# ΑΛΓΟΡΙΘΜΟΙ ΚΑΙ ΠΟΛΥΠΛΟΚΟΤΗΤΑ

## 1η ΣΕΙΡΑ ΓΡΑΠΤΩΝ ΑΣΚΗΣΕΩΝ

Αλέξανδρος Κουλάκος

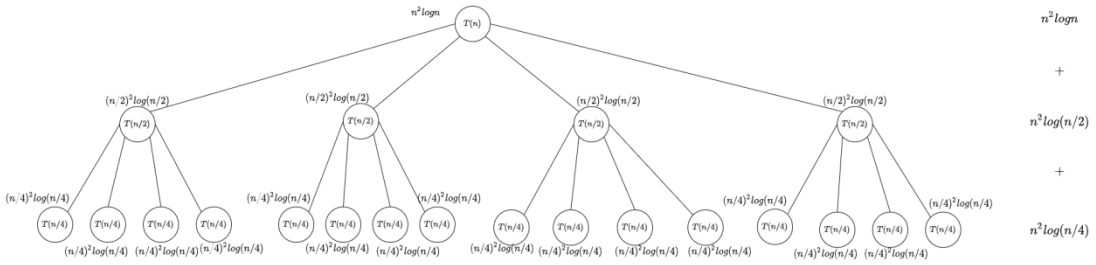
03118144

ΣΗΜΜΥ, 7ο Εξάμηνο

Νοέμβριος, 2021

## Άσκηση 1

**1.** $T(n) = 4T(n/2) + \Theta(n^2 \log n)$




Κατασκευάζοντας το δέντρο της αναδρομής για 3 ενδεικτικά επίπεδα, εύκολα συμπεραίνουμε ότι στο επίπεδο $i$ το συνολικό κόστος αφού επιστρέψει η αναδρομή είναι $cn^2 \log(\frac{n}{2^i})$. Όμως, σε κάθε βήμα της αναδρομής, το μέγεθος του προβλήματος υποδιπλασιάζεται (από το $n$ στο $n/2$, από το $n/2$ στο $n/4$ κοκ μέχρι να φτάσουμε στο μοναδιαίο μέγεθος). Έτσι, το πλήθος των επιπέδων του δέντρου της αναδρομής είναι $\log_2(n) + 1$ (για συντομογραφία θα γράφουμε $\log n$ αντί για $\log_2(n)$) και το συνολικό κόστος του αλγορίθμου υπολογίζεται αν αθροίσουμε το συνολικό κόστος ανά επίπεδο για όλα τα επίπεδα. Συγκεκριμένα,

$$T(n) = \sum_{i=0}^{\log(n)} cn^2 \log(\frac{n}{2^i}) = cn^2 \sum_{i=0}^{\log(n)} [\log n - i]$$

$$= cn^2 \left[\log n\,(\log n + 1) - \frac{1}{2}\log n\,(\log n + 1)\right]$$

$$= cn^2 \left[\log^2 n + \log n - \frac{1}{2}\log^2 n - \frac{1}{2}\log n\right]$$

$$= \frac{1}{2} cn^2 (\log^2 n + \log n) = \frac{1}{2} cn^2 \log n (\log n + 1)$$

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

**2.** $T(n) = 5T(n/2) + \Theta(n^2 \log n)$

Εδώ έχουμε $a = 5,\ b = 2$ και $f(n) = \Theta(n^2 \log n)$. Σχηματίζουμε τη συνάρτηση $n^{\log_b(a)} = n^{\log_2(5)} = n^{k+2}$, όπου $0 < k < 1$, εφόσον $2 < \log_2(5) < 3$. Θεωρούμε το πηλίκο

$$A(n) = \frac{n^{\log_2(5)}}{n^2 \log n} = \frac{n^{k+2}}{n^2 \log n} = \frac{n^k}{\log n}$$

Στο όριο για $n \to \infty$ έχουμε $\log n < n^l$, για κάθε $l > 0$.

Έτσι, επιλέγουμε $l < k$ και το πηλίκο γίνεται

$$A(n) = \frac{n^k}{\log n} > \frac{n^k}{n^l} = n^{k-l}$$

Ονομάζοντας $k - l = \varepsilon > 0$ έχουμε

$$n^{\log_2(5)} > (n^2 \log n) n^{\varepsilon}$$

Επομένως, το $n^{\log_2(5)}$ είναι πολυωνυμικά μεγαλύτερο από το $n^2 \log n$ και σύμφωνα με το $Master\ Theorem$ θα ισχύει ότι

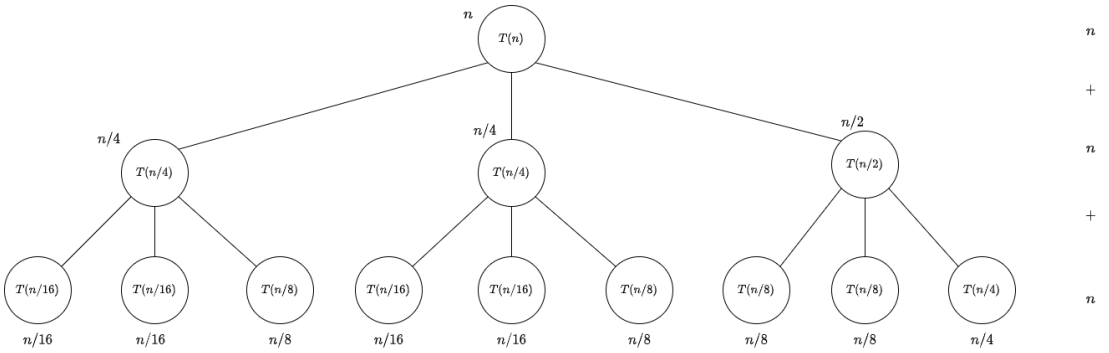
$$T(n) = \Theta\left(n^{\log_2(5)}\right)$$

**3.** $T(n) = T(n/4) + T(n/2) + \Theta(n)$

Εδώ έχουμε $\gamma_1 = 1/4$, $\gamma_2 = 1/2$ και $f(n) = \Theta(n)$. Επίσης, παρατηρούμε ότι $\gamma_1 + \gamma_2 = 3/4$, οπότε υπάρχει $\varepsilon = \frac{1}{5} > 0$ έτσι ώστε $\gamma_1 + \gamma_2 < 1 - \varepsilon$. Άρα, το $T(n)$ υπάγεται στις ειδικές μορφές που ξέρουμε και ως εκ τούτου ισχύει $T(n) = \Theta(n)$.

**Σχόλιο** Πρακτικά, η συνθήκη $\gamma_1 + \gamma_2 < 1 - \varepsilon$ μας λέει ότι καθώς προχωρά η αναδρομή, το μέγεθος του προβλήματος ελαττώνεται, συνεπώς από ένα σημείο και μετά κυριαρχεί το κόστος $f(n) = \Theta(n)$ ύστερα από την επιστροφή της αναδρομής.

