Αλγόριθμοι και Πολυπολοκότητα
1η σειρά Ασκήσεων
Βασίλης Βαρσαμής-el18033

Άσκηση 1

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

Στο i-οστό επίπεδο του δέντρου (η ρίζα είναι στο 0) κάνουμε $cn^2 log(\frac{n}{2^i})$ δουλειά.

Η δουλειά σε κάθε επίπεδο είναι $O(n^2 log n)$ και έχουμε $log n$ επίπεδα οπότε: $T(n) = O(n^2 log^2 n)$

Το άθροισμα των $\frac{log n}{2}$ πρώτων επιπέδων είναι $\Omega(n^2 log^2 n)$ διότι κάθε όρος είναι $\Omega(n^2 log n)$. Οπότε όλων των επιπέδων είναι και αυτό $\Omega(n^2 log^2 n)$.

Συνεπώς $T(n) = \Theta(n^2 log^2 n)$

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

$\frac{n^{\log 5}}{n^2 \log n} = \frac{n^{0.32}}{log n} > n^{0.3}$, οπότε από Master Theorem $T(n) = \Theta(n^{2.32})$

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

Στο i-οστο επίπεδο κάνουμε $(\frac{3}{4})^i cn$. Αθροιστικά για κάθε επίπεδο έχουμε:

$$cn(1 + 3/4 + 9/16 + \ldots + (\frac{3}{4})^{log n}) < 4cn$$

(η αντίστοιχη άπειρη σειρά συγκλίνει στο 4)

Οπότε $T(n) = O(n)$, επίσης είναι προφανές ότι $T(n) = \Omega(n)$, άρα $T(n) = \Theta(n)$.

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

Ακριβώς ισοδύναμη με αυτή της mergesort: $T(n) = \Theta(n log n)$

Κάθε επίπεδο κάνει cn, έχουμε logn επίπεδα.

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

Αν m το μήκος της δυαδικής αναπαράστασης του n, τότε m = logn, και το $n^{1/2}$ "μεταφράζεται" σε m/2.

Επομένως η παραπάνω είναι ισοδύναμη με την:

$T(m) = T(m/2) + \Theta(m)$, η οποία θα δώσει $T(m) = \Theta(m)$

Επομένως $T(n) = \Theta(logn)$

Διαφορετικά αν πάμε με το δέντρο έχουμε:

$clogn(1 + 1/2 + 1/4 + \dots) < 2clogn.$

Άρα $T(n) = \Theta(logn)$.

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

Το $\sqrt{n}$ είναι πολυωνυμικά μεγαλύτερο από το $n^{log_4 1} = 1$, οπότε από το Master Theorem $T(n) = \Theta(\sqrt{n})$

Άσκηση 2

A. Η ιδέα του αλγόριθμου είναι ότι σε κάθε επανάληψη θα βρίσκω το i - οστό μέγιστο στοιχείο στον υποπίνακα από 1 έως n-i και το πολύ με 2 προθεματικές περιστροφές θα το τοποθετώ στη σωστή θέση(αν είναι στην πρώτη θέση θέλω 1, αν είναι στην τελευταία δεν θέλω καμία, σε κάθε άλλη περίπτωση θέλω 2).Αυτός ο αλγόριθμος θα κάνει το πολύ 2n βήματα. Παρατηρούμε ωστόσο ότι αν στον υποπίνακα μας έχουν μείνει δύο στοιχεία, έστω $a_1 \kappa\alpha\iota\ \alpha_2$ τότε θέλουμε το πολύ μία προθεματική περιστροφή(καμία αν $a_1 \leq a_2$ και μια αν $a_1 > a_2$).Συνεπώς η αναδρομική σχέση για τον αλγόριθμο στην χειρότερη περίπτωση είναι η παρακάτω:

$$a_n = a_{n-1} + 2, \qquad a_2 = 1$$

όπου a_n το πλήθος των περιστροφών.

Λύνοντας την παίρνουμε $a_n = 2n - 3$

Επομένως έχουμε το πολύ 2n - 3 προθεματικές περιστροφές.

