Δάσκος Ραφαήλ – Α.Μ.: 03116049

Αλγόριθμοι και Πολυπλοκότητα
Ροή Λ: Λογισμικό Η/Υ
Σ.Η.Μ.Μ.Υ. – Ε.Μ.Π. – 7ο εξάμηνο

Άσκηση 1: Ασυμπτωτικός Συμβολισμός, Αναδρομικές Σχέσεις.

(α) Η σειρά των συναρτήσεων είναι (αντί για $g_1, g_2, \ldots$ θα είναι προς τα κάτω ώστε κάθε προηγούμενη συνάρτηση να είναι $g_i=O(g_{i+1})$ και οι συναρτήσεις ίδιας τάξης θα είναι στην ίδια γραμμή):

- $\sum_{\kappa=1}^{n} \kappa * 2^{-\kappa}$
- $\frac{\log(n!)}{(\log n)^3}$
- $\log\binom{2n}{n}$, $n2^{2^{2^{100}}} = \Theta(n)$
- $2^{(logn)^4}$
- $\sum_{\kappa=1}^{n} \kappa * 2^{\kappa}$, $n\sum_{\kappa=1}^{n}\binom{n}{k} = \Theta(n2^n)$
- $(\sqrt{n})!$

Για την κατάταξη χρησιμοποιούμε ότι:

Αν $\lim_{n\to\infty} \frac{f(n)}{g(n)} = L,\ 0 < L \leq \infty$, τότε $f(n) = \Omega(g(n))$. Αν $0 < L \leq \infty$ τότε $f(n) = \Theta(g(n))$.

Επίσης μερικές χρήσιμες σχέσεις:

1. $(\sqrt{n})! = \Theta(\sqrt{2\pi n}\left(\frac{\sqrt{n}}{e}\right)^{\sqrt{n}})$ (Strirling's approximation)

2. ισχύει $n \leq \log\binom{2n}{n} \leq nlog(2e) = n * (1 + loge)$ καθώς ισχύει ότι $\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \frac{n^k}{k!} \leq \left(\frac{ne}{k}\right)^k$

3. $2^{(logn)^4} = n^{(logn)^3}$

4. $n2^{2^{2^{100}}} = c * n$ άρα θα είναι και $\Theta(n)$

5. $\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)*\ldots*(n-k+1)}{k!}$

6. $\frac{\log(n!)}{(\log n)^3} = \Theta(logn)$, καθώς $\log(n!) = \Theta(\log n^n)$ και για κάθε $\varepsilon>0$, $\log^d n = o(n^{\varepsilon})$

7. $\sum_{\kappa=1}^{n} 2^{-\kappa} * \kappa = \Theta(1)$ γιατί ο αριθμητής αυξάνει γραμμικά ενώ ο παρονομαστής εκθετικά. Και γενικά αν το άθροισμα s_n, *τότε* $s_n = 2 - \frac{n+2}{2^n}$ που στο άπειρο κάνει περίπου 2.

(β)

