ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ - ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2023-2024

PROJECT

Τσάλα Ζαφειρία	1084963	up1084963@ac.upatras.gr	2ο έτος
Γκύλλης Κων/νος	1100526	up1100526@ac.upatras.gr	2ο έτος
Μήλιος Θεόδωρος	1100624	up1100624@ac.upatras.gr	2ο έτος
Τεμπονέρας Γεώργιος	1104775	up1104775@ac.upatras.gr	2ο έτος

PART I – SORTING AND SEARCHING ALGORITHMS

Στο παρόν project , εργαζόμαστε με δεδομένα που τα αντλούμε από το https://www.stats.govt.nz/large-datasets/csv-files-for-download/ . Κατεβάζουμε το ζητούμενο αρχείο το οποίο είναι σε csv μορφή και το μετατρέπουμε σε txt, για να χρησιμοποιήσουμε τις εγγραφές που περιέχει στους κώδικες μας. Δουλεύουμε σε γλώσσα C++.

Το txt μας από το οποίο διαβάζουμε τα δεδομένα μας είναι της μορφής : <Period>,<Birth | Death>,<Region>,<count>

ΕΡΩΤΗΜΑ 1

Στο συγκεκριμένο ερώτημα ταξινομούμε κατά αύξουσα σειρά τις περιοχές , με βάση τις συνολικές τιμές των γεννήσεων με την χρήση των MergeSort και QuickSort αλγορίθμων. Οι μέθοδοι που καλούν τον κάθε sorting αλγόριθμο είναι οι merge & mergesort και quickSort αντίστοιχα και έχουν υλοποιηθεί με βάση τον ψευδοκώδικα που αναλύθηκε τόσο στις διαλέξεις όσο και στο βιβλίο του κ. Τσακαλίδη.

Κάνοντας run τον κώδικα μας , παρατηρούμε πως το sorting γίνεται πολύ πιο γρήγορα όταν χρησιμοποιούμε τον QuickSort παρά τον MergeSort. Παρατίθονται στιγμιότυπα από το runtime.

```
Choose:
1.Exercise1
2.Exercise2
3.Exercise3
4.Exercise4
5.EXIT
1
Choose sorting algorithm: 1 for MergeSort, 2 for QuickSort: 1
MergeSort: 1.4389 ms

Sorted by MergeSort:
Year: 2022, Event Type: Births, Region: "Region not stated or area outside region", Count: 6
Year: 2013, Event Type: Births, Region: "Region not stated or area outside region", Count: 12
Year: 2015, Event Type: Births, Region: "Region not stated or area outside region", Count: 12
Year: 2021, Event Type: Births, Region: "Region not stated or area outside region", Count: 12
Year: 2016, Event Type: Births, Region: "Region not stated or area outside region", Count: 15
Year: 2017, Event Type: Births, Region: "Region not stated or area outside region", Count: 15
Year: 2012, Event Type: Births, Region: "Region not stated or area outside region", Count: 18
Year: 2018, Event Type: Births, Region: "Region not stated or area outs
Year: 2020, Event Type: Births, Region: "Region not stated or area outs
Year: 2014, Event Type: Births, Region: "Region not stated or area outs
Year: 2019, Event Type: Births, Region: "Region not stated or area outs
Year: 2011, Event Type: Births, Region: "Region not stated or area outs
Year: 2005, Event Type: Births, Region: "Region not stated or area outs
Year: 2008, Event Type: Births, Region: "Region not stated or area outs
Year: 2010, Event Type: Births, Region: "Region not stated or area outs
Year: 2009, Event Type: Births, Region: "Region not stated or area outs
Year: 2006, Event Type: Births, Region: "Region not stated or area outs
Year: 2007, Event Type: Births, Region: "Region not stated or area outs
Year: 2022, Event Type: Births, Region: West Coast region, Count: 303
Year: 2020, Event Type: Births, Region: West Coast region, Count: 312
Year: 2016, Event Type: Births, Region: West Coast region, Count: 324
Year: 2018, Event Type: Births, Region: West Coast region, Count: 324
Year: 2021, Event Type: Births, Region: West Coast region, Count: 333
Year: 2005, Event Type: Births, Region: West Coast region, Count: 336
Year: 2019, Event Type: Births, Region: West Coast region, Count: 354
Year: 2017, Event Type: Births, Region: West Coast region, Count: 363
Year: 2014, Event Type: Births, Region: West Coast region, Count: 375
Year: 2015, Event Type: Births, Region: West Coast region, Count: 381
Year: 2013, Event Type: Births, Region: West Coast region, Count: 384
Year: 2006, Event Type: Births, Region: West Coast region, Count: 390
Year: 2007, Event Type: Births, Region: West Coast region, Count: 405
Year: 2014, Event Type: Births, Region: Tasman region, Count: 411
Year: 2012, Event Type: Births, Region: West Coast region, Count: 417
```

```
Choose:
1.Exercise1
2.Exercise2
3.Exercise3
4.Exercise4
5.EXIT
1
Choose sorting algorithm: 1 for MergeSort, 2 for QuickSort: 2
QuickSort: 0.7934 ms

Sorted by QuickSort:
Year: 2022, Event Type: Births, Region: "Region not stated or area outside region", Count: 6
Year: 2021, Event Type: Births, Region: "Region not stated or area outside region", Count: 12
Year: 2015, Event Type: Births, Region: "Region not stated or area outside region", Count: 12
Year: 2013, Event Type: Births, Region: "Region not stated or area outside region", Count: 12
Year: 2017, Event Type: Births, Region: "Region not stated or area outside region", Count: 15
Year: 2016, Event Type: Births, Region: "Region not stated or area outside region", Count: 15
Year: 2020, Event Type: Births, Region: "Region not stated or area outside region", Count: 18
Year: 2012, Event Type: Births, Region: "Region not stated or area outside region", Count: 18
Year: 2018, Event Type: Births, Region: "Region not stated or area outside region", Count: 18
Year: 2019, Event Type: Births, Region: "Region not stated or area outside region", Count: 21
Year: 2014, Event Type: Births, Region: "Region not stated or area outside region", Count: 21
Year: 2011, Event Type: Births, Region: "Region not stated or area outside region", Count: 24
Year: 2005, Event Type: Births, Region: "Region not stated or area outside region", Count: 57
Year: 2008, Event Type: Births, Region: "Region not stated or area outside region", Count: 63
Year: 2010, Event Type: Births, Region: "Region not stated or area outside region", Count: 81
Year: 2009, Event Type: Births, Region: "Region not stated or area outside region", Count: 90
Year: 2006, Event Type: Births, Region: "Region not stated or area outside region", Count: 135
Year: 2007, Event Type: Births, Region: "Region not stated or area outside region", Count: 174
Year: 2022, Event Type: Births, Region: West Coast region, Count: 303
Year: 2020, Event Type: Births, Region: West Coast region, Count: 312
Year: 2016, Event Type: Births, Region: West Coast region, Count: 324
Year: 2018, Event Type: Births, Region: West Coast region, Count: 324
Year: 2021, Event Type: Births, Region: West Coast region, Count: 333
Year: 2005, Event Type: Births, Region: West Coast region, Count: 336
Year: 2019, Event Type: Births, Region: West Coast region, Count: 354
Year: 2017, Event Type: Births, Region: West Coast region, Count: 363
Year: 2014, Event Type: Births, Region: West Coast region, Count: 375
Year: 2015, Event Type: Births, Region: West Coast region, Count: 381
```

MergeSort: ***1.4389 ms***

QuickSort: ***0.7934 ms***

Γενικά, ο Mergesort όπως ήδη γνωρίζουμε έχει βέλτιστη ασυμπτωτική απόδοση στο χειρότερο σενάριο με πολυπλοκότητα $\boldsymbol{O(nlogn)}$**.** Αρχικά χωρίζει την είσοδο σε 2 ισομερή τμήματα , τα ταξινομεί και ύστερα τα συγχωνεύει με αναδρομική κλήση του αλγορίθμου. Οι συγκρίσεις στην χειρότερη περίπτωση που απαιτούνται για να γίνει συγχώνευση είναι $n - 1$ ενώ για την ταξινόμηση μήκους n απαιτεί **το πολύ** $n(logn) + 2^{logn} + 1$ συγκρίσεις.

Ο Quicksort από την άλλη , ανεξάρτητα από τις μεταθέσεις , απαιτεί $n + 1$ συγκρίσεις με μέσο χρόνο συγκρίσεων $\boldsymbol{O(nlogn)}$ (όπως και ο Mergesort). Η διαφορά όμως μεταξύ Quicksort και Mergesort έγκειται στο γεγονός ότι ο Quicksort στο worst case scenario δεν έχει καλή απόδοση (πολυπλοκότητα $O(n^2)$) ενώ ο Mergesort ακόμα και σε αυτό εγγυάται πως έχει $T(n) = O(nlogn)$.

Δεν υπάρχει αντικειμενικά καλύτερος και χειρότερος αλγόριθμος. Όλα εξαρτώνται από το μέγεθος της εισόδου που θέλουμε να ταξινομήσουμε.

- Όταν το input είναι σχετικά μικρό , προτιμούμε τον Quicksort χάρη στην καλή cache απόδοση του.
- Όταν είναι μεγαλύτερο , επιλέγουμε τον Mergesort που εγγυάται πως ακόμα και στο worst case scenario θα έχει πολυπλοκότητα $O(nlogn)$, πιο γρήγορη κατά πολύ από αυτή του Quicksort.

ΕΡΩΤΗΜΑ 2

Σε αυτή την άσκηση ζητείται να ταξινομήσουμε κατά αύξουσα σειρά τις περιοχές , με βάση τις συνολικές τιμές των θανάτων αυτή την φορά , κάνοντας χρήση των HeapSort και CountingSort αλγορίθμων. Καθώς τρέχουμε τον κώδικα μας , παρατηρούμε πως το sorting γίνεται πολύ πιο γρήγορα όταν χρησιμοποιούμε τον Countingsort παρά τον Heapsort.

Από το runtime προκύπτει:

```
Choose:
1.Exercise1
2.Exercise2
3.Exercise3
4.Exercise4
5.EXIT
2
Choose sorting algorithm: 1 for Heap Sort, 2 for Counting sort: 1
HeapSort: 0.9095 ms

Sorted by Heap:
Year: 2018, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 15
Year: 2017, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 21
Year: 2019, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 21
Year: 2012, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 21
Year: 2015, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 24
Year: 2016, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 24
Year: 2013, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 24
Year: 2011, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 27
Year: 2020, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 30
Year: 2022, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 33
Year: 2021, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 39
Year: 2014, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 42
Year: 2010, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 48
Year: 2009, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 66
Year: 2008, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 108
Year: 2005, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 129
Year: 2007, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 132
Year: 2006, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 135
Year: 2006, Event Type: Deaths, Region: West Coast region, Count: 240
Year: 2005, Event Type: Deaths, Region: West Coast region, Count: 243
Year: 2010, Event Type: Deaths, Region: West Coast region, Count: 249
Year: 2012, Event Type: Deaths, Region: West Coast region, Count: 252
Year: 2007, Event Type: Deaths, Region: West Coast region, Count: 255
Year: 2009, Event Type: Deaths, Region: West Coast region, Count: 264
Year: 2016, Event Type: Deaths, Region: West Coast regio
Year: 2008, Event Type: Deaths, Region: West Coast regio
Year: 2020, Event Type: Deaths, Region: West Coast regio
Year: 2014, Event Type: Deaths, Region: West Coast regio
Year: 2013, Event Type: Deaths, Region: West Coast regio
Year: 2018, Event Type: Deaths, Region: West Coast regio
Year: 2011, Event Type: Deaths, Region: West Coast regio
Year: 2015, Event Type: Deaths, Region: West Coast regio
Year: 2008, Event Type: Deaths, Region: Tasman region, C
```

```
Choose:
1.Exercise1
2.Exercise2
3.Exercise3
4.Exercise4
5.EXIT
2
Choose sorting algorithm: 1 for Heap Sort, 2 for Counting sort: 2
CountingSort: 0.3123 ms

Sorted by Counting:
Year: 2018, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 15
Year: 2012, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 21
Year: 2017, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 21
Year: 2019, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 21
Year: 2013, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 24
Year: 2015, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 24
Year: 2016, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 24
Year: 2011, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 27
Year: 2020, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 30
Year: 2022, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 33
Year: 2021, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 39
Year: 2014, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 42
Year: 2010, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 48
Year: 2009, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 66
Year: 2008, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 108
Year: 2005, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 129
Year: 2007, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 132
Year: 2006, Event Type: Deaths, Region: "Region not stated or area outside region", Count: 135
Year: 2006, Event Type: Deaths, Region: West Coast region, Count: 240
Year: 2005, Event Type: Deaths, Region: West Coast region, Count: 243
Year: 2010, Event Type: Deaths, Region: West Coast region, Count: 249
Year: 2012, Event Type: Deaths, Region: West Coast region, Count: 252
Year: 2007, Event Type: Deaths, Region: West Coast region, Count: 255
Year: 2009, Event Type: Deaths, Region: West Coast region, Count: 264
Year: 2016, Event Type: Deaths, Region: West Coast region, Count: 267
Year: 2008, Event Type: Deaths, Region: West Coast region, Count: 273
Year: 2020, Event Type: Deaths, Region: West Coast region, Count: 273
Year: 2014, Event Type: Deaths, Region: West Coast region, Count: 279
Year: 2011, Event Type: Deaths, Region: West Coast region, Count: 291
Year: 2013, Event Type: Deaths, Region: West Coast region, Count: 291
Year: 2018, Event Type: Deaths, Region: West Coast region, Count: 291
```

