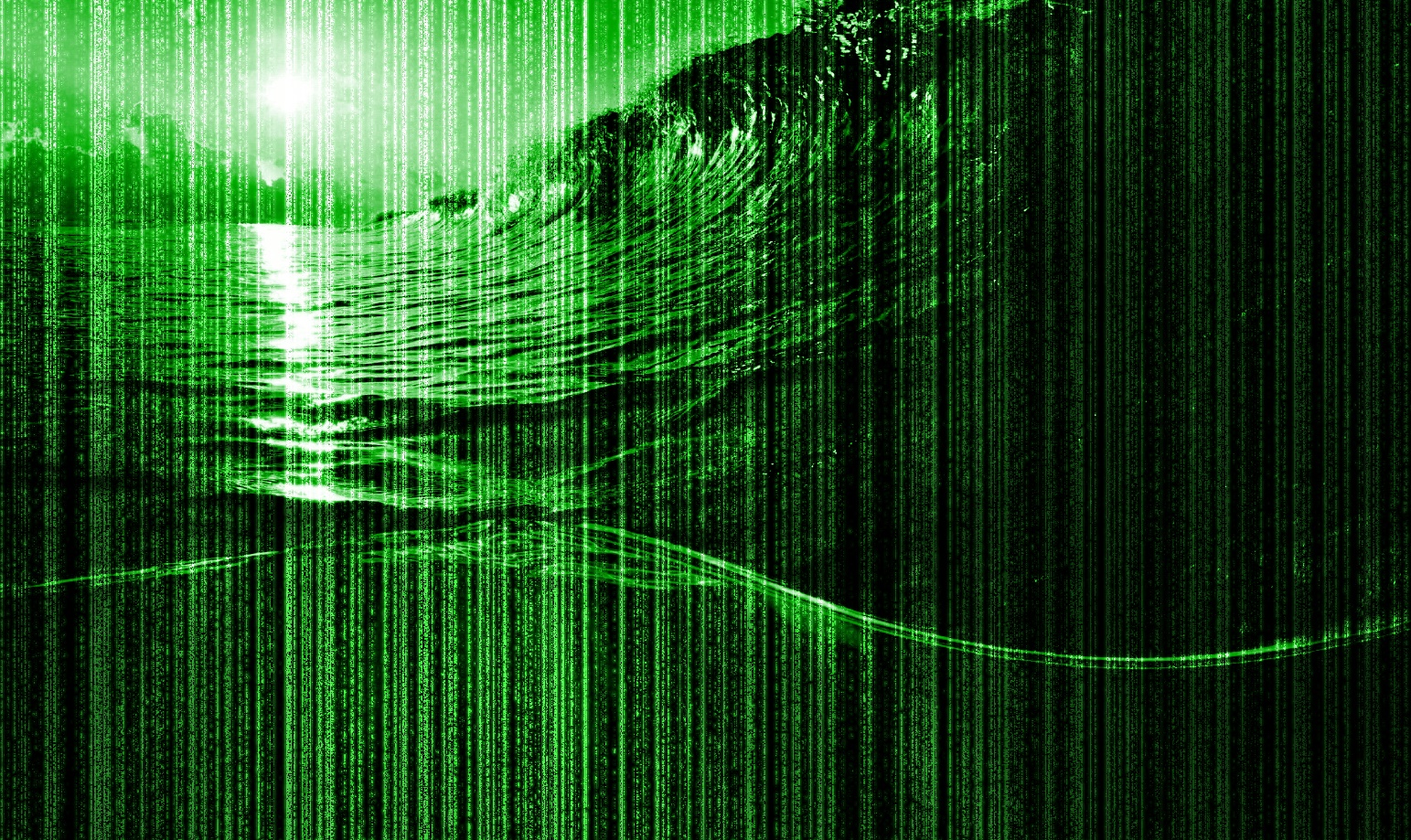
***PROJECT: Δομές δεδομένων 2022***



**Μέλη Ομάδας**

Ονοματεπώνυμο ΑΜ

**Βασίλειος Μπαρδάκης - 1088098**

**Χαρίτων Κικίδης - 1084595**

**Θεόδωρος Πλάτωνας - 1090073**

**Αναστασία Σουλελέ - 1084612**

Το word αυτό αποτελεί περιγραφή της δουλειάς μας πάνω στο project.

Στις επόμενες σελίδες θα βρείτε αναλυτικά τις λειτουργίες του κώδικα με εικόνες και μέσα που μας βοήθησαν να τον υλοποιήσουμε.

Ο κώδικας **δεν** υπάρχει σε αυτό το word, γιατί θεωρήσαμε ότι η προσθήκη του θα καθιστούσε την αναφορά «κουραστική» στον αναγνώστη.

Για να βρείτε τον πλήρη κώδικα καθώς και κάθε κομμάτι της εργασίας ξεχωριστά ανατρέξτε στον φάκελο “Κώδικας”.

# Περιεχόμενα

### PART 1………………………………………………………………………………………………………

(1)

Insertion sort & Quick sort……………………………………………………………………………….. 5

(2)

Heap sort & Counting Sort…………………………………………………………………………………9

(3)

Αλγόριθμοι Δυαδικής Αναζήτησης και Αναζήτησης με παρεμβολή………………………..13

(4)

Αλγόριθμος Δυικής Αναζήτησης Παρεμβολής(BIS)……………………………………………….16

##### PART 2……………………………………………………………………………………………………………

(A,Β)

Δένδρο AVL διατεταγμένο ως προς την ημερομηνία……………………………………………….18

(Γ)

Hashing…………………………………………………………………………………………………………………….23

##### Βιβλιογραφία…………………………………………………………………………………………………..29

# Part.1.1

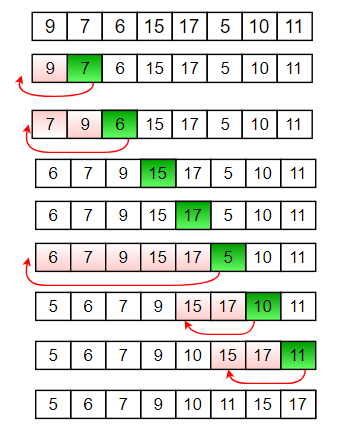
-----------------------------------------(Περιγραφη)-----------------------------------------

