Αλγόριθμοι και Πολυπλοκότητα – 1η Σειρά Ασκήσεων
Κυριακόπουλος Γεώργιος – el18153

Άσκηση 2

(α) Ο πιο απλός αλγόριθμος έχει ως εξής:

- Βρίσκουμε το μεγαλύτερο στοιχείο του Α που δεν έχει ταξινομηθεί ακόμα.
- Με μία προθεματική περιστροφή, το φέρνουμε στην αριστερότερη θέση.
- Με μία προθεματική περιστροφή, το φέρνουμε στη δεξιότερη θέση.
- Επαναλαμβάνουμε τα 3 αυτά bullets αναδρομικά για τα υπόλοιπα στοιχεία, δηλαδή τον υποπίνακα $A[1 \dots (n-1)]$.

Ο παραπάνω αλγόριθμος χρησιμοποιεί το πολύ $2n$ προθεματικές περιστροφές. Γίνεται να βελτιωθεί κατά ένα σταθερό όρο και να φτάσει το $2n-3$ με την παρακάτω διαδικασία.

- Για $n = 1$: δεν χρειάζεται ταξινόμηση.
- Για $n = 2$: μία προθεματική περιστροφή, εάν δεν είναι ήδη ταξινομημένα.
- Για $n \geq 3$: δυο προθεματικές περιστροφές για να φτάσει το μεγαλύτερο μη ταξινομημένο στοιχείο δεξιά και αναδρομικά για τα υπόλοιπα $n-1$, όπως είπαμε παραπάνω.

Έστω T(n) οι περιστροφές που χρειάζονται για να ταξινομηθούν n στοιχεία του πίνακα. Ισχύει ότι:

- $T(1) = 0,$
- $T(2) \leq 1,$
- $T(n) \leq 2 + T(n-1),\ n \geq 3$

Το τελευταίο καταλήγει στο $T(n) \leq 2n-3, n \geq 2$, οπότε έχουμε μία βελτιωμένη πολυπλοκότητα κατά 3 προθεματικές περιστροφές.

(β) Ουσιαστικά το μόνο που χρειάζεται είναι να ακολουθήσουμε τον ίδιο αλγόριθμο με το ερώτημα (α), μόνο που κάθε φορά που θα έχουμε κάνει την πρώτη προσημασμένη προθεματική περιστροφή του μέγιστου αταξινόμητου στοιχείου και αυτό θα βρίσκεται στην αριστερότερη θέση, θα ελέγχουμε εάν χρειάζεται μία επιπλέον προσημασμένη προθεματική περιστροφή μόνο σε αυτό το στοιχείο, ώστε να διορθωθεί το πρόσημο του, πριν γίνει η τελευταία που θα το μετακινήσει στη δεξιότερη θέση. Έτσι θα έχουμε το πολύ $3n$ προσημασμένες προθεματικές περιστροφές (το πολύ $2n$ ή $2n-3$ από το (α) + το πολύ n επιπλέον προσημασμένες προθεματικές περιστροφές για διορθώσεις προσήμου του μέγιστου αταξινόμητου στοιχείου, όπως περιγράψαμε).

(γ) 1. Θεωρούμε δύο περιπτώσεις. Η πρώτη και απλούστερη είναι όταν όλα τα στοιχεία έχουν αρνητικό πρόσημο. Τότε, σύμφωνα και με την εκφώνηση, ο πίνακας δεν είναι ο $[-1, -2, \dots, -n]$, άρα σίγουρα θα υπάρχει ένα στοιχείο $-x$ με $x < n$, για το οποίο το $-(x+1)$ θα βρίσκεται κάπου στα αριστερά του. Σε αυτήν την περίπτωση, με δύο κινήσεις μπορούμε να δημιουργήσουμε ένα νέο συμβατό ζεύγος $(-(x+1), -x)$, φέρνοντας το κομμάτι του πίνακα από $-(x+1)$ έως πριν το $-x$ με μία περιστροφή στην αρχή ως $[(x+1), \dots, -x, \dots]$ και έπειτα με μία ακόμα περιστροφή να φέρουμε το $(x+1)$ αριστερά του $-x$, δηλαδή να έχουμε $[\dots, -(x+1), -x, \dots]$, όπου φαίνεται το συμβατό ζεύγος.