Heapsort*: **0.9095 ms***

Countingsort*: **0.3123 ms***

(σχεδόν **<u>3x</u>** τον χρόνο του Countingsort

χρειάστηκε ο Heapsort!)

Ο Heapsort αλγόριθμος , όπως περιγράφεται και στο βιβλίο «Δομές Δεδομένων» του κ. Τσακαλίδη , επιλέγει αρχικά το μεγαλύτερο στοιχείο από ένα μη ταξινομημένο σύνολο. Έπειτα το αφαιρεί από το σύνολο και το τοποθετεί ως το αριστερότερο και συνεπώς μικρότερο στοιχείο ενός νέου *ταξινομημένου* συνόλου. Μετά επανοργανώνει το πρώτο σύνολο (η εύρεση του μεγαλύτερου στοιχείου καθίσταται αποδοτικότερη με αυτόν τον τρόπο) και εκτελεί ξανά τον αλγόριθμο από την αρχή. Η έξοδος λαμβάνει την τελική της μορφή όταν αδειάσει το πρώτο σύνολο που εισάγαμε ως input. Ο Heapsort γενικά χρειάζεται $\boldsymbol{O(nlogn)}$ συγκρίσεις ακόμα και στο worst case scenario. Επομένως, η ασυμπτωτική του συμπεριφορά είναι βέλτιστη , μιας και ακόμα και στην χειρότερη περίπτωση εκτελεί $\boldsymbol{\Omega(nlogn)}$ συγκρίσεις.

Ο Countingsort από την άλλη είναι ένας αλγόριθμος που χρησιμοποιείται για την ταξινόμηση **integer** και μόνο αριθμών. Ο CountingSort μετράει πόσοι αριθμοί είναι μεγαλύτεροι από έναν δωσμένο αριθμό x για να μπορέσει να προσδιορίσει πού ακριβώς βρίσκεται το x μέσα στον πίνακα.

Το συνολικό του κόστος είναι $O(n+k)$ όπου k το άνω φράγμα του διαστήματος $[1,k]$. Επομένως το αν ο καλύτερος αλγόριθμος για να χρησιμοποιήσουμε στην ταξινόμηση μας είναι ο Countingsort μάς το απαντάει η τιμή του k.

- Αν $\kappa = O(n)$, τότε $T(n) = O(n+n) = O(2n) => T(n) = O(n)$ (ασυμπτωτικά βέλτιστος). Ομοίως αν έχω $\kappa = O(nloglogn)$.
  Γενικά, αν το k είναι μικρός αριθμός (και ακέραιος) , ο Countingsort είναι ακόμα καλύτερος από όλους τους άλλους αλγόριθμους σύγκρισης που έχουμε μάθει.
- Αν όμως $k = O(n^2)$, τότε ο αλγόριθμος απαιτεί μεγάλο χρόνο για να ταξινομήσει στοιχεία (άρα γίνεται πιο αργός και λιγότερο αποδοτικός για την δουλειά μας). Το ίδιο παρατηρούμε και όταν έχουμε ακόμα μεγαλύτερες τιμές για το k , όπως $O(n^3)$, $O(n^4)$ κοκ.

Όπως και στο ερώτημα 1 έτσι και εδώ η επιλογή του κατάλληλου αλγορίθμου πάλι εξαρτάται από το **μέγεθος και το είδος της εισόδου που θέλουμε να ταξινομήσουμε**.

- Αν έχουμε μέσα στα δεδομένα μας floats, chars κλπ (δεν μας νοιάζει αν το n είναι μεγάλο ή μικρό) , προτιμότερο θα ήταν να διαλέξουμε τον Heapsort, καθώς θα είναι πιο γρήγορος σε κάθε πιθανό σενάριο συγκριτικά με τον Countingsort.
- Όταν όμως έχουμε ένα σύνολο n προς ταξινόμηση με $n <<< k$ και μάλιστα **τα n inputs είναι ακέραιοι αριθμοί** , καλύτερα να επιλέξουμε τον Countingsort.

ΕΡΩΤΗΜΑ 3

Στο εξης ερώτημα ταξινομούμε τις περιοχές με βάση τον αριθμό γεννήσεων , ο οποίος ανήκει σε ένα διάστημα τιμών $[b1, b2]$, όπου $b1, b2$ ακέραιοι αριθμοί. Στο πρόγραμμα μας ζητείται από τον χρήστη να εισάγει τα δύο άκρα του διαστήματος με τους ακέραιους αριθμούς που επιθυμεί και πραγματοποιείται η αναζήτηση των περιοχών που έχουν γεννήσεις που ανήκουν στο διάστημα επιλογής.

Για την αναζήτηση χρησιμοποιούμε 2 μεθόδους, αυτή της **δυαδικής αναζήτησης** (*binary search*) και αυτή της **αναζήτησης με παρεμβολή** (*interpolation search*). Παρακάτω επισυνάπτονται στιγμιότυπα από το runtime της άσκησης:

```
Choose:
1.Exercise1
2.Exercise2
3.Exercise3
4.Exercise4
5.EXIT
3
Choose searching algorithm: 1 for Binary Search, 2 for Interpolation Search: 1
Enter the range of birth counts (b1 and b2): 5 14000
InsertionSort: 8.0261 ms

Regions with birth counts in the range [5, 14000] using Binary Search:
Region: "Region not stated or area outside region", Year: 2022, Count: 6
Region: "Region not stated or area outside region", Year: 2013, Count: 12
Region: "Region not stated or area outside region", Year: 2015, Count: 12
Region: "Region not stated or area outside region", Year: 2021, Count: 12
Region: "Region not stated or area outside region", Year: 2016, Count: 15
Region: "Region not stated or area outside region", Year: 2017, Count: 15
Region: "Region not stated or area outside region", Year: 2018, Count: 15
Region: "Region not stated or area outside region", Year: 2012, Count: 18
Region: "Region not stated or area outside region", Year: 2018, Count: 18
Region: "Region not stated or area outside region", Year: 2020, Count: 18
Region: "Region not stated or area outside region", Year: 2012, Count: 21
Region: "Region not stated or area outside region", Year: 2014, Count: 21
Region: "Region not stated or area outside region", Year: 2017, Count: 21
Region: "Region not stated or area outside region", Year: 2019, Count: 21
Region: "Region not stated or area outside region", Year: 2019, Count: 21
Region: "Region not stated or area outside region", Year: 2011, Count: 24
Region: "Region not stated or area outside region", Year: 2013, Count: 24
Region: "Region not stated or area outside region", Year: 2015, Count: 24
Region: "Region not stated or area outside region", Year: 2016, Count: 24
Region: "Region not stated or area outside region", Year: 2011, Count: 27
Region: "Region not stated or area outside region", Year: 2020, Count: 30
Region: "Region not stated or area outside region", Year: 2022, Count: 33
Region: "Region not stated or area outside region", Year: 2021, Count: 39
Region: "Region not stated or area outside region", Year: 2014, Count: 42
Region: "Region not stated or area outside region", Year: 2010, Count: 48
Region: "Region not stated or area outside region", Year: 2005, Count: 57
Region: "Region not stated or area outside region", Year: 2008, Count: 63
Region: "Region not stated or area outside region", Year: 2009, Count: 66
Region: "Region not stated or area outside region", Year: 2010, Count: 81
Region: "Region not stated or area outside region", Year: 2009, Count: 90
Region: "Region not stated or area outside region", Year: 2008, Count: 108
Region: "Region not stated or area outside region", Year: 2005, Count: 129
```

```
Choose:
1.Exercise1
2.Exercise2
3.Exercise3
4.Exercise4
5.EXIT
3
Choose searching algorithm: 1 for Binary Search, 2 for Interpolation Search: 2
Enter the range of birth counts (b1 and b2): 5 14000
InsertionSort: 7.6639 ms

Regions with birth counts in the range [5, 14000] using Interpolation Search:
Region: "Region not stated or area outside region", Year: 2022, Count: 6
Region: "Region not stated or area outside region", Year: 2013, Count: 12
Region: "Region not stated or area outside region", Year: 2015, Count: 12
Region: "Region not stated or area outside region", Year: 2021, Count: 12
Region: "Region not stated or area outside region", Year: 2016, Count: 15
Region: "Region not stated or area outside region", Year: 2017, Count: 15
Region: "Region not stated or area outside region", Year: 2018, Count: 15
Region: "Region not stated or area outside region", Year: 2012, Count: 18
Region: "Region not stated or area outside region", Year: 2018, Count: 18
Region: "Region not stated or area outside region", Year: 2020, Count: 18
Region: "Region not stated or area outside region", Year: 2012, Count: 21
Region: "Region not stated or area outside region", Year: 2014, Count: 21
Region: "Region not stated or area outside region", Year: 2017, Count: 21
Region: "Region not stated or area outside region", Year: 2019, Count: 21
Region: "Region not stated or area outside region", Year: 2019, Count: 21
Region: "Region not stated or area outside region", Year: 2011, Count: 24
Region: "Region not stated or area outside region", Year: 2013, Count: 24
Region: "Region not stated or area outside region", Year: 2015, Count: 24
Region: "Region not stated or area outside region", Year: 2016, Count: 24
Region: "Region not stated or area outside region", Year: 2011, Count: 27
Region: "Region not stated or area outside region", Year: 2020, Count: 30
Region: "Region not stated or area outside region", Year: 2022, Count: 33
Region: "Region not stated or area outside region", Year: 2021, Count: 39
Region: "Region not stated or area outside region", Year: 2014, Count: 42
Region: "Region not stated or area outside region", Year: 2010, Count: 48
Region: "Region not stated or area outside region", Year: 2005, Count: 57
Region: "Region not stated or area outside region", Year: 2008, Count: 63
Region: "Region not stated or area outside region", Year: 2009, Count: 66
Region: "Region not stated or area outside region", Year: 2010, Count: 81
Region: "Region not stated or area outside region", Year: 2009, Count: 90
```

Binary Search: ***8.0261 ms*** ///////// Interpolation Search: ***7.6639 ms***

Ας αναλύσουμε εκτενέστερα τις μεθόδους που χρησιμοποιήθηκαν παραπάνω.

