ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Δομές Δεδομένων - Εργασία 2

Τμήμα Πληροφορικής

Φθινοπωρινό Εξάμηνο 2020-2021

Διδάσκων: Ε. Μαρκάκης

Ταξινόμηση και Ουρές Προτεραιότητας

Σκοπός της 2ης εργασίας είναι η εξοικείωση με τους αλγορίθμους ταξινόμησης και με τη χρήση ουράς προτεραιότητας, μία από τις πιο βασικές δομές δεδομένων στην σχεδίαση αλγορίθμων. Η εργασία αφορά ερωτήματα που, μεταξύ άλλων, συναντώνται σε απλά recommendation systems, όπως συστήματα που προτείνουν ταινίες ή σειρές σε συνδρομητές με βάση κάποιο σκορ για το πόσο ταιριαστή είναι μια ταινία σε έναν χρήστη. Εδώ θα δούμε μια άλλη εφαρμογή, όπου όμως και πάλι το αλγοριθμικό πρόβλημα που θέλουμε να λύσουμε είναι το ίδιο.

Έστω ότι σας δίνεται η ημερήσια αναφορά με τον αριθμό κρουσμάτων covid στις πόλεις της Ελλάδας ή και άλλων χωρών, για κάποια συγκεκριμένη ημερομηνία αναφοράς. Σκοπός είναι, δεδομένης μιας παραμέτρου k, να βρούμε τις k πόλεις με την μεγαλύτερη πυκνότητα κρουσμάτων, συγκεκριμένα θέλουμε τις πόλεις που αντιστοιχούν στους k υψηλότερους αριθμούς κρουσμάτων ανά 50,000 κατοίκους.

Μέρος Α [30 μονάδες]: Μια πρώτη υλοποίηση. Για αρχή, πρέπει να δούμε τι ΑΤΔ χρειαζόμαστε. Θα ορίσετε μια κλάση με όνομα City, που υλοποιεί το παρακάτω interface.

```
public interface CityInterface {
public int getID();
public String getName();
public int getPopulation();
public int getCovidCases();
public void setID(int ID);
public void setName(String name);
public void setPopulation(int population);
public void setCovidCases(int CovidCases);
}
```

Η ερμηνεία των παραπάνω μεθόδων είναι ως εξής:

- Η getID επιστρέφει το id της κάθε πόλης. Ο αριθμός αυτός είναι μοναδικός για κάθε πόλη. Για απλότητα, θεωρούμε ότι το id παίρνει τιμές από το 1 ως το 999.

- Η getName επιστρέφει το όνομα της πόλης. Θα θεωρήσουμε ότι το όνομα δεν υπερβαίνει τους 50 χαρακτήρες (μαζί με τυχόν κενά, π.χ. «Palma de Mallorca»). Επίσης, είναι δυνατόν 2 πόλεις να έχουν ακριβώς το ίδιο όνομα (π.χ. υπάρχει Paris, France, και Paris, Texas, όπως και Athens, Greece, και Athens, Georgia). Επομένως μόνο το ID είναι σίγουρα μοναδικό για κάθε πόλη.
- Η getPopulation επιστρέφει τον πληθυσμό της πόλης (θεωρούμε ότι είναι θετικός και δεν υπερβαίνει τα 10,000,000), και η getCovidCases επιστρέφει τον αριθμό ημερησίων κρουσμάτων. Ο αριθμός κρουσμάτων δεν μπορεί να είναι μεγαλύτερος του πληθυσμού.
- Αντίστοιχα οι setters μέθοδοι απλά δίνουν τιμές στις αντίστοιχες ποσότητες.

Υλοποιήστε ένα πρόγραμμα το οποίο θα διαβάζει από ένα αρχείο την ημερήσια αναφορά κρουσμάτων, θα ζητάει από τον χρήστη την παράμετρο k, και θα τυπώνει τις πόλεις με τους k υψηλότερους αριθμούς κρουσμάτων ανά 50,000 κατοίκους. Το πρόγραμμα θα ονομάζεται Covid_k.java.