* **Αλγόριθμος Insertion sort:**

Ο Insertion Sort είναι ένας απλός αλγόριθμος ταξινόμησης που λειτουργεί παρόμοια με τον τρόπο που ταξινομούμε τα τραπουλόχαρτα στα χέρια μας. Θεωρούμε το 1ο στοιχείο του πίνακα ως ταξινομημένο και το συγκρίνουμε με το 2ο στοιχείο. Αν το 2ο στοιχείο είναι μεγαλύτερο από το 1ο παραμένουν στη θέση τους, αν είναι μικρότερο αφαιρούμε το 2ο στοιχείο , «σπρώχνουμε» τα στοιχεία του πίνακα μια θέση δεξιά και εισάγουμε την τιμή του 2ου στοιχείου στον πίνακα. Επαναλαμβάνουμε την διαδικασία για κάθε μη ταξινομημένο στοιχείο. Οι τιμές των μη ταξινομημένων στοιχείων επιλέγονται και τοποθετούνται στη σωστή θέση.

Worst case : O(n^2) Average case : O(n^2)

Οπτική απεικόνιση λειτουργίας του αλγορίθμου insertion sort:

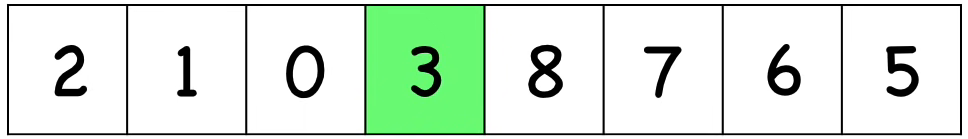


* **Αλγόριθμος Quick sort:**

Ο Quick sort είναι ένας αλγόριθμος «διαίρει και βασίλευε». Λειτουργεί επιλέγοντας ένα στοιχείο ‘pivot’ από τον πίνακα και χωρίζοντας τα άλλα στοιχεία σε δύο υπο-πίνακες, ανάλογα με το αν είναι μικρότερα ή μεγαλύτερα από το ‘pivot’.

Worst case : O(n^2) Average case : O(nlogn)

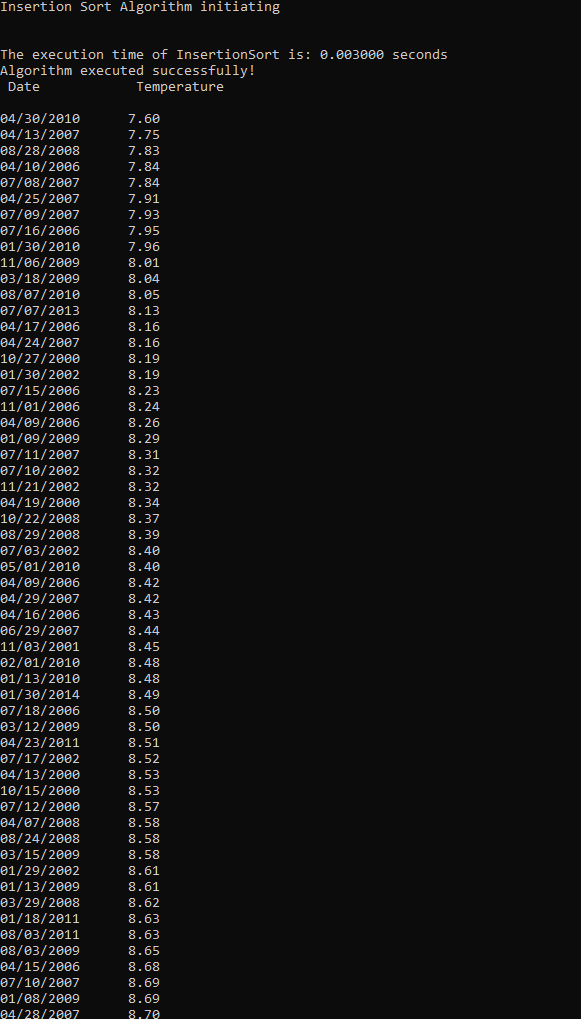
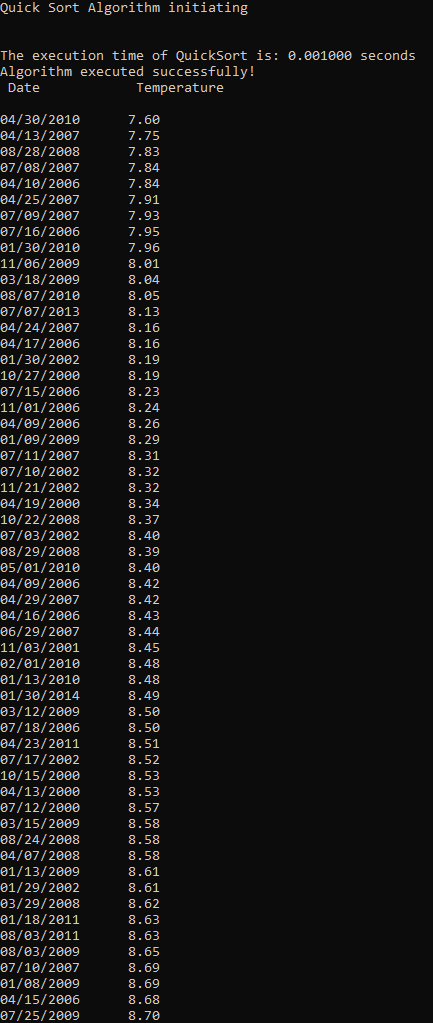
Στον παρακάτω πίνακα έχει επιλεχθεί ο αριθμός 3 ως pivot και ο πίνακας έχει διαχωριστεί ανάλογα, ώστε το 3 να είναι στην σωστή του θέση :



Ο αλγόριθμος Quick sort είναι επαναλαμβανόμενος , οπότε επαναλαμβάνουμε την ίδια διαδικασία για διαφορετικό pivot μέχρι να εξαντλήσουμε όλα τα στοιχεία του πίνακα και να φτάσουμε στο τελικό αποτέλεσμα:



-----------------------------------( screenshot αποτελεσματων κωδικα)--------------------------------------



# Part.1.2

Για τη υλοποίηση του δεύτερου ερωτήματος που αφορά τους αλγορίθμους ταξινόμησης Heap Sort και Counting Sort υλοποιήσαμε ένα struct, το οποίο περιλαμβάνει τα πεδία της ημερομηνίας και των τιμών phosphate, προκειμένου να μπορούμε με μεγαλύτερη ευχέρεια να πραγματοποιήσουμε την αντιστοίχιση μεταξύ της ημερομηνίας με την τιμή phosphate που της αναλογεί.