Αρχικά , το **Binary Search** (δυαδική αναζήτηση) είναι μια μέθοδος «διαίρει και βασίλευε». Χρησιμοποιώντας while loop , χωρίζουμε το **sorted** array σε 2 κομμάτια και συγκρίνουμε συνεχώς με την δωσμένη τιμή μέχρι να την βρούμε ανάμεσα στα 2 άκρα αναζήτησης που εισήγαγε ο χρήστης. Η πολυπλοκότητα του Binary Search είναι $O(logn)$ σε κάθε περίπτωση, τόσο στο average όσο και στο worst case scenario. Γενικά χαρακτηρίζεται ως ένας πολύ efficient τρόπος αναζήτησης, ακόμα και όταν το dataset είναι μεγάλο. Σημαντικό ωστόσο είναι πριν προχωρήσουμε στην δυαδική αναζήτηση να έχουμε κάνει sort το array από πριν (είτε σε αύξουσα είτε σε φθίνουσα σειρά) , διότι ένα unsorted array προκαλεί προβλήματα και ανακριβή αποτελέσματα κατά την εκτέλεση του προγράμματος.

Το **Interpolation Search** (αναζήτηση με παρεμβολή) από την άλλη βρίσκει προσεγγιστικά την θέση του στοιχείου που θέλουμε να αναζητήσουμε, κοιτώντας την ελάχιστη και μέγιστη τιμή του array.

Η αναζήτηση με παρεμβολή είναι προτιμότερη από την δυαδική αναζήτηση ($O_{Interpolation}(loglogn)$), όταν το **dataset είναι ομοιόμορφα κατανεμημένο**, διότι δεν χρειάζεται να κάνει όλες αυτές τις διαδοχικές διαιρέσεις του array για να βρει το key value και να χάσει αρκετό χρόνο (το παρατηρούμε και από τους χρόνους που αναγράφονται στα screenshots). Αν όμως το dataset **δεν είναι ομοιόμορφα κατανεμημένο**, τότε η αναζήτηση με παρεμβολή έχει χειρότερη επίδοση από ο,τι η δυαδική αναζήτηση, με πολυπλοκότητα στο worst case scenario $O_{Interpolation}(n)$ έναντι του $O_{BinSearch}(logn)$.

(Προφανώς $O_{BinSearch}(logn) \gg O_{Interpolation}(n)$)

Συγκεντρωτικός πίνακας για το πότε χρησιμοποιώ την κάθε μέθοδο:

Binary search	Interpolation Search
• Όταν έχω ένα ομοιόμορφα κατανεμημένο και sorted dataset • Όταν το dataset είναι στατικό και δεν αλλάζει. • Όταν θέλω να έχω χαμηλή χρήση μνήμης.	• Όταν έχω ένα μεγάλο, ομοιόμορφα κατανεμημένο και sorted dataset. • Όταν το dataset είναι μεταβαλλόμενο(dynamic). • Όταν το dataset δεν είναι ομοιόμορφα κατανεμημένο αλλά εμφανίζει ομοιομορφία μέχρι έναν βαθμό.

ΕΡΩΤΗΜΑ 4

Σε αυτό το ερώτημα υλοποιούμε ξανά το Ερώτημα 3 , αλλά αυτή την φορά χρησιμοποιούμε την Δυϊκή Αναζήτηση Παρεμβολής (**B**inary **I**nterpolation **S**earch-**BIS**).

Ο μέσος χρόνος αναζήτησης του BIS είναι ο $O(loglogn)$ ενώ στο worst case scenario αποκτά πολυπλοκότητα $O(\sqrt{n})$. Αυτό συμβαίνει, διότι στην πρώτη εκτέλεση του συμβαίνουν το πολύ $\sqrt{n}$ συγκρίσεις, στην επόμενη εκτέλεση $\sqrt{\sqrt{n}}$ κ.ο.κ. Δηλαδή προκύπτει η σχέση $n^{\frac{1}{2}} + n^{\frac{1}{4}} + n^{\frac{1}{8}} + \cdots = O(\sqrt{n})$

Ο χρόνος χειρότερης περίπτωσης ωστόσο μπορεί να βελτιωθεί από $O(\sqrt{n})$ σε $O(logn)$, δίχως να χειροτερεύει ο χρόνος της μέσης περίπτωσης. Αυτό επιτυγχάνεται με την <u>εκθετική αύξηση</u> του i μέσα στο while loop , δηλαδή να έχουμε $i \leftarrow 2 * i$ αντί για $i \leftarrow i + 1$. Έτσι, τα άλματα γίνονται ολοένα μεγαλύτερα, με το αριστερό άκρο να παραμένει διαρκώς σταθερό. Κατά συνέπεια, το τελευταίο υποδιάστημα είναι πολύ μεγαλύτερο από $\sqrt{n}$ και μέσα σε αυτό εφαρμόζουμε δυική αναζήτηση στα στοιχεία (που ανήκουν σε αυτό) που απέχουν κατά $\sqrt{n}$ και έτσι προκύπτει το ζητούμενο υποδιάστημα μεγέθους $\sqrt{n}$.

Ταυτόχρονα ,αν υποθέσουμε πως στο πρώτο βήμα εκτελούμε στην χειρότερη περίπτωση i συγκρίσεις , τότε :

$$2^i\sqrt{n} = n \Rightarrow 2^i = \sqrt{n} \Rightarrow i = \log(\sqrt{n})$$

Άρα, για το 1ο βήμα χρειαζόμαστε χρόνο $O(\log(\sqrt{n}))$, για το 2ο βήμα : $log2^{i-1} = O(i-1) = O(logi) = O(log(\sqrt{n}))$ κ.ο.κ

Επομένως για κάθε επανάληψη του while loop , χρειάζεται $O(logi)$ χρόνος στο worst case scenario.

Συνολικά δηλαδή θα χρειαστεί:

$$\log(\sqrt{n}) + \log\left(\sqrt{\sqrt{n}}\right) + \log\left(\sqrt{\sqrt{\sqrt{n}}}\right) + \cdots = \log n^{\frac{1}{2}} + \log n^{\frac{1}{4}} + \log n^{\frac{1}{8}} + \cdots = \frac{1}{2}\log n + \frac{1}{4}\log n + \frac{1}{8}\log n + \cdots$$

$$= \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots\right)\log n = O(\log n)$$

Σε αυτή την άσκηση βρεθήκαμε αντιμέτωποι με αρκετά προβλήματα. Τα βήματα του αλγορίθμου (που είναι $\sqrt{n}$)ηταν πολλές φορές μεγαλύτερα από τον right υποπίνακα και έτσι παρουσιάζονταν διάφορα exception error.Για να το λύσουμε αυτό,βάλαμε στα while conditions οτι το jump πρέπει να είναι μεγαλύτερο (ή αντίστοιχα μικρότερο) από τον δεξιό πίνακα (αντίστοιχα τον αριστερό) , αλλιώς θα αυξάνεται το jump. Το i αυξάνεται όσο το condition δεν ισχύει ⇒ αυξάνεται και το jump.

```
Choose:
1.Exercise1
2.Exercise2
3.Exercise3
4.Exercise4
5.EXIT
4
QuickSort: 13.0899 ms
Vector contents: 6 12 12 12 15 15 18 18 18 21 21 24 57 63 81 90 135 174 303 312 324 324 333 336 354 363 375 381 384 390 405 411 417 429 432 432 447 447 453
453 459 462 471 474 477 477 477 480 480 483 483 486 486 486 486 489 492 495 498 498 501 504 504 504 507 513 513 519 522 534 534 534 537 540 540 540 543 546
549 552 552 555 558 570 573 600 603 603 627 639 651 663 675 675 699 705 708 711 717 720 723 729 738 738 777 783 801 828 1059 1083 1119 1134 1143 1152 1161 1
200 1212 1215 1224 1233 1245 1332 1338 1350 1356 1389 1404 1410 1455 1458 1467 1473 1488 1503 1506 1533 1542 1548 1554 1587 1596 1596 1611 1611 2037 2037 20
49 2058 2067 2070 2076 2079 2088 2115 2124 2127 2127 2145 2148 2148 2151 2160 2163 2181 2196 2199 2199 2199 2226 2229 2235 2238 2238 2241 2244 2244 2247 225
0 2262 2286 2304 2310 2310 2319 2322 2325 2328 2331 2337 2340 2349 2352 2352 2358 2358 2367 2367 2397 2865 2928 2952 2976 2985 3015 3024 3030 3045 3057 3072
 3093 3153 3165 3171 3297 3297 3372 3504 3696 3771 3840 3864 3876 3879 3939 3951 3957 3969 4005 4047 4053 4086 4116 4173 4257 5592 5610 5667 5688 5691 5706
5745 5769 5889 5904 5916 6015 6051 6054 6057 6057 6066 6147 6195 6204 6225 6231 6231 6234 6237 6324 6330 6387 6423 6435 6441 6528 6543 6573 6603 6633 6693 6
771 6777 6780 6792 6876 6939 6972 7014 7065 7077 7158 7161 7164 7284 7284 7305 7548 20385 20469 20520 20712 20745 21333 21393 21405 21429 21810 21825 22434
22605 22659 22836 23007 23112 23214 57243 57576 57747 58020 58659 58716 58887 59193 59427 59610 59637 61038 61179 61401 62544 63897 64044 64344
Please give the start and end of the space you want to search in (values must be part of the vector):
6 663

Regions with birth counts in the range [0, 91]
Region: "Region not stated or area outside region", Year: 2022, Count: 6
Region: "Region not stated or area outside region", Year: 2013, Count: 12
Region: "Region not stated or area outside region", Year: 2021, Count: 12
Region: "Region not stated or area outside region", Year: 2015, Count: 12
Region: "Region not stated or area outside region", Year: 2016, Count: 15
Region: "Region not stated or area outside region", Year: 2017, Count: 15
Region: "Region not stated or area outside region", Year: 2018, Count: 18
Region: "Region not stated or area outside region", Year: 2020, Count: 18
Region: "Region not stated or area outside region", Year: 2012, Count: 18
Region: "Region not stated or area outside region", Year: 2014, Count: 21
Region: "Region not stated or area outside region", Year: 2019, Count: 21
Region: "Region not stated or area outside region", Year: 2011, Count: 24
Region: "Region not stated or area outside region", Year: 2005, Count: 57
Region: "Region not stated or area outside region", Year: 2008, Count: 63
Region: "Region not stated or area outside region", Year: 2010, Count: 81
Region: "Region not stated or area outside region", Year: 2009, Count: 90
Region: "Region not stated or area outside region", Year: 2006, Count: 135
Region: "Region not stated or area outside region", Year: 2007, Count: 174
Region: West Coast region, Year: 2022, Count: 303
Region: West Coast region, Year: 2020, Count: 312
```

Παραπάνω, επιλέγουμε το διάστημα [6,663] για αναζήτηση.

Εκτυπώνονται οι περιοχές που επιβεβαιώνουν το κριτήριο αναζήτησης.