1. $T(n) = 3T(n/2) + n^2\log n$. Έχουμε $n^{log3} \approx n^{1.6}$. Επίσης, $f(n) = n^2\log n = \Omega(n^{1.6+0.1})$, δηλαδή για $\varepsilon = 0.1 > 0$ έχουμε ότι είναι μικρότερο του n^2 $n^{log3+\varepsilon}$ και αφού $3f(n/2) < f(n)$ για $n \to \infty$, θα έχουμε από Master Theorem $T(n)=\Theta(f(n))=\Theta(n^2\log n)$.
2. $T(n) = 4T(n/2) + n^2\log n$. Η ποσότητα $n^{log4} = 2$ δεν είναι πολυωνυμικά μικρότερη του $n^2\log n$ επομένως θα πρέπει να εφαρμόσουμε δέντρο αναδρομής. Έχουμε ότι, αφού σε κάθε επίπεδο το n υποδιπλασιάζεται, θα υπάρχουν $\log n+1$ επίπεδα. Έχουμε $\log n+1$ επίπεδα στα οποία η συνεισφορά είναι $n^2\log n$ οπότε $n^2\log^2 n$.
3. $T(n) = 5T(n/2) + n^2\log n$. Έχουμε $n^{log5} \approx n^{2.3}$. Για $\varepsilon=0.1$ έχουμε ότι $O(n^{log5+\varepsilon}) = O(n^{2.2})$. $f(n) = n^2\log n = O(n^{2.2})$ καθώς ισχύει ότι $n^{0.2} > \log n$. Οπότε θα ισχύει από Master Theorem ότι $T(n) = \Theta(n^{log5})$.
4. $T(n) = T(n/2) + T(n/3) + n$, με $\gamma_1 + \gamma_2 = \frac{1}{2} + \frac{1}{3} = \frac{5}{6} \approx 0.83 < 0.9 = 1 - 0.1 = 1 - \varepsilon$ για $\varepsilon = 0.1$. Οπότε θα ισχύει ότι $T(n) = \Theta(n)$.
5. $T(n) = T(n/2) + T(n/3) + T(n/6) + n$, με $\gamma_1 + \gamma_2 + \gamma_3 = \frac{1}{2} + \frac{1}{3} + \frac{1}{6} = 1$. Σε δέντρο αναδρομής όμως θα έχουμε ότι το ύψος του είναι φραγμένο από $\log_6 n$ και $\log n$ και η συνεισφορά του κάθε επιπέδου είναι n οπότε θα έχουμε ότι $T(n)=\Theta(n\log n)$.
6. $T(n) = T(n/4) + \sqrt{n}$. $d = ½ > \log_4 1 = 0$. Οπότε $T(n) = \Theta(n^{1/2}) = \Theta(\sqrt{n})$.

Άσκηση 2: Διαστήματα Ελαχίστου Μήκους που Καλύπτει όλους τους Πίνακες

(α) Έχουμε τους δύο ταξινομημένους πίνακες A_1 και A_2. Για να βρούμε τις θέσεις στις οποίες εμφανίζεται η ελάχιστη διαφορά μεταξύ των δύο αυτών πινάκων αρχικά θα τοποθετήσουμε θα κοιτάξουμε τα δύο πρώτα στοιχεία των πινάκων. Η αρχική διαφορά, που είναι μέχρι στιγμής ελάχιστη, είναι η διαφορά αυτών των δύο στοιχείων και οι θέσεις i_1 και i_2 είναι οι πρώτες θέσεις των πινάκων. Στη συνέχεια κοιτάμε ποιο από τα δύο στοιχεία είναι μεγαλύτερο και το κοιτάμε το επόμενο στοιχείο του άλλου πίνακα. Αν θεωρήσουμε ότι έχουμε δύο δείκτες, έναν σε κάθε πίνακα, μετακινούμε τον δείκτη που δείχνει σε μικρότερο στοιχείο. Αν η διαφορά που προκύπτει είναι μικρότερη, τότε κρατάμε τις νέες θέσεις ως i_1 και i_2 και συνεχίζουμε τον αλγόριθμό μας. Πραγματοποιούμε, δηλαδή, συγκρίσεις μεταξύ στοιχείων των πινάκων και κοιτάμε τα επόμενα στοιχεία του πίνακα του οποίου ο τρέχων αριθμός που εξετάζουμε είναι μικρότερος από τον αντίστοιχο του άλλου. Ο αλγόριθμος αυτός τερματίζει όταν ένας εκ

των δύο πινάκων φτάσει στα όρια του, όταν ελέγξουμε δηλαδή το στοιχείο n_1 ή n_2 και μετά χρειαστεί να προχωρήσουμε στον πίνακα που τερμάτισε.

Έχουμε έτσι στο τέλος στις θέσεις i_1 και i_2 την ελάχιστη διαφορά μεταξύ των πινάκων δηλαδή το min$|A_1[i_1] - A_2[i_2]|$.

