

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Αλγόριθμοι και Πολυπλοκότητα 2016-2017

1η Σειρά Γραπτών Ασκήσεων

Όνομα: Παναγιώτης Κωστοπαναγιώτης

A.M:03115196

Άσκηση 1: Ασυμπτωτικός Συμβολισμός, Αναδρομικές Σχέσεις

α) Έχουμε τις εξής σχέσεις:

$$n^{3/2} = O(n\sqrt{n}) \quad , \quad (3^{\log n})^3 = O((3^{\log n})^3) = O(n^3) \quad , \quad \frac{\log(n!)}{\log^3 n} = O(\frac{n \log n}{\log^3 n}) = O(\frac{n}{\log^2 n}) \quad ,$$

$$n2^{2^{2^{2^{2^n}}}} = O(n) \quad , \quad \log^4 n = O(\log^4 n) \quad , \quad n^{(\log(n!))} = O(n^{n \log n}) = O(n^n n^{\log n}) = O(n^{n+1}) = O(n^n) \quad ,$$

$$\sum_{k=1}^n k 2^k \leq \sum_{k=1}^n n 2^k = n \sum_{k=1}^n 2^k = n(2^{n+1} - 1) = O(n2^n) \quad , \quad \sum_{k=1}^n k 2^{-k} \leq \sum_{k=1}^n n 2^{-k} = n \sum_{k=1}^n 2^{-k} \leq n \sum_{k=1}^{\infty} 2^{-k} = O(n)$$

$$n \sum_{k=0}^n \frac{n!}{k!(n-k)!} = n2^n = O(n2^n) \quad , \quad \sqrt{n!} = O(\sqrt{n^{\sqrt{n}}}) = O(n^{\frac{n}{4}}) \quad , \quad \frac{n^3}{\log n^8} = O(\frac{n^3}{\log n^8}) \quad ,$$

$$\log(\frac{2n!}{n!n!}) = \log(2n!) - 2\log(n!) = O(2n \log 2n - 2n \log n) = O(2n \log n - 2n \log n + 2\log 2n) = O(n)$$

$$\log(\frac{n!}{(n-\log n)! \log n!}) = \log(n!) - \log((n-\log n)!) - \log(\log n!) =$$

$$O(n \log n - (n-\log n) \log(n-\log n) - \log n \log \log n) = O(n \log n - n \log n + \log^2 n - \log n \log \log n) = O(\log^2 n)$$

$$\frac{\log^2 n}{\log \log n} = O(\frac{\log^2 n}{\log \log n}) \quad , \quad \frac{n!}{(n-6)! 6!} = O(n^6) \quad .$$

Συνεπώς η διάταξη των συναρτήσεων ως προς την τάξη μεγέθους τους είναι:

$\frac{\log^2 n}{\log \log n}$	$\log(\frac{n!}{(n-\log n)! \log n!})$	$\log^4 n$	$\frac{\log(n!)}{\log^3 n}$
$n2^{2^{2^{2^{2^n}}}}$	$\sum_{k=1}^n k 2^{-k}$	$\log(\frac{2n!}{n!n!})$	$n^{\frac{3}{2}}$
$\frac{n^3}{\log^8 n}$	$3^{(\log n)^3}$	$\frac{n!}{(n-6)! 6!}$	$\log n^{\log n}$

$\sum_{k=1}^n k 2^k$	$n \sum_{k=0}^n \frac{n!}{k!(n-k)!}$	$\sqrt{n!}$	$n^{\log(n!)}$
----------------------	--------------------------------------	-------------	----------------

Οι συναρτήσεις που έχουν ίδια τάξη μεγέθους φαίνονται από τις παραπάνω σχέσεις.

β) 1. Από master theorem λαμβάνουμε: $n \log n = \Omega(n^{\log_3 2})$ Συνεπώς, βρισκόμαστε στην τρίτη περίπτωση του master theorem και άρα $T(n) = \Theta(n \log n)$.

$$2. \text{Έχουμε } T(n) = 3T\left(\frac{n}{3}\right) + n \log n = 3\left(3T\left(\frac{n}{9}\right) + \frac{n}{3} \log\left(\frac{n}{3}\right)\right) + n \log n = 9T\left(\frac{n}{9}\right) + 2n \log(n) - n \log 3$$