**4.** $T(n) = 2T(n/4) + T(n/2) + \Theta(n)$



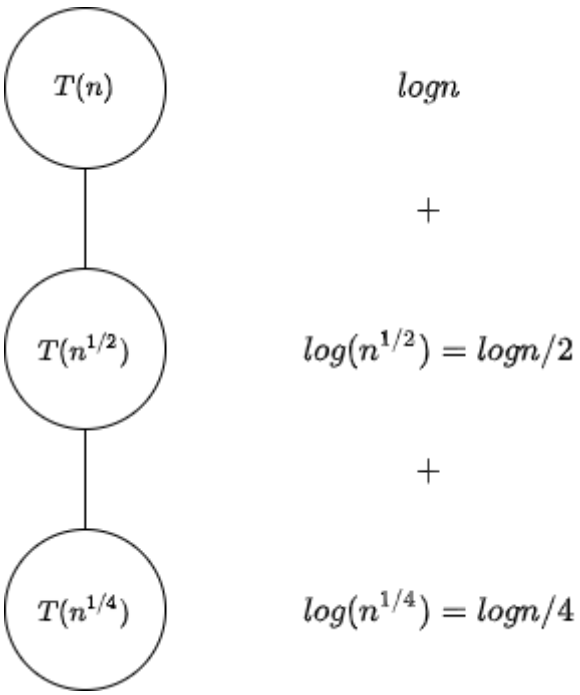

Κατασκευάζοντας το δέντρο της αναδρομής για 3 ενδεικτικά επίπεδα, εύκολα συμπεραίνουμε ότι σε κάθε επίπεδο $i$ το συνολικό κόστος αφού επιστρέψει η αναδρομή είναι $cn$. Όμως, σε κάθε βήμα της αναδρομής, το μέγεθος του προβλήματος υποτετραπλασιάζεται στη χειρότερη περίπτωση (από το $n$ στο $n/4$ και $n/2$, από το $n/2$ στο $n/8$ και $n/4$ κοκ μέχρι να φτάσουμε στο μοναδιαίο μέγεθος). Έτσι, το πλήθος των επιπέδων του δέντρου της αναδρομής είναι $\log_4(n) + 1$ και το συνολικό κόστος του αλγορίθμου υπολογίζεται αν αθροίσουμε το συνολικό κόστος ανά επίπεδο για όλα τα επίπεδα. Συγκεκριμένα,

$$T(n) = \sum_{i=0}^{\log_4(n)} cn = cn(1 + \log_4(n)) = cn\left(1 + \frac{\log_2(n)}{\log_2(4)}\right)$$

$$= cn\left(1 + \frac{1}{2}\log_2(n)\right) \Rightarrow T(n) = \Theta(\mathrm{nlog}\, n)$$

**Σχόλιο** Η παραπάνω αναδρομική σχέση θυμίζει την $Merge\ Sort$. Σε κάθε βήμα το μέγεθος του προβλήματος παραμένει $n$ και δαπανάμε χρόνο $\Theta(n)$ για την αντίστοιχη συγχώνευση.

**5.** $T(n) = T\left(n^{1/2}\right) + \Theta(\log n)$




Κατασκευάζοντας το δέντρο της αναδρομής για 3 ενδεικτικά επίπεδα, εύκολα συμπεραίνουμε ότι σε κάθε επίπεδο $i$ το συνολικό κόστος αφού επιστρέψει η αναδρομή είναι $\frac{\log n}{2^i}$. Όμως, σε κάθε βήμα της αναδρομής, το μέγεθος του προβλήματος τετραγωνίζεται (από το $n$ στο $\sqrt{n} = n^{1/2}$, από το $\sqrt{n}$ στο $\sqrt{\sqrt{n}} = n^{1/4}$, από το $\sqrt{\sqrt{n}}$ στο $\sqrt{\sqrt{\sqrt{n}}} = n^{1/8}$ κοκ μέχρι να φτάσουμε στο μοναδιαίο μέγεθος). Έτσι, το πλήθος των επιπέδων του δέντρου της αναδρομής είναι $\log_2(n) + 1$ (για συντομογραφία θα γράφουμε $\log n$ αντί για $\log_2(n)$) και το συνολικό κόστος του αλγορίθμου υπολογίζεται αν αθροίσουμε το συνολικό κόστος ανά επίπεδο για όλα τα επίπεδα. Συγκεκριμένα,

$$T(n) = \sum_{i=0}^{\log n} \frac{\log n}{2^i} = \log n \sum_{i=0}^{\log n} \left(\frac{1}{2}\right)^i = \log n \frac{1 - \left(\frac{1}{2}\right)^{1+\log n}}{1 - \frac{1}{2}}$$

$$= 2\log n \left(1 - \frac{1}{2}\frac{1}{2^{\log n}}\right) = 2\log n \left(1 - \frac{1}{2n}\right) \Rightarrow T(n) = \Theta(\log n),$$

εφόσον $\lim_{n\to\infty} \frac{1}{n} = 0$, δηλαδή ο όρος $\frac{1}{2n}$ είναι αμελητέος ασυμπτωτικά.

**6.** $T(n) = T(n/4) + \Theta\left(\sqrt{n}\right)$

Εδώ έχουμε $a = 1$, $b = 4$ και $f(n) = \Theta\left(\sqrt{n}\right) = \Theta\left(n^{1/2}\right)$, δηλαδή $f(n) = \Theta(n^d)$, για $d = \frac{1}{2}$.

Παρατηρούμε ότι η εν λόγω αναδρομική σχέση υπάγεται στις ειδικές μορφές και ισχύει $d = \frac{1}{2} > n^{\log_b(a)} = n^{\log_4(1)} = n^0 = 1$. Άρα, ξέρουμε ότι $T(n) = \Theta\left(\sqrt{n}\right)$.

## Άσκηση 2

**(α)** Λόγω του ζητούμενου της ταξινόμησης σε αύξουσα σειρά, στόχος μας είναι στο τέλος ο πίνακας $A$ να περιέχει αριστερά τα «μικρά» στοιχεία $1, 2, 3, \ldots$ και «δεξιά» τα μεγάλα στοιχεία $n, n-1, n-2, \ldots$

Όπως θα φανεί παρακάτω, ο αλγόριθμός μας βασίζεται στη μετακίνηση του μεγαλύτερου κάθε φορά στοιχείου στη δεξιά μεριά του $A$. Αυτή η πρακτική είναι προτιμότερη από τη μετακίνηση του μικρότερου κάθε φορά στοιχείου στην αριστερή μεριά του $A$, διότι το στοιχείο αυτό ανήκει σε κάθε πρόθεμα του $A$, οπότε κάθε προθεματική ταξινόμηση αλλοιώνει τη σωστή του θέση στον πίνακα. Αντίθετα, στοιχεία που ανήκουν στη δεξιά μεριά του $A$ μπορούν να παραμείνουν στη σωστή τους θέση επιλέγοντας προθέματα του πίνακα τα οποία δεν περιλαμβάνουν τα εν λόγω σωστά ταξινομημένα στοιχεία.