Το πρόβλημα αυτό είναι γραμμικού χρόνου καθώς θα έχει πολυπλοκότητα $O(n_1 + n_2)$ που αυτά τα μεγέθη των δύο πινάκων που έχουμε. Αυτό συμβαίνει γιατί σε κάθε επανάληψη προχωράμε είτε στο βάθος του ενός είτε του άλλου πίνακα κατά ένα έως ότου φτάσουμε στο τέλος τους. Αυτό σημαίνει ότι μπορούμε να κάνουμε το πολύ n_1 στο πλήθος επαναλήψεις καθώς μετακινούμε τον δείκτη μας κατά μήκος του πίνακα A_1 και n_2 επαναλήψεις καθώς μετακινούμαστε κατά μήκος του A_2. Γεγονός που εξηγεί το $n_1 + n_2$ στην πολυπλοκότητα.

(β) Αν αυτή τη στιγμή εργαστούμε με όμοιο τρόπο όπως και στο (α) ερώτημα. Δημιουργούμε έναν πίνακα με m θέσεις όπου στην i θέση βρίσκεται το στοιχείο του A_i πίνακα, στην αρχή είναι τα πρώτα στοιχεία. Στη συνέχεια προχωράμε κατά 1 τον δείκτη που δείχνει στο μικρότερο στοιχείο του πίνακα με τα m και επαναλαμβάνουμε έως ότου να εξαντληθούν τα στοιχεία των πινάκων. Σημαντικό είναι κατά την υλοποίηση του αλγορίθμου να διατηρούμε το ελάχιστο διάστημα αλλά και τις θέσεις $i_1,\ldots,i_m$ οι οποίες αντιστοιχούν σε αυτό.

Επειδή σε κάθε επανάληψη χρειαζόμαστε χρόνο ίσο με O(m) για την εύρεση του ελαχίστου αλλά και του μεγίστου (το μέγιστο χρειάζεται για τη διαφορά) από τον πίνακα με τα m στοιχεία και έχουμε συνολικά $N = \sum_{k=1}^{m} n_k$ επαναλήψεις χρειαζόμαστε χρόνο ίσο με O(mN).

(γ) Το πρόβλημα που έχουμε όμως κατά τον παραπάνω αλγόριθμο είναι ότι ο χρόνος υπολογισμός του μεγίστου και του ελαχίστου στον πίνακα με τα m στοιχεία είναι O(m). Για να το αντιμετωπίσουμε αυτό θα χρησιμοποιήσουμε σωρό. Θα χρειαστούμε αρχικά O(m) για να αρχικοποιήσουμε το σωρό με τα πρώτα στοιχεία των πινάκων. Για την εύρεση όμως ελαχίστου αλλά και την εισαγωγή του νέου στοιχείου που προκύπτει από τη μετακίνηση του δείκτη που έδειχνε στον πίνακα στον οποίο άνηκε το ελάχιστο στοιχείο χρειαζόμαστε χρόνο O(logm). Τέλος για την εύρεση του μεγίστου θα έχουμε αρχικά υπολογισμένο το max των αρχικών στοιχείων, O(m) χρόνος, και μετά από κάθε εισαγωγή θα ελέγχουμε αν αυτό έχει αλλάξει με μια σύγκριση με το στοιχείο που βάλαμε.

Επομένως καταλήγουμε ότι θα χρειαστούμε συνολικό χρόνο O(N*logm). Καθώς γίνεται logm για N επαναλήψεις με το N όπως το είχαμε ορίσει από το προηγούμενο ερώτημα.

Άσκηση 3: Παίζοντας Χαρτιά

1. Δεδομένου ότι στο πρόβλημά μας έχουμε n στον αριθμό κάρτες (από το 1 μέχρι το n) το σίγουρο είναι ότι η πολυπλοκότητα που θα έχουμε στον αλγόριθμό μας στο τέλος θα είναι n*O( ), που το O( ) θα εξαρτάται από το πόσο γρήγορα μπορούμε να πραγματοποιήσουμε την εισαγωγή του κάθε στοιχείου στη δομή που χρησιμοποιούμε αλλά και τον τελικό υπολογισμό του αριθμού των στοιβών που έχουμε.

Για να πετύχουμε έναν αποτελεσματικό αλγόριθμο (να βελτιστοποιήσουμε τη λύση μας) θα κάνουμε χρήση AVL δυαδικού δέντρου για την εισαγωγή των στοιχείων στις στοίβες αφού τραβήξουμε ένα φύλλο από την τράπουλα.