Γενικά έχουμε για κάθε κ : $T(n) = 3^\kappa T\left(\frac{n}{3^\kappa}\right) + n \kappa \log n + \Theta(n)$ και για $\kappa = \log_3 n$ λαμβάνουμε:

$$T(n) = n + n \log^2 n + \Theta(n) = \Theta(n \log^2 n)$$

$$5. \text{Έχουμε } T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{3}\right) + T\left(\frac{n}{6}\right) + n = T\left(\frac{n}{2}\right) + T\left(\frac{n}{3}\right) + T\left(\frac{n}{6}\right) + \frac{n}{2} + \frac{n}{3} + \frac{n}{6}. \text{ Θέτουμε:}$$

$$T_1(n) = T\left(\frac{n}{2}\right) + \frac{n}{2}, T_2(n) = T\left(\frac{n}{3}\right) + \frac{n}{3}, T_3(n) = T\left(\frac{n}{6}\right) + \frac{n}{6}. \text{ Συνεπώς } T(n) = T_1(n) + T_2(n) + T_3(n).$$

Είναι, από master theorem $T_1(n) = \Theta(n)$. Ομοίως λαμβάνουμε $T_2(n) = T_3(n) = \Theta(n)$.

Συνεπώς $T(n) = \Theta(n)$.

6. Δεν μπορούμε να εφαρμόσουμε το master theorem σε αυτή την περίπτωση. Εφαρμόζοντας

επαγωγή στο μέγεθος n έχουμε $T(n) = \log n + \log(n-1) + \dots + \log 2 = \log(n!) = \Theta(n \log n)$.

7. Είναι $\Theta(\log n) = c \log n$ για κάποιο $c > 0$. Έχουμε επομένως, για την αναδρομική σχέση:

$$T(n) = T(n^{5/6}) + c \log n = T(n^{25/36}) + c \log n + 5 \frac{c}{6} \log n = \dots = T(n^{(5/6)^k}) + c \log n \sum_{i=0}^k (5/6)^i. \text{ Η παραπάνω σχέση ισχύει για κάθε } k > 0, \text{ για αρκετά μεγάλο } k \text{ λαμβάνουμε:}$$

$$T(n) = \lim_{k \rightarrow \infty} [T(n^{(5/6)^k}) + c \log n \sum_{i=0}^k (5/6)^i] = T(1) + c \log n \sum_{i=0}^{\infty} (5/6)^i = c \log n * \frac{1}{1 - \frac{5}{6}} = 6c \log n = \Theta(\log n)$$

$$8. \text{Έχουμε } T(n) = T\left(\frac{n}{4}\right) + \sqrt{n} = T\left(\frac{n}{8}\right) + \sqrt{n} + \frac{\sqrt{n}}{2}. \text{ Γενικά Έχουμε: } T(n) = T\left(\frac{n}{4^\kappa}\right) + \sqrt{n} \sum_{i=1}^{\kappa} \frac{1}{2^i}.$$

$$\text{Θέτοντας όπου } \kappa = \log n \text{ έχουμε: } T(n) = \sqrt{n} \sum_{i=1}^{\log n} \frac{1}{2^i} \leq \sqrt{n} \sum_{i=1}^{\infty} 2^{-i} = \sqrt{n} \frac{1}{1 - \frac{1}{2}} = 2\sqrt{n}. \text{ Λαμβάνουμε}$$

Συνεπώς $T(n) = \Theta(\sqrt{n})$.

Άσκηση 2: Ταξινόμηση

α) 1. Σκοπός μας είναι να βρούμε $k - 1$ αριθμούς που να διαχωρίζουν τον ταξινομημένο πίνακα σε ίσα μέρη (μεγέθους k το καθένα). Κάνουμε το εξής, αρχικά βρίσκουμε την διάμεσο τιμή και κάνουμε partition τον πίνακα σε 2 ίσα μέρη. Έπειτα, εφαρμόζουμε αναδρομικά τον ίδιο αλγόριθμο

μέχρις ότου να βρεθούμε σε υποπίνακα μεγέθους $\frac{n}{k}$ (επειδή τα n, k είναι δυνάμεις του 2 θα

βρεθούμε απαραίτητα κάποια στιγμή σε υποπίνακα μεγέθους $\frac{n}{k}$). Συνολικά, εφαρμόζουμε μια

διαδικασία σαν την quicksort, με την διαφορά ότι τα επίπεδα της διαδρομής είναι σε πλήθος

$$\log n - \log\left(\frac{n}{k}\right) = \log k , \text{ διότι δεν μας ενδιαφέρει να ταξινομήσουμε τα κομμάτια μεγέθους } \frac{n}{k} ,$$

απλά να τα διατάξουμε με τον τρόπο που περιγράφεται από το πρόβλημα. Σε κάθε επίπεδο κάνουμε συνολικά γραμμικό πλήθος πράξεων και άρα η συνολική πολυπλοκότητα του αλγορίθμου μας είναι $O(n \log k)$.