* **Αλγόριθμος Heap Sort:**

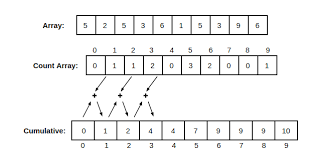
Η ιδέα αυτού του αλγορίθμου ταξινόμησης είναι ότι πρώτα δημιουργούμε τον σωρό τοποθετώντας τα δεδομένα στοιχειά τα οποία έχουμε, ξεκινώντας από το πρώτο στοιχειό τοποθετώντας το στην ρίζα του σωρού και με την ιδία διαδικασία από τα αριστερά προς τα δεξιά μέχρι τα φύλλα. Στην συνέχεια, δημιουργούμε έναν μέγιστο σωρό στον οποίο ο γονέας είναι πάντα μεγαλύτερος ή ίσος από τα παιδιά του. Όταν πλέον δημιουργηθεί ο μέγιστος σωρός το μεγαλύτερο στοιχειό βρίσκεται στην ρίζα και το εναλλάσσουμε με το τελευταίο στοιχειό. Εν τέλει το αφαιρούμε από τον σωρό και επαναλαμβάνουμε την ίδια διαδικασία μέχρι να παραμείνει μόνο ένα στοιχείο.

Worst case : O(nlogn) Average case : O(nlogn)

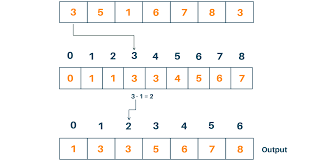
* **Αλγόριθμος Counting Sort:**

Η ιδέα αυτού του αλγορίθμου ταξινόμησης είναι ότι τα στοιχεία ταξινομούνται μετρώντας τον αριθμό εμφανίσεων κάθε μοναδικού στοιχείου και η καταμέτρηση αυτών αποθηκεύεται σε έναν βοηθητικό πίνακα. Η ταξινόμηση γίνεται σε αύξουσα σειρά αντιστοιχίζοντας την καταμέτρηση ως δείκτη του βοηθητικού πίνακα. πίνακα count με μέγεθος την τιμή του μέγιστου στοιχείου+1. Στην συνέχεια, αποθηκεύουμε το πλήθος κάθε στοιχείου -το πόσες φορές εμφανίζεται το στοιχείο-στον πίνακα count. Έπειτα, δημιουργούμε ένα συσσωρευτικό άθροισμα το οποίο προκύπτει με την εξής διαδικασία:

Worst case : O(n + k) Average case : O(n + k)



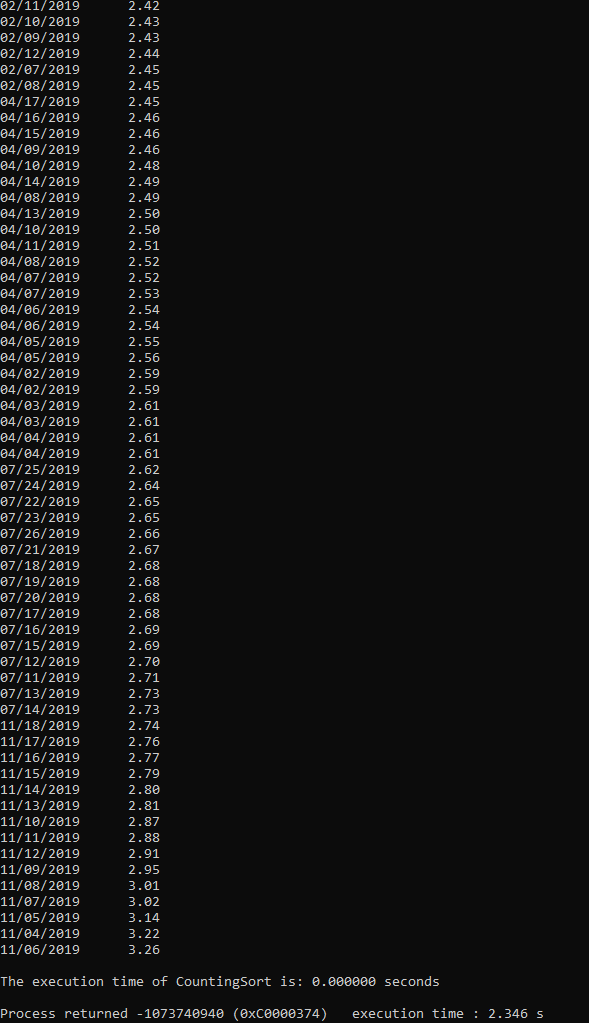
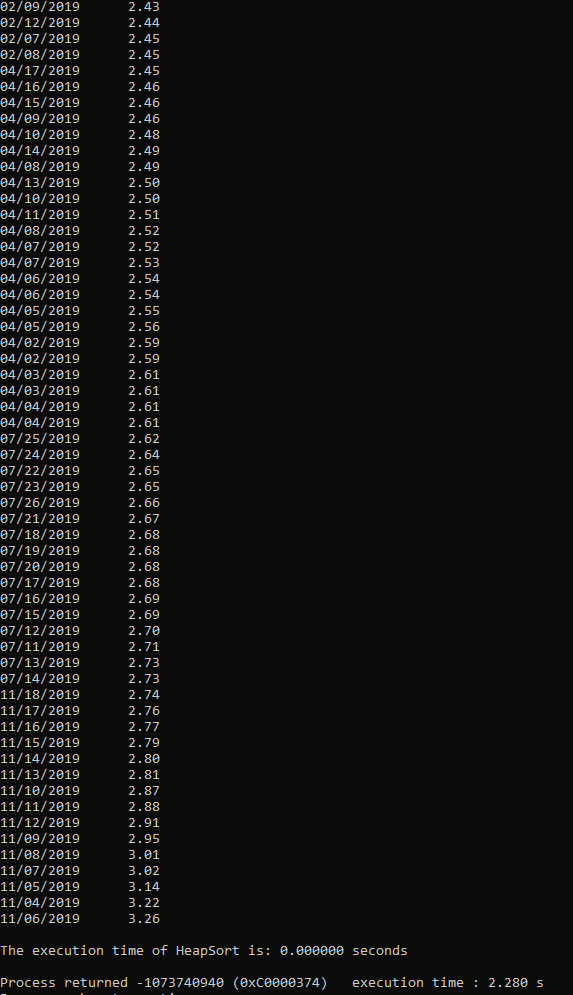
To τελικό αποτέλεσμα προκύπτει με την εξής διαδικασία:



Στον συγκεκριμένο κώδικα, επειδή είχαμε να διαχειριστούμε floating numbers, έπρεπε πρώτα να μετατρέψουμε τους δεκαδικούς σε ακέραιους προκειμένου να μπορέσει να λειτουργήσει ο αλγόριθμος Counting Sort , ο οποίος για την ταξινόμηση χρησιμοποιεί τις θέσεις ενός πίνακα που είναι ακέραιοι.

Τέλος, με τις συναρτήσεις της βιβλιοθήκης time, clock\_t begin=clock() και clock\_t end=clock(), μετρήσαμε τον χρόνο εκτέλεσης των αλγορίθμων Heap Sort και Counting Sort. Με αυτό υπάρχει το πρόβλημα ότι ο χρόνος που κάνουν να τρέξουν οι αλγόριθμοι είναι πάρα πολύ μικρός άρα το μετράει σαν 0s άρα δεν μπορούμε να τους συγκρίνουμε.

-----------------------------------( screenshot αποτελεσματων κωδικα)--------------------------------------

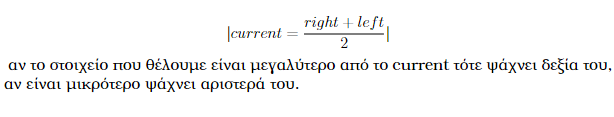


## Part.1.3

* **Αλγόριθμος δυαδικής αναζήτηση:**

Η δυαδική αναζήτηση είναι ένας αποτελεσματικός αλγόριθμος για την εύρεση ενός στοιχείου σε μια ταξινομημένη λίστα στοιχείων. Λειτουργεί διαιρώντας επανειλημμένα στο μισό το τμήμα της λίστας που θα μπορούσε να περιέχει το στοιχείο, μέχρι οι πιθανές τοποθεσίες να περιοριστούν σε μόνο μία.

(screenshot από Latex)



Worst case : O(logn) Average case : O(logn)

Οπτική απεικόνιση λειτουργίας του αλγορίθμου binary search:

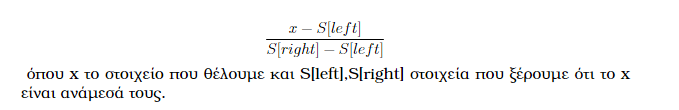


* **Αλγόριθμος αναζήτησης με παρεμβολή:**

Έστω ταξινομημένος πίνακας με **n** ομοιόμορφα κατανεμημένες τιμές **arr[]**. H αναζήτηση παρεμβολής είναι μια βελτίωση σε σχέση με τη δυαδική αναζήτηση, για περιπτώσεις όπου οι τιμές σε έναν ταξινομημένο πίνακα κατανέμονται ομοιόμορφα. Ένα πρακτικό παράδειγμα θα ήταν η αναζήτηση σε ένα λεξικό.Αν ψάχνεις μια λέξη που ξεκινάει από Α θα ανοίξεις το λεξικό στην αρχή και όχι στη μέση (λειτουργία binary search). Η αναζήτηση παρεμβολής μπορεί να μεταβεί σε διαφορετικές τοποθεσίες ανάλογα με το πόσο απέχει αλφαβητικά η σελίδα που βρισκόμαστε από την ζητούμενη λέξη. Για παράδειγμα, εάν η λέξη μας ξεκινάει με Ω, η αναζήτηση παρεμβολής είναι πιθανό να ξεκινήσει την αναζήτηση προς το τέλος του λεξικού.

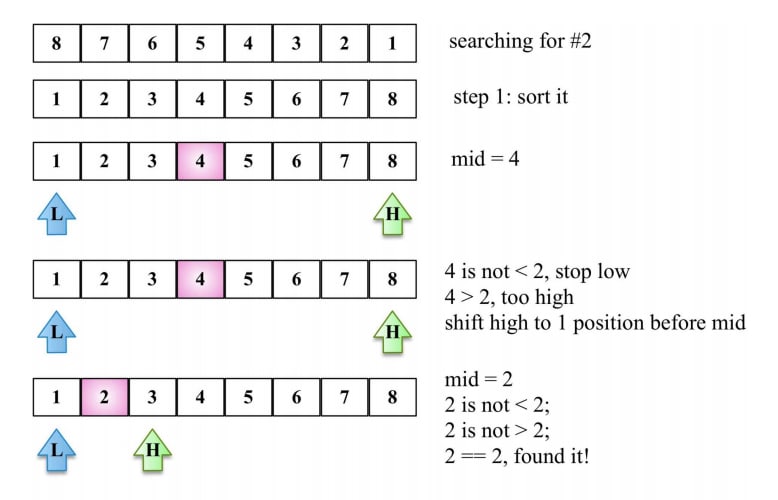
Για να βρει τη θέση προς αναζήτηση, χρησιμοποιεί τον ακόλουθο τύπο:

(screenshot από Latex)



Worst case : O(n) Average case : O(log(logn))

Οπτική απεικόνιση λειτουργίας του αλγορίθμου interpolation search:



-----------------------------------( screenshot αποτελεσματων κωδικα)--------------------------------------



