Δάσκος Ραφαήλ – Α.Μ.: 03116049
Αλγόριθμοι και Πολυπλοκότητα
Ροή Λ: Λογισμικό Η/Υ
Σ.Η.Μ.Μ.Υ. – Ε.Μ.Π. – 7ο εξάμηνο
2η σειρά γραπτών ασκήσεων

Άσκηση 1: Μεταφορά δεμάτων

(α) 1. Δε γίνεται να δημιουργηθεί ασφαλή στοίβαξη ταξινομώντας τα κουτιά με φθίνουσα σειρά βάρους. Αντιπαράδειγμα: αν θεωρήσουμε 2 κουτιά με $w_1 = 5$, $d_1 = 3$ και $w_1 = 4$, $d_1 = 6$. Υπάρχει ασφαλής στοίβαξη αν βάλουμε κάτω το 2ο κουτί, όμως με βάση το βάρος το 1ο δε θα αντέξει το άλλο, οπότε και δεν μπορούμε να τα ταξινομήσουμε σύμφωνα με αυτό το κριτήριο.

2. Δε γίνεται να δημιουργηθεί ασφαλή στοίβαξη ούτε ταξινομώντας τα κουτιά με φθίνουσα σειρά αντοχής. Αντιπαράδειγμα: αν θεωρήσουμε 2 κουτιά με $w_1 = 2$, $d_1 = 6$ και $w_1 = 7$, $d_1 = 4$. Υπάρχει ασφαλής στοίβαξη αν βάλουμε κάτω το 2ο κουτί, όμως με βάση το βάρος το 1ο δε θα αντέξει το άλλο, οπότε και δεν μπορούμε να τα ταξινομήσουμε σύμφωνα με αυτό το κριτήριο.

3. Εφόσον υπάρχει κάποια ασφαλής στοίβαξη μπορούμε να τοποθετήσουμε τα κουτιά με φθίνουσα σειρά του αθροίσματος του βάρους και της αντοχής. Γενικά, αν υπάρχει ασφαλής στοίβαξη σημαίνει ότι $d_i \geq \sum_{j=i+1}^{n} w_j$ για κάθε $i \leq n$. Θα θεωρήσουμε ότι στην αρχή έχουμε δύο πακέτα και θα συγκρίνουμε τι ισχύει για το $w_i + d_i$ του ενός σε σχέση με το άλλο. Επίσης για να έχουμε μια ασφαλή στοίβαξη θα πρέπει $d_1 \geq w_2$ εάν θέλω να τοποθετήσω το 1ο πακέτο κάτω. Έτσι $w_1 + d_1 \geq \sum_{j=1}^{2} w_j = w_1 + w_2$. Αν $w_1 < d_2$ και οδηγούμαστε στο $w_1 + d_1 < w_2 + d_2$ μπορώ να θεωρήσω τα πακέτα μου ανάποδα και ισχύει ο κανόνας και έχω και ασφαλή στοίβαξη. Αν είναι όμως μεγαλύτερο ή ίσο πάλι ισχύει ο κανόνας και θα έχω ασφαλή στοίβαξη.

Βάζω τώρα ένα 3ο πακέτο με d_1 ώστε να έχω ασφαλή στοίβαξη και δεν μπορώ να έχω αλλιώς ασφάλεια, δηλαδή $d_2 < w_1 + w_3$ και $d_3 < w_1 + w_3$ (αυτό είναι το πακέτο 1 και μετά έχω στα πακέτα 2 και 3 τη σχέση που είπαμε με τα 2 πακέτα, δηλαδή $w_3 + d_3 \leq w_2 + d_2$). Έτσι $w_1 + d_1 \geq \sum_{j=1}^{3} w_j = w_1 + w_2 + w_3$. Για να μην ισχύει το κριτήριο θα πρέπει να έχω $w_1 + d_1 < w_i + d_i$ με $i = 1$ ή 2. Ας πούμε ότι ισχύει για το 2. Τότε θα πάρουμε επειδή είπαμε ότι αλλιώς δεν έχω ασφάλεια ότι $w_2 + d_2 < w_1 + w_2 + w_3 \leq w_1 + d_1$, οπότε αυτό δε γίνεται. Ομοίως και αν θεωρήσουμε το 1.