Στην άλλη περίπτωση, όπου υπάρχει έστω και ένα στοιχειό με θετικό πρόσημο, βρίσκουμε το μεγαλύτερο θετικό στοιχείο x. Εάν αυτό είναι το τελευταίο, δηλαδή ίσο με n, τότε με δύο περιστροφές μπορούμε καταχρηστικά να δημιουργήσουμε ένα συμβατό ζεύγος, σύμφωνα και με την εκφώνηση. Αλλιώς, εάν δεν είναι το τελευταίο, τότε θα υπάρχει το $-(x+1)$ και μπορούμε και πάλι με δύο περιστροφές να δημιουργήσουμε νέο συμβατό ζεύγος. Στην περίπτωση που το x βρίσκεται δεξιότερα του $-(x+1)$, κάνουμε την πρώτη περιστροφή μέχρι και το x, οπότε θα βρεθεί στην αριστερότερη θέση $[-x, \dots, (x+1), \dots]$ και με τη δεύτερη περιστροφή μέχρι πριν το $(x+1)$, σχηματίζουμε το νέο συμβατό ζεύγος $[\dots, x, (x+1), \dots]$. Ειδάλλως, το x βρίσκεται αριστερότερα του $-(x+1)$ και τότε χρειάζεται να κάνουμε μια περιστροφή μέχρι και το $-(x+1)$, οπότε θα βρεθεί στην αριστερότερη θέση $[(x+1), \dots, -x, \dots]$ και με τη δεύτερη περιστροφή μέχρι πριν το $-x$, σχηματίζουμε το νέο συμβατό ζεύγος $[\dots, -(x+1), -x, \dots]$.

2. Ακολουθούμε τα παρακάτω βήματα:

- Αντικαταστούμε όλα τα συμβατά ζεύγη αναδρομικά με ένα αντίστοιχα προσημασμένο στοιχείο, ώστε να έχουμε ένα ισοδύναμο αλλά μικρότερο πρόβλημα με n' στοιχεία. Φυσικά, εάν $n' = 0$, τότε η ταξινόμηση έχει ολοκληρωθεί.
- Αν καταλήξουμε σε πίνακα της μορφή $[-1, -2, \dots, -n']$, τότε πρέπει για n' φορές να περιστρέψουμε όλα τα στοιχειά και στη συνέχεια να περιστρέψουμε πάλι τα $n' - 1$ αριστερότερα στοιχεία, ώστε τελικά να ταξινομηθεί ο πίνακας.
- Σε διαφορετική περίπτωση, χρησιμοποιούμε τα βήματα που περιγράψαμε στο ερώτημα 1. για να δημιουργήσουμε κάποιο νέο συμβατό ζεύγος και επαναλαμβάνουμε τον τρέχοντα αλγόριθμο από το πρώτο bullet.

Για να έχουμε το πολύ $2n$ προσημασμένες προθεματικές περιστροφές πρέπει ουσιαστικά να κατασκευάσουμε n συμβατά ζεύγη. Έστω k οι επαναλήψεις του συνολικού αλγορίθμου, τότε θα έχουμε κάνει το πολύ $2k + 2(n-k) = 2n$ περιστροφές. Σε κάθε επανάληψη (εκτός της τελευταίας που ίσως χρειαστούμε $2n' \leq 2(n-k)$ περιστροφές) θα κάνουμε το πολύ 2 κινήσεις και μέσω αυτών μειώνουμε το πλήθος των ζευγών που απομένουν κατά τουλάχιστον 1.