Ύστερα ζητάμε να εκτυπωθεί πού βρίσκεται μέσα στο array η τιμή 81. Η τιμή 81 βρίσκεται στην θέση 14.

```
Region: Tasman region, Year: 2020, Count: 474
Region: Marlborough region, Year: 2020, Count: 477
Region: Tasman region, Year: 2012, Count: 477
Region: Nelson region, Year: 2020, Count: 477
Region: Marlborough region, Year: 2019, Count: 480
Region: Marlborough region, Year: 2022, Count: 480
Region: Tasman region, Year: 2022, Count: 483
Region: Nelson region, Year: 2022, Count: 483
Region: Marlborough region, Year: 2014, Count: 486
Region: Marlborough region, Year: 2006, Count: 486
Region: Marlborough region, Year: 2021, Count: 486
Region: Tasman region, Year: 2013, Count: 486
Region: Marlborough region, Year: 2015, Count: 489
Region: Tasman region, Year: 2021, Count: 492
Region: Marlborough region, Year: 2017, Count: 495
Region: Nelson region, Year: 2021, Count: 498
Region: Tasman region, Year: 2009, Count: 498
Region: Tasman region, Year: 2011, Count: 501
Region: Nelson region, Year: 2017, Count: 504
Region: Marlborough region, Year: 2013, Count: 504
Region: Marlborough region, Year: 2018, Count: 504
Region: Marlborough region, Year: 2012, Count: 507
Region: Tasman region, Year: 2005, Count: 513
Region: Marlborough region, Year: 2007, Count: 513
Region: Nelson region, Year: 2005, Count: 519
Region: Nelson region, Year: 2006, Count: 522
Region: Marlborough region, Year: 2016, Count: 534
Region: Nelson region, Year: 2015, Count: 534
Region: Nelson region, Year: 2014, Count: 534
Region: Nelson region, Year: 2019, Count: 537
Region: Nelson region, Year: 2018, Count: 540
Region: Marlborough region, Year: 2011, Count: 540
Region: Nelson region, Year: 2016, Count: 540
Region: Tasman region, Year: 2010, Count: 543
Region: Nelson region, Year: 2012, Count: 546
Region: Tasman region, Year: 2006, Count: 549
Region: Tasman region, Year: 2007, Count: 552
Region: Marlborough region, Year: 2009, Count: 552
Region: Nelson region, Year: 2013, Count: 555
Region: Tasman region, Year: 2008, Count: 558
Region: Marlborough region, Year: 2008, Count: 570
Region: Marlborough region, Year: 2010, Count: 573
Region: Nelson region, Year: 2011, Count: 600
Region: Nelson region, Year: 2009, Count: 603
Region: Nelson region, Year: 2010, Count: 603
Region: Nelson region, Year: 2008, Count: 627
Region: Nelson region, Year: 2007, Count: 639
Region: Gisborne region, Year: 2014, Count: 651
Region: Gisborne region, Year: 2020, Count: 663
Number to search: 81
81 was found in position 14
Choose:
1.Exercise1
```

Συγκεντρωτικός κώδικας Μέρους Α’

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <string>
#include <cmath>
#include <chrono>

using namespace std;

struct Data {
    int year;
    string eventType;
    string region;
    int count;
};

int stringToInt(const string& str) {
    int result;
    stringstream ss(str);
    ss >> result;
    return result;
}

class FileReader {
public:
    vector<Data> readDataFromFile(const string& filename) {
        vector<Data> data;
        ifstream file(filename);

        if (!file.is_open()) {
            cerr << "Failed to open the file." << endl;
            return data;
        }

        string line;
        while (getline(file, line)) {
            istringstream iss(line);
            string token;
            Data d;

            if (getline(iss, token, ','))
                d.year = stringToInt(token);
            if (getline(iss, token, ','))
                d.eventType = token;
            if (getline(iss, token, ','))
                d.region = token;
            if (getline(iss, token))
                d.count = stringToInt(token);

            data.push_back(d);
        }

        file.close();
        return data;
    }
};

vector<Data> findDeaths(const vector<Data>& data) {
    vector<Data> deathRecords;
    for (const auto& entry : data) {
        if (entry.eventType == "Deaths") {
            deathRecords.push_back(entry);
        }
    }
    return deathRecords;
}

void printRecords(const vector<Data>& data) {
    for (const auto& d : data) {
```

```
            cout << "Year: " << d.year << ", Event Type: " << d.eventType
                << ", Region: " << d.region << ", Count: " << d.count << endl;
        }
    }

    void countingSort(vector<Data>& A, vector<Data>& B, int k) {
        auto start = chrono::steady_clock::now();
        vector<int> C(k + 1, 0);

        for (int j = 0; j < A.size(); ++j)
            C[A[j].count]++;

        for (int i = 1; i <= k; ++i)
            C[i] += C[i - 1];

        for (int j = A.size() - 1; j >= 0; --j) {
            B[C[A[j].count] - 1] = A[j];
            C[A[j].count]--;
        }

        auto end = chrono::steady_clock::now();
        auto diff = end - start;
        cout << "CountingSort: " << chrono::duration <double, milli>(diff).count() << " ms" << endl;
    }

    void merge(vector<Data>& arr, int left, int mid, int right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;

        vector<Data> L(n1);
        vector<Data> R(n2);

        for (int i = 0; i < n1; ++i)
            L[i] = arr[left + i];
        for (int j = 0; j < n2; ++j)
            R[j] = arr[mid + 1 + j];

        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2) {
            if (L[i].count <= R[j].count) {
                arr[k] = L[i];
                ++i;
            }
            else {
                arr[k] = R[j];
                ++j;
            }
            ++k;
        }

        while (i < n1) {
            arr[k] = L[i];
            ++i;
            ++k;
        }

        while (j < n2) {
            arr[k] = R[j];
            ++j;
            ++k;
        }
    }

    void mergeSort(vector<Data>& arr, int left, int right) {
        if (left < right) {
            auto start = chrono::steady_clock::now();

            int mid = left + (right - left) / 2;
            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);
            merge(arr, left, mid, right);

            if (left == 0 && right == arr.size() - 1) { // Check if it's the outermost call
```

```
            auto end = chrono::steady_clock::now();
            auto diff = end - start;
            cout << "MergeSort: " << chrono::duration <double, milli>(diff).count() << " ms" << endl;
        }
    }
}
int partition(vector<Data>& arr, int low, int high) {
    int pivot = arr[high].count;
    int i = low - 1;

    for (int j = low; j <= high - 1; j++) {
        if (arr[j].count < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

void quickSort(vector<Data>& arr, int low, int high) {
    if (low < high) {
        auto start = chrono::steady_clock::now();

        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);

        if (low == 0 && high == arr.size() - 1) { // Check if it's the outermost call
            auto end = chrono::steady_clock::now();
            auto diff = end - start;
            cout << "QuickSort: " << chrono::duration <double, milli>(diff).count() << " ms" << endl;
        }
    }
}

void heapify(vector<Data>& arr, int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left].count > arr[largest].count)
        largest = left;

    if (right < n && arr[right].count > arr[largest].count)
        largest = right;

    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(vector<Data>& arr) {
    auto start = chrono::steady_clock::now();

    int n = arr.size();

    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }

    auto end = chrono::steady_clock::now();
    auto diff = end - start;
    cout << "HeapSort: " << chrono::duration <double, milli>(diff).count() << " ms" << endl;
}

class BinarySearchRegionSearcher {
public:
```

```
    vector<Data> searchRegions(const vector<Data>& data, int b1, int b2) {
        vector<Data> sortedData = data;
        insertionSort(sortedData);

        auto leftIt = binarySearch(sortedData, b1);

        vector<Data> result;
        for (auto it = leftIt; it != sortedData.end() && it->count <= b2; ++it) {
            result.push_back(*it);
        }

        return result;
    }

private:
    void insertionSort(vector<Data>& data) {
        auto start = chrono::steady_clock::now();

        for (size_t i = 1; i < data.size(); ++i) {
            Data key = data[i];
            int j = i - 1;

            while (j >= 0 && data[j].count > key.count) {
                data[j + 1] = data[j];
                j = j - 1;
            }
            data[j + 1] = key;
        }

        auto end = chrono::steady_clock::now();
        auto diff = end - start;
        cout << "InsertionSort: " << chrono::duration <double, milli>(diff).count() << " ms" << endl;
    }

    vector<Data>::iterator binarySearch(vector<Data>& data, int key) {
        int low = 0;
        int high = data.size() - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;

            if (data[mid].count == key)
                return data.begin() + mid;
            else if (data[mid].count < key)
                low = mid + 1;
            else
                high = mid - 1;
        }

        return data.begin() + low;
    }
};

class InterpolationSearchRegionSearcher {
public:
    vector<Data> searchRegions(const vector<Data>& data, int b1, int b2) {
        vector<Data> sortedData = data;
        insertionSort(sortedData);

        auto leftIt = interpolationSearch(sortedData, b1);

        vector<Data> result;
        for (auto it = leftIt; it != sortedData.end() && it->count <= b2; ++it) {
            result.push_back(*it);
        }

        return result;
    }

private:
    void insertionSort(vector<Data>& data) {
        auto start = chrono::steady_clock::now();
```

```
    for (size_t i = 1; i < data.size(); ++i) {
        Data key = data[i];
        int j = i - 1;

        while (j >= 0 && data[j].count > key.count) {
            data[j + 1] = data[j];
            j = j - 1;
        }
        data[j + 1] = key;
    }

    auto end = chrono::steady_clock::now();
    auto diff = end - start;
    cout << "InsertionSort: " << chrono::duration <double, milli>(diff).count() << " ms" << endl;
}

vector<Data>::iterator interpolationSearch(vector<Data>& data, int key) {
    int low = 0;
    int high = data.size() - 1;

    while (low <= high && key >= data[low].count && key <= data[high].count) {
        int pos = low + ((key - data[low].count) * (high - low)) / (data[high].count - data[low].count);
            if (data[pos].count == key) {
                return data.begin() + pos;
            }
            else if (data[pos].count < key) {
                low = pos + 1;
            }
            else {
                high = pos - 1;
            }
        }

        return data.begin() + low;
    }
};
//ASK4#############################################################################
void readDataforask4(const string& filename, vector<Data>& data) {
    ifstream file(filename);
    if (!file.is_open()) {
        cerr << "Error: Unable to open file: " << filename << endl;
        return;
    }

    string line;
    while (getline(file, line)) {
        stringstream ss(line);
        string token;
        getline(ss, token, ',');
        int year = stoi(token);
        getline(ss, token, ',');
        string eventType = token;
        getline(ss, token, ',');
        string region = token;
        getline(ss, token, ',');
        int count = stoi(token);

        if (eventType == "Births") {
            Data d = { year, eventType, region, count };
            data.push_back(d);
        }
    }

    file.close();
}

int access(int x, vector<Data>& S, int start, int end, int flag) {
    int left = start;
    int right = end;

    if (x < S[left].count || x > S[right].count) {
        cout << x << " was not found." << endl;
```

```
    return -1;
}

int size = right - left + 1;
int next = left + ((size * (x - S[left].count)) / (S[right].count - S[left].count));
//removed +1 was causing problems;

while (left <= right) {
    size = right - left + 1;

    if (size <= 3) {
        for (int j = left; j <= right; j++) {
            if (S[j].count == x) {
                cout << x << " was found in position " << j << endl;
                return j;
            }
        }
        cout << x << " was not found." << endl;
        return -1;
    }

    if (x == S[next].count) {
        if (flag = 0) {
            cout << x << " was found in position " << next << endl;
            return next;
        }
        else {
            return next;
        }
    }

    int i = 0;
    int jump = sqrt(size);

    if (x > S[next].count) {
        while (next + i * jump <= right && x > S[next + i * jump].count) {
            i++;
        }
        left = next + (i - 1) * jump + 1;
        right = min(next + i * jump, end);
    }
    else {
        while (next - i * jump >= left && x < S[next - i * jump].count) {
            i++;
        }
        right = next - (i - 1) * jump - 1;
        left = max(next - i * jump, start);
    }

    if (left <= right) {
        next = left + ((right - left) * (x - S[left].count)) / (S[right].count - S[left].count);
        }
        else {
            break;
        }
    }

    cout << x << " was not found." << endl;
    return -1;
}
void printVectorInRange(const vector<Data>& S, int start, int end) {
    cout << "\nRegions with birth counts in the range [" << start << ", " << end << "]\n";
    for (int i = start; i <= end; ++i) {
        cout << "Region: " << S[i].region << ", Year: " << S[i].year << ", Count: "
            << S[i].count << endl;
    }
}
void printVector(const vector<Data>& vec) {
    cout << "Vector contents:";
    for (int i = 0; i < vec.size(); ++i) {
        cout << " " << vec[i].count;
    }
    cout << endl;
```