Β. Η ιδέα είναι παρόμοια με την παραπάνω με κάποιες αλλαγές: βρίσκουμε το μέγιστο κατα απόλυτη τιμή στοιχείο στον υποπίνακα. Με την πρώτη προθεματική αναστροφή θα έρθει στην αρχή, εδώ όμως πρέπει να ελέγξουμε το πρόσημο του στοιχείου: αν είναι αρνητικό τότε με άλλη μία προθεματική αναστροφή έχει πάει στη σωστή θέση με το σωστό πρόσημο, αλλιώς εκτελούμε μια τετριμμένη προθεματική αναστροφή μίας θέσης για να του αλλάξουμε πρόσημο και έπειτα άλλη μια για να πάει στη σωστή θέση.
Στη χειρότερη περίπτωση έχουμε 3 προθεματικές αναστροφές ανά στοιχείο, άρα το πολύ 3n για όλον τον πίνακα.

Γ.

1. Υποθέτουμε ότι ο ενδιάμεσος πίνακας A_t δεν έχει κάποιο συμβατό ζεύγος, διότι αν έχει μπορούμε να το αντικαταστήσουμε ως μια δομική μονάδα που αποτελείται από τα δύο αυτά στοιχεία και με πρόσημο ίδιο με το κοινό πρόσημο των στοιχείων.Με λίγα λόγια αν υπάρχει ήδη συμβατό ζεύγος δεν θα υπάρχει στο μέλλον κάποια περιστροφή που θα αφορά μόνο το 1 από τα 2 στοιχεία του ζεύγους.

Βρίσκουμε το μεγαλύτερο θετικό στοιχείο του A_t και διακρίνουμε περιπτώσεις:

α) Αν max = n, τότε με δύο προθεματικές αναστροφές έχουμε φτιάξει το καταχρηστικό συμβατό ζεύγος της εκφώνησης
β) Αν το max είναι διαφορετικό του n ,τότε το -(max + 1) θα είναι αρνητικό και μάλιστα θα ισχύει ότι |max + 1| > max.Τότε μπορούμε ένα από τα συμβατά ζεύγη (max,|max+1|) και (-(max+1), -max) με 2 προθεματικές περιστροφές.

2.1) Αν το -(max + 1) προηγείται του max τότε:

….,-(max+1), ……, max -> -max, …… ,|max+1| -> …….., max, |max + 1|

2.2) Αν το max προηγείται του -(max + 1) τότε:

….,max,……..,-(max+1) -> |max + 1|,…., -max -> ……..,-(max+1),-max

γ) Αν όλα τα στοιχεία στον πίνακα είναι αρνητικά τότε επειδή από εκφώνηση δεν θα έχουμε το [-1,-2,-3,…,-n], τότε σε οποιαδήποτε διαφορετική μετάθεση θα υπάρχουν

στοιχεία -(x+1) και -x με το -(x+1) να προηγείται του -x.Όμως τότε με δύο περιστροφές φτιάχνω το συμβατό ζεύγος (-(x+1),x):
…..,-(x+1),….,-x -> (x+1),….,-x -> ……..,-(x+1),-x

2. 1) Απαλείφουμε αναδρομικά όλα τα συμβατά ζεύγη και τα αντικαθιστούμε με έναν αριθμό (του σωστού προσήμου). Τώρα έχουμε ρίξει το πρόβλημα στην ταξινόμηση m < n στοίχειωναν m = 0 ο πίνακας είναι ταξινομημένος.

2) Αν ο πίνακας που μας έμεινε είναι ο [-1, -2,-3,…,-m]
Για m φορές:
2.1) Κάνε μια προθεματική περιστροφή όλων των στοιχείων.
2.2) Κάνε μια προθεματική περιστορφή των πρώτων m-1 στοιχείων.
Τελικά είναι ταξινομημένος

3) Αλλιώς φτιάξε ένα νέο συμβατό ζεύγος (το οποίο όπως αποδείξαμε γίνεται) και πήγαινε στο βήμα 1.

Ο αλγόριθμο θα εκτελέσει το πολύ 2n προθεματικές προσημασμένες περιστροφές διότι την τελευταία φορά θα εκτελέσει 2m περιστροφές και μέχρι εκείνη την στιγμή (την στιγμή που εισέρχεται στο βήμα 2) άλλες 2(n-m). Συνολικά έχουμε το πολύ 2n προσημασμένες προθεματικές περιστροφές.

