Αλγόριθμοι & Πολυπλοκότητα

1η σειρά γραπτών ασκήσεων

---

hungrydino ΣΗΜΜΥ 7ο εξάμηνο

Άσκηση 1: Αναδρομικές Σχέσεις

Στην ανάλυση μας παρακάτω θεωρούμε ότι το n είναι δύναμη του εκάστοτε διαιρέτη του για ευκολία, χωρίς αυτό να επηρεάζει την ασυμπτωτική μας ανάλυση.

1) $T(n) = 4T(n/2) + \Theta(n^2 logn)$

Έχουμε: $\frac{n^2 logn}{n^{log4}} = logn < n^\epsilon, \forall \epsilon \in \mathbb{R}^+$, γιατί: $lim_{x\to\infty} \frac{logx}{x^\epsilon} \overset{DLH}{=} lim_{x\to\infty} \frac{1}{\epsilon ln2} x^{-\epsilon} = 0$

Άρα, δεν εφαρμόζεται το Master Theorem. Είναι:

$$T(n) = 4T(n/2) + cn^2 logn, \text{ για κάποιο } c \in \mathbb{R}^*$$

$$\Rightarrow T(n) = 4\left[4T(n/4) + c(\frac{n}{2})^2 log\frac{n}{2}\right] + cn^2 logn$$

Εύκολα από μαθηματική επαγωγή μπορούμε να αποδείξουμε:

$$T(n) = 4^{logn} T(1) + c \sum_{i=0}^{logn-1} n^2 log\frac{n}{2^i}$$

Ή αλλιώς:

$$T(n) = \Theta(n^2) + cn^2 \cdot \sum_{i=0}^{logn-1} log\frac{n}{2^i}$$

Είναι:

$$\sum_{i=0}^{logn/2} log\frac{n}{2^{logn/2}} \leq \sum_{i=0}^{logn-1} log\frac{n}{2^i} \leq logn \cdot logn$$

$$\iff \frac{1}{2} logn \cdot log\sqrt{n} \leq \sum_{i=0}^{logn-1} log\frac{n}{2^i} \leq log^2 n$$

$$\Rightarrow \sum_{i=0}^{logn-1} log\frac{n}{2^i} = \Theta(log^2 n)$$

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

$$\Longrightarrow \boxed{T(n) = \Theta(n^2 log^2 n)}$$

2) $T(n) = 5T(n/2) + \Theta(n^2 logn)$

Είναι:

$$\frac{n^{log5}}{n^2 logn} = \frac{n^{log5-2}}{logn} \geq \frac{n^{log5-2}}{10^{-10}} = n^{log\frac{5}{4+10^{-10}}} > n^{log\frac{5}{4.5}}$$

Άρα εφαρμόζεται το Master Theorem και έχουμε:

$$\boxed{T(n) = \Theta(n^{log5})}$$

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

Είναι:

$$T(n) = 2T(n/4) + T(n/8) + cn/2 + cn$$

$$= 3T(n/8) + T(n/16) + cn/4 + cn/2 + cn$$

$$\vdots$$

$$\Rightarrow T(n) = kT(n/2^k) + T(n/2^{k+1}) + \sum_{i=0}^{k-1} \frac{cn}{2^i}$$

$$\overset{k=logn-1}{\Longrightarrow} T(n) = (logn - 1)T(2) + T(1) + cn \sum_{i=0}^{logn-1} \frac{1}{2^i}$$

Όμως: $\sum_{i=0}^{\infty} \frac{1}{2^i} = 2$ και $\sum_{i=0}^{logn-1} \frac{1}{2^i} \geq 1$ Δηλαδή: $T(n) = \Theta(logn) + \Theta(n)$

$$\Longrightarrow \boxed{T(n) = \Theta(n)}$$

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

Αν κατασκευάσουμε το δέντρο αναδρομής θα δούμε ότι κάθε επίπεδο εκτελεί "δουλειά" Θ(n). Το μικρότερο μονοπάτι από τη ρίζα σε φύλλο έχει μήκος $log_4 n$. Ενώ το μεγαλύτερο έχει μήκος $log_2 n$. Άρα, αν $f(n)$ ο χρόνος που απαιτείται για τη "δουλειά" των επιπέδων τότε: $cnlog_4 n \leq f(n) \leq cnlog_2 n \Rightarrow f(n) = \Theta(nlogn)$.