Συνολικά υπάρχουν $(k!)^{\frac{n}{k}}$ ακολουθίες n αριθμών ταξινομημένες κατά k -μέρη. Κάθε συγκριτικός

αλγόριθμος ταξινόμησης κάνει συνολικά τουλάχιστον $\log((k!)^{\frac{n}{k}})$ συγκρίσεις. Συνεπώς, κάθε

συγκριτικός αλγόριθμος ταξινόμησης έχει πολυπλοκότητα $\Omega\left(\frac{n}{k} \log(k!)\right) = \Omega(n \log k)$.

2. Δεδομένου ότι ο πίνακας μας είναι ήδη ταξινομημένος κατά k -μέρη, μπορούμε να εφαρμόσουμε σε κάθε μέρος ξεχωριστά κάποιον αλγόριθμο ταξινόμησης (π.χ mergesort) και να το ταξινομήσουμε

ξεχωριστά. Η πολυπλοκότητα του αλγορίθμου μας θα είναι $\Theta(k \frac{n}{k} \log \frac{n}{k}) = \Theta(n \log \frac{n}{k})$.

Θα δείξουμε ότι ο αλγόριθμός μας είναι βέλτιστος. Συνολικά, έχουμε $\left(\frac{n}{k}!\right)^k$ πιθανές

αναδιατάξεις των στοιχείων μας, εφ' όσον ο πίνακας μας είναι ήδη ταξινομημένος κατά k -μέρη.

Συνεπώς, κάθε συγκριτικός αλγόριθμος ταξινόμησης, θα κάνει συνολικά τουλάχιστον

$$\log\left(\left(\frac{n}{k}!\right)^k\right) \text{ βήματα και συνεπώς θα έχει πολυπλοκότητα } \Omega(k \log\left(\frac{n}{k}!\right)) = \Omega(n \log\left(\frac{n}{k}\right)) .$$

β) 1. Θα χρησιμοποιήσουμε την τεχνική sliding-window καθώς και έναν σωρό για να ταξινομήσουμε την ακολουθία μας. Αρχικά, δημιουργούμε έναν σωρό που περιέχει τα πρώτα k -στοιχεία. Έπειτα, διαδοχικά αφαιρούμε το μικρότερο στοιχείο του σωρού και εισάγουμε το αμέσως επόμενο στην σειρά, μέχρι να φτάσουμε στο τέλος του πίνακα.

Ορθότητα: Κάθε στοιχείο του πίνακα βρίσκεται k θέσεις, το πολύ, μακριά από την σωστή του θέση στον ταξινομημένο πίνακα. Συνεπώς, αν το στοιχείο βρίσκεται στην θέση i , οι πιθανές θέσεις του βρίσκονται στο διάστημα $[i - k + 1, i + k - 1]$. Εξετάζοντας στον σωρό διαστήματα μεγέθους k , κάθε στοιχείο έχει διάρκεια παραμονής στον σωρό το πολύ $2k$ και συνεπώς εξετάζονται όλες οι πιθανές θέσεις στις οποίες θα μπορούσε να είναι η θέση του στοιχείου αυτού και άρα ο αλγόριθμος μας πράγματι ταξινομεί τον πίνακα.

Πολυπλοκότητα: Η πολυπλοκότητα του αλγορίθμου μας είναι $O(k + n \log k) = O(n \log k)$.

2. Για κάθε πίνακα που είναι k -προταξινομημένος υπάρχουν τουλάχιστον $\left(\frac{k}{2}\right)^n$ πιθανές έγκυρες αναδιατάξεις του, μια από τις οποίες είναι και η ταξινομημένη. Για κάθε σύγκριση που κάνει ένας συγκριτικός αλγόριθμος ταξινόμησης, χωρίζουμε σε 2 σύνολα ίσου μεγέθους τις πιθανές αναδιατάξεις του πίνακα, από τα οποία η σύγκριση μας επιτρέπει να μάθουμε σε ποιό από τα 2 βρίσκεται η ταξινομημένη αναδιάταξη. Συνολικά, λαμβάνουμε ένα πλήρες δυαδικό δέντρο με

τουλάχιστον $\left(\frac{k}{2}\right)^n$ φύλλα. Συνεπώς, κάθε μονοπάτι είναι μεγέθους $\log\left(\left(\frac{k}{2}\right)^n\right) = \Omega(n \log k)$ που είναι και το φράγμα του χρόνου εκτέλεσης των συγκριτικών αλγορίθμων ταξινόμησης k -προταξινομημένων πινάκων.