Στην ουσία του αλγορίθμου, σε κάθε επανάληψη αναζητούμε το κατά περίπτωση μεγαλύτερο στοιχείο του πίνακα, ας το ονομάσουμε $m$ (π.χ. στην επανάληψη υπ' αριθμόν $0$ αναζητούμε το στοιχείο $n$, στην επανάληψη υπ' αριθμόν $1$ αναζητούμε το $n-1$ και εν γένει στην επανάληψη υπ' αριθμόν $i$ αναζητούμε το $n-i$). Εφαρμόζοντας προθεματική ταξινόμηση για τη θέση του $A$ στην οποία βρίσκεται το $m$, το $m$ μετακινείται στην πρώτη θέση του πίνακα και στη συνέχεια με άλλη μια προθεματική ταξινόμηση για τη θέση $n-i$, το $m$ μετακινείται στην υπ' αριθμόν $i$ θέση από τα δεξιά του $A$ (π.χ. στην επανάληψη υπ' αριθμόν $0$ το στοιχείο $m = n$ θα τοποθετηθεί στη θέση $n$, στην επανάληψη υπ' αριθμόν $1$ το στοιχείο $m = n-1$ θα τοποθετηθεί στη θέση $n-1$ κοκ). Η παραπάνω διαδικασία επαναλαμβάνεται έως ότου $m = 0$. Η αναλλοίωτη συνθήκη που εγγυάται την ορθότητα του αλγορίθμου είναι ότι από τη στιγμή που το στοιχείο $m$ εισάγεται στη σωστή του θέση, δεν πρόκειται να μετακινηθεί ποτέ ξανά από τον αλγόριθμο. Αυτό ισχύει επαγωγικά για όλα τα $m = n-i$, με $i = 1, 2, \ldots, n$ και έτσι ο αλγόριθμος παράγει την επιθυμητή ταξινόμηση σε αύξουσα σειρά.

Τέλος, από τα παραπάνω γίνεται φανερό ότι για να τοποθετηθεί το στοιχείο $m$ απαιτούνται ακριβώς $2$ προθεματικές ταξινομήσεις. Όμως, το $m$ μπορεί να λάβει $n$ διαφορετικές τιμές, οπότε για την ταξινόμηση του πίνακα $A$ θα χρειαστούν $2n$ προθεματικές ταξινομήσεις. Η παραπάνω

επίδοση μπορεί να βελτιωθεί αν στον αλγόριθμό μας προσθέσουμε μια συνθήκη η οποία ελέγχει αν το στοιχείο $m = n - i$ με $i = 1, 2, \dots, n$ βρίσκεται κάθε φορά εκ των προτέρων στη σωστή του θέση (δηλαδή στη θέση $n - i$). Έτσι, η τοποθέτηση κάθε στοιχείου $m$ στη σωστή του θέση απαιτεί **στη χειρότερη περίπτωση** δύο προθεματικές ταξινομήσεις, εφόσον το $m$ δεν ανήκει εκ των προτέρων στη σωστή του θέση. Επίσης, παρατηρώντας ότι η ταξινόμηση των $n - 2$ μεγαλύτερων στοιχείων του $A$ εγγυάται ότι τα δύο εναπομείναντα στοιχεία μπορούν να ταξινομηθούν με το πολύ μία κίνηση, συμπεραίνουμε ότι ο αλγόριθμός μας εφαρμόζει το πολύ $2(n - 2) + 1 = 2n - 3$ προθεματικές ταξινομήσεις.

**(β)** Αρκεί να επεκτείνουμε τον παραπάνω αλγόριθμο, έτσι ώστε μετά την τοποθέτηση του στοιχείου $m$ στη σωστή θέση του πίνακα όλα τα στοιχεία του $A$ να έχουν θετικά πρόσημα. Με αυτό τον τρόπο, η ίδια διαδικασία που περιγράφεται στο ερώτημα (α) μπορεί να επαναλαμβάνεται για κάθε $m = n - i$, με $i = 1, 2, \dots, n$ και έτσι ο αλγόριθμος να εγγυάται ότι θα παράξει ορθά τον $A$ ταξινομημένο σε αύξουσα σειρά.

Έστω ότι συναντάμε το στοιχείο $m = n - i$ για κάποιο $i = 1, 2, \dots, n$ στη θέση $j$ του πίνακα $A$, δηλαδή $m = A[j]$. Τότε, εφαρμόζοντας προσημασμένη προθεματική ταξινόμηση για τη θέση $j$, θα έχουμε $A[k] < 0$, για κάθε $k = 1, 2, \dots, j - 1, j$, ενώ τα στοιχεία $A[k]$, με $k = j + 1, \dots, n - 1, n$ θα παραμείνουν αναλλοίωτα ως προς το πρόσημό τους. Εφαρμόζοντας άλλη μια προσημασμένη προθεματική ταξινόμηση για τη θέση $n - i$ έχουμε ότι τα στοιχεία που ήταν προηγουμένως αρνητικά γίνονται θετικά, ενώ τα στοιχεία που ήταν προηγουμένως θετικά γίνονται αρνητικά και βρίσκονται στις $n - j$ πρώτες θέσεις του πίνακα. Έτσι εφαρμόζοντας και μια ακόμη προσημασμένη προθεματική ταξινόμηση για τη θέση $n - j$ τα πρόσημά τους διορθώνονται και όλα τα στοιχεία του πίνακα $A$ είναι θετικά.

Συνεπώς, η επέκταση του αλγορίθμου είναι απλά ότι σε κάθε $m$ αποθηκεύουμε τη θέση στην οποία τον βρήκαμε στον $A$ και αφού τοποθετήσουμε το $m$ στη σωστή του θέση σύμφωνα με τη διαδικασία του (α) ερωτήματος, εφαρμόζουμε και μια ακόμη προθεματική ταξινόμηση για να διορθώσουμε τα αρνητικά πρόσημα. Έτσι, η πολυπλοκότητα του αλγορίθμου αυξάνεται κατά μία προθεματική ταξινόμηση σε κάθε

επανάληψη. Άρα, ο αλγόριθμος εφαρμόζει το πολύ $3(n-1) = 3n-3$ προσημασμένες προθεματικές ταξινομήσεις.

**Σχόλιο** Το πρόσημο του κατά περίπτωση μεγαλύτερου στοιχείου δεν επηρεάζεται όταν το στοιχείο αυτό τοποθετείται στη σωστή του θέση, διότι σε κάθε επανάληψη συμμετέχει σε δύο προσημασμένες προθεματικές ταξινομήσεις, οπότε το πρόσημο παραμένει θετικό όπως ήταν και στην αρχή της εκάστοτε επανάληψης.