Αν θεωρήσουμε την είσοδο σαν ένα κουβά με 3 μπαλάκια, τότε αυτή χωρίζεται σε 3 κουβάδες με ν/4, ν/2, ν/4 μπαλάκια αντίστοιχα, οι οποίοι χωρίζονται περαιτέρω με τον ίδιο τρόπο, φτιάχνοντας ένα δέντρο και φτάνοντας μέχρι τα φύλλα (κουβάς με 1 μπάλα). Επομένως, ο αριθμός των φύλλων θα είναι n και άρα η "δουλειά" τους είναι Θ(n).
Δηλαδή: $T(n) = \Theta(n) + \Theta(nlogn)$

$$\Longrightarrow \boxed{T(n) = \Theta(nlogn)}$$

5) $T(n) = T(n^{1/2}) + \Theta(logn)$

Είναι:

$$T(n) = T(n^{1/2}) + clogn \Rightarrow T(n) = T(n^{1/2^k}) + c\sum_{i=0}^{k-1} logn^{1/2^i}$$

Γνωρίζουμε ότι: $lim\ \sqrt[n]{n} = 1$, άρα για μεγάλα n:

$$T(n) = T(1) + c\sum_{i=0}^{logn-1} logn^{1/2^i}$$

$$\iff T(n) = T(1) + clogn\sum_{i=0}^{logn-1} \frac{1}{2^i}$$

Όμως: $\sum_{i=0}^{\infty} \frac{1}{2} = 2$, επομένως:

$$\boxed{T(n) = \Theta(logn)}$$

6) $T(n) = T(n/4) + \Theta(\sqrt{n})$

Θα είναι:

$$T(n) = T(1) + c\sum_{i=0}^{log_4 n-1} \sqrt{\frac{n}{4^i}}$$

$$= T(1) + c\sum_{i=0}^{log_4 n-1} \frac{\sqrt{n}}{2^i}$$

$$= T(1) + c\sqrt{n}\sum_{i=0}^{log_4 n-1} 2^{-i}$$

Για τον ίδιο λόγο με πριν:

$$\boxed{T(n) = \Theta(\sqrt{n})}$$

Άσκηση 2: Προθεματικές Περιστροφές

α) Ο αλγόριθμος μας έχει ως εξής:

▷ Για i από n έως 3:

- Αν $A[i] = i$ συνέχισε στο επόμενο i (αν $i = 3$ τελειώνουμε)
- Βρες τη θέση n για την οποία $A[k] = i$
- Αν $k \neq 1$ προθεματική μετάθεση k στοιχείων
- Προθεματική μετάθεση i στοιχείων

- Αν $\mathrm{A}[2] \neq 2$ προθεματική μετάθεση 2 στοιχείων

Ο αλγόριθμος δουλεύει σαν selection sort. Κάθε φορά μετά το τέλος της j-στης επανάληψης ο αριθμός n-j+1 τοποθετείται στην αντίστοιχη θέση του πίνακα. Ο αριθμός αυτός έπειτα δεν θα αλλάξει ποτέ θέση στον πίνακα, γιατί προχωράμε από τους μεγάλους αριθμούς στους μικρούς. Δηλαδή η επόμενη μετάθεση (αν υπάρχει) θα έχει το πολύ n - j στοιχεία, αφού δεξιά του n-j+1 αποκλείεται να βρίσκεται ο n-j (έχουμε μόνο μεγαλύτερους από τον n-j+1).