```
}
int main() {
    FileReader reader;
    vector<Data> data = reader.readDataFromFile("data.txt");
    vector<Data> data2;
    readDataforask4("data.txt", data2);
    int choice, startIndex, endIndex, elementIndex;
    int start, end;
    int element;

    do {

        cout << "Choose:\n"
            "1.Exercise1\n"
            "2.Exercise2\n"
            "3.Exercise3\n"
            "4.Exercise4\n"
            "5.EXIT\n";
        cin >> choice;

        switch (choice) {
        case 1: {
            int sortingChoice;
            cout << "Choose sorting algorithm: 1 for MergeSort, 2 for QuickSort: ";
            cin >> sortingChoice;

            vector<Data> data_births;
            for (const auto& elements : data) {
                if (elements.eventType == "Births") {
                    data_births.push_back(elements);
                }
            }

            int n = data_births.size();

            if (sortingChoice == 1) {
                mergeSort(data_births, 0, n - 1);
                cout << "\nSorted by MergeSort:\n";
            }
            else if (sortingChoice == 2) {
                quickSort(data_births, 0, n - 1);
                cout << "\nSorted by QuickSort:\n";
            }
            else {
                cout << "Invalid choice. Exiting." << endl;
                return 1;
            }
            printRecords(data_births);
            break;
        }
        case 2: {
            int sortingChoice;
            cout << "Choose sorting algorithm: 1 for Heap Sort, 2 for Counting sort: ";
            cin >> sortingChoice;

            vector<Data> data_deaths;
            for (const auto& elements : data) {
                if (elements.eventType == "Deaths") {
                    data_deaths.push_back(elements);
                }
            }

            if (sortingChoice == 1) {
                heapSort(data_deaths);
                cout << "\nSorted by Heap:\n";
            }
            else if (sortingChoice == 2) {

                int maxCount = 0;
                for (const auto& d : data_deaths) {
                    if (d.count > maxCount) maxCount = d.count;
                }
```

```
            vector<Data> sortedDeaths(data_deaths.size());

            countingSort(data_deaths, sortedDeaths, maxCount);

            data_deaths = sortedDeaths;

            cout << "\nSorted by Counting:\n";
        }
        else {
            cout << "Invalid choice. Exiting." << endl;
            return 1;
        }
        printRecords(data_deaths);
        break;
    }
    case 3: {
        int searchChoice;
        cout << "Choose searching algorithm: 1 for Binary Search, 2 for Interpolation Search: ";
        cin >> searchChoice;

        int b1, b2;
        cout << "Enter the range of birth counts (b1 and b2): ";
        cin >> b1 >> b2;

        switch (searchChoice) {
        case 1: {
            BinarySearchRegionSearcher binarySearcher;
            vector<Data> regionsBinary = binarySearcher.searchRegions(data, b1, b2);

            cout << "\nRegions with birth counts in the range [" << b1 << ", " << b2
                << "] using Binary Search:\n";
            for (const auto& region : regionsBinary) {
                cout << "Region: " << region.region << ", Year: " << region.year << ", Count: "
                    << region.count << endl;
            }
            break;
        }
        case 2: {
            InterpolationSearchRegionSearcher interpolationSearcher;
            vector<Data> regionsInterpolation = interpolationSearcher.searchRegions(data, b1, b2);

            cout << "\nRegions with birth counts in the range [" << b1 << ", " << b2
                << "] using Interpolation Search:\n";
            for (const auto& region : regionsInterpolation) {
                cout << "Region: " << region.region << ", Year: " << region.year << ", Count: "
                    << region.count << endl;
            }
            break;
        }
        default:
            cout << "Invalid choice. Exiting." << endl;
            return 1;
        }
        break;
    }
    case 4:
        while (true) {
            quickSort(data2, 0, data2.size() - 1);
            printVector(data2);
            cout << "Please give the start and end of the space you want to search in (values must be part of
the vector):" << endl;
            cin >> start >> end;

            startIndex = access(start, data2, 0, data2.size() - 1, 0);
            endIndex = access(end, data2, 0, data2.size() - 1, 0);

            if (startIndex != -1 && endIndex != -1) {
                printVectorInRange(data2, startIndex, endIndex);
                break;
            }
            else {
```

```
                cout << "Invalid start or end. Please try again." << endl;
            }
        }

        cout << "Number to search: ";
        cin >> element;

        elementIndex = access(element, data2, 0, data2.size() - 1, 1);
        startIndex = access(start, data2, 0, data2.size() - 1, 1);
        endIndex = access(end, data2, 0, data2.size() - 1, 1);
        if (elementIndex < startIndex || elementIndex > endIndex) {
            cout << element << " was not found." << endl;
        }
        else {
            access(element, data2, startIndex, endIndex, 0);
        }
        break;
    case 5:
        cout << "Exiting...";
    default:
        cout << "Invalid choice. Exiting." << endl;
        return 1;
    }
    } while (choice != 5);
    return 0;
}
```

PART II – BSTs && HASHING

ΕΡΩΤΗΜΑ Α

Σε αυτό το ερώτημα μας ζητείται να δημιουργήσουμε ένα Δυαδικό Δένδρο Αναζήτησης (ΔΔΑ) στο οποίο κάθε κόμβος του διατηρεί την εγγραφή (Region, Period, Count_of_Births) , ύστερα διατάσσεται ως προς το πεδίο Region και υλοποιείται με δυναμική διαχείριση μνήμης.

Αρχικά , το **Δυαδικό Δένδρο Αναζήτησης** είναι μια δομή δεδομένων που αποτελείται από κόμβους και έχει τα παρακάτω χαρακτηριστικά:

1. Κάθε δέντρο έχει στην κορυφή έναν κόμβο ρίζας .
2. Ο ριζικός κόμβος έχει μηδέν, έναν ή δύο θυγατρικούς κόμβους.
3. Κάθε θυγατρικός κόμβος έχει μηδέν, έναν ή δύο θυγατρικούς κόμβους.
4. Κάθε κόμβος το πολύ δύο παιδιά.
5. Για κάθε κόμβο, οι αριστεροί απόγονοί του είναι μικρότεροι από τον τρέχοντα κόμβο, ο οποίος είναι μικρότερος από τους δεξιούς απόγονους.

Το **AVL (A**delson-**V**elskii-**L**andis) δέντρο είναι ένα δυαδικό ισοζυγισμένο δέντρο με την διαφορά ότι: για κάθε κόμβο του ισχύει ότι τα ύψη των υποδέντρων του διαφέρουν το **πολύ κατά ένα (συντελεστής ισορροπίας)**. Έστω δηλαδή εσωτερικός κόμβος u και $L(u), R(u)$ τα υποδέντρα με ρίζες το αριστερό και δεξί παιδί του u αντίστοιχα. Σε κάθε u ορίζουμε την ποσότητα που ονομάζεται ισοζύγιση του u (συμβολίζεται με $hb(u)$) και δίνεται από την σχέση:

$$hb(u) = Ύψος(R(u)) - Ύψος(L(u))$$

[bonus ερώτημα] Στην δικιά μας εργαστηριακή άσκηση, εκτός από τη διάταξη με τα regions, έχουμε διατάξει τις χρονιές και μετά τα births.

1.Screenshots από την απεικόνιση του ΔΔΑ με ενδο-διατεταγμένη διάσχιση. Κάθε απεικόνιση περιέχει μια επικεφαλίδα με τον τίτλο της περιοχής της εγγραφής που απεικονίζεται

```
What would you like to do ?
1. Organise by Births
2. Organise by region
3.Use hashing to search, insert or delete
4. Exit
2

What would you like to do?
1. Show AVL tree in order
2. Search for birth population given region and year
3. Modify birth population given region and year
4. Delete a region
5. Exit
1
In Order Traversal of the created AVL Tree is:
REGION : "Region not stated or area outside region"
2005, Births, "Region not stated or area outside region", 57
2006, Births, "Region not stated or area outside region", 135
2007, Births, "Region not stated or area outside region", 174
2008, Births, "Region not stated or area outside region", 63
2009, Births, "Region not stated or area outside region", 90
2010, Births, "Region not stated or area outside region", 81
2011, Births, "Region not stated or area outside region", 24
2012, Births, "Region not stated or area outside region", 18
2013, Births, "Region not stated or area outside region", 12
2014, Births, "Region not stated or area outside region", 21
2015, Births, "Region not stated or area outside region", 12
2016, Births, "Region not stated or area outside region", 15
2017, Births, "Region not stated or area outside region", 15
2018, Births, "Region not stated or area outside region", 18
2019, Births, "Region not stated or area outside region", 21
2020, Births, "Region not stated or area outside region", 18
2021, Births, "Region not stated or area outside region", 12
2022, Births, "Region not stated or area outside region", 6
2005, Deaths, "Region not stated or area outside region", 129
2006, Deaths, "Region not stated or area outside region", 135
2007, Deaths, "Region not stated or area outside region", 132
2008, Deaths, "Region not stated or area outside region", 108
2009, Deaths, "Region not stated or area outside region", 66
2010, Deaths, "Region not stated or area outside region", 48
2011, Deaths, "Region not stated or area outside region", 27
2012, Deaths, "Region not stated or area outside region", 21
2013, Deaths, "Region not stated or area outside region", 24
```

```
What would you like to do?
1. Show AVL tree in order
2. Search for birth population given region and year
3. Modify birth population given region and year
4. Delete a region
5. Exit
1
In Order Traversal of the created AVL Tree is:
REGION : "Region not stated or area outside region"
2005, Births, "Region not stated or area outside region", 57
2005, Births, "Region not stated or area outside region", 57
2006, Births, "Region not stated or area outside region", 135
2006, Births, "Region not stated or area outside region", 135
2007, Births, "Region not stated or area outside region", 174
2007, Births, "Region not stated or area outside region", 174
2008, Births, "Region not stated or area outside region", 63
2008, Births, "Region not stated or area outside region", 63
2009, Births, "Region not stated or area outside region", 90
2009, Births, "Region not stated or area outside region", 90
2010, Births, "Region not stated or area outside region", 81
2010, Births, "Region not stated or area outside region", 81
2011, Births, "Region not stated or area outside region", 24
2011, Births, "Region not stated or area outside region", 24
2012, Births, "Region not stated or area outside region", 18
2012, Births, "Region not stated or area outside region", 18
2013, Births, "Region not stated or area outside region", 12
2013, Births, "Region not stated or area outside region", 12
2014, Births, "Region not stated or area outside region", 21
2014, Births, "Region not stated or area outside region", 21
2015, Births, "Region not stated or area outside region", 12
2015, Births, "Region not stated or area outside region", 12
2016, Births, "Region not stated or area outside region", 15
2016, Births, "Region not stated or area outside region", 15
2017, Births, "Region not stated or area outside region", 15
2017, Births, "Region not stated or area outside region", 15
2018, Births, "Region not stated or area outside region", 18
2018, Births, "Region not stated or area outside region", 18
2019, Births, "Region not stated or area outside region", 21
2019, Births, "Region not stated or area outside region", 21
2020, Births, "Region not stated or area outside region", 18
2020, Births, "Region not stated or area outside region", 18
2021, Births, "Region not stated or area outside region", 12
2021, Births, "Region not stated or area outside region", 12
2022, Births, "Region not stated or area outside region", 6
2022, Births, "Region not stated or area outside region", 6
2005, Deaths, "Region not stated or area outside region", 129
2005, Deaths, "Region not stated or area outside region", 129
2006, Deaths, "Region not stated or area outside region", 135
2006, Deaths, "Region not stated or area outside region", 135
2007, Deaths, "Region not stated or area outside region", 132
2007, Deaths, "Region not stated or area outside region", 132
2008, Deaths, "Region not stated or area outside region", 108
2008, Deaths, "Region not stated or area outside region", 108
2009, Deaths, "Region not stated or area outside region", 66
2009, Deaths, "Region not stated or area outside region", 66
2010, Deaths, "Region not stated or area outside region", 48
2010, Deaths, "Region not stated or area outside region", 48
2011, Deaths, "Region not stated or area outside region", 27
2011, Deaths, "Region not stated or area outside region", 27
2012, Deaths, "Region not stated or area outside region", 21
2012, Deaths, "Region not stated or area outside region", 21
2013, Deaths, "Region not stated or area outside region", 24
2013, Deaths, "Region not stated or area outside region", 24
2014, Deaths, "Region not stated or area outside region", 42
2014, Deaths, "Region not stated or area outside region", 42
```