Άσκηση 3

Ο αλγόριθμος χρησιμοποιεί μία δομή στοίβας στην υλοποίηση του. Η στοίβα αυτή αρχικοποιείται με τη θέση του πρώτου στοιχείου του πίνακα, δηλαδή 1. Η θέση που κυριαρχεί της θέσης 1 του πίνακα είναι πάντα η 0, όπου A[0] = ∞. Για κάθε μία τις υπόλοιπες θέσεις, πρέπει να επαναλάβουμε τα παρακάτω βήματα:

- Όσο η στοίβα δεν έχει αδειάσει και η τιμή στη θέση της κορυφής της είναι μικρότερη της τιμής της θέσης που εξετάζουμε, βγάλε την κορυφή της στοίβας.
- Εάν μετά το τέλος του πρώτου βήματος η στοίβα είναι άδεια, τότε η θέση που κυριαρχεί της θέσης που εξετάζουμε είναι η 0. Αλλιώς, η θέση που κυριαρχεί της θέσης που εξετάζουμε είναι η κορυφή της στοίβας.
- Τέλος, προστίθεται η θέση που μόλις εξετάσαμε στην κορυφή της στοίβας.

Σχετικά με την ορθότητα του αλγορίθμου ακολουθούν τα παρακάτω σχόλια:

- Σχετικά με τη κυρίαρχη θέση της θέσης 1, αυτή είναι πάντα η 0 (με A[0] = ∞), όπως ειπώθηκε ήδη, αφού δεν υπάρχουν άλλες θέσεις πριν από την 1 στον πίνακα που θα μπορούσαν να εξεταστούν.
- Στη στοίβα προσθέτουμε κάθε θέση στο τέλος κάθε επανάληψης (όταν βρούμε την κυρίαρχη της), καθώς θέλουμε να υπάρχει για την επόμενη επανάληψη, ώστε να ελεγχθεί και αυτή κατά την εύρεση της κυρίαρχης της επόμενης θέσης.
- Μία θέση, που είχε προστεθεί στη στοίβα, αφαιρείται όταν η τιμή της δεν είναι μεγαλύτερη της τιμής της τρέχουσας θέσης, αφού δεν θα αποτελεί και πιθανή κυρίαρχη θέση μίας επόμενης (η τρέχουσα θέση έχει μεγαλύτερη τιμή και μεγαλύτερη θέση, άρα σίγουρα υπάρχει μεγαλύτερη θέση με μεγαλύτερη τιμή για τις επόμενες που θα εξεταστούν, δηλαδή, αρκεί να εξεταστεί μόνο η τρέχουσα και όχι και η προηγούμενη).
- Για κάθε θέση i, ελέγχουμε την κορυφή της στοίβας όπου βρίσκονται οι πιθανές κυρίαρχες θέσεις, δηλαδή προηγούμενες θέσεις που έχουν τιμές μεγαλύτερες από κάποια τιμή άλλης θέσης που έχουμε διατρέξει. Εάν για την κορυφή της στοίβας j δεν ισχύει το A[j] > A[i], την αφαιρούμε και ελέγχουμε την προηγούμενη θέση, μέχρι είτε να βρούμε την κυρίαρχη θέση j που θα ικανοποιεί τα A[j] > A[i], $0 \leq j < i$ και το j να είναι το μέγιστο δυνατό λόγω της σειράς που εισάγονται και αφαιρούνται οι θέσεις στη στοίβα, όπως εξηγήσαμε, είτε να αδειάσει η στοίβα, οπότε η κυρίαρχη θέση θα είναι η 0, αφού δεν θα υπάρχει προηγούμενη θέση με μεγαλύτερη τιμή από την τρέχουσα θέση i.

Η πολυπλοκότητα του παραπάνω αλγόριθμου είναι O(n), όπου n το πλήθος των φυσικών αριθμών του πίνακα εισόδου. Αυτό συμβαίνει, επειδή (όπως εξηγήθηκε παραπάνω) κάθε θέση προστίθεται και αφαιρείται από τη στοίβα το πολύ μία φορά. Έχουμε, δηλαδή, σταθερό αριθμό πράξεων (1 ή 2) για κάθε θέση (συνολικά το πολύ 2n πράξεις) και από αυτό προκύπτει η γραμμική O(n) πολυπλοκότητα του αλγορίθμου.

