Αλγόριθμοι και Πολυπλοκότητα ΣΗΜΜΥ

Ιωάννης Κ. Γεωργακόπουλος

03111512

2018-2019

1η σειρά γραπτών ασκήσεων

Άσκηση 1: Ασυμπτωτικός Συμβολισμός, Αναδρομικές Σχέσεις.

(a)

- $n^2 = \Theta(n^2)$

- $2^{(\log_2 n)^4} = \Theta(2^{\log_2^4 n})$

- $\dfrac{\log(n!)}{(\log n)^3}$

Έχω $n! < n^n \overset{\log\uparrow}{\Rightarrow} \log(n!) < \log n^n \Rightarrow \log n! < n\log n \Rightarrow \log n! = O(n\log n)$

και $\left(\frac{n}{2}\right)^{\frac{n}{2}} < n! \overset{\log\uparrow}{\Rightarrow} \log\left[\left(\frac{n}{2}\right)^{\frac{n}{2}}\right] < \log n! \Rightarrow \frac{n}{2}\log\frac{n}{2} < \log n! \Rightarrow \log n! = \Omega(n\log n)$

άρα $\log(n!) = \Theta(n\log n)$

Οπότε $\dfrac{\log(n!)}{(\log(n))^3} = \Theta\left(\dfrac{n\log n}{(\log n)^{\cancel{3}2}}\right) = \Theta\left(\dfrac{n}{\log^2 n}\right)$

- $n \cdot 2^{2^{2^{100}}} = \Theta(c \cdot n) = \Theta(n)$

- $\log\left(\binom{n}{\log n}\right) = \log\left[\dfrac{n!}{\log n!(n! - \log n!)}\right] = \log n! - \log[\log n!(n! - \log n!)]$

$< \log n^n - \log[(\log n)^{\log n}(n - \log n)^{n-\log n}] = \log n^n - [\log[\log n \cdot \log n] + \log(n - \log n)^{n-\log n}]$

Scanned by CamScanner

$= n\log n - \log(\log^2 n) - (n-\log n)\cdot \log(n-\log n)$

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

$n > n-\log n$ κ' $\log\uparrow$

$< n\log n - \log\log^2 n - \cancel{n\log n} + \log n\cdot \log n$

$= \log^2 n - \log\log^2 n$

$= O(\log^2 n)$

Όμοια δείχνουμε: $\log n! - \log[(\log n)!(n-\log n)!] > \log \frac{n}{2}^{n/2} - \log\left(\left(\frac{\log n}{2}\right)^{\frac{\log n}{2}} \left(\frac{n-\log n}{2}\right)^{\frac{n-\log n}{2}}\right)$

Λαμβάνω $\log\left(\binom{n}{\log n}\right) = \Omega(\log^2 n)$

Άρα τελικά $\log\left(\binom{n}{\log n}\right) = \Theta(\log^2 n)$

- $\dfrac{(\log n)^2}{\log\log n} = \Theta\left(\dfrac{(\log n)^2}{\log\log n}\right)$

- $\log^4 n = \Theta(\log^4 n)$

- $(\sqrt{n})!$

Γνωρίζω ότι $\frac{1}{2} n\log n < \log(n!) < n\log n$

Άρα έχω $\log(\sqrt{n}!) < \sqrt{n}\cdot\log\sqrt{n} \overset{2^x}{\Rightarrow} \sqrt{n}! < 2^{\sqrt{n}\cdot\log\sqrt{n}} = 2^{\sqrt{n}\cdot \log n^{1/2}} = 2^{\frac{1}{2}\sqrt{n}\log n}$

άρα $\sqrt{n}! = O(2^{\frac{1}{2}\sqrt{n}\cdot\log n})$

και $\log(\sqrt{n}!) > \frac{1}{2}\sqrt{n}\cdot\log\sqrt{n} \overset{2^x}{\Rightarrow} \sqrt{n}! > 2^{\frac{1}{2}\sqrt{n}\cdot\log\sqrt{n}} = 2^{\frac{1}{4}\sqrt{n}\cdot\log n}$

άρα $\sqrt{n}! = \omega(2^{\frac{1}{4}\sqrt{n}\cdot\log n})$

τελικά $\sqrt{n}! = o(2^{\frac{1}{2}\sqrt{n}\log n})$ και $\sqrt{n}! = \omega(2^{\frac{1}{4}\sqrt{n}\cdot\log n})$

