Φροντιστήριο 11 – Σκελετοί Λύσεων

Άσκηση 1

Θα επεκτείνουμε τη δομή δεδομένων *AVL-δένδρο* έτσι ώστε να λαμβάνει υπόψη τον αριθμό επαναλήψεων στοιχείων ως εξής. Σε κάθε θέση του δένδρου φυλάγουμε ζεύγη της μορφής (k, n) όπου k είναι το αποθηκευμένο κλειδί και n είναι το πλήθος των επαναλήψεων του κλειδιού k μέσα στο δένδρο. Οι πράξεις AVL-δένδρου επηρεάζονται ως εξής:

`Insert(k, T)`: Αναζήτησε στο δένδρο το κλειδί k. Αν υπάρχει σε ζεύγος (k, n) ανανέωσε την πληροφορία αυτή έτσι ώστε να λάβει υπόψη τη νέα εμφάνιση του k, δηλαδή (k, n) := (k, n + 1). Διαφορετικά, αν το k δεν υπάρχει στο δένδρο, εφάρμοσε τη γνωστή διαδικασία εισαγωγής σε AVL-δένδρο για το ζεύγος (k,1).

`InOrder(T)`: Τύπωσε τα στοιχεία του δένδρο σε αύξουσα σειρά με τη διαφορά ότι για κάθε κόμβο με κλειδί (k,n) τυπώνουμε το στοιχείο k, n φορές.

Προφανώς η διαδικασία `Insert` εξακολουθεί να διατηρεί χρόνο εκτέλεσης της τάξης O(lg n) όπου n είναι ο αριθμός των κόμβων του δένδρου, ενώ η διαδικασία `InOrder` εξαρτάται και από την πολλαπλότητα των στοιχείων του δένδρου.

Υποθέτοντας υλοποίηση της δομής με όνομα `MultiAVL` ορίζουμε τον πιο κάτω αλγόριθμο.

```
int A[n];
MultiAVL B;

MakeEmpty(B);
for (i = 0; i < n; i++)
   Insert(A[i], B);

InOrder(B);
```

Ο αλγόριθμος αυτός τοποθετεί τα στοιχεία του πίνακα σε ένα `MultiAVL`-δένδρο μέσω n διαδοχικών κλήσεων της `Insert` και στη συνέχεια τα τυπώνει σε αύξουσα σειρά μέσω κλήσης της `InOrder`.

Παρατηρούμε ότι αφού ο πίνακας A περιέχει μόνο lg n διαφορετικά στοιχεία, ο αριθμός κόμβων του δένδρου B αφού δημιουργηθεί θα είναι lg n. Επομένως ο χρόνος εκτέλεσης της Insert θα είναι της τάξης O(lg lg n) και της InOrder O(n). Έτσι, ο χρόνος εκτέλεσης του πιο πάνω αλγόριθμου ταξινόμησης είναι
O(n lg lg n + n) = O(n lg lg n), όπως ζητείται από την άσκηση.

Άσκηση 2

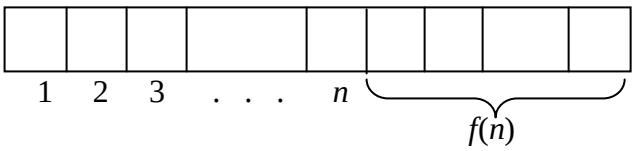
Ας υποθέσουμε ότι ο καθηγητής έχει δίκαιο. Θεωρήστε τον πιο κάτω αλγόριθμο ο οποίος υποθέτει υλοποίηση της δομής του καθηγητή με όνομα `TurboHeap`:

```
int A[n];
TurboHeap B;

for (i = 0; i < n; i++)
   Insert(B, A[i]);
for (i = 0; i < n; i++)
   x = FindMax(B);
   A[i] = x;
   DeleteMax(B);
```

Παρατηρούμε ότι, υποθέτοντας πως οι Insert, FindMax και DeleteMax έχουν χρόνο εκτέλεσης O(1), ο αλγόριθμος αυτός επιτυγχάνει ταξινόμηση του πίνακα A (σε φθίνουσα σειρά) σε χρόνο O(n). Αυτό όμως έρχεται σε αντίφαση προς το γνωστό θεώρημα που δηλώνει ότι ο χρόνος εκτέλεσης οποιουδήποτε σειριακού αλγόριθμου ταξινόμησης είναι Ω(n lg n). Επομένως ο καθηγητής έχει λάθος.

Άσκηση 3



Γνωρίζουμε ότι τα πρώτα n στοιχεία ενός πίνακα είναι ταξινομημένα, ενώ τα υπόλοιπα $f(n)$ είναι άσχετα με τα n πρώτα και βρίσκονται σε τυχαία σειρά. Στις πιο κάτω περιπτώσεις προτείνονται οι πιο κάτω μέθοδοι για ταξινόμηση του ολικού πίνακα.

(a) Αν $f(n) \in O(1)$, εισήγαγε κάθε ένα από τα τελευταία $f(n)$ στοιχεία στη σωστή θέση στον αρχικό πίνακα (χρησιμοποιώντας την βασική ιδέα του InsertionSort). Για την εισαγωγή κάθε στοιχείου, απαιτούνται το πολύ n πράξεις. Επομένως συνολικά ο χρόνος εκτέλεσης του αλγορίθμου απαιτεί $f(n) \times n$ πράξεις, δηλαδή, είναι της τάξης $O(n)$.

(b) Αν $f(n) \in O(\lg n)$, ταξινόμησε τα τελευταία $f(n)$ στοιχεία (με το Mergesort ή το Heapsort σε χρόνο $O(\lg n \cdot (\lg \lg n))$ και, στη συνέχεια, συγχώνευσε τα δύο κομμάτια σε χρόνο $O(n + \lg n)$).
Γνωρίζουμε ότι οποιαδήποτε δύναμη λογαρίθμου έχει σαν άνω φράγμα της την συνάρτηση n, επομένως ο αλγόριθμος αυτός είναι της τάξης $O(n)$.

(c) Αν $f(n) \in O(\sqrt{n})$, ταξινόμησε τα τελευταία $f(n)$ στοιχεία (με το Mergesort ή το Heapsort σε χρόνο $O(\sqrt{n} \cdot (\lg \sqrt{n}))$ και, στη συνέχεια, συγχώνευσε τα δύο κομμάτια σε χρόνο $O(n+\sqrt{n})=O(n)$.
Αφού $\lg \sqrt{n} \le \sqrt{n}$, $\sqrt{n} \cdot \lg \sqrt{n} \le n$ και ο αλγόριθμος είναι της τάξης $O(n)$.