ΕΚΠΑ, Τμήμα Πληροφορικής και Τηλεπικοινωνιών Αλγόριθμοι και Πολυπλοκότητα

Τμήμα ΄Αρτιων Εαρινό Εξάμηνο 2019-2020 Εργασία 1

Ιωαννης Ροβιθάκης sdi1800164

1. Αρχικά βρίσκουμε την αναδρομική εξίσωση χρόνου εκτέλεσης κάθε ενός από τους δεδομένους αλγορίθμους:

i) **A: $T_A(n) = 4\,T_A(\frac{n}{2}) + \frac{n^2}{logn}$**

Θα κάνουμε αναδίπλωση:

{ $T_A(\frac{n}{2}) = 4\,T_A(\frac{n}{4}) + \frac{n^2}{4logn-1}$, $T_A(\frac{n}{4}) = 4\,T_A(\frac{n}{8}) + \frac{n^2}{16logn-2}$ }

$T_A(n) = 4\,T_A(\frac{n}{2}) + \frac{n^2}{logn} = 4\,(\,4\,T_A(\frac{n}{4}) + \frac{n^2}{4logn-1}\,) + \frac{n^2}{logn} =$

$4^2\,T_A(\frac{n}{4}) + \frac{n^2}{logn-1} + \frac{n^2}{logn} = 4^2\,(\,4\,T_A(\frac{n}{8}) + \frac{n^2}{16logn-2}\,) + \frac{n^2}{logn-1} + \frac{n^2}{logn} =$

$4^3\,T_A(\frac{n}{8}) + \frac{n^2}{logn-2} + \frac{n^2}{logn-1} + \frac{n^2}{logn} =$ .....

Οπότε γενικά για κάποιο k: **... = $4^k\,T_A(\frac{n}{2^k}) + n^2 \sum_{i=0}^{k-1} \frac{1}{logn-i}$**

Ο αλγόριθμος τερματίζει για $\frac{n}{2^k}$ = 1 => n = 2^k => **k = logn**

Με αντικατάσταση προκύπτει:

$4^{logn}\,T_A(1) + n^2 \sum_{i=0}^{logn-1} \frac{1}{logn-i} = 2^{2logn}\,TA(1) + n^2 \sum_{i=0}^{logn-1} \frac{1}{logn-i} =$

$2^{log\,n^2}\,T_A(1) + n^2 \sum_{i=0}^{logn-1} \frac{1}{logn-i}$ =>

Αφού $T_A(1)$ = 1 και το αθροισμα έχει logn όρους =>

$T_A(n) = n^2 + n^2\,logn$

Άρα τελικά η πολυπλοκότητα του Α θα είναι: $O(n^2\,logn)$

ii) **B: $T_B(n) = 9\,T_B(\frac{n}{3}) + n^2$**

Δηλαδή είναι της μορφής **$T(n) = aT(\frac{n}{b}) + f(n)$** με a**=9, b=3, f(n)= n^2**

έχουμε ότι **$\log_b a = \log_3 9 = 2$** οπότε **$f(n)= n^2 = \Theta(n^2)$**. Συνεπώς,

σύμφωνα με την **2η** περίπτωση του Θεωρήματος της Κυριαρχίας, η λύση της αναδρομικής εξίσωσης είναι $\mathbf{T_B(n) = \Theta(n^2 \log n)}$

iii) **C:** $\mathbf{T_C(n) = \sqrt{n}\, T_C(\sqrt{n}) + n = n^{\frac{1}{2}}\, T_C(n^{\frac{1}{2}}) + n}$

Θα κάνουμε αναδίπλωση:

$\{\ \mathbf{T_C}(n^{\frac{1}{2}}) = n^{\frac{1}{4}}\, \mathbf{T_C}(n^{\frac{1}{4}}) + n^{\frac{1}{2}}\ ,\ \mathbf{T_C}(n^{\frac{1}{4}}) = n^{\frac{1}{8}}\, \mathbf{T_C}(n^{\frac{1}{8}}) + n^{\frac{1}{4}}\ \}$

$\mathbf{T_C(n) = n^{\frac{1}{2}}\, T_C(n^{\frac{1}{2}}) + n = n^{\frac{1}{2}}\,(\, n^{\frac{1}{4}}\, T_C(n^{\frac{1}{4}}) + n^{\frac{1}{2}}\,) + n = n^{\frac{3}{4}}\, T_C(n^{\frac{1}{4}}) + 2n =}$
$\mathbf{n^{\frac{3}{4}}\,(\, n^{\frac{1}{8}}\, T_C(n^{\frac{1}{8}}) + n^{\frac{1}{4}}\,) + 2n = n^{\frac{7}{8}}\, T_C(n^{\frac{1}{8}}) + 3n = ...}$

