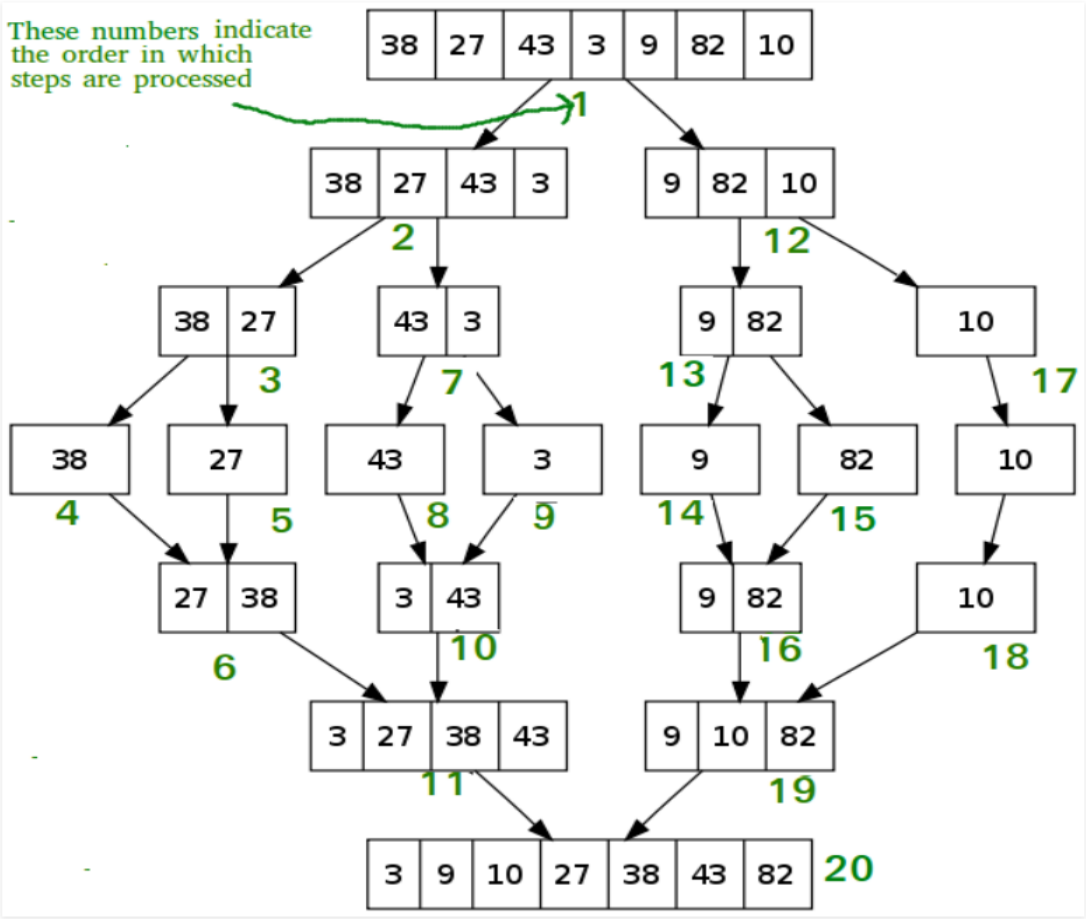
ΜΕΛΟΙ ΟΜΑΔΑΣ:
P3170117 ΤΖΕΝΗ ΜΠΟΛΕΝΑ
P3170122 ΚΩΝΣΤΑΝΤΙΝΟΣ ΝΙΚΟΛΟΥΤΣΟΣ

2η ΕΡΓΑΣΙΑ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

ΜΕΡΟΣ Α

Στο μέρος Α υλοποιήσαμε τον αλγόριθμο Merge Sort ο οποίος ανήκει στην κατηγορία διαίρει και βασίλευε. Ο Merge Sort υλοποιείται αναδρομικά. Κάθε φορά βρίσουμε το μέσο του πίνακα μας και καλούμε την Merge sort για τον κάθε υποπίνακα, η διαδικασία αυτή γίνεται έως ότου μπορούμε να σπάμε τον πίνακα μας στα δυο. Στη συνέχεια καλείται η Merge η οποία συγκρίνει τα στοιχεία των δυο υποπινάκων με την βοήθεια δυο auxiliary arrays, και έτσι ταξινομεί ένα υπομέρος του αρχικού πίνακα. Η merge όμως σε αυτήν την περίπτωση συγκρίνει αντικείμενα και για αυτό καλεί την CompareTo. Και επειδή μερικές φορές το να εξηγείς κάτι γραπτώς δεν είναι η καλύτερη μέθοδος , ας δούμε ένα παράδειγμα εκτέλεσης του αλγόριθμου merge sort για αύξουσα ταξινόμιση με χρήση σχημάτων.