Τώρα θα πούμε ότι έχω n αντικείμενα ταξινομημένα, στα οποία ισχύει το κριτήριο μας και βρισκόμαστε σε ασφαλή στοίβαξη, και θέλω να βάλω άλλο 1 για να έχω πάλι ασφαλή στοίβαξη τοποθετώντας το στην αρχή. Για να ισχύει αυτό πρέπει $w_1 + d_1 \geq \sum_{j=1}^{n+1} w_j$ αφού έχω n+1 κιβώτια. Θα πρέπει για να μην ισχύει το κριτήριο να έχω ότι $w_1 +$

$d_1 < w_i + d_i$ για κάποιο $i \leq n+1$. Γι' αυτό το i θα έχω $w_i + d_i < \sum_{j=i}^{n+1} w_j < \sum_{j=1}^{n+1} w_j \leq w_1 + d_1$, επομένως πρέπει να ισχύει το κριτήριο. Επομένως επαγωγικά παίρνουμε ότι αν υπάρχει κάποια ασφαλής στοίβαξη μπορούμε να τη βρούμε αν τοποθετήσουμε τα κιβώτια που έχουμε κατά φθίνουσα σειρά αθροίσματος βάρους και αντοχής

(β) Αρχικά ταξινομούμε τα στοιχεία κατά αύξουσα σειρά όπως το κριτήριο 3 του (α). Απλά σε αντίθεση με το 3 θεωρούμε το 1 στοιχείο ότι είναι αυτό με το μικρότερο άθροισμα που σημαίνει ότι μπορεί να μπει μόνο στην κορυφή της στοίβαξής μας. Κατά την ταξινόμηση αυτή υπολογίζουμε και ποια είναι η μέγιστη τιμή του d_i που εμφανίζεται.
Στη συνέχεια φτιάχνουμε έναν πίνακα που θα έχει ως γραμμές τα στοιχεία μας και ως στήλες την αντοχή d που θα κυμαίνεται μεταξύ 0 και d_{max} που έχουμε βρει. Πολύ απλά αυτό που μας λέει η κάθε στήλη θα είναι αν είχαμε κάτω από τα n κουτιά ένα ακόμα τι κέρδος θα παίρναμε αναλόγως με την αντοχή που θα είχε. Επειδή από κάτω έχουμε το δάπεδο (θεωρητικά άπειρη αντοχή) η λύση θα είναι το κέρδος που θα υπολογίσουμε στο κελί (n, d_{max}).
Τώρα για να δούμε πως επηρεάζει η κάθε γραμμή την επόμενη θα πρέπει να βρούμε πως γεμίζει η κάθε στήλη του πίνακα για τη βελτιστοποίηση του κόστους που θα έχουμε στο τέλος. Με το $1^ο$ στοιχείο είναι πολύ απλό. Μέχρι να φτάσουμε στο $d = w_1$ θα έχουμε 0 στα στοιχεία της γραμμής αλλά από εκεί και πέρα θα έχουμε κέρδος p_1 μέχρι το d_{max}. Όταν θα πάμε στο επόμενο στοιχείο θα έχουμε ότι μέχρι το $d = w_2$ θα παίρνουμε τη βέλτιστη λύση της προηγούμενης γραμμής και μετά από αυτό, επειδή μπορούμε να πάρουμε το $2^ο$ στοιχείο θα έχουμε το $y = \min\{d-w_2, d_2\}$ που είναι να μπορεί το επόμενο στοιχείο να σηκώσει τα δύο πρώτα και το $2^ο$ το $1^ο$. Έπειτα θα ελέγχουμε αυτό που έχουμε στην προηγούμενη γραμμή για το εκάστοτε d και το y και θα επιλέγουμε τη βέλτιστη τιμή. Οπότε η βέλτιστη λύση για κάποιο i (αριθμό στοιχείων) και κάποιο d είναι:

$$opt(i,d) = \max\{opt(i-1,d), opt(i-1, \min\{d-w_i, d_i\})\}$$

Εμάς η λύση όμως μας δίνεται από το κελί n, d_{max} όπως είπαμε, οπότε θα έχουμε $i = n$ και $d = d_{max}$.
Το κόστος αυτού του αλγορίθμου είναι $\Theta(n*d_{max})$ καθώς θα έχουμε τόσα γεμίσματα στον πίνακά μας.
Όσον αφορά την ορθότητα του αλγορίθμου μας, ισχύει ότι έχουμε σε κάθε περίπτωση του d τη βέλτιστη τιμή του να μην πάρουμε το στοιχείο => να έχουμε ότι και με τα προηγούμενα στοιχεία ή να το πάρουμε και να δούμε τι συμβαίνει αν προστεθεί και αυτό στο βέλτιστο κέρδος.