# Σχόλια:

Στο 3 και στο 4 έχουμε βάλει ένα #define list temp το οποίο δείχνει τι ταξινομούμε/ψάχνουμε στο πρόγραμμα μας. Αν το αλλάξουμε σε #define list pho θα ταξινομεί και θα βρίσκει ημερομηνία για τιμή phosphate. Κάλλιστα θα μπορούσαμε να είχαμε φτιάξει δύο πίνακες που αποθηκευόντουσαν οι τιμές temp/phosphate και να επιλέγαμε ποιόν θα ψάξουμε και ποιόν θα ταξινομίσουμε, ή θα μπορούσαμε να είχαμε φτιάξει δύο διαφορετικές συναρτήσεις που η μία έκανε ταξινόμηση temp ενώ η άλλη pho. Επιλέξαμε όμως να το κάνουμε με αυτόν τον τρόπο για να μην γίνει τεράστιο το πρόγραμμα και για να μην χαλάμε περισσότερο run time για την αρχικοποίηση δύο πινάκων 1405 στοιχείων

# Part.1.4

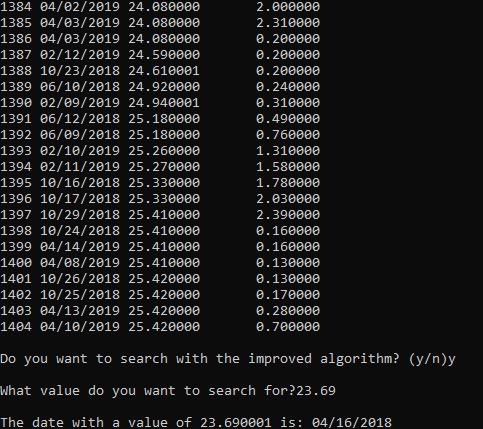
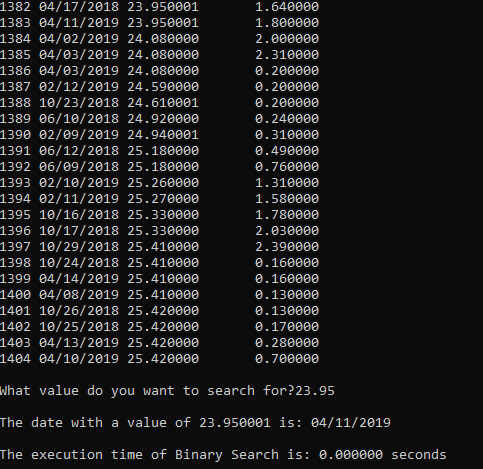
-----------------------------------------(Περιγραφη)-----------------------------------------

* **Αλγόριθμος Δυικής Αναζήτησης Παρεμβολής(BIS)**

Ο αλγόριθμος Δυϊκής αναζήτησης παρεμβολής είναι μια ένωση των δύο προηγούμενων αλγορίθμων που προτάθηκε από τους Perl and Reingold. Κάνουμε βήματα παρεμβολής και μετά ψάχνουμε γραμμικά κάνοντας βήματα **√size** για να βρούμε το διάστημα που περιέχει το x.

Worst case : O(n) Average case : O(log(logn))

-----------------------------------( screenshot αποτελεσματων κωδικα)--------------------------------------



# Σχόλια:

Προσθέσαμε την περίπτωση που η μεταβλητή που χρειαζόμαστε είναι η πρώτη ή η τελευταία επειδή δεν μπορούσε να τις βρει ο αλγόριθμος.

Ο αλγόριθμος crashαρει σε περίπτωση που η τιμή που ψάξαμε δεν υπάρχει στο διάστημα τιμών οπότε προσθέσαμε μια if πριν τον καλέσουμε.

Υπήρξαν περιπτώσεις που ο αλγόριθμος λόγω του εκθετικού βήματος ξεπέρναγε τις διαστάσεις του πίνακα, γιαυτό πρόσθεσαμε στις while την συνθήκη να τρέχει όσο είναι στα όρια του πίνακα

Ακόμα για λίγες τιμές όμως ο δεύτερος improved αλγόριθμος δεν μπορεί να βρεί την τιμή, υποθέτουμε ότι αυτό έχει να κάνει με την κατασκεύη του αλγόριθμου επειδή δεν μπορούσαμε να βρούμε κάποιο λάθος στην λογική του κώδικα.

Στον υπόλοιπο αλγόριθμο δεν έχουμε βάλει σχολιασμούς μιας και είναι ο ψευδοκώδικας του βιβλίου.

PART.2.1 + PART 2.2

-----------------------------------------(Περιγραφη)-----------------------------------------

* **Αλγόριθμος δέντρου AVL**

Ένα δέντρο AVL είναι ένας τύπος **binary search tree**. Έχοντας πάρει το όνομά του από τους εφευρέτες τους Adelson, Velskii και Landis, τα δέντρα AVL έχουν την ιδιότητα της δυναμικής αυτοεξισορρόπησηςκ,εκτός από όλες τις άλλες ιδιότητες που παρουσιάζονται από τα δυαδικά δέντρα αναζήτησης.

Ένα **BST** είναι μια δομή δεδομένων που αποτελείται από κόμβους. Διαθέτει τις εξής εγγυήσεις:

* Κάθε δέντρο έχει έναν κόμβο ρίζας (στην κορυφή)
* Ο ριζικός κόμβος έχει μηδέν, έναν ή δύο θυγατρικούς κόμβους
* Κάθε θυγατρικός κόμβος έχει μηδέν, έναν ή δύο θυγατρικούς κόμβους
* Κάθε κόμβος έχει έως δύο παιδιά
* Για κάθε κόμβο, οι αριστεροί απόγονοί του είναι μικρότεροι από τον τρέχοντα κόμβο, ο οποίος είναι μικρότερος από τους δεξιούς απόγονους

Τα δέντρα **AVL** έχουν επιπλέον εγγύηση:

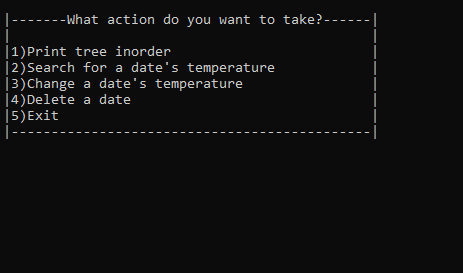
* Η διαφορά μεταξύ του βάθους του δεξιού και του αριστερού υποδέντρου δεν μπορεί να είναι περισσότερο από ένα. Αυτή η διαφορά ονομάζεται συντελεστής ισορροπίας.