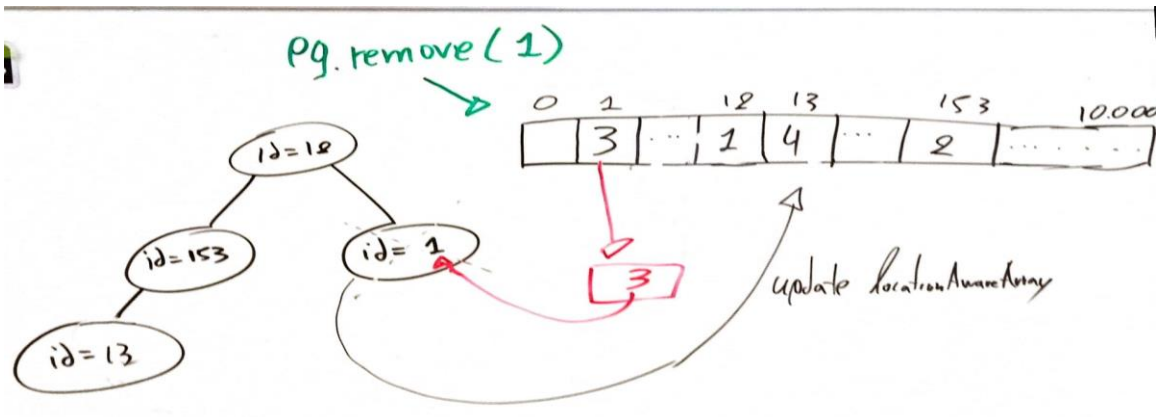


ΜΕΡΟΣ Β

Υλοποίηση της remove:

Σκοπός όταν υλοποιούμε την remove είναι να αφαιρέσουμε ένα στοιχείο από τη σωρό χωρίς όμως να χαθεί η μεγιστοστρεφή ιδιότητα της σωρού . Έχουμε στην κλάση μας έναν πίνακα με integers ο οποίος για κάθε id γνωρίζει σε ποια θέση του heapArray βρίσκεται το τραγούδι που αντιστοιχεί αυτό το id(τα id είναι μοναδικά). Π.χ αν στην θέση 45 του array με integers έχουμε τον αριθμό 10 σημαίνει ότι το τραγούδι με id 45 βρίσκεται στην θέση 10 του heapArray(πίνακς με τα Song). Οπότε αμέσως ξέρουμε ποιο τραγούδι να αφαιρέσουμε. Αφού με το id ξέρουμε το position του συγκεκριμένου song(an δεν υπάρχει song , αυτό το id πετάμε exception) τότε το μόνο που απομένει είναι να το αφαιρέσουμε.