Άσκηση 2: Αναμνηστικά

Όπως μας λέει η εκφώνηση θεωρούμε ότι μπορούμε να πάρουμε τουλάχιστον ένα αναμνηστικό από την κάθε χώρα που επισκεπτόμαστε χωρίς στο τέλος να ξεπεράσουμε το συνολικό ζητούμενο κέρδος.

Έχουμε, λοιπόν, μια παραλλαγή του Knapack 0-1 μόνο που πρέπει να παίρνουμε μόνο ένα στοιχείο από κάθε χώρα.
Έτσι με τη λογική του δυναμικού προγραμματισμού θα έχουμε έναν δισδιάστατο πίνακα n*C, με δυνατό n πλήθος αναμνηστικών και C το ποσό που μπορούμε να διαθέσουμε. Με τα k_i αναμνηστικά να βρίσκονται το ένα μετά το άλλο και με τη σειρά των χωρών.
Πηγαίνουμε τώρα στην πρώτη χώρα και φτιάχνουμε τη γραμμή της βέλτιστης συναισθηματικής αξίας που κάποια στιγμή σε μια τιμή c θα πάρει τη βέλτιστη αξία του πιο φθηνού αναμνηστικού και από εκεί και πέρα θα συγκρίνει με τα υπόλοιπα (αν υπάρχουν), μέχρι την τιμή C. Για την επόμενη χώρα όμως θα θεωρήσουμε ως τιμή 0 την τιμή εκκίνησης των αναμνηστικών της προηγούμενης χώρας (έτσι, αν είχαμε κόστος 4 και ελάχιστο κόστος στη $2^η$ χώρα 2 θα ξεκινήσουμε να προσθέτουμε αξίες στην τιμή 6). Έτσι θα προκύψει η γραμμή για τις πρώτες δύο χώρες με κενό εκεί που δεν μπορούμε να πάρουμε δύο αναμνηστικά και τιμή εκεί που αυτό είναι δυνατό.
Στο τέλος της αναδρομής θα έχουμε τη ζητούμενη λύση στο κελί C του τελευταίου συνόλου.
Ο αλγόριθμος τρέχει σε χρόνο $\Theta(\sum_{i=1}^{n} k_i * C)$, καθώς για την εύρεση της πρώτης γραμμής έχουμε χρόνο k_1*C και πάει λέγοντας.
Η ορθότητα του αλγορίθμου είναι ευνόητη, καθώς έτσι βλέπουμε για κάθε τιμή του κόστους ποια είναι η βέλτιστη συναισθηματική αξία συνδυάζοντας προϊόντα από τις χώρες που μας δίνονται. Για την αναδρομή μπορούμε να χρησιμοποιήσουμε τη σχέση:

$opt(k_i, c) = \max(p_{ij} + opt(k_{i-1}, c-c_{ij})\}$, που i δείχνει τη χώρα και j το αναμνηστικό.

Άσκηση 3: Σοκολατάκια

Απαιτήσεις του προβλήματος είναι να μην ανοίξουμε κουτί με ίδια σοκολατάκια που έχουμε φάει ή με λιγότερα (ή ίδια) στο πλήθος σοκολατάκια με κάποιο προηγούμενο.
Επίσης αν το q_i στο κουτί που είμαστε είναι μεγαλύτερο από το q που έχουμε να φάμε για να είμαστε ικανοποιημένοι τότε δε χρειάζεται να κάνουμε άλλα βήματα. Επίσης θα θεωρήσουμε ότι αν δεν μπορούμε με κάποιον τρόπο να έχουμε τις απαιτήσεις του προβλήματός μας, τότε θα έχουμε άπειρο κόστος.
Φτιάχνω έναν πίνακα με n*Q. Έτσι το ελάχιστο κόστος θα είναι:

$$opt(i,q) = \begin{cases} 0, \text{αν το } q_i \geq Q \\ min\{opt(j, q - q_i) + |j-1|\}, \text{ανμπορούμεναπΦμεκΦπουαλλού} \\ \infty, \text{αν δεν μπορούμε να ικανοποιήσουμε κάποιο από τα κριτήρια} \end{cases}$$