Άσκηση 4

Ο αλγόριθμος βασίζεται σε δύο μικρότερα τμήματα, ένα δυαδικής αναζήτησης στο s* ώστε να βρούμε το ελάχιστο πλήθος υποδοχών φόρτισης και ένα υπολογισμού εάν το s* που έχουμε διαλέξει ικανοποιεί την υπόσχεση σχετικά με την καθυστέρηση κάθε αυτοκινήτου, η οποία δεν πρέπει να ξεπερνάει το d. Για τη δυαδική αναζήτηση έχουμε:

- Ορίζουμε ως $left$ το 1 και ως $right$ το $ceiling(\frac{n}{d})$.
- Εάν το $left > right$, τότε η δυαδική αναζήτηση τερματίζει και πάμε στο τελευταίο bullet. Αλλιώς, προχωράμε τη δυαδική αναζήτηση, με $middle = floor(\frac{left+right}{2})$ (όπου $floor$ η συνάρτηση πάτωμα).
- Εάν για $s* = middle$ ο έλεγχος της υπόσχεσης επιστρέψει true, θέτουμε το $right$ ίσο με $middle - 1$. Αλλιώς, εάν για $s* = middle$ ο έλεγχος της υπόσχεσης επιστρέψει false, θέτουμε το $left$ ίσο με $middle + 1$.
- Εφόσον έχει τελειώσει η αναζήτηση, έχουμε φτάσει στο αποτέλεσμα που θέλουμε. Εάν ο έλεγχος υπόσχεσης για $s* = middle$ επιστρέψει true, τότε το ελάχιστο πλήθος $s*$ υποδοχών φόρτισης είναι το $middle$. Αλλιώς, εάν ο έλεγχος υπόσχεσης για $s* = middle$ επιστρέψει false, τότε το ελάχιστο πλήθος $s*$ υποδοχών φόρτισης είναι το $middle + 1$.

Σχετικά με την ορθότητα του αλγορίθμου ακολουθούν τα παρακάτω σχόλια:

- Προσπαθούμε με τη χρήση της δυαδικής αναζήτησης να βρούμε το $s*$, με τη βοήθεια του ελέγχου της υπόσχεσης που υλοποιείται παρακάτω. Συγκεκριμένα, ξεκινάμε τα δύο άκρα της αναζήτησης στο 1 και $ceiling(\frac{n}{d})$ αντίστοιχα, διότι το ελάχιστο δυνατό πλήθος υποδοχών φόρτισης είναι 1 και το μέγιστο που πιθανόν να χρειαστεί είναι στην περίπτωση όπου και τα n αυτοκίνητα θα φτάσουν μία χρονική στιγμή a_i, επομένως θα πρέπει να εξυπηρετηθούν n αυτοκίνητα τις επόμενες d χρονικές στιγμές, άρα χρειάζονται $ceiling(\frac{n}{d})$ υποδοχές φόρτισης, όπου $ceiling$, η συνάρτηση οροφή (πχ. εάν έρθουν και τα n=15 αυτοκίνητα τη χρονική στιγμή a_3 με d=4, χρειάζονται $ceiling\left(\frac{15}{4}\right) = ceiling(3.75) = 4$ υποδοχές φόρτισης ώστε να τηρηθεί η υπόσχεση).
- Μέσω της δυαδικής αναζήτησης προσπαθούμε να πλησιάσουμε κοντά στο ελάχιστο πλήθος $s*$ μέσω του $middle$, ώστε όταν τερματίσει η αναζήτηση, ελέγχοντας τι επιστρέφει ο έλεγχος της υπόσχεσης για το $s*$ ίσο με το τελευταίο $middle$, να μπορούμε να αποφανθούμε για το ποιο είναι το ελάχιστο πλήθος $s*$ υποδοχών φόρτισης. Αυτό το κάνουμε διότι, πέραν της περιπτώσεως που η δυαδική αναζήτηση βρίσκει και τελειώνει πάνω στο ελάχιστο πλήθος που απαιτείται, υπάρχει η περίπτωση να ελέγξει το πραγματικό ελάχιστο πλήθος, να επιστρέψει true και να προχωρήσει παρακάτω σε χαμηλότερη περιοχή και να τελειώσει εκεί μέσα (συγκεκριμένα ακριβώς πριν το πραγματικό ελάχιστο πλήθος), οπότε σε αυτήν την περίπτωση ο έλεγχος θα

επιστρέψει false και θα πάρουμε το πραγματικό ελάχιστο πλήθος, ως το σημείο που τελείωσε η αναζήτηση συν μία υποδοχή φόρτισης.