Οδηγίες για το Μέρος Α:

Δεδομένα εισόδου: Η είσοδος θα είναι ένα txt αρχείο, το όνομα του οποίου θα δίνεται από τον χρήστη, και το περιεχόμενό του θα είναι όπως στο παρακάτω παράδειγμα:

```
23 Manchester 510746 1200
31 Amsterdam 821752 678
58 Athens 3218218 3504
124 Karditsa 56747 78
6 Arta 43166 5
```

Το format αυτό έχει την εξής ερμηνεία: σε κάθε γραμμή, το πρώτο πεδίο είναι το id της πόλης, στη συνέχεια ακολουθεί το όνομα, μετέπειτα ο πληθυσμός, και τέλος ο συνολικός αριθμός κρουσμάτων για την ημερομηνία αναφοράς. Εδώ π.χ. η πόλη του Manchester, έχει id ίσο με 23, έχει πληθυσμό 510,746 κατοίκους και εμφάνισε 1200 κρούσματα στην ημερομηνία αναφοράς.

Εκτός από το αρχείο, μέρος της εισόδου θα είναι και η παράμετρος k, η οποία συνήθως θα είναι μια σχετικά μικρή σταθερά σε σχέση με τον αριθμό των πόλεων (σίγουρα μικρότερη από τον συνολικό αριθμό πόλεων). Αν το k είναι μεγαλύτερο από τον συνολικό αριθμό πόλεων του αρχείου, θα τυπώνετε μήνυμα λάθους και το πρόγραμμα θα τερματίζει.

Επίλυση μεσω ταξινόμησης: Για το Μέρος Α, η υλοποίηση θα πρέπει να γίνει χρησιμοποιώντας κάποιον αλγόριθμο ταξινόμησης. Δηλαδή θα ταξινομήσετε τις πόλεις και μετά θα επιλέξετε τις k πόλεις με τον υψηλότερο αριθμό ανά 50,000 κατοίκους. Ο αλγόριθμος που θα χρησιμοποιηθεί θα είναι είτε η Heapsort είτε η Quicksort. **Απαγορεύεται να χρησιμοποιήσετε έτοιμες μεθόδους ταξινόμησης που παρέχονται από την Java.** Θα πρέπει να υλοποιήσετε την δική σας μέθοδο, είτε με χρήση πίνακα είτε με χρήση λίστας.

- Αν ο ΑΜ σας λήγει σε άρτιο αριθμό, θα πρέπει να υλοποιήσετε την Heapsort.
- Αν λήγει σε περιττό, θα υλοποιήσετε την Quicksort.
- Αν είστε σε ομάδα και λήγουν και οι 2 ΑΜ σε άρτιο ή και οι 2 σε περιττό, ακολουθείτε τα παραπάνω.
- Σε διαφορετική περίπτωση διαλέξτε εσείς όποια από τις 2 μεθόδους θέλετε.

Για να συγκρίνετε πόλεις μεταξύ τους, θα χρησιμοποιήσετε κατάλληλα τον πληθυσμό και τα ημερήσια κρούσματα για να κάνετε την αναγωγή σε κρούσματα ανά 50,000 κατοίκους. Θα βγει έτσι ένας πραγματικός αριθμός, τον οποίο θα στρογγυλοποιήσετε ώστε να έχει 2 δεκαδικά ψηφία. Είναι βολικό αν θέλετε να έχετε μια μέθοδο calculateDensity που να κάνει τον παραπάνω

υπολογισμό. Σε περίπτωση ισοβαθμίας μεταξύ πόλεων, θεωρήστε πιο υψηλά στην κατάταξη αυτήν που προηγείται αλφαβητικά (π.χ. η πόλη Amsterdam προηγείται της πόλης Athens). Αν τυχόν 2 πόλεις έχουν και το ίδιο όνομα και την ίδια πυκνότητα κρουσμάτων, τότε προηγείται αυτή με το μικρότερο ID. Συνοψίζοντας, η κλάση City, εκτός από το CityInterface, θα πρέπει να υλοποιεί και το interface Comparable<City> (και συνεπώς της συνάρτησης compareTo), έτσι ώστε να μπορέσετε να την χρησιμοποιήσετε για σύγκριση και ταξινόμηση.