Οι περιορισμοί που έχουμε στην εισαγωγή των στοιχείων είναι:

1. πως όταν τραβήξουμε μεγαλύτερο φύλλο από όλες τις υπάρχουσες κεφαλές των στοιβών να δημιουργήσουμε μια νέα στοίβα με κεφαλή το φύλλο αυτό
2. στην περίπτωση που υπάρχει τουλάχιστον μία κεφαλή μεγαλύτερη από το χαρτί που διαλέξαμε, τότε να το τοποθετούμε στην κορυφή της υπάρχουσας στοίβας από την οποία το στοιχείο απέχει το λιγότερο. Αν για παράδειγμα έχουμε τις στοίβες [(5, 4), (7)] και τραβήξουμε το φύλλο 2 τότε θα το τοποθετήσουμε στην πρώτη στοίβα και θα πάρουμε [(5, 4, 2), (7)]

Αυτό μας επιτρέπει τον ελάχιστο αριθμό στοιβών καθώς στην περίπτωση που μετά τραβήξουμε στο παράδειγμα που έδωσα το 6 δε θα χρειαστεί να δημιουργήσουμε εκ νέου στοίβα.

Για να επιτύχουμε τον 1^{ο} περιορισμό απλά εάν το στοιχείο που επιλέξαμε πάει να τοποθετηθεί τελείως δεξιά (το μεγαλύτερο στοιχείο) στο δέντρο μας, του επιτρέπουμε να δημιουργήσει μια έξτρα στοίβα.

Για τον 2^{ο} περιορισμό, εφόσον βρεθεί κόμβος από τον οποίο το στοιχείο είναι μικρότερο τον κρατάμε γιατί ίσος σε αυτόν να πρέπει να τοποθετήσουμε ως κεφαλή το χαρτί που τραβήξαμε. Στη συνέχεια συνεχίζουμε την αναζήτηση που θα έπρεπε να τοποθετηθεί το στοιχείο μας και κρατάμε πάντα το τελευταίο κόμβο από τον οποίο το στοιχείο μας ήταν μικρότερο. Εάν στο τέλος πάει να τοποθετηθεί ως δεξί παιδί, τότε αντικαθιστούμε τον κόμβο που έχουμε κρατήσει με το στοιχείο μας, αλλιώς σημαίνει ότι ο τελευταίος κόμβος ήταν αυτός που είναι πιο κοντά στο στοιχείο μας και θα πάμε να το τοποθετήσουμε αντί γι' αυτόν. Σε κάθε περίπτωση δε θα χρειαστεί καμία άλλη αλλαγή στο δέντρο γιατί με βάση τον περιορισμό μας σημαίνει ότι το στοιχείο που τοποθετούμε είναι πιο κοντά στην κεφαλή που πάμε να το βάλουμε οπότε τα παιδιά του κόμβου στον οποίο τοποθετήθηκε θα είναι πάλι μικρότερα αν είναι αριστερά (γιατί αλλιώς θα έμπαινε το στοιχείο σε ένα από αυτά) και μεγαλύτερα αν είναι δεξιά (γιατί ούτως ή άλλως είναι μεγαλύτερα από την προηγούμενη κεφαλή που ήταν μεγαλύτερη του στοιχείου που τοποθετούμε).

Δεδομένου όλων αυτών έχουμε O(logn) για την τοποθέτηση του κάθε στοιχείου στη στοίβα που ανήκει. Στο τέλος για τον υπολογισμό του πλήθους των στοιβών που θα έχουμε χρειάζεται απλώς να υπολογίσουμε τον αριθμό των κόμβων του δέντρου μας. Οι

αλγόριθμοι υπολογισμού του πλήθους των κόμβων (in-order, post-order, pre-order search) έχουν πολυπλοκότητα O(n).