Οπότε γενικά για κάποιο k: $\mathbf{.. = n^{\frac{k-1}{k}}\, T_C(n^{\frac{1}{2^k}}) + kn = n^{1-\frac{1}{k}}\, T_C(n^{\frac{1}{2^k}}) + kn}$

Ο αλγόριθμος τερματίζει για $\mathbf{n^{\frac{1}{2^k}} = 2}$ (βολεύει στις πραξεις το 2 με 1 κολλαει) => $\mathbf{\frac{1}{2^k}\log n = 1 \Rightarrow \log n = 2^k \Rightarrow k = \log\log n}$

Με αντικατάσταση προκύπτει:

$\mathbf{n^{1-\frac{1}{\log\log n}}\, T_C(2) + n \log\log n = \frac{n}{n^{\log\log n}}\, T_C(2) + n \log\log n}$

$T_C(2) = \theta(1)$ (Μας είπε ο κύριος εμίρης την ώρα του μαθήματος ότι μπορούμε να χρησιμοποιήσουμε και αυτό χωρίς απόδειξη)

$\mathbf{\frac{n}{n^{\log\log n}} = O(n \log\log n)}$ **{ Αφού:** $\mathbf{\lim_{n\to\infty} \frac{n \log\log n}{n^{\frac{n}{\log\log n}}} = \infty\ \}}$

Οπότε τελικά:$T_C(n) = O(n \log\log n)$

Τελικά, συγκρίνοντας τα τρία αποτελέσματα καταλήγουμε ότι ο αλγόριθμος C είναι ο καλύτερος.

2. Έχουμε k = 2^r (άρτια δύναμη του 2) πλήθος ταξινομημένων ακολουθιών ν στοιχείων. Θέλουμε να τις συνδυάσουμε σε μια ταξινομημένη ακολουθία kn στοιχείων.

Εφόσον το πλήθος των ακολουθιών είναι άρτια **δύναμη του 2**, σε κάθε αναδρομική κλήση χωρίζω τις ακολουθίες σε "**ζευγάρια**" η επιλογή των οποίων είναι αυθέρετη και στη συνέχεια κάνω **ταξινομημένα merge** το κάθε ζευγάρι με αποτέλεσμα να καταλήξουμε μετά την πρώτη αναδρομική κλήση σε $\frac{k}{2}$ ταξινομημένες ακολουθίες των **2n** στοιχείων η κάθε μια. Συνεχίζουμε όμοια μέχρι να φτάσουμε σε 2 ακολουθίες με $\frac{\kappa}{2}$ $\boldsymbol{n}$ στοιχεία συνδιάζοντας τις οποίες καταλήγω στην τελική επιθυμιτή ακολουθία η οποία είναι ταξινομημένη λόγω του τρόπου με τον οποίο κάναμε merge τις ακολουθίες.

Σχόλια:

i) Αφού το πλήθος είναι δύναμη του 2 με κάθε υποδιαίρεση σε ζευγάρια, παντα θα υπάρχουν διαθέσιμα ζευγάρια για όλες τις ακολουθίες. (πχ άμα το πρόβλημα αναφερόταν γενικά σε άρτιους θα υπήρχε πρόβλημα πχ στο 6 θα έμενα 3 υποσύνολα οπότε δεν θα μπορούσαν να δημιουργηθόυν ακριβώς ζευγάρια και θα ήθελε διαφορετική αντιμετώπιση).

ii) Το merge του κάθε ζευγαριού γίνεται βηματικά ώστε να διατηρηθεί η αρχική ταξινόμηση των ακολουθιών. Γίνεται με δύο διαφορετικούς δείκτες έναν για κάθε ακολουθία. Αφού οι ακολουθίες ειναι απο πριν ταξινομημένες αρκέι να συγκρίνουμε την τιμή στην οποία δείχνει ο ένας δείκτης με την τιμή που δείχνει ο άλλος και στη συνέχεια να τοποθετήσουμε την μικρότερη τιμή (ή μεγαλύτερη ανάλογα με το είδος της ταξινόμησης) στην τελική ακολουθία με μόνο τον δείκτη του στοιχείου που επιλέχθηκε να μετακινείται. Για ζευγάρι ακολουθιών μεγέθους **n** η διαδικασία αυτή απαιτεί **n+n = 2n** βήματα. Στην τελική, στο worst case scenario θα απαιτηθούν **kn** βήματα.