Άσκηση 3: Αναζήτηση

α) Εφαρμόζουμε δυαδική αναζήτηση. Ας συμβολίσουμε ως $\text{rot}(a)$ τον περιστραμμένο πίνακα a . Σε κάθε φάση της αναζήτησης μπορούμε να αποφανθούμε για το αν βρισκόμαστε στο διάστημα $[1, k]$ ή στο διάστημα $[k + 1, n]$ συγκρίνοντας την τιμή του στοιχείου a_{mid} με το στοιχείο a_{k+1} (δηλαδή το πρώτο στοιχείο δεδομένου ότι ο πίνακας έχει περιστραφεί).

Ψευδοκώδικας:

input: $A = \text{rot}(\alpha)$
output: idx, τέτοιο ώστε $A[\text{idx}] = \min\{A[1], A[2], \dots, A[n]\}$

lo = 1, hi = idx = n

Όσο $\text{lo} <= \text{hi}$

 mid = $(\text{lo} + \text{hi}) / 2$

$\mathbf{Αν} \ A[\text{mid}] < A[1]$

$\text{idx} = \text{mid}$

$\text{hi} = \text{mid} - 1$

$\mathbf{Άλλιως}$

$\text{lo} = \text{mid} + 1$

Ορθότητα: Λόγω της μονοτονίας του πίνακα α , ο περιστραμμένος πίνακας αποκτά ιδιότητες που μας επιτρέπουν να αναζητήσουμε το στοιχείο αποδοτικά μέσω παραλλαγμένης δυαδικής αναζήτησης. Επομένως το στοιχείο α_1 διαμερίζει τον $\text{rot}(\alpha)$ σε δύο γνησίως αύξοντα υποδιαστήματα. Χρησιμοποιώντας το στοιχείο α_{k+1} μπορούμε να αποφανθούμε για την θέση ενός δεδομένου στοιχείου στον $\text{rot}(\alpha)$ ως εξής. Έστω πως επιλέγουμε ένα τυχαίο στοιχείο, έστω α_l , του περιστραμμένου πίνακα. Διακρίνουμε δύο περιπτώσεις:

1. Αν $\alpha_l < \alpha_{k+1}$ τότε το στοιχείο α_l βρίσκεται δεξιότερα του α_1 στον $\text{rot}(\alpha)$, είναι δηλαδή ένα εκ των $\alpha_1, \alpha_2, \dots, \alpha_k$
2. Αν $\alpha_l \geq \alpha_{k+1}$ τότε το στοιχείο α_l βρίσκεται αριστερότερα του α_1 στον $\text{rot}(\alpha)$, είναι δηλαδή ένα εκ των $\alpha_{(k+1)}, \alpha_{(k+2)}, \dots, \alpha_n$.

Συνεπώς, αν για κάθε στοιχείο του πίνακα καταγράψουμε σε ποια κατηγορία ανήκει η ακολουθία - αυτή θα μοιάζει ως εξής: 2, 2, ..., 2, 1, ..., 1. Ενδιαφερόμαστε για την πρώτη αλλαγή που γίνεται από 2 σε 1, συνεπώς αυτή η δυικότητα μας επιτρέπει να χρησιμοποιήσουμε δυαδική αναζήτηση για τον εντοπισμό του στοιχείου.

Πολυπλοκότητα: Η πολυπλοκότητα της λύσης μας προκύπτει άμεσα από την πολυπλοκότητα της δυαδικής αναζήτησης. Συνεπώς έχουμε $O(\log n)$ συνολική πολυπλοκότητα αφού για κάθε βήμα της δυαδικής αναζήτησης απαιτείται σταθερό πλήθος πράξεων.

