ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ – ΕΡΓΑΣΙΑ 2 (ΑΝΑΦΟΡΑ ΠΑΡΑΔΟΣΗΣ)

Μέρος Α

Για το πρώτο μέρος της εργασίας χρησιμοποιήθηκε η μέθοδος ταξινόμησης HeapSort. Η HeapSort είναι ένας αποτελεσματικός αλγόριθμος ταξινόμησης που βασίζεται σε δυαδικό σωρό (binary heap). Ο δυαδικός σωρός που χρησιμοποιείται στο HeapSort είναι ένας «σωρός μέγιστου» (max heap) ή «σωρός ελάχιστου» (min heap).

Η διαδικασία ταξινόμησης HeapSort περιλαμβάνει τα εξής βήματα:

1) **Κατασκευή του σωρού:** Αρχικά, δημιουργούμε έναν σωρό από τα δεδομένα που θέλουμε να ταξινομήσουμε. Αυτό επιτυγχάνεται ανταλλάσοντας σταδιακά στοιχεία έτσι ώστε κάθε υποδέντρο να είναι ένας σωρός
2) **Ταξινόμηση:** Αφού κατασκευαστεί ο σωρός, το μεγαλύτερο (σε περίπτωση σωρού μέγιστου) ή το μικρότερο (σε περίπτωση σωρού ελάχιστου) στοιχείο βρίσκεται στην κορυφή. Αυτό το στοιχείο αφαιρείται από τον σωρό και τοποθετείται στο τέλος του πίνακα.
3) **Επανάληψη:** Επαναλαμβάνεται η διαδικασία ταξινόμησης για τον υπολοιπόμενο σωρό, χωρίς το πρόσφατο τοποθετημένο στοιχείο. Αυτή η διαδικασία επαναλαμβάνεται μέχρι ο σωρός να αδειάσει.

Η πολυπλοκότητα του HeapSort είναι O(nlogn) για όλες τις περιπτώσεις, όπου n είναι ο αριθμός των στοιχείων που πρέπει να ταξινομηθούν. Αυτό καθιστά το HeapSort αποδοτικό για μεγάλα σύνολα δεδομένων.

Μέρος Β

Για το Μέρος Β, η υλοποίηση της **“remove”** στον κώδικα της κλάσης **“PQ.java”** βασίζεται στην εξής ιδέα:

1) Βρίσκουμε την θέση του στοιχείου προς αφαίρεση χρησιμοποιώντας τον πίνακα **“index”**.
2) Αν το στοιχείο δεν υπάρχει στο heap (η θέση στον πίνακα **“index”** είναι 0), τότε επιστρέφουμε **null**.
3) Διαφορετικά, αντικαθιστούμε το στοιχείο που θέλουμε να αφαιρέσουμε με το τελευταίο στοιχείο του heap.
4) Καθορίζουμε τον νέο χώρο του στοιχείου που μετακινήθηκε στην θέση του αφαιρεθέντος.

5) Χρησιμοποιούμε την συνάρτηση **“sink”** για να διατηρήσουμε τη σωστή δομή του heap.
6) Επιστρέφουμε το αφαιρεθέν στοιχείο.

Όσον αφορά την πολυπλοκότητα:

Η πολυπλοκότητα του προγράμματος εξαρτάται από τον τρόπο που υλοποιήθηκε η **PQ**.

Αναλυτικότερα:

- Οι συναρτήσεις **size, isEmpty, min** έχουν πολυπλοκότητα O(1), διότι ο χρόνος εκτέλεσής τους δεν εξαρτάται από το πόσα στοιχεία υπάρχουν στην δομή.
- Οι συναρτήσεις **insert, getMin, remove** έχουν πολυπλοκότητα O(logn). Συγκεκριμένα, οι **insert** και **remove** περιλαμβάνουν προσθήκη/αφαίρεση στοιχείου στην/από τον σωρό και στη συνέχεια καλούν τις μεθόδους **swim** και **sink** αντίστοιχα, οι οποίες εχουν πολυπλοκότητα O(logn), διότι περιλαμβάνουν σύγκριση και ανταλλαγή στοιχείων με τα παιδιά τους μέχρι να αποκατασταθεί η ιδιότητα του σωρού. Γι’ αυτό και οι αρχικές συναρτήσεις αποκτούν πολυπλοκότητα O(logn). Με παρόμοιο τρόπο αιτιολογείται και η πολυπλοκότητα της **getMin**, καθώς και εκείνη χρησιμοποιεί την μέθοδο **sink.**
- Η συνάρτηση **resize** έχει πολυπλοκότητα O(n), καθώς περιλαμβάνει την δημιουργία ενός νέου σωρού με διπλάσιο μέγεθος και την αντιγραφή των στοιχείων απο τον παλιό σωρό στον νέο. Αυτή η λειτουργία έχει γραμμική πολυπλοκότητα χρόνου O(n), όπου n είναι ο αριθμός των στοιχείων στον σωρό.

Μέρος Γ

Αναφορικά για το Μέρος Γ, η υλοποίηση είναι η εξής:

1) Διαβάζουμε το αρχείο και δημιουργούμε ένα αντικείμενο **“City”** για κάθε γραμμή.
2) Χρησιμοποιούμε έναν **Comparator** που βασίζεται στα κρούσματα για τον προσδιορισμό των top k πόλεων.
3) Χρησιμοποιούμε την **PQ** για την διατήρηση των top k πόλεων, δηλαδή κατά την ανάγνωση κάθε γραμμής, εισάγουμε την πόλη στην **PQ**
4) Εκτυπώνουμε τις top k πόλεις κάθε 5 γραμμές

Η συνολική πολυπλοκότητα του προγράμματος είναι εξαρτημένη από τον αριθμό των πόλεων και τον αριθμό των top k πόλεων που θέλουμε να διατηρήσουμε. Εάν το k είναι πολύ μικρό σε σχέση με τον συνολικό αριθμό των πόλεων, τότε το κόστος θα είναι σχετικά χαμηλό, με αποτέλεσμα να έχει συμφέρον η συγκεκριμένη υλοποίηση σε τέτοια περίπτωση.

Εν γένει, η πολυπλοκότητα εξαρτάται από την αναλογία του k στον συνολικό αριθμό των πόλεων και την απόδοση της υλοποίησης της PQ.

ΣΥΝΤΑΚΤΗΣ ΕΡΓΑΣΙΑΣ – ΑΝΑΦΟΡΑΣ ΠΑΡΑΔΟΣΗΣ: ΧΑΡΑΛΑΜΠΙΔΗΣ ΝΑΠΟΛΕΩΝ - 3220225