Scanned by CamScanner

- $\binom{n}{6} = \frac{n!}{6!(n-6)!} = \frac{(n-5)\cdot(n-4)\cdot(n-3)\cdot(n-2)\cdot(n-1)\cdot n}{6!} = \Theta\left(\frac{n^6}{6!}\right) = \Theta(n^6)$

- $\frac{n^3}{(\log n)^8} = \Theta\left(\frac{n^3}{(\log n)^8}\right)$

- $(\log_2 n)^{\log_2 n} = 2^{\log_2[(\log_2 n)^{\log_2 n}]} = 2^{\log_2(n)\cdot\log_2(\log_2 n)} = \Theta\left(2^{\log_2(n)\cdot\log_2(\log_2 n)}\right)$

- $\log\left(\binom{2n}{n}\right) = \log\left[\frac{(2n)!}{n!(2n-n)!}\right] = \log\left[\frac{(2n)!}{\underbrace{n!\,n!}_{2\cdot(n!)}}\right] = \log\left[\frac{\cancel{n!}(n+1)\cdots(2n-1)\cdot(2n)}{\cancel{n!}}\cdot\frac{1}{2}\right]$

$= \log\left[(n+1)\cdot(n+2)\dots(2n-1)\cdot(2n)\right] - \log 2$

$= \underbrace{\log(n+1)+\log(n+2)+\log(n+3)+\dots+\log(2n-1)+\log(2n)}_{n \text{ πλήθος}} - 1$

$< n\cdot\log(2n) - 1 = n\cdot(\log 2 + \log n) - 1 = n + n\cdot\log n - 1 = O(n\log n)$

Επίσης $\log(n+1)+\dots\log(2n-1)+\log(2n) - 1 > n\log n - 1 = \Omega(n\log n)$

Άρα τελικά $\log\left(\binom{2n}{n}\right) = \Theta(n\log n)$

- $n\cdot\sum_{k=0}^{n}\binom{n}{k}$

Γνωρίζω ότι από διωνυμικό τύπο, για $a=b=1$ ισχύει:
$(1+1)^n = \sum_{k=0}^{n}\binom{n}{k}1^{n-k}\cdot 1^k \Rightarrow \sum_{k=0}^{n}\binom{n}{k} = 2^n$

άρα τελικά $n\cdot\sum_{k=0}^{n}\binom{n}{k} = n\cdot 2^n = 2^{\log(n\cdot 2^n)} = 2^{\log n + \log 2^n} = 2^{\log n + n\cdot\log 2}$
$= 2^{n+\log n}$
$= \Theta(2^{n+\log n})$

Scanned by CamScanner

• $(\sqrt{n})^{\log_2\log_2(n!)} = 2^{\log[(\sqrt{n})^{\log\log(n!)}]} = 2^{\log\log(n!)\cdot\log\sqrt{n}}$
$= 2^{\log(n^{1/2})\cdot\log\log(n!)} = 2^{\frac{1}{2}\cdot\log n\cdot\log\log(n!)}$

Γνωρίζω ότι $\left(\frac{n}{2}\right)^{\frac{n}{2}} < n! < n^n$ και 2^n, $\log \nearrow$

άρα $2^{\frac{1}{2}\log n\cdot\log\log(n!)} < 2^{\frac{1}{2}\log n\cdot\log\log(n^n)} = 2^{\frac{1}{2}\log n\cdot\log(n\log n)} = 2^{\frac{1}{2}\log n\cdot(\log n+\log\log n)}$

=

και $2^{\frac{1}{2}\log n\cdot\log\log(n!)} > 2^{\frac{1}{2}\log n\cdot\log\log(\frac{n}{2})^{\frac{n}{2}}} = 2^{\frac{1}{2}\log n\cdot\log[\frac{n}{2}\cdot\log\frac{n}{2}]} = 2^{\frac{1}{2}\log n\cdot(\log\frac{n}{2}+\log\log\frac{n}{2})}$

άρα $(\sqrt{n})^{\log_2\log_2(n!)} = o\left(2^{\frac{1}{2}\log n(\log n+\log\log n)}\right)$ κ'
$(\sqrt{n})^{\log_2\log_2(n!)} = \omega\left(2^{\frac{1}{2}\log n(\log\frac{n}{2}+\log\log\frac{n}{2})}\right)$

• $\sum_{k=1}^{n} k\cdot 2^k$

Προφανώς ισχύει $\sum_{k=1}^{n} k\cdot 2^k \geq n\cdot 2^n = 2^{\log(n\cdot 2^n)} = 2^{n+\log n}$