**(γ1)** Διακρίνουμε τις εξής δύο περιπτώσεις:

• Έστω ότι ο πίνακας $A_t$ έχει τουλάχιστον ένα θετικό στοιχείο. Έτσι, έχει νοήμα να ορίσουμε το μεγαλύτερο θετικό στοιχείο (ας το ονομάσουμε $x$) που συναντάμε στον $A_t$. Αν $x = n$, τότε με βάση την ανάλυση που προηγήθηκε στα ερωτήματα (α) και (β), χρησιμοποιώντας δύο προσημασμένες προθεματικές ταξινομήσεις μετακινούμε το $x$ στην τελευταία θέση του $A_t$ δημιουργώντας καταχρηστικά συμβατό ζεύγος. Διαφορετικά, ορίζουμε το στοιχείο $y = -(x+1)$ το οποίο είναι αρνητικό και θα δημιουργήσουμε συμβατό ζεύγος εστιάζοντας στα στοιχεία $x$, $y$ (π.χ. $x = 3$ και $y = -4$). Αν το $y$ βρίσκεται αριστερά από το $x$ στον πίνακα $A_t$, εφαρμόζουμε μια προσημασμένη προθεματική ταξινόμηση για τη θέση του $A_t$ στην οποία ανήκει το $x$. Έτσι, το στοιχείο $-x$ βρίσκεται τώρα στην αρχή του $A_t$ και ο πίνακας έχει τη μορφή $A_t = \{-x, \dots, x+1, \dots\}$. Εφαρμόζοντας ακόμα μια προσημασμένη προθεματική ταξινόμηση για τη θέση στην οποία ανήκει το στοιχείο αριστερά από το $x+1$, ο $A_t$ γίνεται $A_t = \{\dots, x, x+1, \dots\}$ και το συμβατό ζεύγος είναι το $(x, x+1)$ π.χ. $(3,4)$. Αν το $y$ βρίσκεται δεξιά από το $x$ στον πίνακα $A_t$, εφαρμόζουμε μια προσημασμένη προθεματική ταξινόμηση εφαρμόζουμε μια προσημασμένη προθεματική ταξινόμηση για τη θέση του $A_t$ στην οποία ανήκει το $y$. Έτσι, το στοιχείο $-y = x+1$ βρίσκεται τώρα στην αρχή του $A_t$ και ο πίνακας έχει τη μορφή $A_t = \{x+1, \dots, -x, \dots\}$. Εφαρμόζοντας ακόμα μια προσημασμένη προθεματική ταξινόμηση για τη θέση στην οποία ανήκει το στοιχείο αριστερά από το $-x$, ο $A_t$ γίνεται $A_t = \{\dots, -(x+1), -x, \dots\}$ και το συμβατό ζεύγος είναι το $(-(x+1), -x)$ π.χ. $(-4, -3)$.

• Έστω ότι ο πίνακας $A_t$ έχει μόνο αρνητικά στοιχεία. Τότε, από υπόθεση έχουμε ότι ο $A_t$ δεν είναι ταξινομημένος σε φθίνουσα σειρά, συνεπώς για κάποιο στοιχείο $x < 0$ το $x-1$ θα βρίσκεται στα αριστερά του

(ειδάλλως, αν δηλαδή το αμέσως μικρότερο στοιχείο κάθε στοιχείου βρίσκεται στα δεξιά του, επαγωγικά προκύπτει ότι ο $A_t$ είναι ταξινομημένος κατά φθίνουσα σειρά, το οποίο είναι άτοπο). Δηλαδή, ο $A_t$ είναι της μορφής $A_t = \{\dots, x-1, \dots, x\}$. Έτσι, εφαρμόζουμε μια προσημασμένη προθεματική ταξινόμηση για τη θέση στην οποία ανήκει το $x-1$ και ο $A_t$ γίνεται $A_t = \{-(x-1), \dots, x\}$. Εφαρμόζοντας ακόμα μια προσημασμένη προθεματική ταξινόμηση για τη θέση στην οποία ανήκει το στοιχείο αριστερά από το $x$, ο $A_t$ γίνεται $A_t = \{\dots, x-1, x, \dots\}$ και το συμβατό ζεύγος είναι το $(x-1, x)$.

Επομένως, σε κάθε περίπτωση, για τη δημιουργία συμβατού ζεύγους απαιτούνται το πολύ δύο προσημασμένες προθεματικές ταξινομήσεις.

■

**(γ2)**

Ο αλγόριθμος διατυπώνεται ως εξής:

Βήμα 1 (Απλοποίηση προβλήματος)

Απαλείφουμε αναδρομικά τα συμβατά ζεύγη αντικαθιστώντας καθένα από αυτά με κατάλληλο αριθμό το πρόσημο του οποίου ταυτίζεται με το κοινό πρόσημο των δύο στοιχείων του ζεύγους (π.χ. αν $[3, 4, 5]$ υποπίνακας του $A$, τότε στο συμβατό ζεύγος $(3, 4)$ δίνουμε την αντιπροσωπευτική τιμή 4 και απομένει $[4, 5]$ και στο συμβατό ζεύγος $(4, 5)$ δίνουμε την αντιπροσωπευτική τιμή 5). Έτσι, η διάσταση του πίνακα γίνεται $n' < n$, δηλαδή ελαττώνεται τουλάχιστον κατά ένα, οπότε το πρόβλημα απλοποιείται σημαντικά. Αν $n' = 0$, τότε ο πίνακας έχει ταξινομηθεί.

Βήμα 2

(α) Αν ο πίνακας $A$ είναι της μορφής $A = [-1, -2, \dots, -n']$, οπότε δεν είναι εφικτή η δημιουργία συμβατού ζεύγους με το πολύ δύο προσημασμένες επιθεματικές ταξινομήσεις, τότε ταξινομούμε τον πίνακα $A$ επαναλαμβάνοντας την παρακάτω διαδικασία $n'$ φορές:

• Εφαρμόζουμε μια προσημασμένη προθεματική ταξινόμηση σε ολόκληρο τον πίνακα.

• Εφαρμόζουμε μια προσημασμένη προθεματική ταξινόμηση στα $n' - 1$ αριστερότερα στοιχεία

Εύκολα φαίνεται ότι στην επανάληψη υπ' αριθμόν $i$, $i = 1, 2, \dots, n'$ το στοιχείο 1 τοποθετείται στη θέση $n' - (i - 1)$ και όλες οι θέσεις δεξιά από αυτή είναι ταξινομημένες σε αύξουσα σειρά, οπότε στην επανάληψη $n'$ ο πίνακας $A$ θα είναι ταξινομημένος.

(β) Αν ο πίνακας $A$ δεν είναι της μορφής $A = [-1, -2, \dots, -n']$, τότε δημιουργούμε συμβατό ζεύγος χρησιμοποιώντας το πολύ δύο προσημασμένες επιθεματικές ταξινομήσεις. Επιστροφή στο Βήμα 1

Έστω ότι η επαναληπτική δομή του αλγορίθμου εκτελείται $k$ φορές. Καθεμιά από αυτές τις φορές δημιουργείται συμβατό ζεύγος με δύο το πολύ κινήσεις, οπότε συνολικά απαιτούνται $2k$ προσημασμένες προθεματικές ταξινομήσεις για τη δημιουργία $k$ ζευγών. Όμως, για να ταξινομηθεί ο $A$ πρέπει και αρκεί να δημιουργηθούν ακριβώς $n$ συμβατά ζεύγη. Αν σε κάθε επανάληψη το πλήθος των εναπομείναντων ζευγών μειώνεται κατά ένα, τότε μετά το πέρας των $k$ επαναλήψεων, απομένουν το πολύ $n - k$ συμβατά ζεύγη για τη δημιουργία των οποίων απαιτούνται $2(n - k)$ προσημασμένες προθεματικές ταξινομήσεις. Άρα, ο αλγόριθμος συνολικά εφαρμόζει $2k + 2(n - k) = 2n$ προσημασμένες προθεματικές ταξινομήσεις.