Συνολικά θα έχουμε το πολύ 2(n-2) + 1 = $\boxed{2n - 3 \text{ μεταθέσεις}}$

β) Ο αλγόριθμος εδώ θα είναι παρόμοιος:

▷ Για i από n έως 2:

- Αν $A[i] = i$ συνέχισε στο επόμενο i, εκτός αν $i = 2$ οπότε τελειώνουμε
- Βρες τη θέση k για την οποία $A[k] = \pm i$
- Αν $k \neq 1$ προθεματική μετάθεση k στοιχείων
- Αν $\mathrm{A}[1] > 0$ προθεματική μετάθεση 1 στοιχείου
- Προθεματική μετάθεση i στοιχείων

- Αν $A[1] = -1$ προθεματική μετάθεση 1 στοιχείου

Ο αλγόριθμος δουλεύει με την ίδια λογική (σαν selection sort). Αν το στοιχείο που φέραμε ή υπήρχε ήδη στην πρώτη θέση έχει τώρα θετικό πρόσημο, κάνουμε τετριμμένη μετάθεση 1 στοιχείου (αντιστρέφουμε το πρόσημο) και έπειτα το τοποθετούμε στην θέση του (θα έχει θετικό πρόσημο). Ο αλγόριθμος εκτελεί το πολύ 3(n -1) + 1 = $\boxed{3n - 2 \text{ μεταθέσεις}}$

γ) (ι) Θεωρούμε το πρώτο στοιχείο του πίνακα A_t. Ελέγχουμε αν δημιουργεί συμβατό ζεύγος με το επόμενο στοιχείο (άρα αν αυτό είναι ο διάδοχος του). Αν ναι, εξετάζουμε το δεύτερο και ούτω καθ'εξής μέχρι να βρούμε κάποιο στοιχείο που δεν δημιουργεί συμβατό ζευγάρι με τον διάδοχο του. Έστω ότι εξετάζουμε τελικά τον αριθμό a και είναι το k_a-στο στοιχείο. Δεξιότερα του, έστω στη θέση $k_m > k_a + 1$, θα υπάρχει ο διάδοχος του (*), δηλαδή το $a+1$ (είτε με το ίδιο είτε με αντίθετο πρόσημο).
(Αν δεν υπάρχει ο διάδοχος $\rightarrow$ σημείωση)*

- Αν αυτός έχει το ίδιο πρόσημο τότε:

- Εκτελούμε προθεματική μετάθεση k_a στοιχείων.
- Εκτελούμε προθεματική μετάθεση $k_m - 1$ στοιχείων.

Προσέχουμε ότι η πρώτη μετάθεση δεν χαλάει την "αλυσίδα" συμβατών ζευγαριών μέχρι το α, αν αυτή υπάρχει. Η δεύτερη μετάθεση την επεκτείνει προσθέτοντας το $a+1$ και τα υπόλοιπα συμβατά του ζεύγη (αν υπάρχουν). Τα ζεύγη ανάμεσα στο a και το $a+1$, αν υπήρχαν δεν "έσπασαν", απλά άλλαξαν πρόσημο, σειρά και τοποθετήθηκαν πίσω από την αρχική αλυσίδα.

Έστω τα παρακάτω:

$$\begin{aligned} [X] &:= \text{αλυσίδα στοιχείων (πιθανόν κενή)} \\ [-X!] &:= \text{η αλυσίδα αντεστραμμένη και με αντίθετα στοιχεία} \\ \sim &:= \text{σύνδεση} \end{aligned}$$

Τότε σχηματικά έχουμε:

$$[A] \sim a \sim [B] \sim a+1 \sim [C] \rightarrow$$
$$-a \sim [-A!] \sim [B] \sim a+1 \sim [C] \rightarrow$$
$$[-B!] \sim [A] \sim a \sim a+1 \sim [C]$$

- Αν δεν έχει το ίδιο πρόσημο τότε:

- Εκτελούμε προθεματική μετάθεση k_m στοιχείων.
- Εκτελούμε προθεματική μετάθεση $k_m - k_a$ στοιχείων.