Η πολυπλοκότητα του παραπάνω αλγόριθμου είναι O(logn), όπου n το πλήθος των ηλεκτρικών αυτοκινήτων. Αυτό συμβαίνει, επειδή χρησιμοποιούμε δυαδική αναζήτηση πάνω στο διάστημα (1, $\frac{n}{d}$), η οποία, ως γνωστόν, έχει πολυπλοκότητα O(log($\frac{n}{d}$))= O(logn) με $\frac{n}{d}$ το μήκος του διαστήματος.

Για τον έλεγχο της υπόσχεσης έχουμε:

- Δεδομένου του δοσμένου $s*$, αρχικοποιούμε μια μεταβλητή in_queue στο 0.
- Τρέχουμε μία λούπα για τις επόμενες T χρονικές μονάδες.
- Για κάθε χρονική μονάδα, μόλις μπει ένα αυτοκίνητο, υπολογίζουμε τον χρόνο που αυτό θα μείνει στο σταθμό μέσω της σχέσης $delay = waiting + charging = floor\left(\frac{in\_queue}{s*}\right) + 1.$
- Εάν αυτός ο χρόνος για κάποιο αυτοκίνητο είναι μεγαλύτερος του d, ο έλεγχος επιστρέφει false. Διαφορετικά, αυξάνεται η in_queue κατά 1 και επαναλαμβάνουμε για τα υπόλοιπα αυτοκίνητα της ίδιας χρονικής μονάδας, εάν υπάρχουν, ειδάλλως, προχωράμε χρονική μονάδα, αφού πρώτα μειώσουμε το in_queue κατά $s*$ (ή να το θέσουμε ίσο με μηδέν σε περίπτωση αρνητικής τιμής), λόγω των αυτοκινήτων που φόρτισαν αυτήν τη χρονική στιγμή και αποχωρούν στη συνέχεια.
- Εάν τελειώσουν όλες οι χρονικές στιγμές και για κανένα αυτοκίνητο δεν έχει παραβιαστεί η υπόσχεση, τότε ο έλεγχος επιστρέφει true.

Σχετικά με την ορθότητα του αλγορίθμου ακολουθούν τα παρακάτω σχόλια:

- Ουσιαστικά ελέγχουμε για κάθε αμάξι που έρχεται στο σταθμό μετά από πόσες χρονικές στιγμές θα εξυπηρετηθεί και το συγκρίνουμε με το d. Συγκεκριμένα, προσθέτουμε τη μονάδα λόγω του χρόνου φόρτισης και το πάτωμα του συνολικού αριθμού των αυτοκινήτων που είναι στην ουρά διά το πλήθος των υποδοχών φόρτισης, που μας δίνει πόσες χρονικές στιγμές χρειάζονται ώστε να υπάρξει έστω και μία κενή θέση στις υποδοχές φόρτισες. Για αυτό και χρησιμοποιούμε το floor, ώστε εάν πχ έχουμε 15 αυτοκίνητα στην αναμονή και 4 υποδοχές, να μας γυρίσει 3 (=floor(3.75)), καθώς θα φορτίσουν 4+4+4+3 αυτοκίνητα, επομένως θα μπορεί την 4η χρονική στιγμή (φτάνει στο σταθμό την 1η) να μπει να φορτίσει το αυτοκίνητο που εξετάζουμε.
- Ο έλεγχος αυτός γίνεται με χρονική σειρά για όλα τα αυτοκίνητα που φτάνουν στο σταθμό, αφού πρώτα γίνουν οι κατάλληλες ανανεώσεις (πχ αύξηση της in_queue μετά την άφιξη και τον υπολογισμό του χρόνου, διότι καλύπτεται άλλη μία θέση στην αναμονή για φόρτιση, ή μείωση της in_queue κατά το πολύ $s*$, αφού στο τέλος κάθε χρονικής στιγμής φορτίζουν το πολύ $s*$ αυτοκίνητα (εάν έχουμε 2 αυτοκίνητα στην αναμονή και 3 υποδοχές φόρτισης προφανώς το in_queue θα γίνει 0 και όχι -1). Επομένως, έχουμε ανά πάσα στιγμή μία ρεαλιστική αποτύπωση των συνθηκών στις