Έτσι, ο πίνακας γεμίζει από τον ελάχιστο χρόνο που μπορείς να φας q στο πλήθος σοκολατάκια αν ξεκινάς από το κουτί που είναι για την κάθε γραμμή.
Αν δεν μπορείς να το πραγματοποιήσεις είναι άπειρο και αν δεν πρέπει να κάνεις βήμα είναι 0.
Η βέλτιστη λύση τελικά στο πρόβλημά μας δίνεται από την τελευταία στήλη του πίνακα συναρτήσει της αρχικής θέσης από την οποία είμαστε.

Συνολικά ο αλγόριθμος αυτός θέλει χρόνο Θ(n) για να υπολογίσει κάθε στοιχείο του πίνακα και αφού έχουμε n*Q στοιχεία θα θέλουμε συνολικό χρόνο $\Theta(n^2*Q)$.
Η ορθότητα αποδεικνύεται επαγωγικά. Για 1 κουτί ισχύει κανονικά ότι η βέλτιστη λύση είναι να φάμε τα σοκολατάκια του κουτιού. Υποθέτουμε ότι για n κουτιά μπορούμε να φάμε τα Q σοκολατάκια με τα ελάχιστα βήματα. Τώρα θα δούμε τι συμβαίνει στα n+1 κουτιά. Αν η βέλτιστη λύση δεν περιέχει το νέο κουτί τότε έχουμε τη βέλτιστη από το n. Αν, όμως, το περιέχει, τότε θα πρέπει να συμπεριληφθεί το κουτί και αφού διαλέγουμε πάντα το ελάχιστο κουτί θα διατηρηθεί ο κανόνας της βελτιστότητας.

Άσκηση 4: Απόσταση επεξεργασίας σε Αριθμητικές Εκφράσεις

Χρειάζεται να βρούμε ποια προβλήματα μπορεί να έχει η έκφραση x ώστε να μην ανήκει στη γλώσσα E_s:

- Να έχει διαφορετικό αριθμό παρενθέσεων που ανοίγουν και που κλείνουν.
- Να έχει συνεχόμενα μηδενικά χωρίς να προηγείται άλλο ψηφίο, π.χ. 00.
- Να έχει συνεχόμενους τελεστές, π.χ. ++.
- Να έχει αριθμό πριν από παρένθεση που ανοίγει, π.χ. 1(, ή μετά από παρένθεση που κλείνει, π.χ.)1.
- Να έχει τελεστή πριν από παρένθεση που κλείνει, π.χ. +), ή μετά από παρένθεση που ανοίγει, π.χ. (+.
- Να ξεκινάει ή να τελειώνει με τελεστή, π.χ. 1+ ή +1.
- Να έχει παρένθεση χωρίς στοιχείο ενδιάμεσα

Τώρα να δούμε πως μπορούμε να εξαλείψουμε αυτά τα προβλήματα που έχουμε. Αρχικά με ένα πέρασμα στη λέξη σε χρόνο n μπορούμε να βρούμε εάν έχουμε θέμα με τις παρενθέσεις και να διατηρήσουμε ποια είναι αυτή η διαφορά, αν πρέπει π.χ. να προσθέσουμε 2) που αντιστοιχεί και στο να αφαιρέσουμε 2 (.
Αν το πρώτο μας στοιχείο είναι αποδεκτό, δηλαδή είτε παρένθεση είτε αριθμός, διαφορετικό του 0 τότε δε χρειάζεται να πραγματοποιήσουμε κάποια αλλαγή, σε αντίθετη περίπτωση το αποτέλεσμά μας είναι το ελάχιστο των μεταβολών που θα έχουμε για την υπόλοιπη συμβολοσειρά (x χωρίς το αρχικό στοιχείο) με το να αλλάξουμε το στοιχείο, να προσθέσουμε κάτι πριν από αυτό ή να το διαγράψουμε.
Στις επόμενες επαναλήψεις θα κοιτάμε ποιο είναι το προηγούμενο στοιχείο και αν χρειάζεται κάποια αλλαγή το στοιχείο που ελέγχουμε εξαιτίας αυτού θα έχουμε μια από τις κινήσεις που μπορούμε να πραγματοποιήσουμε και συνεπακόλουθα θα αυξήσουμε το αποτέλεσμα που μας επιστρέφει. Πάλι θα έχουμε το ελάχιστο της υπόλοιπης συμβολοσειράς και κρατάμε και το στοιχείο που κοιτάξαμε για να μπορέσουμε να τσεκάρουμε μετά το επόμενο.
Συνεχίζουμε με τον ίδιο τρόπο μέχρι να φτάσουμε σε μια τετρημένη συμβολοσειρά που επειδή είναι το τελευταίο στοιχείο μπορεί να είναι αριθμός ή κλείσιμο παρένθεσης, εφόσον έχει ανοίξει εκείνη κάπου πριν. Για να σιγουρευτούμε ότι έχουμε σωστό αριθμό