Προκειμένου να διατηρηθεί αυτή η εγγύηση, μια εφαρμογή ενός AVL θα περιλαμβάνει έναν αλγόριθμο για την εξισορρόπηση του δέντρου όταν η προσθήκη ενός πρόσθετου στοιχείου θα διαταράξει αυτήν την εγγύηση

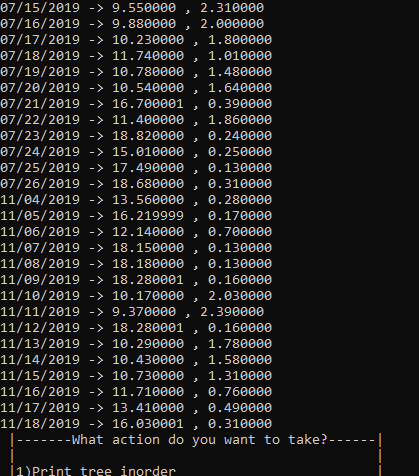
Insertion,Deletion: O(logn) Και Searching: O(n)

-----------------------------------( screenshot αποτελεσματων κωδικα)--------------------------------------

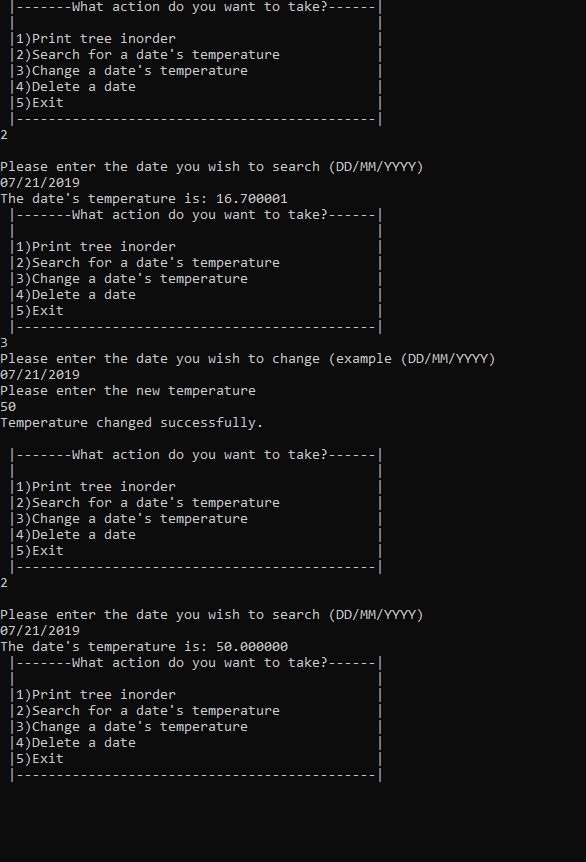
Menu της επιλογής AVL με ημερομηνία



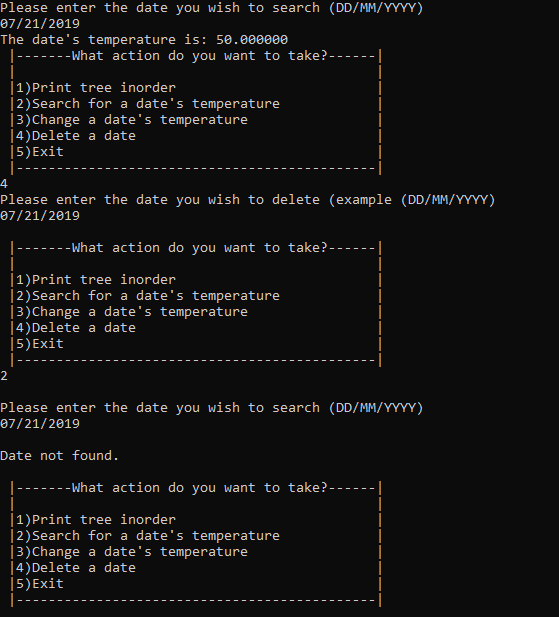
Λειτουργία Print Tree Inorder



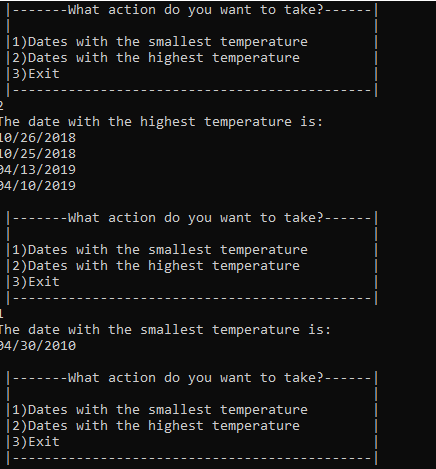
Λειτουργία Search,Change



Λειτουργία Search,Delete

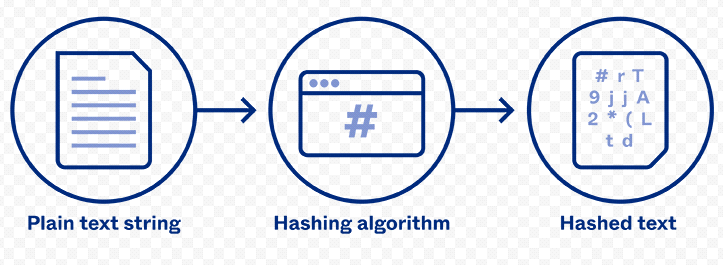


Menu και λειτουργίες της επιλογής AVL με θερμοκρασία



PART 2.3

***Hashing***

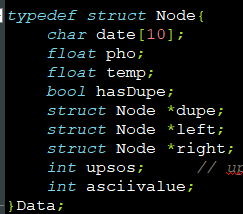


Υλοποίηση με Hashing algorithm: Για την μέθοδο του hashing χρησιμοποίησα την συνάρτηση που ήταν προτεινόμενη στην instruction set δηλαδή το άθροισμα των ημερομηνιών με την διαίρεση ενός πρώτου αριθμού για να αποφύγουμε τα πολλά collisions.