Έξοδος: Το πρόγραμμα θα τυπώνει στη σειρά τις k υψηλότερες πόλεις σε πυκνότητα, ξεκινώντας από αυτήν με τον υψηλότερο αριθμό κρουσμάτων ανά 50,000 κατοίκους και συνεχίζοντας. Με k=3, στο παράδειγμα που φαίνεται παραπάνω η εκτύπωση του προγράμματος θα πρέπει να είναι στη μορφή (επιβεβαιώστε το και μόνοι σας):

```
The top k cities are:
Manchester
Karditsa
Athens
```

Μέρος Β [25 μονάδες]: ΑΤΔ ουράς προτεραιότητας. Ένα μειονέκτημα της παραπάνω προτεινόμενης υλοποίησης είναι ότι πρέπει να αποθηκεύσετε πρώτα την σχετική πληροφορία για **όλες** τις πόλεις, για να αποφασίσετε ποιες έχουν την υψηλότερη πυκνότητα κρουσμάτων μέσω ταξινόμησης. Αυτό φαίνεται να μην είναι αποδοτικό σχετικά με την χρήση μνήμης, ειδικά όταν το k είναι μικρό (σκεφτείτε πώς θα λύνατε το πρόβλημα όταν k=1 ή k=2). Εκτός από αυτό το μειονέκτημα, θα θέλαμε επίσης να μπορούμε να λύσουμε μια πιο δυναμική εκδοχή του προβλήματος ως εξής: μετά το διάβασμα κάθε γραμμής του αρχείου θα θέλαμε να ξέρουμε ποιες είναι οι k υψηλότερες πόλεις μέχρι εκείνη τη στιγμή, και να ενημερώνουμε δυναμικά αυτή την πληροφορία χωρίς να χρειάζεται να αποθηκεύσουμε κάπου όλες τις πόλεις του αρχείου (αυτό είναι γενικά σημαντικό σε εφαρμογές με streams από δεδομένα). Μπορεί π.χ. σε real time, αντί για ένα ενιαίο αρχείο με όλες τις πόλεις, τα δεδομένα να έρχονται σε διαφορετικές στιγμές στον ΕΟΔΥ και να θέλουμε να έχουμε σε κάθε χρονική στιγμή τις k υψηλότερες πόλεις με βάση τα υπάρχοντα δεδομένα. Με αυτό τον τρόπο, αν μας δοθεί αργότερα κι ένα επόμενο αρχείο ή ένα stream με επιπλέον πόλεις, θα μπορούμε εύκολα να συνεχίσουμε την επεξεργασία μας από εκεί που σταματήσαμε. Τέλος, ένα άλλο μειονέκτημα του Μέρους Α είναι ότι επειδή στηρίζεται σε αλγόριθμο ταξινόμησης, θα έχετε χρονική πολυπλοκότητα τουλάχιστον O(NlogN) αν έχετε N πόλεις. Όταν όμως η παράμετρος k είναι μικρή σε σχέση με τον αριθμό των πόλεων, αξίζει να δούμε πώς θα μπορούσαμε να έχουμε μια καλύτερη υλοποίηση.

Για να αντιμετωπίσουμε όλα τα παραπάνω, θα χρειαστείτε μια ουρά προτεραιότητας, όπου θα αποθηκεύετε αντικείμενα τύπου City. Η υλοποίηση της ουράς θα γίνει με χρήση μεγιστοστρεφούς σωρού, όπως έχουμε δει στο μάθημα και στο εργαστήριο. Μπορείτε να βασιστείτε στην ουρά προτεραιότητας του Εργαστηρίου 6 ή σε ό,τι είδαμε στην Ενότητα 11 των διαφανειών, με κατάλληλες τροποποιήσεις, ή μπορείτε να φτιάξετε εξ' ολοκλήρου την δική σας ουρά. Η ουρά θα πρέπει να διαθέτει, εκτός από τις λειτουργίες *insert* και *getmax*, που είδαμε και στο μάθημα, και κάποιες επιπλέον μεθόδους, που περιγράφονται παρακάτω (μπορείτε να έχετε κι άλλες βοηθητικές μεθόδους στην υλοποίησή σας):