```
2017, Deaths, Auckland region, 8577
2017, Deaths, Auckland region, 8577
2018, Deaths, Auckland region, 8586
2018, Deaths, Auckland region, 8586
2019, Deaths, Auckland region, 8619
2019, Deaths, Auckland region, 8619
2020, Deaths, Auckland region, 8328
2020, Deaths, Auckland region, 8328
2021, Deaths, Auckland region, 8907
2021, Deaths, Auckland region, 8907
2022, Deaths, Auckland region, 9783
2022, Deaths, Auckland region, 9783
REGION : Bay of Plenty region
2005, Births, Bay of Plenty region, 3771
2005, Births, Bay of Plenty region, 3771
2006, Births, Bay of Plenty region, 3840
2006, Births, Bay of Plenty region, 3840
2007, Births, Bay of Plenty region, 4053
2007, Births, Bay of Plenty region, 4053
2008, Births, Bay of Plenty region, 4116
2008, Births, Bay of Plenty region, 4116
2009, Births, Bay of Plenty region, 3939
2009, Births, Bay of Plenty region, 3939
2010, Births, Bay of Plenty region, 3957
2010, Births, Bay of Plenty region, 3957
2011, Births, Bay of Plenty region, 3951
2011, Births, Bay of Plenty region, 3951
2012, Births, Bay of Plenty region, 3876
2012, Births, Bay of Plenty region, 3876
2013, Births, Bay of Plenty region, 3696
2013, Births, Bay of Plenty region, 3696
2014, Births, Bay of Plenty region, 3504
2014, Births, Bay of Plenty region, 3504
2015, Births, Bay of Plenty region, 3879
2015, Births, Bay of Plenty region, 3879
2016, Births, Bay of Plenty region, 3864
2016, Births, Bay of Plenty region, 3864
2017, Births, Bay of Plenty region, 4086
2017, Births, Bay of Plenty region, 4086
2018, Births, Bay of Plenty region, 3969
2018, Births, Bay of Plenty region, 3969
2019, Births, Bay of Plenty region, 4047
2019, Births, Bay of Plenty region, 4047
2020, Births, Bay of Plenty region, 4005
2020, Births, Bay of Plenty region, 4005
2021, Births, Bay of Plenty region, 4173
2021, Births, Bay of Plenty region, 4173
2022, Births, Bay of Plenty region, 4257
2022, Births, Bay of Plenty region, 4257
2005, Deaths, Bay of Plenty region, 1956
2005, Deaths, Bay of Plenty region, 1956
2006, Deaths, Bay of Plenty region, 2151
2006, Deaths, Bay of Plenty region, 2151
2007, Deaths, Bay of Plenty region, 2139
2007, Deaths, Bay of Plenty region, 2139
2008, Deaths, Bay of Plenty region, 2241
2008, Deaths, Bay of Plenty region, 2241
2009, Deaths, Bay of Plenty region, 2226
2009, Deaths, Bay of Plenty region, 2226
2010, Deaths, Bay of Plenty region, 2139
2010, Deaths, Bay of Plenty region, 2139
2011, Deaths, Bay of Plenty region, 2202
2011, Deaths, Bay of Plenty region, 2202
2012, Deaths, Bay of Plenty region, 2211
2012, Deaths, Bay of Plenty region, 2211
2013, Deaths, Bay of Plenty region, 2238
2013, Deaths, Bay of Plenty region, 2238
```

2.Αναζήτηση του αριθμού γεννήσεων για συγκεκριμένη χρονική περίοδο και περιοχή που δίνονται από το χρήστη.

Αναζητούμε την περιοχή West Coast το 2022 και μας εμφανίζει 303 γεννήσεις.

```
What would you like to do ?
1. Organise by Births
2. Organise by region
3.Use hashing to search, insert or delete
4. Exit
2

What would you like to do?
1. Show AVL tree in order
2. Search for birth population given region and year
3. Modify birth population given region and year
4. Delete a region
5. Exit
2
Enter the region you want to find: West Coast region
Enter the year you want to find: 2022

Node found: 2022, Births, West Coast region, 303
```

Εδώ αναζητούμε μία άλλη εγγραφή με όνομα Australia , την χρονιά 2024 και αφού δεν υπάρχει κάπου τότε εμφανίζεται το παρακάτω μήνυμα:

```
What would you like to do?
1. Show AVL tree in order
2. Search for birth population given region and year
3. Modify birth population given region and year
4. Delete a region
5. Exit
2
Enter the region you want to find: Australia
Enter the year you want to find: 2024

Node with region Australia and year 2024 for Births not found.
```

3. Τροποποίηση του πεδίου αριθμού γεννήσεων για συγκεκριμένη χρονική περίοδο και περιοχή που δίνονται από το χρήστη

```
What would you like to do?
1. Show AVL tree in order
2. Search for birth population given region and year
3. Modify birth population given region and year
4. Delete a region
5. Exit
3
Enter the region you want to modify population for:
 West Coast region
Enter the year: 2020
Enter the new population: 541
Population for region West Coast region in year 2020 updated to 541

What would you like to do?
1. Show AVL tree in order
2. Search for birth population given region and year
3. Modify birth population given region and year
4. Delete a region
5. Exit
1
In Order Traversal of the created AVL Tree is:
REGION : "Region not stated or area outside region"
2005, Births, "Region not stated or area outside region", 57
2006, Births, "Region not stated or area outside region", 135
2007, Births, "Region not stated or area outside region", 174
2008, Births, "Region not stated or area outside region", 63
2009, Births, "Region not stated or area outside region", 90
2010, Births, "Region not stated or area outside region", 81
2011, Births, "Region not stated or area outside region", 24
2012, Births, "Region not stated or area outside region", 18
2013, Births, "Region not stated or area outside region", 12
2014, Births, "Region not stated or area outside region", 21
2015, Births, "Region not stated or area outside region", 12
2016, Births, "Region not stated or area outside region", 15
2017, Births, "Region not stated or area outside region", 15
2018, Births, "Region not stated or area outside region", 18
2019, Births, "Region not stated or area outside region", 21
2020, Births, "Region not stated or area outside region", 18
2021, Births, "Region not stated or area outside region", 12
2022, Births, "Region not stated or area outside region", 6
2005, Deaths, "Region not stated or area outside region", 129
2006, Deaths, "Region not stated or area outside region", 135
2007, Deaths, "Region not stated or area outside region", 132
2008, Deaths, "Region not stated or area outside region", 108
2009, Deaths, "Region not stated or area outside region", 66
2010, Deaths, "Region not stated or area outside region", 48
2011, Deaths, "Region not stated or area outside region", 27
2012, Deaths, "Region not stated or area outside region", 21
2013, Deaths, "Region not stated or area outside region", 24
2014, Deaths, "Region not stated or area outside region", 42
2015, Deaths, "Region not stated or area outside region", 24
```

```
REGION : West Coast region
2005, Births, West Coast region, 336
2006, Births, West Coast region, 390
2007, Births, West Coast region, 405
2008, Births, West Coast region, 453
2009, Births, West Coast region, 432
2010, Births, West Coast region, 429
2011, Births, West Coast region, 432
2012, Births, West Coast region, 417
2013, Births, West Coast region, 384
2014, Births, West Coast region, 375
2015, Births, West Coast region, 381
2016, Births, West Coast region, 324
2017, Births, West Coast region, 363
2018, Births, West Coast region, 324
2019, Births, West Coast region, 354
2020, Births, West Coast region, 541
2021, Births, West Coast region, 333
2022, Births, West Coast region, 547
2005, Deaths, West Coast region, 243
2006, Deaths, West Coast region, 240
2007, Deaths, West Coast region, 255
2008, Deaths, West Coast region, 273
2009, Deaths, West Coast region, 264
2010, Deaths, West Coast region, 249
2011, Deaths, West Coast region, 291
2012, Deaths, West Coast region, 252
2013, Deaths, West Coast region, 291
2014, Deaths, West Coast region, 279
2015, Deaths, West Coast region, 294
2016, Deaths, West Coast region, 267
2017, Deaths, West Coast region, 309
2018, Deaths, West Coast region, 291
2019, Deaths, West Coast region, 300
2020, Deaths, West Coast region, 273
2021, Deaths, West Coast region, 336
2022, Deaths, West Coast region, 363
```

<u>Αλλαγή των γεννήσεων για την περιοχή West Coast το 2020 σε 541</u>

4. Διαγραφή μιας εγγραφής βάσει της περιοχής που δίνεται από τον χρήστη.

```
What would you like to do?
1. Show AVL tree in order
2. Search for birth population given region and year
3. Modify birth population given region and year
4. Delete a region
5. Exit
4
Give region to delete:
 West Coast region
```

Επιλέγουμε να διαγράψουμε τις εγγραφές που αφορούν την περιοχή

West Coast *και εκτυπώνουμε πάλι τις περιοχές ταξινομημένες.*

Βλέπουμε πως πράγματι έχει διαγραφεί η West Coast.

```
2006, Deaths, Tasman region, 342
2007, Deaths, Tasman region, 324
2008, Deaths, Tasman region, 297
2009, Deaths, Tasman region, 300
2010, Deaths, Tasman region, 309
2011, Deaths, Tasman region, 348
2012, Deaths, Tasman region, 342
2013, Deaths, Tasman region, 372
2014, Deaths, Tasman region, 339
2015, Deaths, Tasman region, 375
2016, Deaths, Tasman region, 384
2017, Deaths, Tasman region, 372
2018, Deaths, Tasman region, 402
2019, Deaths, Tasman region, 450
2020, Deaths, Tasman region, 366
2021, Deaths, Tasman region, 420
2022, Deaths, Tasman region, 444
REGION : Waikato region
2005, Births, Waikato region, 5667
2006, Births, Waikato region, 5592
2007, Births, Waikato region, 6231
2008, Births, Waikato region, 6423
2009, Births, Waikato region, 6147
2010, Births, Waikato region, 6324
2011, Births, Waikato region, 5916
2012, Births, Waikato region, 6051
2013, Births, Waikato region, 5745
2014, Births, Waikato region, 5688
2015, Births, Waikato region, 6234
2016, Births, Waikato region, 6066
2017, Births, Waikato region, 6204
2018, Births, Waikato region, 6054
2019, Births, Waikato region, 6237
2020, Births, Waikato region, 6195
2021, Births, Waikato region, 6330
2022, Births, Waikato region, 6387
2005, Deaths, Waikato region, 2520
2006, Deaths, Waikato region, 2754
2007, Deaths, Waikato region, 2787
2008, Deaths, Waikato region, 2871
2009, Deaths, Waikato region, 2760
2010, Deaths, Waikato region, 2772
2011, Deaths, Waikato region, 2937
2012, Deaths, Waikato region, 2931
2013, Deaths, Waikato region, 2919
2014, Deaths, Waikato region, 2955
2015, Deaths, Waikato region, 3105
2016, Deaths, Waikato region, 3066
2017, Deaths, Waikato region, 3378
2018, Deaths, Waikato region, 3420
2019, Deaths, Waikato region, 3393
2020, Deaths, Waikato region, 3357
2021, Deaths, Waikato region, 3600
2022, Deaths, Waikato region, 4101
REGION : Wellington region
2005, Births, Wellington region, 6225

What would you like to do?
```

ΕΡΩΤΗΜΑ Β

Σε αυτό το ερώτημα τροποποιήσαμε κατάλληλα τον κώδικα του (Α), ώστε το αρχείο να διαβάζεται στο ΔΔΑ με βάση τον αριθμό γεννήσεων (Count_of_Births, Period, Region). Το ΔΔΑ διατάσσεται ως προς το πεδίο $Count\_of\_Births$ και υλοποιείται με δυναμική διαχείριση μνήμης.

Παρακάτω υπάρχει στιγμιότυπο που εμφανίζει το μενού με τις επιλογές:

1. Εύρεση Περιοχής/Περιοχών με τον **μέγιστο αριθμό γεννήσεων**.

2. Εύρεση Περιοχής/Περιοχών με τον **ελάχιστο αριθμό γεννήσεων**.