Για να το αφαιρέσουμε αρχικά το αποθηκεύουμε σε μια temp μεταβλητή έτσι ώστε μετά να το κάνουμε return, αλλάζουμε το song αυτό με το song που βρίσκεται στο τέλος του heap και μειώνουμε το πλήθος των ενεργών στοιχείων κατά 1, αφού πρώτα κάνουμε Null το στοιχείο προς αφαίρεση. Πρέπει επίσης να ενημερώσουμε τον πίνακα που κρατάει τις θέσεις που βρίσκονται τα song με κάποιο id. Αφού αφαιρέσαμε το song με το id που είχαμε σαν παράμετρο πάμε στον πίνακα με Integers και κάνουμε το locationAwareArray[id]=0, επειδή πλέον δεν υπάρχει song με αυτό το id. Το στοιχείο που το κάναμε αλλαγή με το στοιχείο προς αφαίρεση τώρα του αλλάξουμε στον πίνακα με integers τη θέση που βρίσκεται στην σωρό . Αφού κάναμε και αυτό το μόνο που απομένει είναι να κάνουμε ένα sink το στοιχείο που φέραμε πάνω έτσι ώστε να διατηρηθεί η μεγιστοστρεφής ιδιότητα της σωρού. Το κάνουμε sink διότι το αντικείμενο αυτό πριν βρισκόταν στα φύλλα της σωρού, οπότε τώρα πρέπει να πάει σε μία θέση ώστε το παιδί της (αν έχει) να έχει μικρότερη προτεραιότητα από αυτό. Η **remove έχει πολυπλοκότητα O(logN).**



ΜΕΡΟΣ Γ

Υλοποίηση Top_k_withPQ:

Σκοπός του προγράμματος αυτού είναι η ουρά μας να έχει το πολύ κ στοιχεία και αυτά τα κ στοιχεία να είναι τα καλύτερα . Για να το κάνουμε αυτό αρχικά ορίζουμε ότι μέγιστη προτεραιότητα έχει το στοιχείο εκείνο που έχει τα λιγότερα Likes και το μεγαλύτερο title. Ο λόγος που το κάνουμε αυτό είναι διότι θέλουμε στην κορυφή της σωρού να βρίσκεται το στοιχείο με ta min likes(εξηγούμε παρακάτω γιατί θέλουμε στην κορυφή να υπάρχει αυτό το song). Τώρα δημιουργούμε στην main ένα αντικείμενο τύπου PQ στο constructor του οποίου περνάμε σαν όρισα εκτός από το κ(πλήθος κ καλύτερων τραγουδιών) και έναν comperator<Song> και υλοποιούμε το interface αυτό μέσω μιας anonymous κλάσης η οποία υλοποιείται αμέσως μετά την δημιουργία του αντικειμένου PQ. Μέσα στην anonymous κλάση υλοποιούμε την compare(Song s1, Song s2) μέθοδο που έχει η comperator. Επειδή θέλουμε μέγιστη προτεραιότητα να έχει το τραγούδι που έχει τα λιγότερα Likes και το μεγαλύτερο title όταν θα γίνει compare δυο τραγουδιών(με τον comperator) θα χρησιμοποιώ την compareTo της song αλλά θα επιστρέφω αντίθετο αποτέλεσμα δλδ, η compare που βρίσκεται στην anonymous κλάση του comperator θα επιστρέφει (-s1.compareTo(s2)).