μεταβλητές μας και για αυτό βγαίνουν ορθά συμπεράσματα σχετικά με την τήρηση ή όχι της υπόσχεσης μέγιστης καθυστέρησης στο σταθμό για κάθε αυτοκίνητο.

Η πολυπλοκότητα του παραπάνω αλγόριθμου είναι O(n), όπου n το πλήθος των ηλεκτρικών αυτοκινήτων. Αυτό συμβαίνει, επειδή για κάθε αυτοκίνητο κάνουμε μόλις μερικές σταθερές πράξεις-υπολογισμούς, όπως ο υπολογισμός του χρόνου καθυστέρησης και η δύο φορές αλλαγή της τιμής της μεταβλητής in_queue. Άρα έχουμε γραμμική πολυπλοκότητα ως προς το πλήθος των αυτοκινήτων.

Τέλος, ο αλγόριθμος που λύνει το πρόβλημα συνδυάζει τους δύο παραπάνω με μία απλή λογική όπως έχει περιγραφεί ήδη. Συγκεκριμένα, κάνουμε τη δυαδική αναζήτηση στο ελάχιστο πλήθος $s *$ υποδοχών φόρτισης σύμφωνα με τον πρώτο αλγόριθμο, και ως έλεγχο της υπόσχεσης χρησιμοποιούμε το δεύτερο αλγόριθμο. Φυσικά, το αποτέλεσμα είναι να βρούμε το ελάχιστο πλήθος $s *$ υποδοχών φόρτισης για το οποίο θα ικανοποιείται ο έλεγχος για όλα τα αυτοκίνητα. Η πολυπλοκότητα του παραπάνω αλγόριθμου είναι O(nlogn), αφού έχουμε μία δυαδική αναζήτηση πολυπλοκότητας O(logn) με κλήσεις του ελέγχου υπόσχεσης με πολυπλοκότητα O(n) σε κάθε βήμα της αναζήτησης, άρα συνολικά O(nlogn).

Άσκηση 5

(α) Ο αλγόριθμος βασίζεται στη δυαδική αναζήτηση μέσω της Fs στο σύνολο S. Αναλυτικότερα τα βήματα έχουν ως εξής:

- Ορίζουμε ως $left$ το 1 και ως $right$ το M.
- Εάν το $left > right$, τότε η δυαδική αναζήτηση τερματίζει και πάμε στο τελευταίο bullet. Αλλιώς, προχωράμε τη δυαδική αναζήτηση, με $middle = floor(\frac{left+right}{2})$ (όπου $floor$ η συνάρτηση πάτωμα).
- Εάν το $Fs(middle) < k$, θέτουμε το $left$ ίσο με $middle + 1$ και επαναλαμβάνουμε το 2ο bullet. Αλλιώς, εάν το $Fs(middle) \geq k$, θέτουμε το $right$ ίσο με $middle - 1$ και επαναλαμβάνουμε το 2ο bullet.
- Εφόσον έχει τελειώσει η αναζήτηση, έχουμε φτάσει στο αποτέλεσμα που θέλουμε. Εάν $Fs(middle) < k$, το k-οστό μικρότερο στοιχείο είναι το $middle + 1$. Αλλιώς, εάν $Fs(middle) \geq k$, το k-οστό μικρότερο στοιχείο είναι το $middle$.