Προσέχουμε ότι το $-(a+1)$ δεν μπορεί να είναι "αρχή" αλυσίδας συμβατών ζευγών, γιατί αυτό θα σήμαινε ότι έχει στα δεξιά του το $-(a+1)+1 = -a$, όμως το a είναι αριστερά του, άρα η 1η μετάθεση που κάνουμε δεν "σπάει" κάποια αλυσίδα. Η δεύτερη μετάθεση θα μεταφέρει τα στοιχεία που υπήρχαν μεταξύ των $a, -(a+1)$, αυτούσια στην αρχή. Άρα δεν "σπάει" κάποια αλυσίδα.

Σχηματικά:

$$[A] \sim a \sim [B] \sim -(a+1) \sim [C] \rightarrow$$

$$a+1 \sim [-B!] \sim -a \sim [-A!] \sim [C] \rightarrow$$

$$[B] \sim -(a+1) \sim -a \sim [-A!] \sim [C] \rightarrow$$

Σημείωση στα παραπάνω:

- Θεωρούμε ότι ο αριθμός n έχει τον διάδοχο του ($n+1$) "fixed" μετά το τέλος του πίνακα.

- Αν στην αρχή του πίνακα έχουμε αλυσίδα που καταλήγει σε -1 κάνουμε υποψήφιο το επόμενο στοιχείο (ο διάδοχος του -1 είναι το 0 που δεν υπάρχει στον πίνακα). Αν παρ'όλα αυτά όλα τα δεξιότερα στοιχεία έχουν τον διάδοχο τους αριστερά τους ή είναι ήδη σε συμβατό ζευγάρι με τον διάδοχο, δεν γίνεται να φτιάξουμε ζευγάρι σε λιγότερες απο 3 κινήσεις.
Παράδειγμα: $[-3,-2,-1,-4,5]$ *και γενικότερα:*
$A_{(k,d,n)} := [-k,\ -(k-1),\ ...\ -1,\ \ -(k+1),\ -(k+2),\ ...\ -(k+d),\ \ (k+d+1),\ (k+d+2),\ ...,\ n]$

(ιι) Ο αλγόριθμος μας είναι απλός: τοποθετούμε το n στο τέλος με 2 κινήσεις (αν είναι ήδη στο τέλος δεν κάνουμε κάτι). Έπειτα χρησιμοποιούμε την παραπάνω διαδικασία $n-1$ φορές για να δημιουργήσουμε κάθε φορά ένα νέο συμβατό ζεύγος χωρίς να σπάσουμε τα προηγούμενα. Στο τέλος θα έχουμε n συμβατά ζεύγη, δηλαδή έναν ταξινομημένο πίνακα.

Αν σε κάποια κίνηση προκύψει πίνακας της μορφής $A_{(k,d,n)}$ που περιγράψαμε παραπάνω, εστιάζουμε στον υποπίνακα αριστερά του στοιχείου $k+d+1$ και εκτελούμε τον παρακάτω αλγόριθμο:

- Προθεματική $k+d$ στοιχείων (όλα τα στοιχεία του υποπίνακα).
- Προθεματική d στοιχείων.
-Εκτέλεσε d φορές:

- Προθεματική $k+d$ στοιχείων (όλα τα στοιχεία του υποπίνακα).
- Προθεματική $k+d-1$ στοιχείων.

Για να αποδείξουμε την ορθότητα αυτού αντικαθιστάμε νοητικά τον υποπίνακα $[-k,\ -(k-1),\ ...\ ,-1]$ με το $[-\mathrm{K}]$, δηλαδή ο πίνακας που εξετάζουμε τώρα είναι ο $[-K,\ -(k+1),\ ...,\ -(k+d)]$.