Αφού τώρα ο comperator μας κάνει τις συγκρίσεις που θέλουμε πάμε να δούμε τι γίνεται με τις εισαγωγές των song , ώστε να έχουμε μια ουρά με κ στοιχεία κα αυτά τα κ στοιχεία να είναι ανά πάσα στιγμή τα κ καλύτερα. Όταν στην ουρά μας έχουμε λιγότερα από κ στοιχεία τότε απλά εισάγουμε κάθε τραγούδι. Ωστόσο το όλο θέμα προκύπτει όταν έχουμε ήδη κ ενεργά στοιχεία στην ουρά μας και βλέπουμε πως ένα άλλο song θέλει να μπεί στην ουρά. Αυτή τη στιγμή στην ουρά υπάρχουν τα κ καλύτερα τραγούδια , και το τραγούδι το οποίο είναι το «χειρότερο» είναι αυτό που βρίσκεται στην κορυφή της ουράς , αφού παραπάνω ορίσαμε ότι μέγιστη προτεραιότητα έχει εκείνο το song που έχει τα λιγότερα Likes και το μεγαλύτερο title. Τώρα για να δούμε αν θα εισάγουμε το νέο στοιχείο στην ουρά και αν θα αφαιρέσουμε αυτό που βρίσκεται στην κορυφή αρκεί να κάνουμε μια απλή σύγκριση του στοιχείου προς εισαγωγή με το στοιχείο στην κορυφή της ουράς. Η σύγκριση θα γίνει με την compareTo της Song, αφού θέλουμε να δούμε αν το στοιχείο προς εισαγωγή έχει περισσότερα likes (αν ίσα Likes, τότε μικρότερο title), από αυτό της κορυφής της ουράς. Αν αυτό ισχύει τότε θα κάνουμε getMax(), δλδ θα αφαιρέσουμε το στοιχείο κορυφής της ουράς, και θα εισάγουμε το νέο song. Η ίδια διαδικασία θα γίνει και για τα επόμενα τραγούδια προς εισαγωγή. Ακολουθώντας την παραπάνω διαδικασία θα έχουμε καταφέρει η ουρά μας να έχει κάθε χρονική στιγμή τα κ καλύτερα τραγούδια.

Η παραπάνω υλοποίηση έχει πολυπλοκότητα O(NlogN) , διότι η insert και getMax και max έχουν πολυπλοκότητα O(logn), και επειδή η διαδικασίες αυτές γίνονται για N songs, συνολική πολυπλοκότητα είναι O(NlogN) . Προφανώς και η υλοποίηση αυτή συμφέρει περισσότερο από την υλοποίηση του Α μέρους (πολυπλοκότητα merge sort O(Nlogn)), διότι για πολύ μικρά κ σε σύγκριση με το σύνολο των τραγουδιών δεσμεύουμε ελάχιστη μνήμη, ενώ στο Α μέρος δεσμεύουμε υποχρεωτικά μνήμη όση το πλήθος των τραγουδιών.

ΜΕΡΟΣ Δ

<u>ΠΟΛΥΠΛΟΚΟΤΗΤΑ ΥΠΟΛΟΓΙΣΜΟΥ MEDIAN</u>: η πολυπλοκότητα του υπολογισμού του median είναι **O(1)** αφού όπως εξηγήθηκε παραπάνω με τον τρόπο που κάνουμε τις εισαγωγές γνωρίζουμε ότι κάθε φορά ο median βρίσκεται στο upperHalfPQ και συγκεκριμένα στην κορυφή του οπότε μέσω της upperHalfPq.max() παίρνουμε το median(πολυπλοκότητα της max() είναι και αυτή O(1)) .

<u>ΠΟΛΥΠΛΟΚΟΤΗΤΑ INSERT</u>:logN, επειδή καλεί την insert της PQ η οποία έχει πολυπλοκότητα logN , διότι κάνει το πολύ sawaps όσο το βάθος του heap, δλδ logN. Η insert του dynamicMedian καλεί μια φορά την balance , η οποία καλεί το getMax που έχει πολυπλοκότητα logN και την insert της PQ που έχει πολυπλοκότητα logN, οπότε η **πολυπλοκότητα της INSERT είναι logN + logN + logN = 3logN άρα O(logN).**

<u>ΣΥΝΟΛΙΚΗ ΠΟΛΥΠΛΟΚΟΤΗΑΤ ΠΡΟΓΡΑΜΜΑΤΟΣ</u>:NlogN επειδή έχουμε N εισαγωγές και η κάθε μια διαρκεί logN , οπότε N*logN , δίνει πολυπλοκότητα O(NlogN).