Επίσης ισχύει $\sum_{k=1}^{n} k\cdot 2^k \leq n\sum_{k=1}^{n} 2^k \leq n\cdot n\cdot 2^n = 2n2^n = 2^{n+\log n+1}$

άρα $\sum_{k=1}^{n} k\cdot 2^k = \Theta(2^{n+\log n})$

• $\sum_{k=1}^{n} k\cdot 2^{-k}$

Έστω σταθερά C

Έχω $\sum_{k=1}^{n} k\cdot 2^{-k} \leq \sum_{k=1}^{\infty} k\cdot 2^{-k} \leq C + \sum_{k=1}^{\infty} (0.6)^k \leq C + \frac{10}{4}$, αφού $k \in o(2^k(0.6)^k)$

άρα $\sum_{k=1}^{n} k\cdot 2^{-k} = \Theta(1)$

Άρα τελικά →

Scanned by CamScanner

Ταξινομώντας σε αύξουσα σειρά τάξης μεγέθους, λαμβάνουμε την παρακάτω διάταξη:

$\sum_{k=1}^{n} k\cdot 2^{-k}$, $\frac{(\log n)^2}{\log\log n}$, $\log\left(\binom{n}{\log n}\right)$, $\log^4 n$, $\frac{\log(n!)}{(\log(n))^3}$,

$n\cdot 2^{2^{2^{100}}}$, $\log\binom{2n}{n}$, n^2, $\frac{n^3}{(\log n)^8}$, $\binom{n}{6}$, $(\log_2 n)^{\log_2 n}$,

$(\sqrt{n})^{\log_2\log_2(n!)}$, $2^{(\log_2 n)^4}$, $(\sqrt{n})!$, $n\cdot\sum_{k=0}^{n}\binom{n}{k}$, $\sum_{k=1}^{n} k\cdot 2^{k}$

Εκ των οποίων οι $n\cdot\sum_{k=0}^{n}\binom{n}{k}$ και $\sum_{k=1}^{n} k\cdot 2^{k}$ έχουν ίδια τάξη μεγέθους $(\Theta(2^{n+\log n}))$.

(β) 1. $T(n)=2T(n/3)+n\log n=\Theta(n\log n)$, αφού από master theorem έχω:

- $n^{\log_3 2}=n^{\log 2/\log 3}<n^1$, άρα $n\log n=\Omega(n^{\log_3 2+\varepsilon})$, $\varepsilon>0$ } περίπτωση 3.
- $2\frac{n}{3}\cdot\log\frac{n}{3}<n\log n$

2. $T(n)=3T(n/3)+n\log n=\Theta(n\log^2 n)$ από δέντρο αναδρομής

3. $T(n)=4T(n/3)+n\log n=\Theta(n^{\log_3 4})$, αφού από master theorem έχω:

$n^{\log_3 4}=n^{\log 4/\log 3}>n^1$, 1η περίπτωση

4. $T(n)=T(n/2)+T(n/3)+n$ από αναδρομικό δέντρο, σε κάθε level έχουμε $\frac{5}{6}$ του κόστους του προηγούμενου

άρα $T(n)=\Theta\left(n\cdot\left(\underbrace{1+\frac{5}{6}+\left(\frac{5}{6}\right)^2+\dots\left(\frac{5}{6}\right)^h}_{\text{φραγμένο από σταθερά αφού } \log_3 n<h\le\log_2 n}\right)\right)$

άρα τελικά $T(n)=\Theta(n)$

Scanned by CamScanner

5. $T(n/2)+T(n/3)+T(n/6)+n$ αντιστοιχεί με το προηγούμενο από αναδρομικό δέντρο σε κάθε επίπεδο έχουμε $\Theta(n)$ κόστος, όσο το προηγούμενο. Ισχύει πάλι $\log_6 n \le h \le \log_2 n$. Άρα, $T(n)=\Theta(n\log n)$

6. $T(n)=T(n^{5/6})+\Theta(\log n)$ από αναδρομικό δέντρο βλέπουμε κάθε επίπεδο να έχει κόστος $5/6$ του προηγούμενου.
Άρα $T(n)=\Theta(\log n(\underbrace{1+\frac{5}{6}+(\frac{5}{6})^2+\dots(\frac{5}{6})^h}_{\text{φραγμένο}}))$. Όμως $h=\Theta(\log\log n)$