- `boolean isEmpty()`: έλεγχος για το αν είναι άδεια η ουρά.
- `int size()`: επιστρέφει τον αριθμό ενεργών στοιχείων στην ουρά.
- `void insert(City x)`: εισαγωγή αντικειμένου. Η εισαγωγή θα πρέπει να καλεί και μια μέθοδο resize που θα κάνει διπλασιασμό του πίνακα όταν η ουρά έχει γεμίσει κατά το 75%.
- `City max()`: επιστρέφει το στοιχείο με τη μέγιστη προτεραιότητα χωρίς να το αφαιρεί από την ουρά.
- `City getmax()`: Αφαιρεί και επιστρέφει το αντικείμενο με τη μέγιστη προτεραιότητα.

- `City remove(int id)`: Αφαιρεί και επιστρέφει την πόλη με το συγκεκριμένο id. Πρέπει μετά την αφαίρεση να αποκατασταθεί η ιδιότητα του σωρού για να μην καταστραφεί η ουρά.

Η πολυπλοκότητα που πρέπει να έχει η κάθε μέθοδος, σε μια ουρά με n αντικείμενα δίνεται στον παρακάτω πίνακα.

Μέθοδος	Πολυπλοκότητα
`size, isEmpty`	O(1)
`insert`	O(logn) (*δειτε σχόλια*)
`max`	O(1)
`getmax`	O(logn)
`remove`	O(logn) (*δειτε σχόλια*)

Σχόλια:

- Η υλοποίηση της ουράς προτεραιότητας θα πρέπει να είναι στο αρχείο PQ.java.
- Για την insert, η απαίτηση για O(logn) είναι μόνο όταν δεν χρειάζεται να καλέσει τη resize για την επέκταση του πίνακα.
- Η υλοποίηση της remove αναδεικνύει το trade-off που υπάρχει εδώ μεταξύ χώρου και χρόνου. Με όσα έχουμε πει για τις ουρές προτεραιότητας, χωρίς καμία προσθήκη βοηθητικής μνήμης, υπάρχει ένας απλοϊκός τρόπος να κάνετε την remove σε O(n). Για να υλοποιήσετε την remove σε O(logn), χρειάζεται να κάνετε κάποιες προσθήκες στα χαρακτηριστικά της ουράς και στον τρόπο λειτουργίας των insert και getmax. Για τον σκοπό αυτό, επιτρέπεται να χρησιμοποιήσετε βοηθητική μνήμη, στην μορφή 1 απλού πίνακα ακεραίων. Δεν υπάρχει αυστηρό άνω όριο στο μέγεθος του πίνακα (αλλά μην χρησιμοποιήσετε όλη την μνήμη του υπολογιστή σας). Σκεφτείτε την ελάχιστη δυνατή πληροφορία που χρειάζεστε. Επίσης, οι τροποποιήσεις που θα κάνετε δεν θα πρέπει να επιβαρύνουν τις insert και getmax παραπάνω από έναν O(1) παράγοντα, ώστε να παραμένει λογαριθμική η πολυπλοκότητά τους.
- Αν δυσκολεύεστε να κάνετε την remove σε O(logn), μπορείτε να την κάνετε σε O(n), χωρίς την χρήση έξτρα μνήμης, για τις μισές μονάδες από αυτές που αντιστοιχούν στην remove (θα σας κοστίσει πολύ λίγο στον τελικό βαθμό της εργασίας).
- Για τους πιο εξοικειωμένους με τις δομές της Java: Υπάρχει τρόπος για να κάνουμε την remove σε O(logn), όπου η έξτρα μνήμη μπορεί να περιοριστεί περαιτέρω σε χώρο O(n), για ουρά με n ενεργά αντικείμενα. Αυτό όμως απαιτεί την χρήση πιο σύνθετων δομών όπως π.χ. τα δέντρα κόκκινου-μαύρου, που δεν έχουμε καλύψει ακόμα. Όποιοι θέλετε μπορείτε να το κάνετε και με αυτό τον τρόπο είτε χρησιμοποιώντας αντικείμενα τύπου TreeMap (που αποτελούν υλοποίηση δέντρων κόκκινου-μαύρου) ή φτιάχνοντας μόνοι σας από την αρχή ό,τι χρειάζεστε. **Προσοχή:** Δεν θα βαθμολογηθείτε για αυτό, είναι υποχρεωτικό να μας παραδώσετε την υλοποίηση με χρήση πίνακα. Μπορούμε όμως να ελέγξουμε την ορθότητα του κώδικά σας, για να γνωρίζετε ότι το κάνατε σωστά. Αν κάνετε και $2^{η}$ υλοποίηση βάλτε την σε ξεχωριστό αρχείο με όνομα PQ2.java.