**Σχόλιο** Η αναλλοίωτη συνθήκη που εγγυάται την ορθότητα του αλγορίθμου είναι ότι άπαξ δημιουργηθεί ένα συμβατό ζεύγος, τότε δεν υπάρχει περίπτωση να εφαρμοστεί περιστροφή η οποία το διαχωρίζει. Έτσι, παρατηρούμε ότι κάθε ζεύγος μπορεί να αντιπροσωπεύεται από μια τιμή μέσω της οποίας απλοποιείται σημαντικά το πρόβλημα.

## Άσκηση 3

Το πρόβλημα λύνεται άμεσα με την κατασκευή του καρτεσιανού δέντρου $C$ για τον πίνακα $A$, όπου το $C$ έχει την ιδιότητα του δυαδικού δέντρου αναζήτησης για τις **θέσεις του $A$** (τις οποίες θεωρούμε τιμές κλειδιά) και του σωρού μεγίστου για **τις τιμές του $A$**.

Συγκεκριμένα, κατασκευάζουμε βήμα – βήμα το καρτεσιανό δέντρο $C_i$ για κάθε $i = 0, 1, 2, \dots, n$. Για να βρούμε τη θέση που κυριαρχεί της θέσης $i \geq 1$, αρκεί να εντοπίσουμε τον κόμβο $i$ στο δέντρο $C_i$ και να επιστρέψουμε την τιμή κλειδιού του πατέρα αυτού του κόμβου. Η ίδια διαδικασία επαναλαμβάνεται για όλα τα $i$.

Η ορθότητα του παραπάνω αλγορίθμου δικαιολογείται από τη φύση της κατασκευής του καρτεσιανού δέντρου. Έστω π.χ. ότι έχουμε κατασκευάσει σωστά το δέντρο $C_i$ και θέλουμε να εισάγουμε σε αυτό τον κόμβο $i + 1$. Τότε, ο κόμβος αυτός τοποθετείται ως δεξιότατος κόμβος στο $C_i$ και αν η τιμή του $A[i + 1]$ είναι μεγαλύτερη από την τιμή του πατέρα του $A[i]$, ο κόμβος $i + 1$ ανεβαίνει επίπεδο στο $C_i$ αφήνοντας τον κόμβο που ξεπέρασε στα αριστερά του. Αναδρομικά, ο κόμβος $i + 1$ θα συγκριθεί με την τιμή του καινούριου πατέρα του $A[i']$, όπου η θέση $i'$ κυριαρχεί της $i$ (δηλαδή $A[i'] > A[i]$, λόγω της επαγωγικής υπόθεσης του $C_i$), και είναι υποψήφια κυρίαρχη θέση της $i + 1$. Η διαδικασία επαναλαμβάνεται μέχρι η τιμή του πατέρα του κόμβου $i + 1$ (έστω $j$) να είναι αυστηρά μεγαλύτερη από την τιμή του κόμβου $i + 1$. Τότε και μόνο τότε έχει ολοκληρωθεί η κατασκευή του $C_{i+1}$ και μπορούμε με ασφάλεια να επιστρέψουμε τη θέση $j$ ως κυρίαρχη της θέσης $i + 1$.

Με βάση την παραπάνω ανάλυση είναι προφανές ότι η χρονική πολυπλοκότητα χειρότερης περίπτωσης του αλγορίθμου ταυτίζεται με τη χρονική πολυπλοκότητα κατασκευής του καρτεσιανού δέντρου $C$ για τον πίνακα $A$ η οποία ως γνωστόν είναι $O(n)$. Αυτό βέβαια δε σημαίνει ότι κατά μέσο όρο η εισαγωγή κάθε στοιχείου στο δέντρο απαιτεί σταθερό χρόνο $O(1)$. Η πραγματική εξήγηση είναι ότι η πολυπλοκότητα μετριέται με βάση τις συγκρίσεις στις οποίες το ένα στοιχείο «χάνει» και το άλλο «κερδίζει». Κάθε φορά που ένα στοιχείο «χάνει» σε μια σύγκριση μετατοπίζεται στο αριστερό μέρος του δέντρου, οπότε δε δικαιούται να συμμετάσχει σε καμιά άλλη σύγκριση. Έτσι, κάθε στοιχείο του πίνακα $A$ ενδέχεται να «χάσει» το πολύ σε μια σύγκριση και εφόσον ο $A$ έχει $n$

στοιχεία, θα υπάρχουν το πολύ $n$ συγκρίσεις που οδηγούν σε «χαμένο» και «κερδισμένο».

**Σχόλιο** Επειδή το δέντρο $C$ έχει την ιδιότητα του σωρού μεγίστου ως προς τις τιμές του πίνακα $A$, ως ρίζα του θα έχει την τιμή κλειδί 0 που αντιστοιχεί στο στοιχείο $A[0] = \infty$ που είναι και το μέγιστο του πίνακα $A$.

## Άσκηση 4

Το συγκεκριμένο πρόβλημα ανήκει στην κατηγορία των προβλημάτων βελτιστοποίησης. Αρχικά, θα ανάγουμε το πρόβλημα βελτιστοποίσης στο αντίστοιχο πρόβλημα απόφασης. Δηλαδή, έστω κάποιο $s = 1,2, \dots, n$, τότε αναρωτιόμαστε αν για πλήθος σταθμών φόρτισης ίσο με $s$ ικανοποιείται η υπόσχεση που δώσαμε στους πελάτες. Έχοντας κατασκευάσει αλγόριθμο που λύνει το παραπάνω πρόβλημα απόφασης, στη συνέχεια κάνοντας ένα $binary\ search$ στα $s$, μπορούμε να βρούμε το ελάχιστο $s^*$ που απαντάει στο αρχικό πρόβλημα βελτιστοποίησης.

Έστω $A$ πίνακας με $n$ στοιχεία στον οποίο αποθηκεύονται οι χρονικές στιγμές αφίξεως των αυτοκινήτων. Δηλαδή, $A[i] = a_i$, $i = 1,2, \dots, n$.

Έστω $B$ πίνακας με $n$ στοιχεία στον οποίο αποθηκεύονται οι χρονικές στιγμές όπου το κάθε αυτοκίνητο ξεκινά τη φόρτισή του (με αφετηρία πάντα τη μηδενική χρονική μονάδα). Ο πίνακας $B$ αρχικά τίθεται ίσος με τον $A$, δηλαδή $B[i] = A[i]$, για κάθε $i = 1, 2, \dots, n$.

Έχοντας ορίσει τους παραπάνω πίνακες, εύκολα προκύπτει ότι ο χρόνος αναμονής του αυτοκινήτου $i$ στο σταθμό είναι $T_i = B[i] - A[i]$. Τότε, για να απαντήσουμε με «ΝΑΙ» στο πρόβλημα απόφασης πρέπει να διασφαλίσουμε ότι $T_i \leq d - 1$, για κάθε $i = 1,2 \dots, n$.