```
What would you like to do ?
1. Organise by Births
2. Organise by region
3.Use hashing to search, insert or delete
4. Exit
1

What would you like to do ?
1. Show most Births
2. Show least Births
3.Show AVL tree in order
4. Exit
1
Node with maximum population: 2008, Births, New Zealand, 64344
What would you like to do ?
1. Show most Births
2. Show least Births
3.Show AVL tree in order
4. Exit
2
Node with minimum population: 2022, Births, "Region not stated or area outside region", 6
What would you like to do ?
1. Show most Births
2. Show least Births
3.Show AVL tree in order
4. Exit
```

ΕΡΩΤΗΜΑ Γ

Στο τελευταίο ερώτημα , υλοποιήσαμε πάλι το ερώτημα (Α) κάνοντας χρήση **hashing με αλυσίδες** αυτή την φορά αντί για ΔΔΑ. Σύμφωνα με την εκφώνηση, η συνάρτηση κατακερματισμού υπολογίζεται ως το υπόλοιπο ($modulo$) της διαίρεσης του αθροίσματος των κωδικών ASCII των επιμέρους χαρακτήρων που απαρτίζουν την ΠΕΡΙΟΧΗ (Region) με ένα περιττό αριθμό m που συμβολίζει το πλήθος των κάδων ($buckets$).

Π.χ. για ΠΕΡΙΟΧΗ=*"Northland region"* και $m = 11$, ισχύει:

$$Hash("Northland\ region") = [ASCII('N') + ASCII('o') + ASCII('r') + \dots\dots + ASCII('i') + ASCII('o') + ASCII('n')]\ mod\ 11.$$

Παρακάτω επισυνάπτονται screenshots από το runtime της εφαρμογής

1.Αναζήτηση του αριθμού γεννήσεων για συγκεκριμένη χρονική περίοδο και περιοχή που δίνονται από το χρήστη.

Αναζητούμε το West Coast region,2022 (εμφανίζει 303 births) και το West Coast region, 2024 (δεν υπάρχει η εγγραφή και δεν βρίσκει κάτι)

```
What would you like to do ?
1. Organise by Births
2. Organise by region
3.Use hashing to search, insert or delete
4. Exit
3

Menu:
1. Search for an event
2. Remove an entry
3. Change amount for an event
4. Exit
Enter your choice: 1
Enter the place you want to find: West Coast region
Enter the date you want to find: 2022

 found: 2022, Births, West Coast region, 303
```

```
Menu:
1. Search for an event
2. Remove an entry
3. Change amount for an event
4. Exit
Enter your choice: 1
Enter the place you want to find: West Coast region
Enter the date you want to find: 2024

Place West Coast region and date 2024 was not found or was all Deaths.
```

2. Τροποποίηση του πεδίου αριθμού γεννήσεων για συγκεκριμένη χρονική περίοδο και περιοχή που δίνονται από τον χρήστη.

```
Menu:
1. Search for an event
2. Remove an entry
3. Change amount for an event
4. Exit
Enter your choice: 3
Enter the place: Auckland region
Enter the date: 2020
Please enter new number below:
58

Menu:
1. Search for an event
2. Remove an entry
3. Change amount for an event
4. Exit
Enter your choice: 1
Enter the place you want to find: Auckland region
Enter the date you want to find: 2020

 found: 2020, Births, Auckland region, 58

Menu:
1. Search for an event
2. Remove an entry
3. Change amount for an event
4. Exit
Enter your choice: 4
Exiting program.

What would you like to do ?
1. Organise by Births
2. Organise by region
3.Use hashing to search, insert or delete
4. Exit
4
Exiting programm...
Process returned 0 (0x0)   execution time : 230.356 s
Press any key to continue.
```

3. Διαγραφή μιας εγγραφής βάσει της περιοχής που δίνεται από τον χρήστη