Μέρος Γ [35 μονάδες]. Αυτό είναι το πιο ενδιαφέρον κομμάτι της εργασίας. Χρησιμοποιώντας την ουρά, θέλουμε τώρα να διαβάζουμε το αρχείο εισόδου γραμμή προς γραμμή, και μετά την ανάγνωση κάθε γραμμής, να διατηρούμε τις k πόλεις με την υψηλότερη πυκνότητα κρουσμάτων μέχρι εκείνη τη στιγμή. Το πρόγραμμα και πάλι θα τυπώνει μόνο τις k υψηλότερες πόλεις στο τέλος της ανάγνωσης όλου του αρχείου όπως και στο Μέρος Α (για debugging όμως και έλεγχο ορθότητας, είναι καλό να τυπώνετε μετά από κάθε 5 γραμμές του αρχείου εισόδου, τις k πόλεις με την υψηλότερη πυκνότητα κρουσμάτων). Για το πρόγραμμά σας, επιτρέπεται να χρησιμοποιήσετε μια ουρά προτεραιότητας **που θα περιέχει καθ'όλη την διάρκεια του προγράμματος το πολύ k ενεργά αντικείμενα**. Δεν επιτρέπεται να χρησιμοποιήσετε δομή που να αποθηκεύει όλες τις πόλεις του αρχείου εισόδου (θεωρούμε πάντα

ότι το k είναι μικρότερο από τις γραμμές του αρχείου). Τονίζουμε ότι το μέγεθος του πίνακα της ουράς μπορεί να είναι παραπάνω από k λόγω της resize στις εισαγωγές, αυτό που θέλουμε όμως είναι το πλήθος των ενεργών στοιχείων στην ουρά να μην είναι ποτέ πάνω απο k (μπορείτε αν θέλετε να αρχικοποιήσετε την ουρά με μέγεθος 2k για να μην χρειαστεί να γίνει ποτέ η resize).
Η απαίτηση για την πολυπλοκότητα του προγράμματος είναι ότι δεν πρέπει να είναι χειρότερη από την πολυπλοκότητα του Μέρους Α, και όταν k=O(1) θα πρέπει σίγουρα να είναι καλύτερη.

Hint: Η ουρά που θα χρησιμοποιήσετε θα βασίζεται σε μεγιστοστρεφή σωρό, δηλαδή η getmax φέρνει πάντα το στοιχείο με την μέγιστη προτεραιότητα. Αν ακολουθήσετε το υπόδειγμα του Εργαστηρίου 6, σκεφτείτε πως θα ορίσετε την προτεραιότητα μιας πόλης για τις ανάγκες του συγκεκριμένου προβλήματος και κατασκευάστε τον κατάλληλο Comparator (με χρήση της compareTo της κλάσης City) για να κάνετε συγκρίσεις στην ουρά προτεραιότητας.