Με την πρώτη μετάθεση φτιάχνουμε τον $[(k+d),\ ,(k+d-1),\ ,...,\ (k+1),\ K]$.
Η δεύτερη μετάθεση δημιουργεί τον $[\ -(k+1),\ ...,\ -(k+d),\ K]$.

Από εκεί και πέρα εκτελούμε d φορές μια μετάθεση $k+d$ στοιχείων (όλων) και μετά μια μετάθεση $k+d-1$ στοιχείων. Αυτό θα έχει ως αποτέλεσμα την πρώτη φορά τον πίνακα $[-(k+2),\ ...,\ -(k+d),\ K,\ (k+1)]$, έπειτα τον πίνακα $[-(k+3),\ ...,\ -(k+d),\ K,\ (k+1),\ (k+2)]$. Βλέπουμε δηλαδή ότι μετά το πέρας των d επαναλήψεων θα έχει δημιουργηθεί ο πίνακας $[K,\ k+1,\ ...,\ k+d]$, δηλαδή ο $[1,\ ,2,\ ,...,\ k,\ k+1,\ ...,\ k+d]$. Άρα, δηλαδή ταξινομήσαμε των υποπίνακα (και έτσι τον συνολικό πίνακα) σε $2(d+1)$ μεταθέσεις. Ο αρχικός πίνακας είχε φτάσει σε αυτή την κατάσταση μετά από το πολύ $2\{(k-1)+[n-(k+d)]\}$, αφού τόσα ήταν τα συμβατά ζευγάρια. Άρα ταξινομήσαμε τον συνολικό πίνακα τελικά σε το πολύ: $2(d+1)+2(k-1+n-k-d) = \underline{2n \text{ μεταθέσεις}}$.

Άσκηση 3: Υπολογισμός Κυρίαρχων Θέσεων

Η "αφελής" λύση είναι να ελέγξουμε για κάθε στοιχείο, το αμέσως αριστερά του. Αν δεν είναι το κυρίαρχο κοιτάμε το επόμενο και ούτω καθ'εξής. Αυτό δίνει πολυπλοκότητα $O(\sum_{i=1}^{n-1} i) = O(\frac{n(n-1)}{2}) = O(n^2)$.

Μια καλύτερη λύση είναι μέσω στοίβας. Δηλαδή:

Έστω s αρχικά κενή στοίβα, $B = [0...0]$ πίνακας n θέσεων.
$\triangleright$ Για i από 2 έως n

- $s.add(i-1)$
- Όσο $s \neq$ κενή
  - Αν $A[s.top()] \leq A[i] \rightarrow s.pop()$
  - Αλλιώς $break$
- Αν $s \neq$ κενή $\rightarrow B[i] = s.top()$

Ο B στο τέλος θα έχει σε κάθε θέση i αυτήν που κυριαρχεί της i στον πίνακα A.
Ο αλγόριθμος είναι ορθός γιατί σε κάθε επανάληψη i έχουμε την εξής αναλλοίωτη:
"Στην στοίβα s υπάρχουν από πάνω προς τα κάτω με σειρά προτεραιότητας οι μόνες θέσεις του πίνακα που μπορεί να κυριαρχούν της θέσης i"

- Για $i = 2$ ισχύει το παραπάνω (το μόνο στοιχείο που έχει είναι το πρώτο).
- Έστω ισχύει για κάποιο $i \geq 1$.
- Επανάληψη i: αν $A[i] > A[s.top()]$ τότε η θέση $s.top()$ αποκλείεται να είναι κυρίαρχος της $i+1$, αφού η i είναι πιο κοντά και έχει μεγαλύτερη τιμή στον πίνακα.