```
Menu:
1. Search for an event
2. Remove an entry
3. Change amount for an event
4. Exit
Enter your choice: 2
Enter the place to remove: West Coast region
West Coast region
Entry removed.
```

Σχόλια για την άσκηση

Κατά την υλοποιηση των ασκήσεων του Part B αντιμετωπίσαμε μερικές δυσκολίες ,αλλά καταφέραμε να τις επιλύσουμε.

- Συγκεκριμένα, στα AVL δέντρα είχαμε πολλές περιπτώσεις όπου ένας κόμβος ήταν null και προσπαθούσε να γινει rotation σε εκείνο το σημείο. Για να το λυσουμε, βάλαμε ενα απλό debugging που όταν το node μας είναι null να γυρνάει null ξανά για το καινούριο node.
- Πολλές φορές το πρόγραμμα έπρεπε να διαβάσει μια γραμμή με κενά , γι'αυτό και κάναμε χρήση της $\boldsymbol{getline}()$.Όμως αυτό δημιουργούσε προβλήματα καθώς δεν μπορούσε να διαβάσει από τον χρήστη ένα οποιοδήποτε input.Βρεθήκαμε αντιμέτωποι με αυτό το θέμα σε πολλές περιστάσεις , ειδικά στο hashing κομμάτι της εργασίας . Βλέπαμε exception errors "$\boldsymbol{Can't\ divide\ with\ 0}$" και δεν γνωρίζαμε από που προέρχονταν. Αφού ψάξαμε παραπάνω, καταλάβαμε πως το getline() διάβαζε χαρακτήρες όπως "$\backslash n$" και έπρεπε να καθαρίζουμε τον buffer πριν πάρουμε κάποια νέα τιμή.Όπως φαινεται και στον κώδικα μας , κάναμε εν τέλει χρήση της $\boldsymbol{cin.ignore}()$.
- Αντιμετωπίσαμε επίσης και ένα λογικό λάθος στα rotate του tree .Τα ύψη των κόμβων δεν έκαναν σωστά update, διότι είχαμε ξεχάσει να τα αυξήσουμε κατά 1.

Σχόλια για τα bonus ερωτήματα

Στο AVL δέντρο μας , εκτός από τη διάταξη με τα regions,έχουμε διατάξει τις χρονιές και τα births. Οπότε έχει πιο περίπλοκη σειρά, δεν είναι μόνο η σειρά του AVL με τα regions.

Αναζήτηση σε AVL δέντρο

Η αναζήτηση σε ένα AVL δέντρο είναι παρόμοια με την αναζήτηση σε ένα τυπικό δυαδικό δέντρο αναζήτησης:

1. Ξεκινάμε από τη ρίζα του δέντρου.
2. Συγκρίνουμε το στοιχείο που αναζητούμε με την τιμή του τρέχοντος κόμβου.
3. Εάν το στοιχείο είναι μικρότερο, προχωράμε στον αριστερό υποκόμβο.
4. Εάν το στοιχείο είναι μεγαλύτερο, προχωράμε στον δεξιό υποκόμβο.
5. Επαναλαμβάνουμε τα βήματα 2-4 μέχρι να βρούμε το στοιχείο ή να φτάσουμε σε ένα φύλλο (και να διαπιστώσουμε ότι το στοιχείο δεν υπάρχει στο δέντρο).

Εξισορρόπηση του AVL δέντρου

Η εξισορρόπηση σε ένα AVL δέντρο επιτυγχάνεται με την διατήρηση της ιδιότητας του ισοζυγισμένου ύψους. Για κάθε κόμβο στο δέντρο, η διαφορά ύψους μεταξύ των δύο υποκόμβων του (balance factor) πρέπει να είναι -1, 0 ή 1. Εάν η εισαγωγή ή η διαγραφή κόμβων προκαλέσει ανισορροπία, εκτελούνται περιστροφές για την επαναφορά της ισορροπίας.

Υπάρχουν τέσσερις τύποι περιστροφών που μπορούν να εφαρμοστούν:

1. **Μονή Περιστροφή προς τα Δεξιά (Right Rotation):** Χρησιμοποιείται όταν έχουμε ανισορροπία αριστερά-αριστερά (LL case).
2. **Μονή Περιστροφή προς τα Αριστερά (Left Rotation):** Χρησιμοποιείται όταν έχουμε ανισορροπία δεξιά-δεξιά (RR case).
3. **Διπλή Περιστροφή προς τα Αριστερά-Δεξιά (Left-Right Rotation):** Χρησιμοποιείται όταν έχουμε ανισορροπία αριστερά-δεξιά (LR case).
4. **Διπλή Περιστροφή προς τα Δεξιά-Αριστερά (Right-Left Rotation):** Χρησιμοποιείται όταν έχουμε ανισορροπία δεξιά-αριστερά (RL case).

Εμείς στην υλοποιήση μας ψάχνουμε αρχικά το αριστερό δέντρο και μετά το δεξί και φιλτράρουμε αν υπάρχει birth ή death.

Συγκεντρωτικός κώδικας Μέρους Β’

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <vector>
#include <list>


using namespace std;
//HASHING*********************************************************************************
struct info {
    int date;
    string type;
    string place;
    int amount;
};

int stringToIntH(const string& str) {
    int result;
    stringstream ss(str);
    ss >> result;
    return result;
}

// Define the HashTable class
class HashTable {
private:
    int size;
    vector<list<info>> table;

    // Hash function to determine index
    int hash(const string& key) {
        size = key.size();

        int hashValue = 0;
        for (char c : key) {
            hashValue += c;
        }
        return hashValue % size;
    }


public:
    // Constructor
    HashTable(int size) : size(size), table(size) {}

    // Insertion method
    void insert(const info& data) {
        int index = hash(data.place);
        table[index].push_back(data);
    }

    // Search method
    info* searchHASH(string place, int date, string event) {
        int index = hash(place);
        for (auto& element : table[index]) {
            if (element.place == place && element.date == date && element.type == event) {
                return &element;
            }
        }
        return NULL;
    }
    // Deletion method
    void remove(const string& region) {
        int index = hash(region);
        for (auto it = table[index].begin(); it != table[index].end();) {
            if (it->place == region) {
                it = table[index].erase(it);
            }
            else {
                it++;
```

```
            }
        }
    }
    info* change_amount(string place, int date, string event) {
        int index = hash(place);
        int num;
        for (auto& element : table[index]) {
            if (element.place == place && element.date == date && element.type == event) {
                cout << "Please enter new number below:\n";
                cin >> num;
                element.amount = num;
                return &element;
            }
        }
        return NULL;
    }
};


void readFiletoHash(HashTable& hashTable, string filename) {
    ifstream file(filename.c_str());
    if (!file.is_open()) {
        cout << "Failed to open the file\n";
        return;
    }

    string line;
    while (getline(file, line)) {
        string datestr, type, place;
        int amount;
        stringstream ss(line);
        getline(ss, datestr, ',');
        getline(ss, type, ',');
        getline(ss, place, ',');
        ss >> amount;
        int date = stringToIntH(datestr);
        info data = { date , type, place, amount };
        hashTable.insert(data);
    }
    file.close();
}

void searchMod(HashTable& hashTable) {
    string searchplace;
    int searchdate;
    cin.ignore();
    cout << "Enter the place you want to find: ";
    getline(cin, searchplace);

    cout << "Enter the date you want to find: ";
    cin >> searchdate;

    info* result = hashTable.searchHASH(searchplace, searchdate, "Births");

    if (result != NULL) {
        cout << "\n found: " << result->date << ", " << result->type << ", " << result->place << ", " << result-
>amount << endl;
    }
    else {
        cout << "\nPlace " << searchplace << " and date " << searchdate << " was not found or was all Deaths." <<
endl;
    }
}

//AVL TREE*****************************************************************************************
struct Node {
    int year;
    string name;
    string region;
    int population;
    Node* left;
    Node* right;
    int height;
```

```
};

//used for the readfromfile function
int stringToInt(const string& str) {
    int result;
    stringstream ss(str);
    ss >> result;
    return result;
}

//constuctor
Node* newNode(int year, string name, string region, int population) {
    Node* node = new Node;  // Use new instead of malloc
    node->year = year;
    node->name = name;
    node->region = region;
    node->population = population;
    node->left = NULL;
    node->right = NULL;
    node->height = 1;
    return node;
}

int max(int a, int b) {
    return (a > b) ? a : b;
}

int height(Node* N) {
    if (N == NULL)
        return 0;
    return N->height;
}

Node* rightRotate(Node* y) {
    if (y == nullptr) {
        // Handle null pointer case
        return nullptr;
    }

    Node* x = y->left;
    if (x == nullptr) {
        // Handle null pointer case
        return nullptr;
    }

    Node* T2 = x->right;

    x->right = y;
    y->left = T2;

    // Update heights
    y->height = std::max(height(y->left), height(y->right)) + 1;
    x->height = std::max(height(x->left), height(x->right)) + 1;

    return x;
}


Node* leftRotate(Node* x) {
    if (x == nullptr) {
        // Handle null pointer case
        return nullptr;
    }

    Node* y = x->right;
    if (y == nullptr) {
        // Handle null pointer case
        return nullptr;
    }

    Node* T2 = y->left;

    y->left = x;
```

```
    x->right = T2;

    // Update heights
    x->height = std::max(height(x->left), height(x->right)) + 1;
    y->height = std::max(height(y->left), height(y->right)) + 1;

    return y;
}



int getBalance(Node* N) {
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}

Node* insertNode(Node* node, int year, string name, string region, int population) {
    if (node == NULL) {
        return newNode(year, name, region, population);
    }
    // Compare region first
    if (region < node->region)
    node->left = insertNode(node->left, year, name, region, population);
else if (region > node->region)
    node->right = insertNode(node->right, year, name, region, population);
else { //Region is the same, compare name type
    if (name < node->name)
        node->left = insertNode(node->left, year, name, region, population);
    else if (name > node->name)
        node->right = insertNode(node->right, year, name, region, population);
    else { //Region and name type are the same, compare year
        if (year < node->year)
            node->left = insertNode(node->left, year, name, region, population);
        else
            node->right = insertNode(node->right, year, name, region, population);
    }
}


node->height = 1 + max(height(node->left), height(node->right));


int balance = getBalance(node);


if (balance > 1 && year < node->left->year)
    return rightRotate(node);


if (balance < -1 && year > node->right->year)
        return leftRotate(node);


    if (balance > 1 && year > node->left->year) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }


    if (balance < -1 && year < node->right->year) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    return node;
}
Node* insertNodeButBirths(Node* node, int year, string name, string region, int population) {
    // If the node is a "Death" node, ignore it
    if (name == "Deaths") {
        return node;
    }
```

```
if (node == NULL) {
    return newNode(year, name, region, population);
}

// Compare populations
if (population < node->population) {
    node->left = insertNodeButBirths(node->left, year, name, region, population);
}
else if (population > node->population) {
    node->right = insertNodeButBirths(node->right, year, name, region, population);
}
else { // If populations are the same
    // Compare regions
    if (region < node->region) {
        node->left = insertNodeButBirths(node->left, year, name, region, population);
    }
    else if (region > node->region) {
        node->right = insertNodeButBirths(node->right, year, name, region, population);
    }
}

// Update height of the current node
node->height = 1 + max(height(node->left), height(node->right));

    // Get the balance factor of this node
    int balance = getBalance(node);

    // Perform rotations if necessary
    if (balance > 1 && population < node->left->population)
        return rightRotate(node);

    if (balance < -1 && population > node->right->population)
        return leftRotate(node);

    if (balance > 1 && population > node->left->population) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    if (balance < -1 && population < node->right->population) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    // If no rotations needed, return the unchanged node
    return node;
}
```

```
void inOrder(Node* root, string& currentRegion) {
    if (root != NULL) {
        inOrder(root->left, currentRegion);
        if (root->region != currentRegion) {
            currentRegion = root->region;
            cout << "REGION : " << currentRegion << endl;
        }
        cout << root->year << ", " << root->name << ", " << root->region << ", " << root->population << endl;
        inOrder(root->right, currentRegion);
    }
}

void readFileToAVLTree(Node **root, string filename,int select) {
    ifstream file(filename.c_str());
    if (!file.is_open()) {
        cout << "Failed to open the file\n";
        return;
    }

    string line;
    while (getline(file, line)) {
```

```
        string yearstr, name, region;
        int population;
        stringstream ss(line);
        getline(ss, yearstr, ',');
        getline(ss, name, ',');
        getline(ss, region, ',');
        ss >> population;
        int year = stringToInt(yearstr);
        if (select == 1) {
            *root = insertNodeButBirths(*root, year, name, region, population);
        }
        else {
            *root = insertNode(*root, year, name, region, population);
        }
    }

    file.close();
}

Node* search(Node* root, string region, int year, string event) {
    if (root == NULL) {
        return NULL;
    }

    Node* leftResult = search(root->left, region, year, event);
    if (leftResult != NULL)
        return leftResult;

    // Check if region, year, and event match
    if (root->region == region && root->year == year && root->name == event)
        return root;

    // If not matched, continue searching in the right subtree
    return search(root->right, region, year, event);
}
```

```
void search(Node* root) {
    string searchRegion, searchEvent;
    int searchYear;

    cout << "Enter the region you want to find: ";
    getline(cin, searchRegion);
    cout << "Enter the year you want to find: ";
    cin >> searchYear;

    Node* foundNode = search(root, searchRegion, searchYear, "Births");

    if (foundNode != NULL) {
        cout << "\nNode found: " << foundNode->year << ", " << foundNode->name << ", " << foundNode->region << ",
" << foundNode->population << endl;
    }
    else {
        cout << "\nNode with region " << searchRegion << " and year " << searchYear << " for Births not found." <<
endl;
    }
}
```

```
Node* deleteNode(Node* root, string deleteRegion) {
    if (root == NULL) {
        return root;
    }

    if (deleteRegion < root->region)
        root->left = deleteNode(root->left, deleteRegion);
```

```
else if (deleteRegion > root->region)
    root->right = deleteNode(root->right, deleteRegion);
else {

    root->left = deleteNode(root->left, deleteRegion);
    root->right = deleteNode(root->right, deleteRegion);
    delete root;
    return NULL;
}


root->height = 1 + max(height(root->left), height(root->right));


int balance = getBalance(root);

//If this node becomes unbalanced, then there are 4 cases


if (balance > 1 && getBalance(root->left) >= 0)
    return rightRotate(root);

    if (balance > 1 && getBalance(root->left) < 0) {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }


    if (balance < -1 && getBalance(root->right) <= 0)
        return leftRotate(root);


    if (balance < -1 && getBalance(root->right) > 0) {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }

    return root;
}

Node* findMaxPopulation(Node* root) {
    if (root == NULL)
        return NULL;

    while (root->right != NULL)
        root = root->right;

    return root;
}

Node* findMinPopulation(Node* root) {
    if (root == NULL)
        return NULL;

    while (root->left != NULL)
        root = root->left;

    return root;
}


void modifyPopulation(Node* root) {
    string region;
    int year, newPopulation;

    cout << "Enter the region you want to modify population for:\n ";
    getline(cin, region);

    cout << "Enter the year: ";
    cin >> year;
```

```
    cout << "Enter the new population: ";
    cin >> newPopulation;


    if (year < 0 || newPopulation < 0) {
        cout << "Invalid input! Year and population must be non-negative." << endl;
        return;
    }

    Node* nodeToUpdate = search(root ,region, year,"Births");


    if (nodeToUpdate != NULL && nodeToUpdate->name == "Births") {
        nodeToUpdate->population = newPopulation;
        cout << "Population for region " << region << " in year " << year << " updated to " << newPopulation <<
endl;
    }
    else {
        cout << "Node with region " << region << " and year " << year << " not found or there were only deaths" <<
endl;
    }
}
```

```
int main() {
    string region_to_delete;
    string region = " ";
    Node* root = NULL;
    int num,question,most_least;
    int choice;
    HashTable hashTable(100);
    readFiletoHash(hashTable, "data.txt");
   do{
       cout << "\nWhat would you like to do ? \n"
            "1. Organise by Births\n"
            "2. Organise by region\n"
            "3.Use hashing to search, insert or delete\n"
            "4. Exit\n";
    cin >> question;
    switch (question)
    {
    case(1):
        readFileToAVLTree(&root, "data.txt", question);
        do {
            cout << "\nWhat would you like to do ? \n"
                "1. Show most Births\n"
                "2. Show least Births\n"
                "3.Show AVL tree in order\n"
                "4. Exit\n";
            cin >> most_least;
            Node* maxNode = findMaxPopulation(root);
            Node* minNode = findMinPopulation(root);
            switch (most_least)
            {
            case(1):
                if (maxNode != NULL)
                    cout << "Node with maximum population: " << maxNode->year << ", " << maxNode->name << ", " <<
maxNode->region << ", " << maxNode->population;
                else
                    cout << "AVL tree is empty.";
                break;
            case(2):
                if (minNode != NULL)
                    cout << "Node with minimum population: " << minNode->year << ", " << minNode->name << ", " <<
minNode->region << ", " << minNode->population;
                else
                    cout << "AVL tree is empty.";
                break;
            case(3):
                cout << "In Order Traversal of the created AVL Tree is:\n";
                inOrder(root, region);
                break;
```

```
            case(4):
                break;
            default:
                cout << "Invalid input\n";
                break;
            }
        } while (most_least != 4);
        break;
    case(2):
        readFileToAVLTree(&root, "data.txt", question);
        do {

            cout << "\nWhat would you like to do?\n"
                "1. Show AVL tree in order\n"
                "2. Search for birth population given region and year\n"
                "3. Modify birth population given region and year\n"
                "4. Delete a region\n"
                "5. Exit\n";
            cin >> num;

            switch (num) {
            case 1:
                cout << "In Order Traversal of the created AVL Tree is:\n";
                inOrder(root, region);
                break;
            case 2:
                cin.ignore();
                search(root);
                break;
            case 3:
                cin.ignore();
                modifyPopulation(root);
                break;
            case 4:
                cin.ignore();
                cout << "Give region to delete:\n ";
                getline(cin, region_to_delete);
                root = deleteNode(root, region_to_delete);
                break;
            case 5:
                cout << "Exiting the program.\n";
                break;
            default:
                cout << "Invalid option. Please try again.\n";
            }
        } while (num != 5);
        break;
    case(3):
        do {
            cout << "\nMenu:\n";
            cout << "1. Search for an event\n";
            cout << "2. Remove an entry\n";
            cout << "3. Change amount for an event\n";
            cout << "4. Exit\n";
            cout << "Enter your choice: ";
            cin >> choice;

            switch (choice) {
            case 1:
                searchMod(hashTable);
                break;
            case 2: {
                string removed;
                cout << "Enter the place to remove: ";
                 cin.ignore();
                 getline(cin, removed);
                cout << removed;
                hashTable.remove(removed);
                cout << "\nEntry removed.\n";
                break;
            }
            case 3: {
                string place;
```

```
            int date;
            cout << "Enter the place: ";
            cin.ignore();
            getline(cin,place);
            cout << "Enter the date: ";
            cin >> date;
            hashTable.change_amount(place, date, "Births");
            break;
        }
        case 4:
            cout << "Exiting program.\n";
            break;
        default:
            cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 4);
    break;
case(4):
    cout << "Exiting programm...";
    break;
default:
    cout << "Invalid input\n";
    break;
    }
}while (question != 4);


    return 0;
}
```

Βιβλιογραφία

[1] Αθανάσιος Κ. Τσακαλίδης: Δομές Δεδομένων , Πανεπιστήμιο Πατρών
https://eclass.upatras.gr/modules/document/index.php?course=CEID1158&openDir=/5e81d479ayvm

[2] https://www.geeksforgeeks.org/interpolation-search/

[3] https://www.geeksforgeeks.org/binary-search/

[4] https://www.javatpoint.com/interpolation-search-vs-binary-search

[5] Χρ. Μακρής – Σ. Σιούτας: Διαφάνειες e-Class(CEID_NY233) 2023-2024 https://eclass.upatras.gr/courses/CEID1158/

[6] Kurt Mehlhorn, Peter Sanders: Αλγόριθμοι και Δομές Δεδομένων-Τα βασικά εργαλεία, εκδόσεις Κλειδάριθμος