Άσκηση 3

Φτιάχνουμε ένα Cartesian tree το οποίο είναι max heap στις τιμές και BST στους δείκτες σε O(n) χρόνο.
Διατρέχουμε όλο το Cartesian tree και αν ήμαστε στη ρίζα του δέντρου τυπώνουμε 0.
Αν είμαστε στο αριστερό παιδί βάζουμε την ίδια τιμή με αυτή του πατέρα, ενώ αν είμαστε στο δεξί παιδί βάζουμε τον δείκτη του πατέρα (αυτό θα γίνει σε O(n)).
Αν $p[i]$ ο δείκτης του πατέρα του παιδιού με δείκτη i, τότε αν το i είναι αριστερό παιδί : $A[i] = A[p[i]]$, ενώ αν είναι δεξί παιδί: $A[i] = p[i]$.
Στην περίπτωση του δεξιού παιδιού, ο πατέρας του είναι σίγουρα μεγαλύτερος του (αφού είναι max heap) και είναι και ο μεγαλύτερος δείκτης που ισχύει αυτό γιατί αν δεν ήταν τότε λόγω του ότι είναι BST, ο πιο μεγάλος δείκτης θα παρεμβαλλόταν ανάμεσα στο $p[i]$ και το i.
Στην περίπτωση του αριστερού παιδιού έχουμε δείκτη μικρότερο από τον πατέρα και επίσης και μικρότερη τιμή, επομένως $A[i] = A[p[i]]$, διότι ο p[i] έχει επισκεφθεί πρώτος και εφόσον η τιμή του παιδιού είναι μικρότερη από αυτή του πατέρα, ο κοντινότερος δείκτης στον πατέρα ταυτίζεται με αυτόν του παιδιού.

Τελικά έχουμε χρόνο O(n).

Άσκηση 4

Μετατρέπουμε το δοθέν στο αντίστοιχο πρόβλημα απόφασης: Γίνεται να το κάνουμε με s υποδοχές φόρτισης; Επειδή το s είναι στο [1,n] και επειδή αν γίνεται να το κάνουμε με s τότε γίνεται για κάθε άλλο s' > s ενώ αν δεν γίνεται με s τότε δεν γίνεται για κανένα s' < s τότε μπορούμε για τον προσδιορισμό του s* να κάνουμε binary search: Παίρνουμε το μέσο το πίνακα [1,n] και αν το πρόβλημα απόφασης επιστρέψει ΝΑΙ κάνουμε αναδρομικά το ίδιο για το αριστερό μισό του πίνακα, αλλιώς αν επιστρέψει ΟΧΙ κάνουμε αναδρομικά το ίδιο για το δεξιό μισό του πίνακα.Αυτό έχει πολυπλοκότητα O(logn).
Πως λύνουμε το πρόβλημα απόφασης:
Για να ισχύει αυτό που διατείνεται η εκφώνηση πρέπει $t_{εισοδου} - t_{αφιξης} \leq d - 1.$ (1)
Ο αλγόριθμος έχει ως εξής:

1) Αρχικοποίησε το $t_{εισοοδου} = a_1 - 1.$
2) Αύξησε το $t_{εισοοδου}$ κατά 1 και πάρε την επόμενη s-άδα.
3) Έλεγξε αν ισχύει η συνθήκη (1) για όλα τα στοιχεία της s-αδας.
4) Αν ναι πήγαινε στο βήμα 2,αλλιώς σταμάτησε και απαντά όχι.
5) Απάντα ναι.

Σημείωση: η τελευταία s-αδα μπορεί να μην είναι και ολόκληρη, αλλά αυτό δεν έχει ουσιαστική επιρροή στον αλγόριθμο, απλώς πρέπει να προστεθεί ως συνθήκη για να μην βγούμε από τα όρια.Η πολυπλοκότητα του παραπάνω είναι O(n).
Συνολικά έχουμε πολυπλοκότητα O(nlogn) γιατί κάνουμε τον παραπάνω αλγόριθμο για O(logn) υποψήφια s.

