά που θα είχαν συγκρουστεί, επομένως επιθυμούμε να διώξουμε τα a που είναι αριστερότερα(1)

3. Για να γίνει αυτό κρατάμε το σωματίδιο b_j και βρίσκουμε τους χρόνους σύγκρουσης για όλα τα a. Έχοντας σαν pivot τον χρόνο της σύγκρουσης του (a_i, b_j) , διώχνουμε κάθε φορά τα σωματίδια a με μεγαλύτερο χρόνο σύγκρουσης.
4. Έπειτα βρίσκουμε το σωματίδιο a με τον μικρότερο χρόνο σύγκρουσης και κάνουμε το ίδιο, διώχνοντας τα b με μεγαλύτερο χρόνο σύγκρουσης, καθώς κατ' αναλογία θα ισχύει η υπόθεση (1)
5. Αυτή η εναλλαγή γίνεται έως ότου μείνουν δύο στοιχεία με τον ελάχιστο χρόνο σύγκρουσης

Γνωρίζουμε ότι η πιθανοτική Quickselect αφήνει περίπου τα ¾ των στοιχείων σε κάθε επανάληψη και μας δίνει κατά μέση περίπτωση μια πολυπλοκότητα της τάξης Θ(n).

<u>Πιο αναλυτικά:</u>

Αν η είσοδος, που είναι **2n**, τεθεί ίσον με **k, k = 2n,** τότε έχω:

T(k) = T(3k/4) + Θ(n) = Θ(n)