Σχετικά με την ορθότητα του αλγορίθμου ακολουθούν τα παρακάτω σχόλια:

- Ψάχνουμε να βρούμε ένα στοιχείο που ανήκει στο σύνολο S, επομένως ξεκινάμε τα δύο άκρα της δυαδικής αναζήτησης από το 1 και από το M, που είναι το μικρότερο και το μεγαλύτερο, αντίστοιχα, δυνατό στοιχείο που μπορεί να περιέχει το σύνολο S. Αυτό ισχύει διότι το σύνολο S περιέχει θετικούς ακέραιους (ο μικρότερος τέτοιος είναι το 1) που δεν ξεπερνούν τη θετική ακέραια τιμή M (άρα ο μεγαλύτερος είναι το M).
- Μέσω της δυαδικής αναζήτησης προσπαθούμε να πλησιάσουμε κοντά στο k-οστό στοιχείο μέσω του $middle$, ώστε όταν τερματίσει η αναζήτηση, ελέγχοντας την τιμή της Fs για την τελευταία επιλογή $middle$, σε σχέση με το k, να μπορούμε να αποφανθούμε για το ποιο είναι το k-οστό μικρότερο στοιχείο, όπως αναλύεται στο επόμενο bullet.
- Συγκεκριμένα, εάν $Fs(middle) < k$, τότε θα ισχύει ότι $Fs(middle + 1) \geq k$, αφού η δυαδική αναζήτηση έχει τελειώσει, δηλαδή έχουμε πλησιάσει χωρίς να μπορούμε να βρούμε κατά την διάρκεια της αναζήτησης το στοιχείο που ψάχνουμε, ξέροντας όμως ότι κατά την αναζήτηση η παραπάνω περιοχή που περιλαμβάνει το $middle + 1$ έχει αποκλειστεί διότι είχε τιμή μεγαλύτερη του k και λόγω του αλγορίθμου η αναζήτηση συνέχισε σε χαμηλότερη περιοχή. Σε αυτήν την περίπτωση, το k-οστό μικρότερο στοιχείο είναι το $middle + 1$, δεδομένου ότι ενώ για $middle$ έχουμε λιγότερα από k στοιχεία, δηλαδή δεν περιλαμβάνεται το k-οστό μικρότερο, για $middle + 1$ έχουμε τουλάχιστον k στοιχεία, δηλαδή περιλαμβάνεται το k-οστό μικρότερο και μάλιστα είναι ίσο με $middle + 1$, διότι μόνο το πλήθος των $middle + 1$ στοιχείων προστέθηκε στην απάντηση της Fs και την διαμόρφωσε ως έχει.
- Ομοίως, εάν $Fs(middle) \geq k$, τότε θα ισχύει και ότι $Fs(middle - 1) < k$, για τους ίδιους λόγους σχετικά με τις περιοχές που διαλέγει, αποκλείει και τελικά αναζητάει σε αυτές ο αλγόριθμος. Σε αυτήν την περίπτωση το k-οστό μικρότερο στοιχείο είναι το $middle$, αφού μόνο για $middle$ και πάνω η Fs επιστρέφει τουλάχιστον k στοιχεία,

δηλαδή περιλαμβάνει το k-οστό μικρότερο, ενώ για $middle - 1$ δεν το περιλαμβάνει, καθώς επιστρέφει λιγότερα από k στοιχεία.

Η πολυπλοκότητα του παραπάνω αλγόριθμου είναι O(logM), όπου M ο δεδομένος ακέραιος που έχει μεγαλύτερη ή ίση τιμή από όλα τα στοιχεία του συνόλου S. Αυτό συμβαίνει, επειδή χρησιμοποιούμε δυαδική αναζήτηση πάνω στο διάστημα (1, M), η οποία, ως γνωστόν, έχει πολυπλοκότητα O(logn) με n το μήκος του διαστήματος.