Πρόσθετες οδηγίες υλοποίησης για τα μέρη Α, Β και Γ:

- Το πρόγραμμα για το μέρος Γ **πρέπει να λέγεται** DynamicCovid_k_withPQ.java.
- Αν θέλετε, μπορείτε να υλοποιήσετε την ουρά προτεραιότητας με generics.
- Για το διάβασμα του αρχείου εισόδου, μπορείτε να χρησιμοποιήσετε έτοιμες μεθόδους της Java για άνοιγμα αρχείων και για να διαχωρίσετε τα δεδομένα κάθε γραμμής. Επιπλέον, μπορείτε να χρησιμοποιήσετε κώδικα από τα εργαστήρια ή την $1^{η}$ εργασία σας: συνδεδεμένη λίστα (μονή ή διπλή), ουρές, κτλ. **Δεν επιτρέπεται να χρησιμοποιήσετε έτοιμες υλοποιήσεις δομών τύπου λίστας, στοίβας, ουράς, από την βιβλιοθήκη της Java** (π.χ. Vector, ArrayList, κτλ).
- Δεν χρειάζεται να ασχοληθείτε με ανίχνευση λαθών στο format του αρχείου εισόδου, παρά μόνο σε ότι αφορά τους αριθμητικούς περιορισμούς για το id, τον πληθυσμό και τα κρούσματα. Σε περίπτωση λάθους, το πρόγραμμα τερματίζει με μήνυμα λάθους.

Μέρος Δ [10 μονάδες] Προαιρετικό – Bonus. Έστω ότι διαβάζουμε και πάλι ένα αρχείο εισόδου με το ίδιο format, και μας ενδιαφέρει να απαντάμε σε ερωτήσεις της μορφής: «Πόσο μεγάλη πυκνότητα κρουσμάτων χρειάζεται να έχει μια πόλη για να είναι στο top 50% των πόλεων που εξετάζουμε;» (για να μπει σε κίτρινη ζώνη π.χ.). Αυτό σημαίνει ότι πρέπει να υπολογίζουμε το median από τις πυκνότητες κρουσμάτων που βλέπουμε. Ο median μιας ακολουθίας ακεραίων είναι ο αριθμός που θα βρίσκεται στη μέση της ακολουθίας αν την ταξινομήσουμε σε αυξουσα σειρά. Π.χ. στην ακολουθία 13, 17, 11, 24, 19, 8, 14, ο median είναι ο 14. Αν η ακολουθία έχει άρτιο αριθμό ακεραίων, τότε θα κάνουμε τη σύμβαση να παίρνουμε ως median τον μεγαλύτερο από τους 2 μεσαίους, δηλαδή στην ακολουθία 13, 17, 11, 24, 19, 8, θεωρούμε ως median τον 17.

Θέλουμε να υλοποιήσουμε μια δομή, η οποία να μπορεί να διατηρεί ενημερωμένο τον median δυναμικά, όπως επεξεργάζεται κάθε νέα πόλη από το αρχείο εισόδου. Θέλουμε δηλαδή μετά την ανάγνωση κάθε γραμμής του αρχείου εισόδου, να μπορούμε εύκολα να υπολογίσουμε τον median αριθμό κρουσμάτων ανά 50,000 κατοίκους, με βάση τις πόλεις που έχουμε επεξεργαστεί μέχρι εκείνη την χρονική στιγμή (γενικεύοντας, η δυναμική ενημέρωση του median σε ενα stream από δεδομένα είναι σημαντικό πρόβλημα σε αρκετές άλλες εφαρμογές).
Hint: Χρησιμοποιήστε κατάλληλα 2 ουρές προτεραιότητας.

Οδηγίες υλοποίησης:

- Το πρόγραμμα σας **πρέπει να λέγεται** Dynamic_Median.java.
- Για να μην είναι πολύ μεγάλη η εκτύπωση της εξόδου, απαιτείται να τυπώνετε τον median μόνο μετά από κάθε 5 γραμμές του αρχείου εισόδου. Το πρόγραμμά σας θα πρέπει με την ανάγνωση κάθε γραμμής εισόδου να κάνει κάποια επεξεργασία, αλλά ο

υπολογισμός και η εκτύπωση του median θα γίνεται μόνο όταν (i%5 == 0). Π.χ. στο παράδειγμα που υπάρχει στο Μέρος Α, θα τυπωνόταν μόνο 1 φορά στο τέλος ο median.