Άσκηση 5

1. Η ιδέα του αλγόριθμου είναι ίδια με αυτή της binary search: Υπολογίζουμε το $F_S(M/2)$, αν το $k > F_S(M/2)$, κάνουμε το ίδιο αναδρομικά για το "δεξί κομμάτι" δηλαδή για τιμές απο M/2 έως M, αλλιώς αν $k \leq F_S(M/2)$ τότε κάνουμε το ίδιο αναδρομικά για το "αριστερό κομμάτι". Έχει σημασία να παίρνουμε την ισότητα στο "αριστερό κομμάτι" διότι αυτό εξασφαλίζει ότι δεν θα επιστρέψουμε κάποιο στοιχείο το οποίο δεν εμφανίζεται στο S.Πιο αναλυτικά, αν ένα στοιχείο x δεν εμφανίζεται στο S τότε θα έχει ακριβώς ίδια συνάρτηση F με το x-1. Πηγαίνοντας στην ισότητα αριστερά, εξασφαλίζουμε ότι θα βρούμε το πιο αριστερό στοιχείο που είναι ίσο με k, το οποίο εφόσον είναι το πιο αριστερό είναι σίγουρο ότι υπάρχει στο S. Σημειώνουμε ότι μπορούμε να κάνουμε binary

search γιατί η συνάρτηση F είναι αύξουσα (όχι απαραίτητα γνησίως).Το παραπάνω έχει πολυπλοκότητα O(logM).

2. Βρήκαμε αλγόριθμο ο οποίος αν έχουμε έτοιμη την κατανομή F_S σε O(logM). Πως όμως υπολογίζουμε την F_s ;
Αν ο πίνακας A[1…n] είναι ταξινομημένος τότε μπορούμε εύκολα σε χρόνο O(n) να υπολογίσουμε την F_s με τον παρακάτω αλγόριθμο:

1) Αρχικοποίησε δύο δείκτες που αρχικά και οι δύο δείχνουν στο πρώτο στοιχείο του πίνακα.Θα καλούμε τον έναν $left$και τον άλλον $right$.Επίσης αρχικοποίησε το τελικό αποτέλεσμα σε 0.
2) Μετακίνησε τον δείκτη $right$ δεξιά μέχρι να παραβιαστεί η συνθήκη $A[right] - A[left] \leq k$, ο right δείχνει δηλαδή το πρώτο στοιχείο που παραβιάζει την συνθήκη, τότε πρόσθεσε στο τελικό αποτέλεσμα το $right - left - 1$. Εδώ πρέπει να προσέξουμε ότι αν το $right$ γίνει κάποια στιγμή n, για να μην βγούμε από τα όρια του πίνακα επιστρέφουμε και$n - right$ δεν τον μετακινούμε άλλο.
3) Μετακίνησε τον $left$ μια θέση δεξιά, χωρίς να πειράξεις το $right$ και πήγαινε στο βήμα 2.Επαναλαμβάνουμε μέχρι το left να φτάσει στο τέλος του πίνακα.

Ο αλγόριθμος αυτός έχει πολυπλοκότητα O(n) γιατί ο $right$ δεν πρόκειται να περάσει δεύτερη φορά από το ίδιο στοιχείο.
Μια σύντομη αιτιολόγηση για την ορθότητα του παραπάνω αλγορίθμου είναι ότι επειδή ο πίνακας είναι ταξινομημένος, αν το $right$ είναι σε μια συγκεκριμένη θέση, και το $left$ έχει προχωρήσει μία θέση δεξιά (δείχνει σε μεγαλύτερο στοιχείο απ'οτι πριν) τότε όλα τα στοιχεία ανάμεσα στο $left$ και το $right$ ικανοποιούν την συνθήκη(γιατί την ικανοποιούσαν και πριν που το $A[left]$ ήταν μικρότερο) και δεν χρειάζεται να τα ξαναελέγξουμε.

Ο τελικός αλγόριθμος είναι ο εξής:

1) Ταξινόμησε τον πίνακα A[1…n]
2) Εκτέλεσε τον αλγόριθμο του ερωτήματος (1) υπολογίζοντας την F κάθε φορά με τον παραπάνω αλγόριθμο.