Έστω $t$ η πιο πρόσφατη χρονική μονάδα στην οποία ένας τουλάχιστον φορτιστής του σταθμού άρχισε να χρησιμοποιείται. Τότε, το μέγεθος $t + 1$ εκφράζει τη χρονική μονάδα ύστερα από την οποία όλοι οι φορτιστές είναι διαθέσιμοι. Αρχικά, θέτουμε $t = B[1]$.

Ο αλγόριθμος διατυπώνεται παρακάτω:

Αρχικά, παίρνουμε το αυτοκίνητο υπ' αριθμόν 1 για φόρτιση και έτσι το πλήθος των διαθέσιμων φορτιστών μειώνεται κατά 1, δηλαδή γίνεται $s - 1$.

Ξεκινάμε να σαρώνουμε τον πίνακα $B[2 \dots n]$ από αριστερά προς τα δεξιά. Αν ικανοποιείται η συνθήκη $B[i] \geq t + 1$, αυτό θα σημαίνει ότι το αυτοκίνητο που κατέλαβε πιο πρόσφατα το σταθμό έχει ολοκληρώσει τη φόρτισή του τη στιγμή που καταφτάνει το αυτοκίνητο υπ' αριθμόν $i$. Αν δεν αληθεύει η παραπάνω συνθήκη, αυτό θα σημαίνει ότι το αυτοκίνητο $i$

φτάνει στο σταθμό ταυτόχρονα με το αυτοκίνητο που βρίσκεται σε φόρτιση.

Αν, λοιπόν, ισχύει ότι $B[i] \geq t + 1$, τότε θέτουμε $t = B[i]$ και το αυτοκίνητο $i$ μπαίνοντας για φόρτιση θα αφήσει $s - 1$ το πλήθος διαθέσιμους φορτιστές, ενώ αν ισχύει ότι $B[i] < t + 1$, τότε το αυτοκίνητο $i$ μπαίνοντας για φόρτιση θα ελαττώσει το πλήθος των διαθέσιμων φορτιστών κατά ένα.

Αν κάποια στιγμή μηδενιστεί το πλήθος των διαθέσιμων φορτιστών λόγω των $s$ αυτοκινήτων που έφτασαν στο σταθμό ταυτόχρονα τη χρονική στιγμή $t$, αυτό θα δημιουργήσει αναμονή σε όλα τα υπόλοιπα αυτοκίνητα που κατέφθασαν επίσης τη χρονική στιγμή $t$ (και ενδεχομένως σε άλλα που έφτασαν παραπλήσιες χρονικές στιγμές), οπότε ο πίνακας $B$ για τις θέσεις αυτών των αυτοκινήτων πρέπει να ενημερωθεί κατάλληλα. Συγκεκριμένα, έστω ότι το αυτοκίνητο υπ' αριθμόν $j$ είναι το τελευταίο από τα ταυτόχρονα εμφανιζόμενα αυτοκίνητα που άρχισε να φορτίζεται τη χρονική στιγμή $t$. Τότε, για $i > j$ αν $B[i] < t + 1$ πρέπει $B[i] = t + 1$, δηλαδή αν το αυτοκίνητο $i$ κατέφθασε νωρίτερα από τη χρονική στιγμή στην οποία τα $s$ αυτοκίνητα ολοκληρώνουν τη φόρτισή τους, αυτό θα αναμένει την ολοκλήρωση της φόρτισης της φουρνιάς των $s$ αυτοκινήτων. Όμως, αυτή η τροποποίηση του πίνακα $B$ μπορεί να οδηγήσει ξανά σε αναμονή, εφόσον υπάρχουν $s - \acute{\alpha}\delta\varepsilon\varsigma$ αυτοκίνητων που πλέον δικαιούνται να εξυπηρετηθούν την ίδια χρονική στιγμή (π.χ. αν αρχικά $B = \{4, 4, 4, 4, 4, 5, 5, 5\}$ και $s = 3$, τότε ο $B$ στο τέλος θα έχει τη μορφή $B = \{4, 4, 4, 5, 5, 5, 6, 6\}$, δηλαδή η αναμονή που προκάλεσαν τα πρώτα τρία αυτοκίνητα «εξαναγκάζει» το τέταρτο και πέμπτο αυτοκίνητο να εξυπηρετούνται την ίδια χρονική στιγμή με το έκτο αυτοκίνητο, γεγονός που δημιουργεί περαιτέρω αναμονή). Ενημερώνουμε τη μεταβλητή $t$ ώστε να αποθηκεύει τη νέα αυτή χρονική στιγμή (στο προηγούμενο παράδειγμα θα ισχύει $t = 5$). Η παραπάνω διαδικασία ενημέρωσης του $B$ συνεχίζεται μέχρι να βρεθεί κάποιο $i$ τέτοιο ώστε $B[i] \geq t + 1$, όπου η αναμονή μηδενίζεται και στην περίπτωση αυτή οι διαθέσιμοι φορτιστές είναι $s$ το πλήθος, οπότε κατά κάποιο τρόπο το πρόβλημα ξεκινά από την αρχή όσον αφορά τη διαθεσιμότητα των φορτιστών.

Μετά το τέλος της παραπάνω διαδικασίας, ο πίνακας $B$ περιλαμβάνει τις ακριβείς και ορθές χρονικές μονάδες στις οποίες κάθε αυτοκίνητο ξεκινά να φορτίζεται. Έτσι, ελέγχουμε επαναληπτικά τη διαφορά $B[i] - A[i]$, για κάθε $i = 1, 2, \dots, n$ και αν υπάρχει $i$ ώστε $B[i] - A[i] > d - 1$, τότε ο αλγόριθμος απαντά με «ΟΧΙ», αλλιώς απαντά με «ΝΑΙ».

Η αναλλοίωτη συνθήκη που εγγυάται την εγκυρότητα του αλγορίθμου είναι ότι κάθε φορά που μηδενίζεται το πλήθος των διαθέσιμων φορτιστών, δηλαδή κάθε φορά που τουλάχιστον $s$ το πλήθος αυτοκίνητα δικαιούνται να εξυπηρετηθούν την ίδια χρονική στιγμή $t$, ελέγχουμε αν η χρονική μονάδα άφιξης των επόμενων αυτοκινήτων είναι μικρότερη από την ελάχιστη χρονική μονάδα στην οποία όλοι οι φορτιστές θα ξαναγίνουν διαθέσιμοι $(t + 1)$. Αν ισχύει κάτι τέτοιο, το πρόβλημα ξεκινά από την αρχή όσον αφορά τη διαθεσιμότητα των φορτιστών, ενώ στην αντίθετη περίπτωση ο πίνακας $B$ ενημερώνεται έτσι ώστε αν $B[i] < t + 1$, τότε $B[i] = t + 1$

Η χρονική πολυπλοκότητα χειρότερης περίπτωσης του αλγορίθμου αποτελεί άμεση συνάρτηση του κόστους κατασκευής του πίνακα $B$, καθώς και του ελέγχου των διαφορών $B[i] - A[i]$ για κάθε $i = 1, 2, \dots, n$. Ο πίνακας $B$ λαμβάνει την τελική του μορφή σε χρόνο $O(n)$, διότι κάθε στοιχείο $B[i]$ ενημερώνεται μια μόνο φορά σε σταθερό χρόνο. Επίσης, για τον έλεγχο των διαφορών $B[i] - A[i]$ απαιτείται $O(n)$, άρα η υπολογιστική πολυπλοκότητα του αλγορίθμου που λύνει το πρόβλημα απόφασης είναι $O(n)$.