- Για απλότητα, δεν θα χρησιμοποιήσουμε αρχεία εισόδου άνω των 500 γραμμών.

Μέρος Ε - Αναφορά παράδοσης [10 μονάδες]. Ετοιμάστε μία σύντομη αναφορά σε pdf αρχείο (μην παραδώσετε Word ή txt αρχεία!) με όνομα project2-report.pdf, στην οποία θα αναφερθείτε στα εξής:

a. Εξηγήστε συνοπτικά ποιον αλγόριθμο ταξινόμησης υλοποιήσατε στο μέρος Α (άνω όριο 1 σελίδα).
b. Για τα Μέρη Β και Γ, σχολιάστε αρχικά πώς υλοποιήσατε την remove του Μέρους Β. Μετέπειτα, εξηγήστε αναλυτικά την ιδέα για την υλοποίηση του προγράμματος DynamicCovid_k_withPQ στο Μέρος Γ. Περιγράψτε τι πολυπλοκότητα αναμένεται να έχει το Μέρος Γ και αν είναι τελικά συμφέρον σε περιπτώσεις όπου το k είναι αρκετά μικρότερο του συνολικού αριθμού πόλεων (άνω όριο 3 σελίδες).
c. Ομοίως για το (προαιρετικό) μέρος Δ. Σχολιάστε την πολυπλοκότητα του προγράμματός σας. Π.χ. σε κάθε γραμμή i, με (i%5 ==0), όπου πρέπει να τυπώσουμε τον median, τι πολυπλοκότητα έχει ο υπολογισμός του median; (άνω όριο 2 σελίδες)
d. Φροντίστε να εξηγήσετε με ποιο τρόπο δίνετε το μονοπάτι για το αρχείο εισόδου, καθώς και το k. Ως παράδειγμα, αν στο project σας ο κώδικας είναι στο φάκελο src, και το αρχείο εισόδου έχει όνομα inputfile.txt και είναι στον φάκελο Data, η εκτέλεση της main του Μέρους Α από τη γραμμή εντολών μέσα στον φάκελο src μπορεί να γίνεται ως εξής:

```
java Covid_k k ../Data/inputfile.txt
```

Το συνολικό μέγεθος της αναφοράς πρέπει να είναι τουλάχιστον 2 σελίδες και το πολύ 6 σελίδες.

Οδηγίες Παράδοσης

Η εργασία σας θα πρέπει να μην έχει συντακτικά λάθη και να μπορεί να μεταγλωττίζεται. Εργασίες που δεν μεταγλωττίζονται χάνουν το **50%** της συνολικής αξίας.

Η εργασία θα αποτελείται από:

1. Τον πηγαίο κώδικα (source code). Τοποθετήστε σε ένα φάκελο με όνομα **src** τα αρχεία java που έχετε φτιάξει. Χρησιμοποιήστε τα ονόματα των κλάσεων όπως δίνονται. Επιπλέον, φροντίστε να συμπεριλάβετε όποια άλλα αρχεία πηγαίου κώδικα φτιάξατε και απαιτούνται για να μεταγλωττίζεται η εργασία σας. Φροντίστε επίσης να προσθέσετε επεξηγηματικά σχόλια όπου κρίνετε απαραίτητο στον κώδικά σας.

2. Την αναφορά παράδοσης

Όλα τα παραπάνω αρχεία θα πρέπει να μπουν σε ένα αρχείο zip. Το όνομα που θα δώσετε στο αρχείο αυτό θα είναι ο αριθμός μητρώου σας πχ. 3130056_3130066.zip ή 3130056.zip (αν δεν είστε σε ομάδα). Στη συνέχεια, θα υποβάλλετε το zip αρχείο σας στην περιοχή του μαθήματος «Εργασίες» στο e-class. Δεν χρειάζεται υποβολή και από τους 2 φοιτητές μιας ομάδας.

Η προθεσμία παράδοσης της εργασίας είναι Παρασκευή, 8 Ιανουαρίου 2021 και ώρα 23:59.