παρενθέσεων θα πρέπει να κοιτάμε το αν μπορούμε να αλλάξουμε κάποια από τα στοιχεία με παρένθεση στο πέρασμά μας (χωρίς να δημιουργούμε προβλήματα) και να οδηγηθούμε σε καλό αποτέλεσμα.
Η κλήση του αλγορίθμου μας έχει ως αποτέλεσμα n^2 επαναλήψεις για κάθε ένα από τα στοιχεία του προβλήματος και επειδή έχουμε n στο πλήθος γράμματα θα έχουμε εν τέλει $\Theta(n^3)$ πράξεις. Ο αλγόριθμος αυτός μας δίνει στο τέλος το επιθυμητό αποτέλεσμα καθώς έχουμε πάρει κάθε φορά την ελάχιστη τιμή από το υποπρόβλημα που έχουμε (την x χωρίς το πρώτο στοιχείο και από αυτό κοιτάμε με το προηγούμενο πως το επηρεάζει.

Άσκηση 5: Καλύπτοντας ένα Δέντρο

(α) φτιάχνουμε έναν πίνακα n*n. Γεμίζουμε τον πίνακα επιλέγοντας με βάση το πώς επηρεάζει ένας κόμβος έναν άλλον, αν επιλέξουμε τον πρώτο να είναι στους K κόμβους. Επιλέγουμε τα μέγιστα όλων των γραμμών και διαλέγουμε τον κόμβο που έχει το ελάχιστο από αυτά τα μέγιστα. Στη συνέχεια κοιτάμε πως επηρεάζουν οι κόμβοι τους υπόλοιπους διατηρώντας σταθερά τα στοιχεία που επηρεάζουν οι προηγούμενα εισαγόμενοι κόμβοι και όχι αυτός. Αν το επηρεάζουν και οι δύο παίρνουμε το καλύτερο κόστος. Και συνεχίζουμε έτσι n φορές και στο τέλος το συνολικό κόστος θα είναι το μέγιστο κόστος που εμφανίζεται στον πίνακα. Αυτός όμως ο αλγόριθμος έχει πολυπλοκότητα $\Theta(n^2*k)$ αφού το γέμισμα του πίνακα γίνεται σε χρόνο n^2 και γίνεται για k φορές.

(β) Θέλουμε να βρούμε έναν αποδοτικό αλγόριθμο που με είσοδο το δέντρο T (V, E), τη ρίζα r του T και έναν ακέραιο αριθμό $z \leq n$ να υπολογίζει το ελάχιστο μέγεθος συνόλου $K \subseteq V$ για το οποίο $cost(K) \leq z$. Για να έχουμε βέλτιστο αλγόριθμου δυναμικού προγραμματισμού χρειάζεται να αρχίσουμε από τα φύλλα γιατί όσο πάμε προς κάτω το δέντρο «ανοίγει». Αρχικά θα κάνουμε μια αναδρομή σε όλους τους κόμβους του δέντρου για να βρούμε ποιοι από αυτούς είναι φύλλα και θα τους τοποθετήσουμε σε μια ουρά. Δε μας ενδιαφέρει η σειρά που θα τους βάλουμε. Αλλά κατά το πέρασμά μας χρειάζεται να κρατάμε και πόσα παιδιά έχει ο κάθε πατέρας (numChild), γιατί μόνο όταν βγουν και τα δύο παιδιά (ή όσα έχει) ενός κόμβου θα τον τοποθετήσουμε στην ουρά μας για να τον ελέγξουμε πιο μετά. Επίσης θα έχουμε αποθηκεύσει και σε έναν πίνακα με n στοιχεία τον πατέρα του κάθε κόμβου (myFather). Και τέλος θα έχουμε έναν ακόμα πίνακα n που θα έχει το ποιο κόστος (maxCost) του ήρθε από το παιδί του αρχικοποιημένο στο 0.
Σε αυτό το σημείο έχουμε πολυπλοκότητα O(n) καθώς οι πίνακες κάθε φορά γεμίζουν σε O(1). Τώρα μας ενδιαφέρει να δούμε ποια θα είναι η αναδρομή μας. Πώς, δηλαδή, θα επιλέξουμε ποια στοιχεία θα βάλουμε για να βρούμε το πλήθος του K;
Σκεφτόμαστε ότι για να έχουμε το πολύ κόστος z χρειάζεται να ανέβουμε από το πιο απομακρυσμένο παιδί κατά z και να βάλουμε εκείνο τον κόμβο αυξάνοντας το K.