Πλέον, μπορούμε να βρούμε το ελάχιστο $s^*$ που απαντά στο πρόβλημα βελτιστοποίησης εφαρμόζοντας ένα $binary\ search$ στα $s$ του προβλήματος απόφασης. Ξεκινάμε παρατηρώντας ότι αρχικά το $s$ ανήκει στο εύρος τιμών $\{1, 2, \dots, n\}$. Επιλέγουμε τη μεσαία τιμή αυτού του διαστήματος και λύνουμε το πρόβλημα απόφασης για αυτή την τιμή, δηλαδή για $s = \frac{n}{2}$. Αν ο αλγόριθμος που περιγράψαμε παραπάνω απαντήσει με «ΝΑΙ», αυτό σημαίνει ότι $n/2$ φορτιστές επαρκούν για να τηρηθεί η υπόσχεση προς τους πελάτες. Έτσι, κάθε πλήθος φορτιστών που ξεπερνά το $n/2$ θα είναι επίσης αρκετό, οπότε αναρωτιόμαστε μήπως υπάρχει ένα «καλύτερο» $s < n/2$ που λύνει το πρόβλημα απόφασης. Συνεπώς, απορρίπτουμε όλες τις τιμές $s > n/2$ και εστιάζουμε στις τιμές $1 \leq s \leq n/2$. Όμως, αν ο αλγόριθμος απαντήσει

με «ΟΧΙ», αυτό σημαίνει ότι δεν επαρκούν $n/2$ φορτιστές για να τηρηθεί η υπόσχεση προς τους πελάτες. Έτσι, κάθε πλήθος φορτιστών μικρότερο του $n/2$ θα είναι επίσης ανεπαρκές, οπότε η λύση του προβλήματος απόφασης δίνεται για $s > n/2$. Συνεπώς, αγνοούμε όλες τις τιμές $s < n/2$ και εστιάζουμε στις τιμές $\frac{n}{2} \leq s \leq n$. Πρατηρούμε ότι είτε ο αλγόριθμος απαντά με «ΝΑΙ» είτε απαντά με «ΟΧΙ» έχουμε καταφέρει να απομονώσουμε το πρόβλημα σε ένα εύρος τιμών που είναι το υποδιπλάσιο από το αρχικό. Η ίδια διαδικασία επαναλαμβάνεται για το ενημερωμένο εύρος τιμών του $s$, μέχρι αυτό να εκφυλιστεί σε μια μόνο τιμή η οποία είναι και η βέλτιστη λύση $s^*$ που επιστρέφει ο αλγόριθμος.

Εφόσον μετά από κάθε εκτέλεση της $binary\ search$ το εύρος τιμών (δηλαδή το μέγεθος του προβλήματος) υποδιπλασιάζεται, είναι φανερό ότι στη χειρότερη περίπτωση θα χρειαστεί να κάνουμε $\log n$ δυαδικές αναζητήσεις μέχρι να βρούμε το ελάχιστο $s^*$. Όμως, σε κάθε δυαδική αναζήτηση λύνουμε εκ νέου το πρόβλημα απόφασης σε γραμμικό χρόνο. Επομένως, το πρόβλημα βελτιστοποίησης λύνεται σε χρόνο $O(n \log n)$.

**Σχόλιο** Εφόσον τα $a_i$ είναι ακέραια, περίπτωση αναμονής στο σταθμό θα συμβεί αν και μόνο αν τουλάχιστον $s$ το πλήθος αυτοκίνητα καταφτάσουν ταυτόχρονα στο σταθμό.

## Άσκηση 5

**(α)** Αρχικά, σημειώνουμε ότι η συνάρτηση $F_S$ είναι αύξουσα, διότι ισχύει ότι $F_S(x) \leq F_S(x+1)$, για κάθε $x$.

Έστω $t$ τυχαίο στοιχείο του συνόλου $S$. Τότε, λαμβάνοντας υπόψη τα πιθανά duplicates, το $t$ είναι **ενδεχομένως** το $k-οστό$ μικρότερο στοιχείο του $S$ αν ισχύει $F_S(t) \geq k$ και σε αυτή την περίπτωση η αναζήτηση έχει νόημα να συνεχιστεί για $t' \leq t$. Κάθε άλλο στοιχείο $t' > t$ απορρίπτεται ως το $k-οστό$ μικρότερο, λόγω της μονοτονίας της $F_S$. Ομοίως, αν $F_S(t) < k$, τότε η αναζήτηση έχει νόημα να συνεχιστεί για $t' > t$. Κάθε $t' \leq t$ αποκλείεται να είναι το $k-οστό$ μικρότερο στοιχείο του $S$, λόγω της μονοτονίας της $F_S$.

Με βάση τις παραπάνω παρατηρήσεις, βλέπουμε ότι καλώντας κάθε φορά τη συνάρτηση $F_S$ για ένα θετικό ακέραιο αριθμό, μπορούμε να ελαττώσουμε σημαντικά το μέγεθος του προβλήματος εστιάζοντας κάθε φορά σε ένα εύρος τιμών στοιχείων στο οποίο **ενδέχεται** να ανήκει το $k-οστό$ μικρότερο του συνόλου $S$. Η ιδέα που περιγράφεται παραπάνω αποτυπώνει τον αλγόριθμο του $binary\ search$ στο $M$. Πράγματι, το $M$ είναι μια καλή επιλογή $upper\ bound$, διότι είναι ανεξάρτητο του $n$ και ορίζει αρχικά το εύρος τιμών $\{1,\ 2, \dots, M\}$ μέσα στο οποίο γνωρίζουμε από υπόθεση ότι ανήκουν όλα τα στοιχεία του $S$. Ξεκινάμε επιλέγοντας τη μεσαία τιμή $v$ αυτού του συνόλου, δηλαδή $v = M/2$. Αν $F_S(v) \geq k$ επαναλαμβάνουμε τη διαδικασία για το νέο εύρος τιμών $\{1,\ 2,\ \dots,\ \frac{M}{2}\}$. Αν $F_S(v) < k$ επαναλαμβάνουμε τη διαδικασία για το νέο εύρος τιμών $\{\frac{M}{2} + 1, \dots,\ M\}$. Η αναζήτηση επαναλαμβάνεται μέχρι το εύρος τιμών να εκφυλιστεί σε μια μόνο τιμή, έστω $t$, η οποία είναι και η λύση του προβλήματος. Για να οδηγηθούμε σε αυτή την τιμή θα έχει προηγηθεί ότι $F_S(t-1) < k$ και $F_S(t) \geq k$. Τότε, επιστρέφουμε το στοιχείο $t$ ως το $k-οστό$ μικρότερο του συνόλου $S$ και ξέρουμε με βεβαιότητα ότι $t \in S$, διότι αν $t \notin S$ θα ίσχυε $F_S(t) = F_S(t-1) < k$.