Συνεπώς η τελική πολυπλοκότητα υπολογίζεται ως εξής:

Στην πράξη, αφού ξεκινάμε από **k** πίνακες και σε κάθε αναδρομή/επίπεδο το πλήθος τους μειώνεται στο μισό, σχηματίζεται μία μορφή «δέντρου» με **logk** επίπεδα. Όπως είδαμε και παραπάνω, το sorted merge έχει μέγιστο κόστος **nk**. Οπότε τελικά η πολυπλοκότητα του αλγορίθμου αυτού θα είναι

$T_2(n) = O(nk \log k)$

3. Ως πρώτο βήμα θα χρειαστεί να ταξινομήσουμε τα διαστήματα (χωρίς βλάβη γενικότητας) αύξοντα, με κριτήριο την αρχή τους **(a_i)** με κόστος **O(nlogn)** άμα για παράδειγμα χρησιμοποιήσουμε **merge sort**.
Μετά την ταξινόμηση παρατηρούμε ότι τα σύνολα στά ¨δεξιά¨ ενός συνόλου είναι όλα υποψήφια υποσύνολα. Στο σημείο αυτό θα χρειαστεί να χρησιμοποιήσουμε ως κριτήριο το τέλος του κάθε διαστήματος **(b_j).**
Εφαρμόζοντας στα **(b_j)** την γνωστή από το μάθημα μέθοδο διαίρει και βασίλευε για την **μέτρηση αντιστρόφων** στα στοιχεία b_i οδηγούμαστε σε σωστό αποτέλεσμα με συνολικό κόστος **O(nlogn).**

Ο προαναφερθής αλγόριθμος για την μέτρηση αντιστρόφων λειτουργεί περιληπτικά ως εξής: Διαιρέι τα σύνολά σε δύο ίσα τμήματα, υπολογίζει αναδρομικά τις αντιστροφές σε αυτα, μετρά τις αντιστροφές με a_i, b_j σε διαφορετικά τμήματα και στην συνέχεια επιστρέφει το άθροισμα των τριών.
(Ο αλγόριθμος περιγράφεται με λεπτομέρεια στις διαφάνειες του μαθήματος του κυρίου Εμίρη 10dc.pdf διαφάνειες 24 με 30)

Ο αλγόριθμος που πρότεινα για το πρόβλημα αυτό συνολικά, δίνει σωστό αποτέλεσμα διότι έχοντας ταξινομήσει τα διαστήματα ως προς το ένα άκρο, στην πράξη μετρώντας τις αντιστροφές που απαιτούνται για να γίνει ταξινομημένος, μετράει για κάθε στοιχείο πόσους μικρότερους όρους/τέλη-διανυσματος έχει μετά από αυτό – «δεξιά» του και συνεπώς πόσα υποσύνολα έχει το κάθε ένα από τα σύνολα.

4. Η βέλτιστη λύση που μπόρεσα να σκεφτώ βασίζεται σε ένα άθροισμα που είχαμε διδαχτεί πέρσι στα διακριτά μαθηματικά: $\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$.
Η λογική μου έχει ως εξής: Με χρήση του παραπάνω τύπου μπορουμε εύκολα να υπολογίσουμε το συνολικό άθροισμα όλων των όρων από το 0 μέχρι το n. Έχοντας το άθροισμα αυτό, μπορούμε από αυτό να αφαιρέσουμε το συνολικό άθροισμα των όρων που βρίσκονται στον πίνακα καταλήγουμε στον αριθμό που λείπει. Για να βρούμε το άθροισμα των στοιχείων του πίνακα θα χρησιμοποιήσουμε έναν απλό αλγόριθμο τύπου divide and conquer ο οποίος να σπάει σε κάθε αναδρομή τον πίνακα στη μέση, να υπολογίζει αναδρομικά τα αθροίσματα στους δύο υποπίνακες και τελικά να τα προσθέτει για το ολικό σύνολο. (Όταν φτάνει σε πίνακα μεγέθους 1 θα επιστρέφει απλά την τιμή). Έτσι ο υπολογισμός του αθροίσματος του πίνακα θα γίνει σε χρόνο O(logn), οπότε η συνολική πολυπλοκότητα θα είναι **O(logn).**

5. α) A(x): Βρίσκει δείκτη j : x <= j <= n ώστε να μεγιστοποιείται το $\sum_{k=x}^{j} A[k]$

Θα χρειαστούμε μια μεταβλητή να κρατάει το συνολικό αθροισμα των ορων απο την αρχή μέχρι το σημείο ελέγχου(T), μια μεταβλητή για να κρατάμε το μεγαλύτερο εως τώρα άθροισμα(M) και άλλη μια για τον ανάλογο δείκτη (M_i) (αρχικοποίημενες στην τιμή του πίνακα στο x και σε x αντίστοιχα).

Ξεκινόντας από την επόμενη θέση μετά το στοιχείο της θέσης x του πίνακα, διατρέχουμε τον πίνακα σειριακά αθρόιζοντας σε κάθε βήμα το έως τώρα άθροισμα όλων των όρων με τον ανάλογο «επόμενο όρο». Αν το νέο άθροισμα T είναι μεγαλύτερο από το τωρινό μέγιστο άθροισμα M, κρατάμε στο M την τιμή του T και τον ανάλογο δείκτη στο M_i αλλιώς η τιμή M και ο δείκτης της δεν αλλαζουν. Συνεχίζουμε έτσι μέχρι να φτάσουμε στο τέλος του πίνακα οπότε και επιστρέφουμε τον δείκτη M_i.