Άρα $T(n)=\Theta(\log n)$

7. $T(n)=T(n/4)+\sqrt{n}$ Αντίστοιχα, από αναδρομικό δέντρο κάθε επίπεδο έχει κόστος $1/2$ του προηγούμενου
Άρα $T(n)=\Theta(\sqrt{n}(\underbrace{1+\frac{1}{2}+\dots(\frac{1}{2})^h}_{\text{φραγμένο}}))$

Άρα $T(n)=\Theta(\sqrt{n})$

Άσκηση 2: Ταξινόμηση

(a) 1. Αρκεί να εκτελέσουμε ντετερμινιστική quicksort με χρήση διαμέσου μέχρι κάθε υποπίνακας να έχει $\frac{n}{k}$ στοιχεία. Αυτό συμβαίνει σε βάθος $\log k$. Γνωρίζοντας ότι η εύρεση του μέσου και ο διαχωρισμός των στοιχείων σε κάθε επίπεδο θέλει γραμμικό χρόνο ($\Theta(n)$) και έχουμε $\log k$ επίπεδα, ο αλγόριθμος θα έχει συνολικό χρόνο εκτέλεσης $O(n\log k)$.
Όμως κάθε στοιχείο μπορεί να ανήκει σε ένα από τα k σύνολα, άρα έχουμε $\binom{n}{\frac{n}{k}\dots\frac{n}{k}}=\frac{n!}{((\frac{n}{k})!)^k}$, δυνατούς συνδυασμούς, άρα το ύψος του δέντρου θα είναι τουλάχιστον $\log(n!)-k\log((\frac{n}{k})!)\simeq n\log n-n\log\frac{n}{k}\simeq n\log k$. Συνεπώς κάθε συγκριτικός αλγόριθμος έχει χρόνο εκτέλεσης τουλάχιστον $\Omega(n\log k)$. Άρα ο αλγόριθμος μας είναι βέλτιστος

Scanned by CamScanner

2. Αρχικά να προχωρήσουμε την quicksort από το σημείο που τερματίζει (έχει φτάσει βάθος $\log k$). Έως το βάθος $\log n$, όπου θα έχει ταξινομηθεί πλήρως. Άρα $\log n - \log k = \log\frac{n}{k}$ το πλήθος των υπολειπόμενων επιπέδων και εφόσον έχουμε το ίσο κόστος ανά επίπεδο, ο χρόνος εκτέλεσης θα ~~ισούται~~ είναι $O(n\log\frac{n}{k})$.

Προφανώς αυτός ο χρόνος είναι και ο βέλτιστος καθώς με την εφαρμογή των 2 αλγορίθμων έχουμε συνολικό χρόνο εκτέλεσης

$$O(n\log k + n\log\tfrac{n}{k}) = O(n\log k + n\log n - \log k) = O(n\log n)$$

που γνωρίζουμε πως είναι και το κάτω όριο για συγκριτικούς αλγορίθμους της ταξινόμησης του πίνακά μας.

Άρα τελικά $\Theta(n\log\frac{n}{k})$.

(β) Ταξινομώντας κατά k μέρη τον πίνακα A, προκύπτουν το πολύ $\frac{k}{2}$ υποπίνακες με >2 εμφανίσεις διαφορετικών ακεραίων, άρα και τουλάχιστον $\frac{k}{2}$ υποπίνακες με <=2 εμφανίσεις διαφορετικών ακεραίων, οι οποίοι έχουν γραμμικό κόστος ταξινόμησης $\Theta(n)$. Οπότε μετά από κάθε φορά αναδρομής υπολείπονται $\frac{k}{2}$ πίνακες με n/k στοιχεία έκαστος. Άρα ισχύει $T(n) \le T(\frac{n}{2}) + \Theta(n\log k)$

Οπότε $T(n) = O(n\log k)$

Γνωρίζουμε όμως ότι έχουμε $O(\log^d n)$ διαφορετικά στοιχεία άρα αντικαθιστώντας $k = O(\log^d n)$:

$$T(n) = O(n\log(\log^d n)) = O(n d\cdot\log\log n) = O(n\cdot\log\log n)$$

Προφανώς, εφόσον έχουμε πολλαπλές εμφανίσεις, το ασυμπτωτικό συγκριτικό φράγμα θα είναι μικρότερο από αυτό του ερωτήματος 1.