Εφόσον μετά από κάθε εκτέλεση της $binary\ search$ το εύρος τιμών (δηλαδή το μέγεθος του προβλήματος) υποδιπλασιάζεται, είναι φανερό ότι στη χειρότερη περίπτωση θα χρειαστεί να κάνουμε $\log M$ επαναλήψεις μέχρι να βρούμε τη λύση $t$. Όμως, σε κάθε επανάληψη καλούμε και μια φορά τη συνάρτηση $F_S$, άρα για να βρεθεί το $k-οστό$

μικρότερο στοιχείο του συνόλου $S$ απαιτούνται στη χειρότερη περίπτωση $\log M$ κλήσεις της $F_S$.

**(β)** Σε πρώτο στάδιο αναζητούμε έναν τρόπο να υλοποιήσουμε αποδοτικά τη συνάρτηση $F_S$. Παρατηρούμε ότι το πλήθος των μη αρνητικών διαφορών ζευγών στοιχείων του $A$ στη χειρότερη περίπτωση είναι $(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}=O(n^2)$, δηλαδή για να βελτιώσουμε την $F_S$ αναζητούμε ενδεχομένως μια υλοποίηση σε γραμμικό χρόνο. Για να ξεφύγουμε από το $O(n^2)$ αρκεί να διαπιστώσουμε ότι η $F_S$ δεν πρέπει σε καμία περίπτωση να σαρώνει το σύνολο $S$ και αντ' αυτού υποψιαζόμαστε ότι θα σαρώνει τον πίνακα $A$, αφού αυτός υποστεί κατάλληλη επεξεργασία.

Προς αυτό το σκοπό, ταξινομούμε τον πίνακα $A$ (π.χ. με $mergesort$). Έτσι, το ελάχιστο στοιχείο θα είναι το $A[1]$ και το μεγαλύτερο το $A[n]=M$. Γενικά, στον υποπίνακα $A[i\ldots j]$ το ελάχιστο στοιχείο είναι το $A[i]$ και το μέγιστο στοιχείο το $A[j]$, οπότε η μέγιστη (θετική) διαφορά είναι η $A[j]-A[i]$.

Συνεπώς, όσον αφορά την υλοποίηση της $F_S$, αν καλέσουμε $F_S(l)$ και σαρώνοντας τον πίνακα $A$ από τα αριστερά προς τα δεξιά βρούμε $A[j]-A[i]\le l$, τότε επίσης $A[x_1]-A[x_2]\le l$, για κάθε $x_1, x_2=j,\ldots,i$ με $x_1\neq x_2$ και $A[x_1]>A[x_2]$, δηλαδή προσμετρώνται όλες οι ενδιάμεσες (θετικές) διαφορές. Αν $A[j]-A[i]>l$, τότε $A[k]-A[i]>l$, για κάθε $k>j$, οπότε σε αυτή την περίπτωση έχει νόημα να ελέγξουμε τη διαφορά $A[j+1]-A[i]$ και να επαναλάβουμε την ίδια διαδικασία.

Ο αλγόριθμος για την υλοποίηση της $F_S(l)$ διατυπώνεται αναλυτικά παρακάτω:

**<u>Βήμα 1</u>** Έστω $d$ ο κάτω δείκτης και $u$ ο πάνω δείκτης. Οι δύο δείκτες οριοθετούν κάθε φορά τον υποπίνακα $A[d\ldots u]$. Αρχικοποιούμε $d=1$ και $u=2$.

**<u>Βήμα 2</u>** Όσο ο πάνω δείκτης είναι μικρότερος ή ίσος του $n$ και ο πάνω δείκης είναι αυστηρά μεγαλύτερος του κάτω δείκτη:

• Αν $A[u]-A[d]\le l$, τότε προσμετράμε όλες τις ενδιάμεσες (θετικές) διαφορές του πίνακα $A[d\ldots u]$ και αυξάνουμε τον πάνω δείκτη κατά 1 αφήνοντας τον κάτω δείκτη ως έχει, ώστε στην επόμενη επανάληψη να

απομονώσουμε τον υποπίνακα $A[d \dots u+1]$. Επαναλαμβάνουμε το Βήμα 2.

• Αν $A[u] - A[d] > l$, αυξάνουμε τον κάτω δείκτη κατά 1, αφήνοντας τον πάνω δείκτη ως έχει, ώστε στην επόμενη επανάληψη να απομονώσουμε τον υποπίνακα $A[d+1 \dots u]$. Επαναλαμβάνουμε το Βήμα 2.

Ο αλγόριθμος ολοκληρώνεται είτε μόλις $u = n+1$ είτε μόλις $d = u$. Η χειρότερη περίπτωση, όσον αφορά τη χρονική πολυπλοκότητα του αλγορίθμου, δίνεται για $u = n+1$ και $u > d$, διότι τότε έχει σαρωθεί ολόκληρος ο πίνακας από τα αριστερά προς τα δεξιά. Για να έχει συμβεί κάτι τέτοιο, πρέπει ο δείκτης $u$ να έχει αυξηθεί ακριβώς $n-2$ φορές. Παρατηρούμε, όμως, ότι κάθε αύξηση ενός δείκτη στον αλγόριθμο ισοδυναμεί με τον έλεγχο μιας επιπλέον διαφοράς. Έτσι, στη χειρότερη περίπτωση της χειρότερης περίπτωσης, αν ο δείκτης $u$ αυξηθεί $n-2$ φορές, τότε ο δείκτης $d$ μπορεί να αυξηθεί κι εκείνος το πολύ κατά $n-2$ φορές, ώστε να ισχύει $u > d$. Τότε, ο αλγόριθμος ελέγχει το πολύ $2(n-2) + 1 = 2n-3$ διαφορές, οπότε η χρονική πολυπλοκότητα σε κάθε κλήση της $F_S$ είναι $O(n)$.

**Σχόλιο** Ο παραπάνω όρος «+1» αναφέρεται στην πρώτη διαφορά $A[2] - A[1]$ που ελέγχεται κατά την οποία δεν έχει προηγηθεί η αύξηση κανενός δείκτη.

Έχοντας υλοποιήσει αποδοτικά την $F_S$, το πρόβλημα ανάγεται πολύ εύκολα στο πρόβλημα του ερωτήματος (α), δηλαδή εφαρμόζουμε ξανά $binary\ search$ στο $M$, το οποίο είναι πάλι καλή επιλογή $upper\ bound$, διότι κάθε (θετική) διαφορά ζευγών του πίνακα $A$ είναι μικρότερη από το $A[n] = M$, δηλαδή κάθε διαφορά ανήκει στο εύρος τιμών $\{1, 2, \dots, M\}$. Η συνολική χρονική πολυπλοκότητα του αλγορίθμου είναι $O(n \log n)$ για την ταξινόμηση του $A$ με $merge\ sort$ και $O(n \log M)$ για το $binary\ search$, εφόσον η $F_S$ εκτελείται $\log M$ φορές έχοντας κόστος $O(n)$ κάθε φορά. Άρα, συνολικά η πολυπλοκότητα του αλγορίθμου είναι $O(n(\log n + \log M)) = O(n \log n) = O(n \log M)$, αν αναλογιστούμε ότι τα μεγέθη $n$ και $M$ είναι πολυωνυμικά συσχετισμένα.