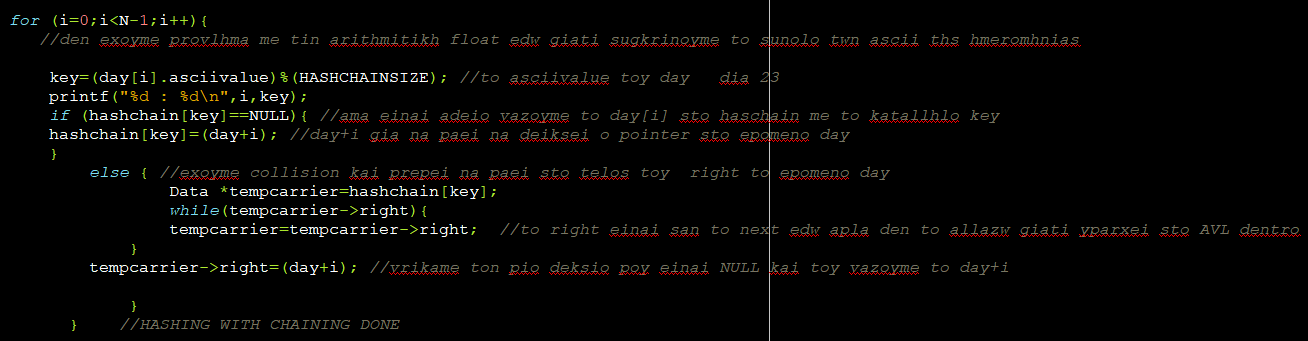
Ο πρώτος αριθμός που επέλεξα σε αυτήν την περίπτωση είναι το 23,καθώς υπολόγισα το μέγιστο και ελάχιστο άθροισμα των ημερομηνιών και κατέληξα σε αυτό το νούμερο. Με το υπόλοιπο απο την διαίρεση καθορίζω το bucket που θα μπει.

Προσθέτω μέσα στο Node μια int μεταβλητή που διατηρεί για κάθε ημερομηνία το

Asciivalue της.



Έτσι,σε κάθε Day θα υπάρχει η θερμοκρασία,το string της ημερομηνίας,ο επόμενος κόμβος που θα δείχνει,καθώς και το asciivalue .

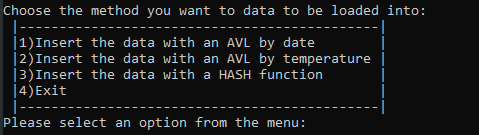


Με αυτόν τον κώδικα υλοποιώ τα buckets που θα μπουν οι ημερομηνίες.Στην αρχή γίνεται ο υπολογισμός του key,το bucket που θα μπει η ημερομηνία.Έπειτα, γίνεται έλεγχος στην περίπτωση που το bucket είναι άδειο προστίθεται κατευθείαν,ειδάλλως «ταξιδεύουμε» μέσω στου tempcarrier στον πιο «δεξί» κόμβο,μέχρι να βρούμε τον next που είναι κενός και να προστεθεί εκεί η ημερομηνία.

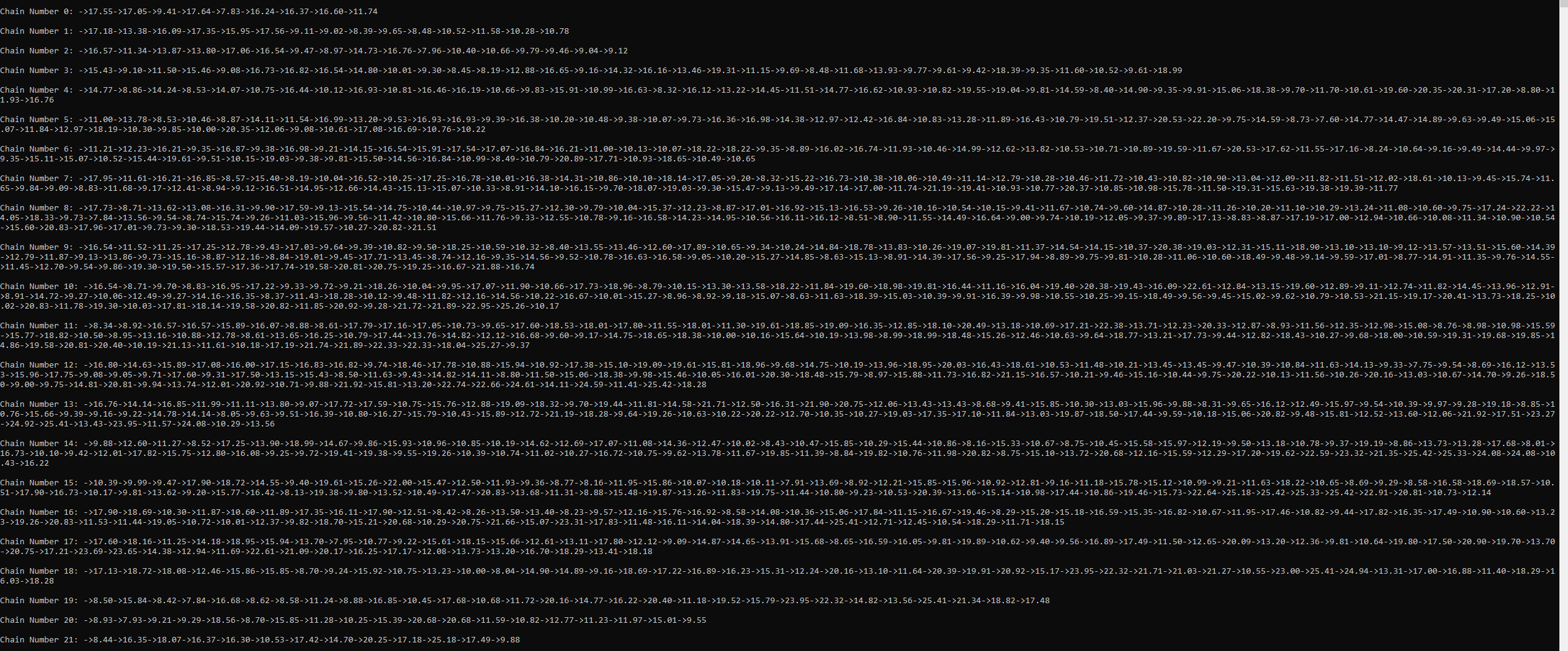
Ένα σημαντικό πλεονέκτημα του hashing είναι ότι η προσθήκη σε ένα hashchain δεν γίνεται με το float της θερμοκρασίας αλλά με το Int του asciivalue. Έτσι,γλυτώνουμε τους πολλάπλασιασμούς για την μετατροπή των float σε int εξοικονομώντας χρόνο στις πράξεις,αφού γίνονται κατευθείαν οι συγκρίσεις.