Έτσι, συνολικά θα έχουμε για τη χειρότερη περίπτωση:

$$O(n*\log n + n) = O(n*\log n)$$

2. Για να μπορέσουμε γρήγορα να ταξινομήσουμε την τράπουλα μας θα πρέπει το δέντρο μας να έχει ως στοιχεία με τα οποία γίνονται οι διάφορες διεργασίες τις κεφαλές της κάθε στοίβας. Επομένως, από τη στιγμή που θα κάνουμε pop το τελευταίο στοιχείο που έχει εισαχθεί εργαζόμαστε σαν να διαγράφουμε τον κόμβο και τον εισάγουμε πάλι στο δέντρο μας. Έτσι ξεκινάμε από το μικρότερο στοιχείο (το 1) που βρίσκεται τελείως αριστερά στο δέντρο μας. Από τη στιγμή που το αφαιρέσουμε τοποθετούμε πάλι τη στοίβα στο δέντρο στη θέση που πρέπει με κεφαλή το δεύτερο της στοιχείο κ.ο.κ. Όταν μια στοίβα δεν έχει άλλα στοιχεία απλώς φεύγει ο κόμβος που είχε από το δέντρο.

Η πολυπλοκότητα της αναζήτησης του ελαχίστου στοιχείου είναι O(logn), της διαγραφής αλλά και της εισαγωγής σε AVL δέντρο είναι πάλι O(logn). Οπότε και πάλι η πολυπλοκότητα αυτού του αλγορίθμου είναι: O(n*logn) καθώς την κάθε μια από αυτές τις διεργασίες την κάνουμε για τα n στοιχεία των στοιβών.

3. Έχουμε την ακολουθία 3, 2, 4, 7, 8, 1, 5, 6. Μέσω του αλγορίθμου που γράψαμε στο ερώτημα 1 θα πάρουμε την εξής ακολουθία στοιβών:

$$[] \rightarrow [(3)] \rightarrow [(3, 2)] \rightarrow [(3, 2), (4)] \rightarrow [(3, 2), (4), (7)] \rightarrow [(3, 2), (4), (7), (8)] \rightarrow$$
$$[(3, 2, 1), (4), (7), (8)] \rightarrow [(3, 2, 1), (4), (7, 5), (8, 6)]$$

Στη συνέχεια για να ταξινομήσουμε τις κάρτες θα έχουμε την εξής ακολουθία στοιβών:

$$[(3, 2, 1), (4), (7, 5), (8, 6)] \rightarrow [(3, 2), (4), (7, 5), (8, 6)] \rightarrow [(3), (4), (7, 5), (8, 6)] \rightarrow$$
$$[(4), (7, 5), (8, 6)] \rightarrow [(7, 5), (8, 6)] \rightarrow [(7), (8, 6)] \rightarrow [(7), (8)] \rightarrow [(8)] \rightarrow []$$

Και οι κάρτες θα είναι: 1, 2, 3, 4, 5, 6, 7, 8.

4. Έστω ότι το πλήθος των στοιβών είναι μικρότερο της μέγιστης αύξουσας υπακολουθίας, τότε αυτό σημαίνει ότι θα υπάρξει ένα τουλάχιστον στοιχείο από αυτά της υπακολουθίας που θα μπει πάνω από κάποιο από τα προηγούμενα. Αυτό όμως είναι άτοπο, εφόσον αναγκαστικά δεδομένου ότι κάθε νέο στοιχείο είναι μεγαλύτερο από τα προηγούμενά του, τότε θα έχουμε ότι θα τοποθετείται σε μια διαφορετική στοίβα από ότι τα άλλα (δε λέω σε καινούργια γιατί μπορεί ήδη να είχαμε τον αριθμό των στοιβών από πριν). Αυτό μας οδηγεί στο συμπέρασμα πως αναγκαστικά θα έχουμε αριθμό στοιβών τουλάχιστον ίσο με το μέγεθος της υπακολουθίας αυτής καθώς τα στοιχεία της είναι υποχρεωμένα να τοποθετηθούν σε διαφορετικές στοίβες.

6. Είναι σαν την Αρχή του Περιστερώνα. Θα πάμε να αποδείξουμε ότι εάν δε συμβαίνει το ένα είναι αναγκαστικό να συμβαίνει το άλλο και το αντίθετο.