Για κάθε στοιχείο του πίνακα που θέλουμε να βρούμε την κυρίαρχη θέση του, θα διατρέξουμε το πολύ όλη την στοίβα, σταματάμε μόνο αν βρούμε μεγαλύτερη τιμή στη θέση που δείχνει το κεφάλι της (τότε έχουμε τον κυρίαρχο της θέσης i). Κάθε φορά που "απορρίπτουμε" στοιχείο της στοίβας σαν πιθανό κυρίαρχο, αυτό δε θα εμφανιστεί για την

επόμενη επανάληψη. Έτσι μετά το τέλος των επαναλήψεων θα έχουμε προσπελάσει την στοίβα συνολικά το πολύ n φορές.
Άρα, ο αλγόριθμος μας έχει πολυπλοκότητα $O(n)$

Άσκηση 4: Φόρτιση Ηλεκτρικών Αυτοκινήτων

Ο αλγόριθμος μας είναι ο επόμενος:

- Γνωρίζουμε ότι ο πίνακας των αφίξεων των αυτοκινήτων $\mathrm{A}[1...n]$ είναι σε αύξουσα σειρά, άρα σε $O(n)$ χρόνο μπορούμε να φτιάξουμε τους πίνακες $Time[1...n]$, $Cars[1...n]$ των ζευγών: *(στιγμή, πλήθος αφίξεων)*. Έστω $arrival\_times$ το πλήθος των παραπάνω στιγμών ($O(n)$ σε πλήθος). Επομένως οι θέσεις πέρα από το $arrival\_times$ δε θα μας χρειαστούν στους πίνακες.
- Θα χρησιμοποιήσουμε $binary\ search$ στο πλήθος s των υποδοχών που θα χρειαστούμε (άκρα: 1 και n). Για το δεδομένο s εκτελούμε τον παρακάτω αλγόριθμο:

▷ Για i από 1 έως $arrival\_times - 1$:

- Αν $Cars[i] > s \cdot d$ τότε $binary\ search$ στο δεξί μισό
  Αν δεν ισχύει η παραπάνω σχέση σημαίνει ότι μπορούμε να φορτίσουμε όλα τα αμάξια της στιγμής αυτής μέσα στον χρόνο d
- Αν $Time[i+1] \geq Time[i] + d$ συνέχισε
  Αν η επόμενη στιγμή που έχουμε αφίξεις είναι d μονάδες μακριά, δεν έχουμε πρόβλημα, συνεχίζουμε
- Αλλιώς $Cars[i+1] \leftarrow Cars[i+1] + max\{0, Cars[i] - s \cdot (Time[i+1] - Time[i])\}$
  Αν η επόμενη στιγμή είναι πιο κοντά, τότε πρέπει να της "φορτώσουμε" τα αμάξια που θα απομείνουν

- Αν $Cars[arrival\_times] > s \cdot d$ τότε $binary\ search$ στο δεξί μισό
- Αλλιώς $binary\ search$ στο αριστερό μισό (δεκτό s).

Η λογική είναι ότι διατρέχουμε τις στιγμές άφιξης, και κάθε φορά εξυπηρετούμε s αμάξια. Τα εναπομείναντα τα "φορτώνουμε" στην επόμενη στιγμή και συνεχίζουμε με τον ίδιο τρόπο. Αν φτάσουμε σε στιγμή όπου έχουμε παραπάνω αμάξια από την υποσχόμενη καθυστέρηση επί τις υποδοχές (δηλαδή το σύνολο των αμαξιών που μπορούμε να εξυπηρετήσουμε μέσα σε d στιγμές), τότε χρειαζόμαστε παραπάνω υποδοχές. Αν τελικά οι υποδοχές είναι αρκετές, κοιτάμε για καλύτερη λύση.