Scanned by CamScanner

Άσκηση 3 : Διάστημα Ελάχιστου μήκους που καλύπτει όλους τους πίνακες

a) min-length algorithm

1. $mini = \infty$, $i_1 = 1$, $i_2 = 1$
2. while $i_1 < n_1$ && $i_2 < n_2$ do:
   i) $mini = \min\{mini, |A_1[i_1] - A_2[i_2]|$
   ii) if $A_1[i_1] < A_2[i_2]$ {
       i_1++
     else {
       i_2++
3. return mini

Ορθότητα: Έστω αυθαίρετα ότι $A_1[1] < A_2[1]$. A_2 sorted, άρα $|A_1[1] - A_2[i_2]|$ ελάχιστο για $i_2 > 1$. Άρα πρέπει να αυξηθεί το i_1 και να προχωρήσει. Με επαγωγή στο $(n_1 + n_2 - 1)$ θα έχουμε σωστό αποτέλεσμα.

Πολυπλοκότητα: Γραμμικού χρόνου, όπως ζητείται (max 2n επαναλήψεις στην χειρότερη περίπτωση)

β) 1. $mini = \infty$, $i_j = 1$ για κάθε $j \in m$
2. While $\bigwedge_{i=1} (i_j < n_j)$
   i) $mini = \min\{mini, \max_j A_j[i] - \min_j A_j[i_j]\}$
   ii) $\arg\min_j A_j[i_j]$++
3. return mini

Ορθότητα: Έστω αυθαίρετα ότι $\arg\min_j A_j[i_j] = 1$.

Άρα προφανώς $\max_{j>1}\{A_1[1], A_j[i_j]\} - \min_j\{A_1[1], A_j[i_j]\} >=$

$\max_j\{A_j[1] - \min A_j[1]$. Οπότε για τα μεγάλα

Scanned by CamScanner

αυξάνουμε το i_1. Με επαγωγή στο υπόλοιπο σύνολο αυτό αποτέλεσμα συνολικό.

Πολυπλοκότητα: $O(m)$ εντολές για εύρεση min, max, και
$O(N)$ επαναλήψεις, μία για κάθε increment
ενός $\arg\min_j A_j[i_j]$.
Άρα συνολικά πολυπλοκότητα $O(mN)$.

γ) Το κόστος που καθορίζει τον αλγόριθμο και μπορούμε να μειώσουμε, είναι το κόστος εύρεσης των min, max. Οπότε, για να βελτιώσουμε τον αλγόριθμο θα χρησιμοποιήσουμε μία σωρό και θα εργαστούμε ως εξής: Θα αποθηκεύουμε σε μία μεταβλητή το μέγιστο, ~~τ~~ για τα min θα χρησιμοποιούμε τη σωρό, την οποία ξεκινάμε αρχικά με τα $A_j[1]$ στοιχεία.

άρα, ο νέος αλγόριθμος

1. $mini = \infty$, $maxi = -\infty$, $i_j = 1$ για κάθε $j \in m$
2. γέμισε τη σωρό με m στοιχεία ($A_j[1]$ αρχικά)
3. while $\bigwedge_{j=1}(i_j < n_j)$

    i) pop $\min_j A_j[i_j]$
    ii) $mini = \min\{mini, maxi - \min_j A_j[i_j]\}$
    iii) $\arg\min_j A_j[i_j]$++
    iv) $w = \arg\min_j A_j[i_j]$
    v) $maxi = \max\{maxi, A_w[i_w]\}$
4. return mini

Ο αλγόριθμος είναι ορθός, αφού έχει την ίδια δομή με πριν. Επίσης, στη σωρό τα pop και push απαιτούν $O(\log m)$.
Άρα η πολυπλοκότητα μειώνεται σε $O(N \cdot \log m)$

Scanned by CamScanner

Άσκηση 4. Αναζήτηση

(a) Το εν λόγω πρόβλημα συναντάται συχνά στα διαγράμματα της λογικής σχεδίασης και ο λόγος είναι προφανής.

Μπορούμε να κωδικοποιήσουμε τις φιάλες σε δυαδική βάση. Θα χρειαστούν $\lceil \log_2 1000.000 \rceil = 20$ εθελοντές. Κάθε φιάλη δοκιμάζεται από τους εθελοντές των οποίων ο αύξων αριθμός αντιστοιχεί σε set bit του κωδικού της. Επομένως μετά από 24 ώρες, βλέποντας τους εθελοντές που έχουν υπερευαισθησία (ή πέθαναν γιατί τελικά το φάρμακο ήταν δηλητήριο), κατασκευάζουμε (αποκωδικοποιούμε) τον κωδικό της φιάλης που έχει το φάρμακο.