β) Έστω πως μπορούμε να πραγματοποιήσουμε το ταξίδι αυτό, καλύπτοντας απόσταση το πολύ d σε κάποια ημέρα. Τότε, σίγουρα μπορούμε να το πραγματοποιήσουμε και με απόσταση το πολύ $d1 > d$ σε κάποια ημέρα. Αντίστροφα, αν δεν μπορούμε να το πραγματοποιήσουμε με απόσταση d το πολύ, σίγουρα δεν θα μπορούμε να το πραγματοποιήσουμε με απόσταση $d2 < d$. Συνεπώς εφαρμόζουμε δυαδική αναζήτηση στην απάντηση. Για να αποφανθούμε κατα πόσο είναι δυνατόν να πραγματοποιήσουμε το ταξίδι μας με απόσταση ανά μέρα το πολύ d, μπορούμε να χρησιμοποιήσουμε έναν άπληστο αλγόριθμο γραμμικού χρόνου ο οποίος να προσπαθεί να “στριμώξει” σε κάθε ημέρα όσο μεγαλύτερη απόσταση γίνεται χωρίς να ξεπερνάει το d. Η ορθότητα του αλγορίθμου αυτού μπορεί να αιτιολογηθεί μέσω επιχειρήματος ανταλλαγής μεταξύ 2 διαδοχικών ημερών.

Πολυπλοκότητα: Για κάθε βήμα της δυαδικής αναζήτησης εκτελούμε γραμμικό πλήθος βημάτων. Συνολικά η δυαδική αναζήτηση εκτελείται $O(\log D)$ φορές όπου D το άθροισμα των αποστάσεων στον πίνακα. Τότε η πολυπλοκότητα του αλγορίθμου μας είναι $O(n \log D)$

Άσκηση 4: Εντοπισμός Επαναλαμβανόμενου Στοιχείου

Έστω πως μπορούσαμε να λύσουμε το πρόβλημα χρησιμοποιώντας $O(n)$ επιπλέον χώρο. Έστω πίνακας A' μεγέθους n . Αρχικά ο πίνακας A' είναι άδειος. Τότε, δεδομένου πως τα στοιχεία μας είναι από 1 έως n μπορούμε να κάνουμε το εξής:

Κάθε στοιχείο του A , έστω $A[i]$ το τοποθετούμε στην θέση $A'[A[i]]$, άμα κάποια στιγμή πάμε να τοποθετήσουμε ένα στοιχείο και στην αντίστοιχη θέση του A' υπάρχει ήδη κάποιο, τότε βρήκαμε την απάντηση. Προκειμένου να κάνουμε την όλη διαδικασία in-place, παρατηρούμε πως μπορούμε απλά κάθε στοιχείο του πίνακα A , έστω $A[i]$ να το εναλλάσσουμε με το στοιχείο $A[A[i]]$. Αν κάποια στιγμή πριν το swap, ισχύει $A[A[i]] = A[i]$, δεδομένου βέβαια πως $i \neq A[i]$, τότε βρήκαμε το στοιχείο που εμφανίζεται τουλάχιστον 2 φορές.

Άσκηση 5: Αθροίσματα Στοιχείων σε Συγκεκριμένο Διάστημα

(α) Εφαρμόζουμε τον ακόλουθο μετασχηματισμό στο πρόβλημα, σε κάθε στοιχείο του πίνακα A , αφαιρούμε το L και η ανισότητα μετασχηματίζεται σε $0 \leq B[j] - A[i] \leq M - L$. Θέτουμε $M - L = K$ και έχουμε $0 \leq B[j] - A[i] \leq K$.

Τώρα, εφαρμόζουμε την τεχνική των 2 pointers. Ξεκινάμε 2 δείκτες, έστω i, j από την αρχή του πίνακα A και B αντίστοιχα. Για σταθερή τιμή του i , προχωράμε τον δείκτη j , μέχρις ώτου η διαφορά $B[j] - A[i]$ να πάρει την μέγιστη δυνατή τιμή χωρίς να ξεπεράσει το K . Μόλις βρούμε αυτή την θέση j , αυξάνουμε την απάντησή μας κατά $j-i+1$, αφού κάθε διάστημα $[i, k]$ με $k \leq j$ είναι έγκυρη λύση. Κατόπιν, αυξάνουμε την τιμή του δείκτη i και συνεχίζουμε την ίδια διαδικασία με τον j από εκεί που σταμάτησε πριν.

Ορθότητα: Για κάθε τιμή του i μετράμε όλα τα πιθανά διαστήματα τα οποία είναι έγκυρα και ξεκινάνε από i , συνεπώς αφού διατρέχουμε όλες τις πιθανές τιμές του δείκτη i ο αλγόριθμός μας θα υπολογίσει σωστά το πλήθος των διαστημάτων με διαφορά μικρότερη ή ίση του K .