Θα αποδείξουμε πρώτα ότι ο παραπάνω αλγόριθμος παράγει σωστές λύσεις (αποδεκτά s). Έστω ότι έχουμε T κουτιά. Το καθένα από αυτό έχει 0 ή παραπάνω αυτοκίνητάκια αριθμένα με τον αύξων αριθμό του κουτιού. Κάθε κουτί αναπαριστά μια στιγμή, αρχικά τα κουτιά έχουν πλήθος αυτοκινήτων ίσο με αυτά που κατέφθασαν τη στιγμή εκείνη. Αν το

s είναι ένας υποψήφιος αριθμός υποδοχών, το ερώτημα ανάγεται στο "Αν κάποιο κουτί έχει παραπάνω από s αυτοκινητάκια, υπάρχει τρόπος να κατανείμουμε τα υπόλοιπα στο μπροστινό του κουτί (αναδρομικά), ώστε καθένα να έχει το πολύ s αυτοκινητάκια και κάθε αυτοκινητάκι με αριθμό i να βρίσκεται το πολύ στο κουτί $i + d - 1$;" Ο αλγόριθμος ελέγχει ακριβώς αυτό. Αρχίζει με το πρώτο κουτί-στιγμή, αν τα αυτοκίνητα δεν είναι παραπάνω από $s \cdot d$ τότε μπορεί να εξυπηρετηθούν όλα μέσα στον χρόνο που υποσχόμαστε. Έτσι ελέγχει πόσα έχει να μοιράσει στο αμέσως επόμενο κουτί που δεν είναι κενό (αν είναι κενό δε μας ενδιαφέρει και κοιτάμε πόσα έχουμε να μοιράσουμε αναδρομικα στο επόμενο μη κενό). Συνεχίζοντας στο επόμενο κουτί-στιγμή κοιτάει αν μπορούν να εξυπηρετηθούν όλα τα αμάξια που έχουν συσσωρευθεί, μέσα σε d στιγμές, δηλαδή αν ισχύει ότι τα αμάξια του 2 είναι λιγότερα από $s \cdot d$ (*). Τα αμάξια που "φορτώθηκαν" στο δεύτερο κουτί είναι το πολύ $(d-1) \cdot s$,΄άρα θα μπορούν να εξυπηρετηθούν μέσα στις επόμενες $d-1$ στιγμές (έχει περάσει ήδη 1 χρονική σχισμή για αυτά), ενώ τα αμάξια που είχε ήδη το κουτί-στιγμή 2 θα εξυπηρετηθούν το αργότερο την στιγμή $d-1+2 = d+1$, δηλαδή θα έχουν τελειώσει την φόρτιση τη στιγμή $d+2$: μέσα στον χρόνο. Αν για κάποιο κουτί δεν ισχύει η (*), σημαίνει ότι δεν υπάρχει τρόπος μοιρασιάς-χρονοπρογραμματισμού. Επαγωγικά από την παραπάνω ανάλυση επαληθεύεται εύκολα η ορθότητα του αλγορίθμου. Η βελτιστότητα της λύσης έρχεται από το $binary\_search$ καθώς αν έχουμε δεκτή λύση συνεχίζουμε μέχρι να βρούμε καλύτερη.

Η πολυπλοκότητα του αλγορίθμου είναι $O(nlogn)$, αφού όπως είπαμε η δημιουργία των πινάκων είναι $O(n)$, το $binary\_search$ είναι προφανώς $O(logn)$ και εσωτερικά κάνει το πολύ $O(n)$ επαναλήψεις.

Άσκηση 5: Επιλογή

α) Ο αλγόριθμος μας έχει ως εξής:

- Έστω $left = 1$, $right = M$, $middle = \left\lfloor \frac{right+left}{2} \right\rfloor$.

▷ Όσο $left < right - 1$:

- Αν $F_s(middle) < k$ τότε: $left \leftarrow middle$
- Αλλιώς: $right \leftarrow middle$
- $middle \leftarrow \left\lfloor \frac{right+left}{2} \right\rfloor$

$Answer = right$

Η αναλλοίωτη του αλγόριθμου είναι ότι για το $right$ πάντα ισχύει $F_s(right) \geq k$, ενώ για το $left$ πάντα ισχύει $F_s(left) < k$. Η απόδειξη αυτού είναι τετριμμένη. Ο αλγόριθμος τερματίζει καθώς σε κάθε επανάληψη τα $left$ και $right$ έρχονται πιο "κοντά", αφού θα εκτελεστεί είτε το "Αν" είτε το "Αλλιώς" εξισώνοντας μια από τις άκρες με