Έστω ότι κάθε στοίβα έχει μέγεθος το πολύ m. Για να έχουμε πλήθος στοιχείων n*m+1 θα τουλάχιστον n+1 στοίβες, καθώς εάν είχαμε n τότε το μέγιστο πλήθος καρτών που θα είχαμε είναι ίσο με n*m δεδομένου ότι και οι n στοίβες είχαν m κάρτες. Οπότε εάν κάθε στοίβα έχει μέγεθος το πολύ m, τότε έχουμε σίγουρα τουλάχιστον n+1 στο πλήθος στοίβες.

Τώρα έστω ότι έχουμε το πολύ n στοίβες. Τότε, για να έχουμε πλήθος στοιχείων n*m+1 θα χρειαστούμε τουλάχιστον m+1 στοιχεία σε κάθε μία από αυτές τις στοίβες, καθώς εάν είχαμε m τότε το μέγιστο πλήθος καρτών που θα είχαμε είναι ίσο με n*m δεδομένου ότι και οι n στοίβες είχαν m κάρτες. Επομένως, εάν έχουμε το πολύ n στο πλήθος στοίβες, υπάρχει τουλάχιστον μία στοίβα με m+1 κάρτες.

Έτσι καταλήγουμε στο ότι για κάθε ανακάτεμα μιας τράπουλας με n*m+1 φύλλα θα έχουμε πάντα είτε τουλάχιστον n+1 στοίβες είτε μία από αυτές θα έχει μέγεθος τουλάχιστον m+1 (μπορεί να είναι βέβαια και περισσότερες από μια αυτές οι στοίβες)

Η πιο απλή ακολουθία 25 καρτών που αλγόριθμος του (1) θα δώσει 5 στοίβες που η καθεμία θα έχει από 5 κάρτες είναι η εξής:

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

Και οι στοίβες που θα δημιουργηθούν είναι οι παρακάτω:

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

Άσκηση 4: Γρήγορη επιλογή στο πεδίο της μάχης

Στο πρόβλημά μας η πλευρά που εκτοξεύει τα σωματίδια a, με τη χρήση του υπερυπολογιστή της, μπορεί να γνωρίζει σε σταθερό χρόνο και με απόλυτη ακρίβεια τη θέση του κάθε σωματιδίου τη χρονική στιγμή $t \geq 0$ ενώ η πλευρά που εκτοξεύει τα σωματίδια b μπορεί να υπολογίσει σε σταθερό χρόνο τη χρονική στιγμή που ένα ζευγάρι σωματιδίων διαφορετικού τύπου συγκρούονται.

Επίσης μπορούμε να ξέρουμε ότι η χρονική στιγμή της σύγκρουσης των δύο πρώτων σωματιδίων είναι μεταξύ των χρονικών στιγμών $t_{min} = \frac{L}{2*Vmax}$ και $t_{max} = \frac{L}{2*Vmin}$ καθώς η μέγιστη ταχύτητα που μπορούν να πάνε τα σωματίδια είναι ίση με V_{max} και η ελάχιστη ταχύτητα που μπορούν να πάνε είναι ίση με V_{min}.

Αρχίζουμε τώρα το πρόβλημα για τον στρατηγό του στρατοπέδου που εκτοξεύει τα σωματίδια a. Και μπορεί να γνωρίζει σε σταθερό χρόνο τη θέση του κάθε σωματιδίου είτε είναι τύπου a είτε τύπου b. Αρχικά καταγράφουμε, με τη βοήθεια της αρχικής θέσης του κάθε σωματιδίου, σε χρόνο n (ίσο με το πλήθος των σωματιδίων) εάν είναι τύπου a, που σημαίνει ότι βρίσκονται στη θέση x = 0, ή τύπου b, που θα είναι στη θέση x = L. Επίσης θα θεωρήσουμε ότι ακόμα και εάν έχει πραγματοποιηθεί σύγκρουση μεταξύ δύο σωματιδίων διαφορετικού τύπου και αυτά έχουν εξαϋλωθεί, ο υπερυπολογιστής του στρατοπέδου αυτού θα γίνει τη θέση που θα είχε το σωματίδιο εάν συνέχιζε την κίνηση του. Έπειτα θα πραγματοποιήσουμε μια δυαδική αναζήτηση στο χρονικό διάστημα που