Η πολυπλοκότητα αυτού είναι: O(nlogn) για την ταξινόμηση, O(nlogM) για το βήμα (2) γιατί έχουμε O(logM) κλήσεις της F και κάθε υπολογισμός της θέλει O(n) χρόνο. Αν τα M,n είναι πολυωνυμικά συσχετισμένα τότε οι πολυπλοκότητες είναι ισοδύναμες οπότε έχουμε O(nlogM). Σημειώνουμε ότι εφόσον ο A αποτελείται από θετικούς ακέραιους, τότε μπορούμε να βελτιώσουμε τον χρόνο της ταξινόμησης σε O(n)(με radix sort).Τελικά πάλι θα έχουμε O(nlogM).

Άσκηση 7

(α) Αν επιλέξω στην τύχη κάποιο bit στην i-οστή θέση των δυαδικών συμβολοσειρών μεγέθους n, τότε το $h(x)$ είναι το η τιμή του i-οστού bit στην πρώτη συμβολοσειρά και αντίστοιχα το $h(y)$ για την δεύτερη συμβολοσειρά.Συνεπώς:
$Prob[h(x) = h(y)] = 1 - Prob[h(x) \neq h(y)]$.
Επειδή όμως $d(x,y) \leq d_1$ αυτό σημαίνει ότι έχουμε το πολύ d_1 διαφορετικά bits.
Επομένως $Prob[h(x) \neq h(y)] \leq \frac{d_1}{n}$, άρα $Prob[h(x) = h(y)] \geq 1 - \frac{d_1}{n}$
Αντίστοχα αφού $d(x,y) \geq d_2$ είναι $Prob[h(x) \neq h(y)] \geq \frac{d_2}{n}$ και
$Prob[h(x) = h(y)] \leq 1 - \frac{d_2}{n}$.
Παρατηρούμε ότι το παραπάνω ισχύει όποιο bit των δύο συμβολοσειρών διαλέξουμε,δηλαδή για κάθε i.

Οπότε μια $(d_1, d_2, 1 - d_1/n, 1 - d_2/n)$-ευαίσθητη οικογένεια hash functions είναι η

$H = \{h(x) = (x/2^i) \bmod 2 \ , i = 1,2,..., \lfloor \log n \rfloor\}$ η οποία στην ουσία απομονώνει το i-οστο bit.

(γ) Παρατηρούμε ότι η ευαίσθητη οικογένεια που μας ζητείται μοιάζει πολύ με αυτή του ερωτήματος (α). Έτσι ψάχνουμε τρόπο να βρούμε μια συνάρτηση που να αντιστοιχεί μοναδικά ένα σύνολο σε μια συμβολοσειρά (η συνάρτηση πρέπει να ναι 1-1 και επί). Ο τρόπος είναι ο εξής, αν το μέγιστο στοιχείο που έχουν τα δύο sets είναι M και υποθέσουμε ότι τα σύνολα δεν είναι multi-sets και περιέχουν θετικούς ακεραίους τότε φτιάχνουμε ένα M-bit μεγέθους δυαδικό αριθμό ο οποίος έχει στην i-οστή θέση 0 αν το $i \notin S$ και 1 αλλιώς. Παίρνουμε στην τύχη ένα bit, το ίδιο από κάθε αναπαράσταση, η πιθανότητα να έχουν ίδια τιμή είναι ακριβώς το Jaccard similarity δηλαδή 1- d(x,y).
Συνεπώς $Prob[h(x) = h(y)] = 1 - d(x,y) \geq 1 - d_1$
Και $Prob[h(x) = h(y)] = 1 - d(x,y) \leq 1 - d_2$.

Παρατηρούμε ότι η υπόθεση περί θετικών ακέραιων μπορεί να απορριφθεί, στην πραγματικότητα μας αρκεί ένα range απο το μικρότερο και των δύο συνόλων έως το μεγαλύτερο των δύο συνόλων.Αν m το μικρότερο και M το μεγαλύτερο τότε ο δυαδικός αριθμός θα έχει μήκος M - m + 1 bits.