τη μέση, η οποία αμέσως μετά επαναυπολογίζεται μέχρι οι άκρες να έχουν διαφορά 1. Έχοντας ως υπ'όψιν την αναλλοίωτη που περιγράψαμε, όταν τερματίσει ο αλγόριθμος έχουμε $F_s(left) = k - m < k$, $F_s(right) = k + M \geq k$ και $left = right - 1$. Άρα, ο πίνακας μας θα έχει $k + M \geq k$ αριθμούς μεγαλύτερους ή ίσους του $right$ και $k - m < k$ αριθμούς μικρότερους του $right - 1$. Δηλαδή θα έχει $M + m \geq 1$ αριθμούς ίσους με $right$ και ο k-οστός μικρότερος αριθμός που ψάχνουμε τότε είναι ο $right$. Η πολυπλοκότητα του αλγορίθμου είναι $\underline{\Theta(logM)}$, αφού σε κάθε επανάληψη έχουμε διαίρεση του μήκους αναζήτησης δια 2, μέχρι αυτό να γίνει 2.

β) Ο υπολογισμός του $F_s(k)$ έχει ως εξής:

- Αρχικά ταξινομούμε τον πίνακα σε $O(n)$ χρόνο με την $radixsort$ (έχουμε ακεραίους).

- Έστω $Answer = 0, B = 2$
▷ Για i από 1 έως $n - 1$:

  - Όσο $(A[R] \leq A[i] + k$ και $B \leq n)$: $B \leftarrow B + 1$
  - $Answer \leftarrow Answer + B - 1 - i$

Αρχίζοντας από το πρώτο στοιχείο εξετάζουμε μέχρι ποιο στοιχείο (από εδώ και πέρα "ακριανό" της θέσης) διαφέρει κατά k το πολύ. Προσμετράμε αυτό τον αριθμό στοιχείων. Για το επόμενο στοιχείο, αφού είναι μεγαλύτερο ή ίσο του πρώτου, ξέρουμε ότι η διαφορά του με το ακριανό που βρήκαμε πριν θα είναι μικρότερη ή ίση του k, άρα και η διαφορά του με τα ενδιάμεσα στοιχεία, αφού ο πίνακας είναι ταξινομημένος. Συνεχίζουμε (αν γίνεται) βρίσκοντας το νέο ακριανό για το δεύτερο στοιχείο και επαναλαμβάνουμε την διαδικασία.
Έτσι προσμετράμε κάθε φορά τα δεκτά ζευγάρια που ανήκει το στοιχείο $A[i]$ και επομένως μετά δεν μας "χρειάζεται" το στοιχείο $A[i]$. Παρατηρούμε ότι ο δείκτης B θα φτάσει το πολύ μέχρι την θέση n από όπου μετά δε θα κουνηθεί, καθώς όλα τα επόμενα στοιχεία σίγουρα θα δημιουργούν ζευγάρια διαφορών μικρότερων/ίσων του k με τα επόμενα τους (αφού "προχωρόντας" τα στοιχεία θα "πλησιάζουν" τα επόμενα τους). Επομένως, η διαδικασία αυτή έχει πολυπλοκότητα $O(n)$.

Ο αλγόριθμος μας για την λύση του προβλήματος θα χρησιμοποιήσει τον αλγόριθμο του υποερωτήματος (α) για την εύρεση του k-οστού μικρότερου στοιχείου του S καλώντας την παραπάνω υλοποίηση της F_s. Ο αλγόριθμος που είχαμε υλοποιήσει είχε πολυπλοκότητα $O(logM)$ και αφού θα καλούμε την F_s κάθε φορά, ο αλγόριθμος του προβλήματος αυτού θα έχει πολυπλοκότητα $\underline{O(nlogM)}$.