(β) Ο αλγόριθμος σχετικά με την εύρεση του k-οστού μικρότερου στοιχείου παραμένει όμοιος με τον αλγόριθμο του ερωτήματος (α). Αυτό που αλλάζει είναι η υλοποίηση της Fs, η οποία θα γίνει με την ακόλουθη λογική, αφού πρώτα ταξινομήσουμε τον πίνακα A:

- Αρχικοποιούμε μία μεταβλητή counter στο 0, και δύο δείκτες i, j στο 0.
- Τρέχουμε μία for με δείκτη i, n φορές (μήκος του πίνακα A) και εσωτερικά της μία while όπου όσο το j είναι μικρότερο του n και η διαφορά $A[j] - A[i]$ είναι μικρότερη ή ίση του k (όρισμα της Fs) αυξάνουμε το δείκτη j.
- Όταν τελειώσει αυτή η while, αυξάνουμε την τιμή του counter κατά $j - i - 1$.
- Επαναλαμβάνουμε για n φορές και τελικά στο counter έχουμε την τιμή της Fs που θέλουμε.

Σχετικά με την ορθότητα του αλγορίθμου ακολουθούν τα παρακάτω σχόλια:

- Για κάθε i, ουσιαστικά ψάχνουμε το μικρότερο j ώστε η διαφορά του A[j] με το A[i] να είναι μεγαλύτερη του k. Τότε, θα έχουμε προχωρήσει το j τόσες θέσεις όσα είναι τα ζευγάρια που έχουν διαφορά μικρότερη ή ίση του k και θα ανήκουν στο S, μείον το i, από όπου ξεκινάνε τα ζευγάρια, μείον 1 λόγω του παραπάνω βήματος για να αποτύχει ο έλεγχος και να βρούμε το πρώτο στοιχείο που δεν θέλουμε να μετρήσουμε.
- Φυσικά, βασικός λόγος που δουλεύει ο αλγόριθμος μας είναι η απαίτηση να έχουμε ταξινομήσει νωρίτερα τον πίνακα, ώστε να είναι τα στοιχεία σε μία σειρά και να μπορούμε να βρούμε τα ζευγάρια A[i], A[j] με περιορισμένη τιμή διαφοράς, κρατώντας σταθερό το A[i] και ψάχνοντας το μεγαλύτερο A[j] που γίνεται προχωρώντας στον πίνακα το δείκτη j.
- Για την υπόλοιπο αλγόριθμο, δηλαδή την εύρεση του k-οστού μικρότερου στοιχείου του S, ισχύουν όσα αναφέρθηκαν και στο ερώτημα (α), καθώς χρησιμοποιούμε αυτόν τον ίδιο αλγόριθμο για την αναζήτηση.

Η πολυπλοκότητα του παραπάνω αλγόριθμου είναι O(nlogM + nlogn) ή αλλιώς O(nlog(max(M,n))), όπου M το μέγιστο στοιχείο του συνόλου S και n το μήκος του πίνακα A. Το nlogn οφείλεται στην ταξινόμηση του πίνακα A πριν αρχίσει ο υπόλοιπος αλγόριθμος εύρεσης του k-οστού μικρότερου στοιχείου του S (πχ. με χρήση merge sort). Το nlogM οφείλεται στον υπόλοιπο αλγόριθμο, συγκεκριμένα στη δυαδική αναζήτηση, logM ως δυαδική αναζήτηση πάνω στο διάστημα (1, M), επί n που είναι η γραμμική πολυπλοκότητα υπολογισμού της Fs, σύμφωνα με τον αλγόριθμο που περιγράφηκε παραπάνω, που καλείται σε κάθε βήμα της

δυαδικής αναζήτησης. Συγκεκριμένα, ο παραπάνω αλγόριθμος έχει πολυπλοκότητα O(n), καθώς χρησιμοποιεί την τεχνική των δύο pointers και παρότι υπάρχουν φωλιασμένα loops, το j συνεχίζει να αυξάνεται μόνο εφόσον αυξηθεί και το i και συνολικά η αύξηση του γίνεται το πολύ n φορές, δίνοντας τη γραμμική πολυπλοκότητα.