είπαμε ότι αναγκαστικά μέσα σε αυτό θα έχουμε την πρώτη σύγκρουση, δηλαδή στο $\{t_{min}\text{-}t_{max}\}$ ξεκινώντας φυσικά από τη στιγμή $\frac{tmax+tmin}{2}$ που είναι το μέσο των δύο στιγμών. Για τη συνέχεια του αλγορίθμου δυαδικής αναζήτησης υπάρχουν τρεις περιπτώσεις:

1. εάν δεν υπάρχει κανένα σωματίδιο a πιο δεξιά από ένα σωματίδιο b τότε θα πάμε σε μεταγενέστερη χρονική στιγμή για να εξετάσουμε.
2. εάν έστω και ένα σωματίδιο a είναι πιο κοντά στο σημείο x = L από κάποιο σωματίδιο b τότε πάει να πει ότι θα εξετάσουμε προγενέστερη χρονική στιγμή.
3. τέλος εάν όλα τα σωματίδια a είναι στην αριστερή μεριά και όλα τα σωματίδια b στη δεξιά και υπάρχουν κάθε τύπο ένα σωματίδιο των οποίων η θέση τους να ταυτίζεται, τότε η έχουμε βρει τη χρονική στιγμή που γίνεται η πρώτη σύγκρουση και τα δύο σωματίδια που συγκρούονται είναι αυτά με την κοινή θέση.

Γενικά σε κάθε ένα από τα βήματα της δυαδικής αναζήτησης έχουμε n χρήσης του υπερυπολογιστή. Και συνολικά θα έχουμε πλήθος $\log(t_{max}\text{-}t_{min}) = \log(\frac{L*(Vmax-Vmin)}{2*Vmax*Vmin})$ που είναι ένας σταθερός αριθμός. Οπότε συνολικά θα είχαμε εάν c η σταθερά χρόνου του υπερυπολογιστή: $T(n) = n*\log(\frac{L*(Vmax-Vmin)}{2*Vmax*Vmin})*c + c_1 = O(n)$ δεδομένου ότι όλα τα άλλα είναι σταθερά. Εκτός αν θεωρήσουμε ότι εξαιτίας των V_{max} και V_{min} ο λογάριθμος αυτός δεν είναι σταθερός.

Τώρα θα κοιτάξουμε τι συμβαίνει με τον στρατηγό του στρατοπέδου με τα σωματίδια b. Αυτό τρέχουμε ένα τυχαίο σωματίδιο b με όλα τα σωματίδια a και από αυτό βρίσκουμε το σωματίδιο με τον ελάχιστο χρόνο. Οπότε αυτή τη στιγμή εάν έχουμε c το σταθερό χρόνο που χρειάζεται ο υπερυπολογιστής μέχρι αυτό το σημείο έχουμε cn+n χρόνο οπότε και έχουμε πολυπλοκότητα ίση με O(n). Στη συνέχεια κοιτάμε το σωματίδιο a που βρήκαμε με όλα τα b και κάνουμε sort ώστε να δούμε ποια σωματίδια έχουν μικρότερο χρόνο να χτυπήσουν με σωματίδια του άλλου τύπου έχουν μικρότερο χρόνο να χτυπήσουν και «πετάμε» τα υπόλοιπα b σωματίδια. Αυτό στη μέση περίπτωση είναι ίσο με n/2 και επειδή ο αλγόριθμός μα είναι πιθανοτικός θα έχουμε τη μέση περίπτωση. Έτσι και σε αυτό το στάδιο έχουμε O(n) καθώς και το sorting παίρνει χρόνο O(n) στη μέση περίπτωση. Αυτό είναι σωστό καθώς θα έχουμε ότι για να χτυπήσει πιο μετά το συγκεκριμένο σωματίδιο πάει να πει ότι βρισκόταν πιο κοντά στην αφετηρία του από το σωματίδιο που ελέγξαμε στην αρχή μιας και χτυπάει πιο μετά. Έτσι δε γίνεται να έχει χτυπήσει και κάποιο άλλο από τα a πιο νωρίς καθώς και όλα τα a σωματίδια πιο κοντά στη δική τους αφετηρία από το σωματίδιο a που πήραμε στο αρχικό min. Εργαζόμενοι με τον ίδιο τρόπο θα καταλήξουμε σε ένα ζεύγος a και b σωματιδίων που θα έχουν και τον ελάχιστο χρόνο που συγκρούστηκαν. Και το πλήθος των επαναλήψεων που θα πραγματοποιηθούν είναι ίσος με logn, αφού κάθε φορά υποδιπλασιάζεται το πλήθος των

σωματιδίων που ελέγχουμε. Οπότε συνολικά θα έχουμε πολυπλοκότητα χρόνου ίση με O(nlogn).