Στο πρόγραμμα Dynamic_Median με χρήση δύο PQ γνωρίζουμε έπειτα από κάθε είσοδο ενός song τον median. Έχουμε το lowerHalfPQ το οποίο είναι μια μεγισοστρεφή ουρά όπου μέγιστη προτεραιότητα έχει το τραγούδι με τα περισσότερα Like (αν Like ίσα, τότε θεωρούμε ως τραγούδι με μέγιστη προτεραιότητα αυτό που έχει το μικρότερο title). Για να υλοποιήσουμε την compare κάνουμε ίδια διαδικασία με το Γ μέρος, απλώς η διαφορά είναι ότι η compare θα επιστρέψει s1.compareTo(s2). Επιπλέον έχουμε και το upperHalfPQ το οποίο είναι μια μεγισοστρεφή ουρά όπου μέγιστη προτεραιότητα έχει το τραγούδι με τα λιγότερα Like (αν Like ίσα, τότε θεωρούμε ως τραγούδι με μέγιστη προτεραιότητα αυτό που έχει το μεγαλύτερο title). Για να υλοποιήσουμε την compare κάνουμε ακριβώς ίδια διαδικασία με το Γ μέρος(εξήγηση στο Γ μέρος).

Αρχικά έχουμε εισάγει σε κάθε ουρά 1 βοηθητικό song, τα οποία δεν επηρεάζουν τον median , αλλά απλώς μας βοηθάνε στον τρόπο με τον οποίο στη συνέχεια θα κάνουμε Insert των στοιχείων στην κατάλληλη ουρά. Ο τρόπος που υλοποιούμε τις ουρές και κάνουμε τις εισαγωγές μας επιτρέπει να έχουμε το εξής πλεονέκτημα: το song που βρίσκεται στην κορυφή της lowerHalfPQ έχει μικρότερη προτεραιότητα από το song που βρίσκεται στην κορυφή του upperHalfPQ(//σύμφωνα με τον ορισμό προτεραιότητας που μας δίνεται για το median στην εκφώνηση της εργασίας), και αυτό μας επιτρέπει να ξέρουμε πώς ανεξάρτητα από τα μεγέθη των ουρών ο median βρίσκεται στο upperHalfPQ.

Ο τρόπος που εισάγουμε τα song στις ουρές είναι ο εξής(μην ξεχνάμε ότι έχουμε τα δύο help songs):

Για να βρούμε ευκολά τον median θέλουμε να ισχύει πάντα ότι το στοιχείο με τι μέγιστη προτεραιότητα στο lowerHalfPQ ναι έχει μικρότερη προτεραιότητα από το στοιχείο με τη μέγιστη προτεραιότητα στο upperHalfPQ και επιπλέον η διάφορα στο μέγεθος των ουρών όταν πάμε να βρούμε τον median να είναι το πολύ 1.

Ξεκινάμε λοιπόν και εισάγουμε στοιχεία. Αν το song προς εισαγωγή έχει προτεραιότητα μικρότερη από αυτή του lowerHalfPQ το εισάγουμε στο lowerHalfPQ, αλλιώς το

εισάγουμε στο UpperHalfPQ. Στη συνέχεια της insert καλούμε την μέθοδο balance η οποία εξισορροπεί τα μεγέθη των ουρών αν τα size τους έχουν διαφορά μεγαλύτερη του 1(δλδ διαφορά των size ==2). Είπαμε και παραπάνω ότι για να ισχύει ότι ο median βρίσκεται στο upperHalfPQ πρέπει οι ουρές να έχουν το πολύ διαφορά στο πλήθος των στοιχείων 1. Η balance απλώς τσεκάρει αν η διαφορά στο μέγεθος των ουρών είναι μεγαλύτερη του 1, αν ναι τότε από την ουρά με το μεγαλύτερο πλήθος στοιχείων αφαίρει το max(δλδ getMax()) και το βάζει στην ουρά με τα λιγότερα στοιχεία.