Αρχίζουμε, λοιπόν, με τα φύλλα στην ουρά. Διαγράφουμε το φύλλο που βρίσκεται στην κεφαλή της ουράς και κοιτάμε τον πατέρα του, μέσω του myFather και από εκεί, αν αφαιρέσουμε 1 από το numChild και είναι ίσο με 0, τότε τοποθετούμε τον πατέρα του στο τέλος της ουράς. Επίσης κοιτάμε και το maxCost = max{maxCost(που ήδη υπήρχε), maxCost(παιδιού που αφαιρέσαμε) + 1}. Αν όμως το maxCost τη στιγμή που τοποθετούμε το στοιχείο στην ουρά είναι ίσο με z, τότε θα πρέπει να αυξήσουμε το K και να του βάλουμε το maxCost = 0. Εάν φτάσουμε στην ρίζα τότε αναγκαστικά την τοποθετούμε και αυξάνουμε κατά 1 το K καθώς εάν δεν το κάναμε θα είχαμε άπειρο συνολικό κόστος. Έτσι έχουμε υπολογίσει το ελάχιστο αριθμό του K των κόμβων που πρέπει να επιλέξουμε, ώστε το δέντρο μας να έχει κόστος ίσο με z.

Η πολυπλοκότητα αυτού του αλγορίθμου είναι ίση με O(n) έχουμε πει ότι μέχρι πριν τους ελέγχους έχουμε χρόνο ίσο με O(n). Οι έλεγχοι, όμως, είναι και αυτοί n στο πλήθος με τις πράξεις που έχουν να εκτελέσουν να έχουν πολυπλοκότητα O(1). Οπότε συνολικά ο αλγόριθμος αυτό είναι O(n).

Ο αλγόριθμος αυτός μπορεί να χρησιμοποιηθεί για την επίλυση σε καλύτερο χρόνο από το (α) το πρόβλημα που μας δίνεται. Θα χρησιμοποιήσουμε δυαδική αναζήτηση για να βρούμε το κόστος του K.

Ξεκινάμε από το z = n/2 και βρίσκουμε ένα K_t που αντιστοιχεί σε αυτό αν το $K_t < K$, τότε πάει να πει ότι το κόστος θα βρίσκεται μεταξύ του (0,n/2) διαστήματος, αλλιώς θα βρίσκεται στο από πάνω διάστημα. Και συνεχίζουμε έτσι έως ότου να βρούμε το ποιο στοιχείο z μας δίνει K κόμβους. Όταν θα βρούμε αυτό κάνουμε μια δυαδική αναζήτηση στο τελευταίο μικρότερο διάστημα που έχουμε, γιατί μπορεί να υπάρχει και άλλο πιο μικρό z που να μας δίνει το ίδιο K και εφόσον θέλουμε το ελάχιστο κόστος, αυτό μας νοιάζει να βρούμε. Αν δε βρούμε άλλο ίδιο κρατάμε το προηγούμενο z αλλιώς το πιο μικρό από τα z που μας ικανοποιούν.

Έχουμε ότι ο αλγόριθμος που χρησιμοποιούμε κάθε φορά έχει κόστος O(n) και θα τον χρησιμοποιήσουμε το πολύ logn φορές οπότε συνολικά ο τελικός μας αλγόριθμος θα έχει πολυπλοκότητα ίση με O(nlogn).