Άσκηση 5: Κρυμμένος Θησαυρός

Στο πρόβλημα μας έχουμε ότι ο θησαυρός βρίσκεται κρυμμένος στη θέση x πάνω στον άξονα $\mathbb{Z}$ και ότι εμείς ξεκινάμε από τη θέση 0. Ο πιο αποδοτικός τρόπος για την εύρεση του θησαυρού είναι να ξεκινήσουμε από το 0 και να πάμε στο +2 μετά στο -4 έπειτα στο +8 κ.ο.κ. Να μετακινούμαστε δηλαδή στην επόμενη δύναμη του 2 αλλάζοντας όμως και μεριά στον άξονα $\mathbb{Z}$ (αν είμαστε στα θετικά μετακινούμαστε στα αρνητικά και το αντίθετο). Αυτό το κάνουμε έως ότου να περάσουμε πάνω από τη θέση στην οποία είναι κρυμμένος ο θησαυρός.

Με αυτό τον τρόπο θα κάνουμε 2 βήματα να φτάσουμε στο 2, 2+4 να φτάσουμε στο -4 και κάποια στιγμή θα κάνουμε μια δύναμη του 2 + |x| για να φτάσουμε στο θησαυρό.

Έτσι συνολικά θα έχουμε κάνει:

$2+(2+4)+(4+8)+\ldots+ (2^{\kappa+1}+|x|) = 4*(1+2+\ldots+2^{\kappa}) +|x| = 4*\sum_{i=0}^{\kappa} 2^{i}+|x| = 4*(2^{\kappa+1}-1)+|x|$ βήματα.

Έβαλα στην αρχή το $2^{\kappa+1}$ ώστε απλώς να έχω μετά στην παρένθεση μέσα μόνο το 2^{κ} και το άθροισμα έως το κ.

Τώρα το σημαντικό είναι να δούμε τη σχέση του κ με το |x| ώστε να δούμε την πολυπλοκότητα του αλγορίθμου αυτού. Δεδομένου όμως ότι το κ έχει σχέση με το λογάριθμο του |x| σημαίνει ότι η πολυπλοκότητα είναι O(|x|).

Για να υπολογίσουμε όμως και το άνω φράγμα c*|x| που ο αλγόριθμος θα υπολογίζει το που βρίσκεται ο θησαυρός μετά από c*|x| βήματα χρειάζεται να υπολογίσουμε ακριβώς τη σχέση του κ με το |x|. Έχουμε πει ήδη ότι πριν φτάσει να κάνει |x| βήματα, ο αλγόριθμός μας έκανε $2^{\kappa+1}$. Αυτό σημαίνει ότι το |x| βρίσκεται μεταξύ του 2^{κ} και του $2^{\kappa+2}$ καθώς το πρώτο είναι η προηγούμενη δύναμη που βρισκόταν στη μεριά του και δεν το ξεπέρασε και το δεύτερο η πρώτη δύναμη από τη μεριά του που το ξεπερνά. Δηλαδή:

$$2^{\kappa} < |x| \leq 2^{\kappa+2} \Leftrightarrow \kappa < \log|x| \leq \kappa+2$$

Οπότε θα έχουμε ότι:

$$4*(2^{\kappa+1}-1)+|x| < 4*(2^{\log|x|+1} - 1) + |x| = 4*(2*2^{\log|x|} - 1) + |x| = 9*|x| - 4$$

και το -4 που είναι σταθερά μπορούμε να το διώξουμε οπότε μπορούμε να πούμε ότι το άνω όριο είναι ίσο με 9*|x| και άρα το c = 9.