Πολυπλοκότητα: Συνολικά, ο δείκτης j διατρέχει ακριβώς μια φορά όλο τον πίνακα B και ο δείκτης i ακριβώς μια φορά τον πίνακα A . Σε κάθε βήμα έχουμε σταθερό πλήθος πράξεων, συνεπώς η πολυπλοκότητα του αλγορίθμου μας είναι $O(n + m)$.

β) Έστω πως έχουμε τον αλγόριθμο του ερωτήματος (α) ως $\text{count}(\text{int } A[], \text{int } B[])$. Τότε κάνουμε το εξής, κάθε φορά διαχωρίζουμε τον πίνακα μας στην μέση και υπολογίζουμε αναδρομικά την απάντηση για τους υποπίνακες στους οποίους τον χωρίσαμε. Έπειτα, στην φάση της σύνθεσης, μένουμε με 2 πίνακες για τους οποίους θέλουμε να υπολογίζουμε τα ζεύγη για τα οποία η διαφορά 2 στοιχείων των πινάκων φράζεται από τα L, M . Δηλαδή έχουμε το πρόβλημα του ερωτήματος (α).

Αλγόριθμος:

Input: Πίνακας A , ακέραιος n .

```
int merge_and_count( int A[], int l, int r ):
    if(l == r) return → 0
    int mid = (l + r) / 2
    res = merge_and_count(A, l, mid) + merge_and_count(A, mid + 1, r)
    res = res + count(A[1 ... mid], A[mid+1 ... r])
    return res
```

Output: $\text{merge_and_count}(A, 1, n)$

Ορθότητα: Η ορθότητα του αλγορίθμου προκύπτει επαγωγικά από την ορθότητα κάθε επιπέδου της αναδρομής. Σε κάθε σημείο δεδομένου ότι το “σπάσιμο” θα μου επιστρέψει σωστή απάντηση για τους 2 υποπίνακες, γνωρίζοντας (από το ερώτημα α) ότι ο αλγόριθμος count είναι ορθός υπολογίζει την απάντηση και για τα ζευγάρια (i, j) για τα οποία ισχύει ($i \leq mid < j$). Συνεπώς, στο τέλος κάθε επίπεδο της αναδρομής μας επιστρέφει την σωστή απάντηση και συνεπώς η αρχική μας κλήση $\text{merge_and_count}(A, 1, n)$ μας υπολογίζει την σωστή απάντηση.

Πολυπλοκότητα: Η πολυπλοκότητα του αλγορίθμου μας είναι η ίδια με την mergesort. Έχουμε για τον χρόνο εκτέλεσης για πίνακα με n στοιχεία:

$$T(n) = 2 * T\left(\frac{n}{2}\right) + O(n) = O(n \log n)$$

γ) Εφαρμόζουμε τον αλγόριθμο του ερωτήματος (β) στον πίνακα των μερικών αθροισμάτων του πίνακα A . Δηλαδή υπολογίζουμε έναν πίνακα S , για τον οποίο ισχύει:

$$S[0] = 0, S[i] = S[i-1] + A[i] \quad \forall i \leq n.$$

Συνεπώς τώρα έχουμε τον εξής μετασχηματισμό του προβλήματος. Αναζητούμε όλα τα ζεύγη (i,j) για τα οποία ισχύει $L \leq \sum_{k=i}^j A[k] \leq M$. Όμως είναι $S[j] - S[i-1] = \sum_{k=i}^j A[k]$, άρα αναζητούμε τα ζεύγη (i,j) για τα οποία ισχύει $L \leq S[j] - S[i-1] \leq M$, το οποίο είναι ισοδύναμο με το πρόβλημα του ερωτήματος (β). Αν ο πίνακας A αποτελείται από θετικούς ακεραίους, τότε ο πίνακας S είναι ταξινομημένος σε αύξουσα σειρά, συνεπώς μπορούμε να εφαρμόσουμε απευθείας τον αλγόριθμο του ερωτήματος (α).

Η πολυπλοκότητα του αλγορίθμου μας είναι $O(n \log n)$, σε περίπτωση που ο πίνακας A περιέχει και αρνητικούς αριθμούς και $O(n)$ στην περίπτωση που αποτελείται μόνο από θετικούς ακέραιους.