Ο συγκεκριμένος αλγόριθμος έχει γραμμική πολυπλοκότητα O(n) αφού διατρέχει αναγκαστικά τον πίνακα μέχρι το τέλος του.

Με τον ίδιο τρόπο μπορούμε να σχεδιάσουμε και έναν ανάλογο αλγόριθμο

Δ(x): Βρίσκει δείκτη i : 1 <= i <= x ώστε να μεγιστοποιείται το $\sum_{k=i}^{x} A[k]$

Ο οποίος να λειτουργεί με τον ίδιο τρόπο, με μόνη διαφορά οτι θα ξεκινά από το τέλος του πίνακα και θα τον διασχιζει αναποδα δηλαδη προς την αρχη.

β) Με χρήση Ωμής βίας μπορούμε απλά να υπολογίσουμε το A(x) για κάθε x από την αρχή μέχρι το τέλος του πίνακα. (Δηλαδή για x=0 : A(0), x=1 : A(1) …) και να κρατήσουμε το μέγιστο A(x) το οποίο θα έχει και την τιμή του μέγιστου αθροίσματος με τους ζητουμενους δείκτες να είναι αυτος που χρησιμοποιήθηκε ως παράμετρος και ο δείκτης που επέστρεψε η Α. (Ο αλγόριθμος που αναγραφω παραπάνω για την Α μπορει εύκολα να επιστρέφει και την τιμή του μέγιστου αθροίσματος). Η τελική πολυπλοκότητα του αλγορίθμου αυτού είναι $O(n^2)$ αφου η διάσχηση του πίνακα καλεί n φορές την Α με κόστος n.

γ) Χρειαζόμαστε έναν αλγόριθμο Διαίρει και βασίλευε που να επιλύει το ίδιο πρόβλημα. Αρχικά χρειαζόμαστε ένα κριτήριο σύμφωνα με το οπόιο να διαιρούμε τον πίνακα σε κομμάτια που να είναι διαχειρίσιμα για εφαρμογή αναδρομής. Η αρχική ιδέα που είχα, την οποία χρησιμοποίησα και τελικά είναι να διαιρούμε με κάθε αναδρομή τον πίνακα στα δύο. (ακέραια διαίρεση)

Με τον διαχωρισμό αυτό, το επιθυμητό ΜΜΑ θα μπορεί να είναι πλέον σε μια από τις ακόλουθες κατηγορίες: 1) τα Μερικά Αθροίσματα (ΜΑ) «αριστερά» του σημείου διαχωρισμού του πίνακα, 2)τα ΜΑ «δεξιά» του και 3) τα ΜΑ τα οπόια έχουν τα άκρα τους και στα δύο άκρα.

Συνεπώς για να βρούμε τον συνολικό ΜΜΑ αρκεί να συγκρίνουμε σε κάθε αναδρομή τους τρείς ανάλογους ΜΑ και ο μεγαλύτερος έιναι η απάντηση.

Τα αριστερά και δεξιά ΜΑ υπολογίζονται με χρήση αναδρομής στο ανάλογο κομματι του πίνακα (2 * $T(\frac{n}{2})$) .

Για το «μεσαίο» ΜΑ μπορούμε να χρησιμοποιήσουμε τους αλγορίθμους Α και Δ από το ερώτημα α. Με τον Α να επιστρέφει το μέγιστο j δεξια του μεσου και τον καθως και το άθροισμά του και όμοια ο Δ να επιστρέφει τον μέγιστο i αριστερά του και το άθροισμά του. Με την πρόσθεση των δυο αυτών τιμών βρίσκουμε τον ΜΑ των ακολουθιών που διαπερνούν το μέσο του πίνακα. (Θ(n) κλήση Α, Δ για τους δύο υποπίνακες με μεγεθος $\frac{n}{2}$ => Θ(n)).

Η αναδρομή τερματίζει όταν ο πίνακας που θα της δωθέι θα έχει πλέον μέσα μόνο ένα στοιχειο, οπότε και θα επιστρέψει την τιμή του.

Η παραπάνω μέθοδος παρατηρούμε ότι ακολουθέι παρόμοια μεθοδολογία-βήματα με την merge sort. Οπότε μπορούμε να πούμε ότι η πολυπλοκότητά της είναι τελικά η επιθυμιτή **O(nlogn).** (Προκύπτει και με τις γνωστές μεθόδους βηματικά $T(n) = 2\,T(\frac{n}{2}) + n$)