(β) Η ελάχιστη δυνατή απόσταση σε μία μέρα είναι $\frac{\Sigma d_i}{k}$ και η μέγιστη Σd_i. Πρώτα θα ελέγξουμε αν η ελάχιστη επιτρεπτή είναι doable σε k days και αν ισχύει. Θα τρέξουμε δυαδική αναζήτηση για την ελάχιστη απόσταση μεταξύ των $\frac{\Sigma d_i}{k}$ και Σd_i.

```
doable(dist)
    1. days=1, length=0, i=1
    2. if l+di ≤ dist
          length=length+di
          i++
          If i>n return T else goto: 2.
    3. days++
       If days ≤k{
          length=0
          goto: 2
       }else{
          return F
       }
```

main algorithm of binary search

1. $l = \frac{\Sigma d_i}{k}$, $h = \Sigma d_i$
2. If double(l) return l
3. While $h-l>1$:

   $mid = \frac{i+h}{2}$

   if double(mid) then h=mid else l=mid
4. return h

Ορθότητα: Το h θα η θα είναι πάντα εφικτό, αφού ενώ η από κάτω των από το l υποδιπλασιάζεται σε κάθε βήμα. Άρα θα έχουμε ορθό αποτέλεσμα.

Πολυπλοκότητα: Αφού θέλουμε γραμμικό χρόνο για το double ($\Theta(n)$) και λογαριθμικό για τη δυαδική αναζήτηση το κόστος θα είναι $\Theta(n \log \Sigma d_i)$.

Άσκηση 5: Επιλογή

(α) binary search

1. $l=1$, $h=M$
2. If $F_s(l) \geq l$ return l
3. while $h-l>1$:

   i) $mid = \frac{h+l}{2}$

   ii) if $F_s(mid) < k$ then l=mid else h=mid.
4. return h.

Ορθότητα: Το k-οστό στοιχείο θα είναι το h, αφού χρησιμοποιώντας τη δοθείσα συνάρτηση F_s θα έχουμε $F_s(h) \geq k$ και $F_s(l) < l$. Άρα το τελικό αποτέλεσμα θα είναι ορθό.

Scanned by CamScanner

Πολυπλοκότητα: Έστω t_{Fs} ο χρόνος εκτέλεσης μιας κλήσης της Fs. Εφόσον εκτελούμε δυαδική αναζήτηση κάνουμε $\log M$ επαναλήψεις, σε κάθε μία από τις οποίες καλείται η συνάρτηση Fs. Άρα χρόνος εκτέλεσης: $\Theta(t_{Fs} \log M)$

(β) Ο πίνακας A δεν δίνεται ταξινομημένος άρα πληρώνουμε $\Theta(n\log n)$ αρχικά ώστε να τον ταξινομήσουμε.

Στη συνέχεια [illegible] τον αλγόριθμο του α) ερωτήματος καλώντας τον παρακάτω για την Fs

```
Fs(l):
  1. sum=0, i=1, j=1
  2. if l<0 return 0.
  3. while j≤n:
     i) while i<n && (A[i+1]-A[j]) ≤ l do: i++
     ii) sum+= i-j
     iii) j++
     iv) if j==i+1 then i++
  4. return sum
```

Ορθότητα: Όσο η διαφορά $A[i]-A[j] < l$, το i αυξάνεται, για δεδομένο j. Άρα μέχρι το μέγιστο i λαμβάνουμε τα ζευγάρια για τα οποία ισχύει $A[i]-A[j] \le l$. Λόγω ταξινομημένου πίνακα, το i αυξάνεται ορθά με την αύξηση του j. Άρα τελικά επιστρέφεται το ορθό αποτέλεσμα (πλήθος των δυνατών αποδεκτών ζευγαριών).

Πολυπλοκότητα: ταξινόμηση: $\Theta(n \log n)$

χρόνος κλήσης: $\Theta(n)$, αφού θα διασχιστούν όλα τα στοιχεία του πίνακα από τους δύο δείκτες.

πλήθος κλήσεων: $\log M$

άρα πολυπλοκότητα: $\Theta(n\log n + n\log M) = \Theta(n(\log n + \log M))$

Scanned by CamScanner