Το αποτέλεσμα είναι το επιθυμητό:

ΜΕΝΟΥ:

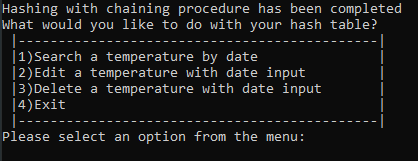


Hashing:



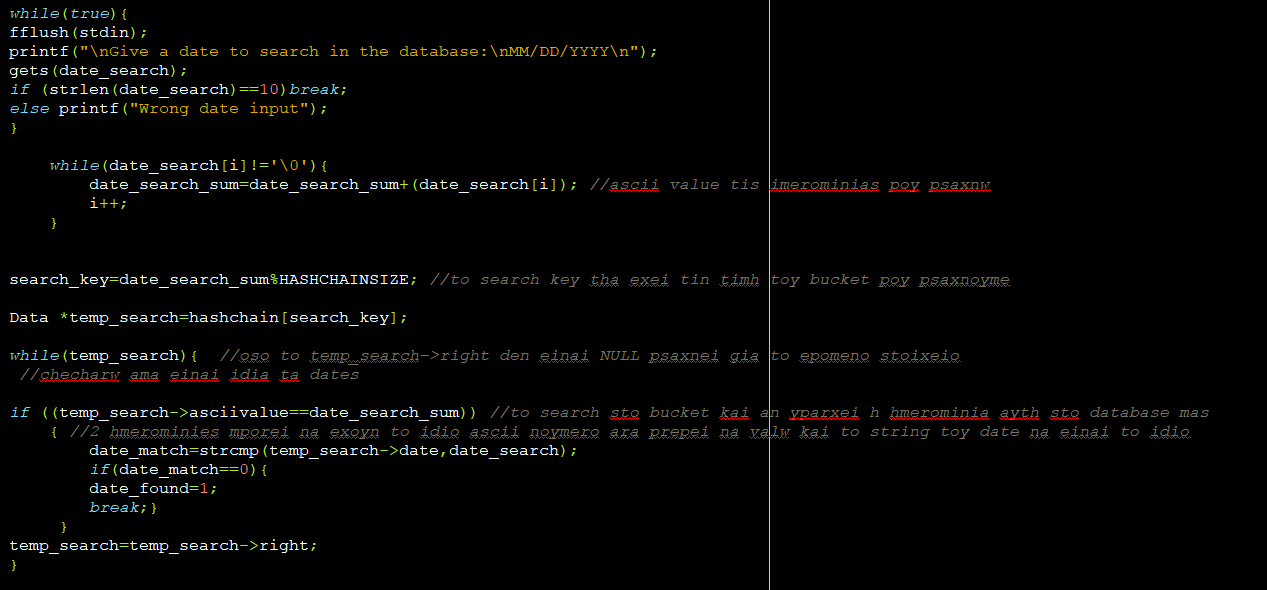
Έχουμε 23 buckets που κάθε ημερομηνία μπορεί να μπει ανάλογα με το key της.

Στην συνέχεια,έχουμε ακόμη ένα μενου που δίνει στον χρήστη την ικανότητα να διαλέξει 4 επιλογές.



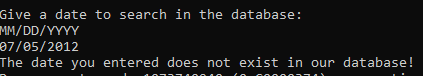
Σε όλες τις υλοποιήσεις ,εκτός από την σύγκριση του asciivalue του string που δίνει ο χρήστης, γίνεται σύγκριση και με το string της ημερομηνίας που ψάχνουμε με αυτή που δόθηκε. Αυτό γινέται,διότι πολλές ημερομηνίες μπορεί να τυχαίνει να έχουν το ίδιο asciivalue σε ένα bucket. Οπότε, άμα δεν γίνει η σύγκριση του date string μπορεί να έχουμε λάθος αποτέλεσμα.

Παρακάτω είναι το βασικό structure που εντοπίζει την ημερομηνία που δώθηκε και στις 3 επιλογές αναζήτηση/τροποποίηση/διαγραφή και γίνονται οι ανάλογες δράσεις:

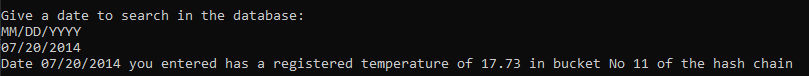


(1) Αναζήτηση θερμοκρασίας μέσω ημερομηνίας:

Στην περίπτωση που δεν υπάρχει:

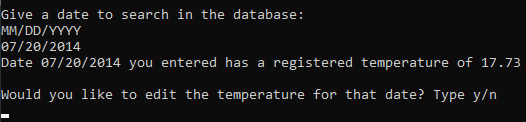


**Εάν υπάρχει η ημερομηνία:**

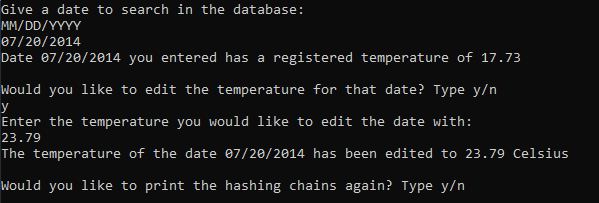


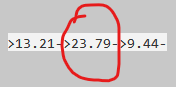
**Εντοπίζει την ημερομηνία και εκτυπώνει την θερμοκρασία και τον αριθμό του bucket που βρίσκεται.**

(2) Τροποποίηση θερμοκρασίας μέσω ημερομηνίας:

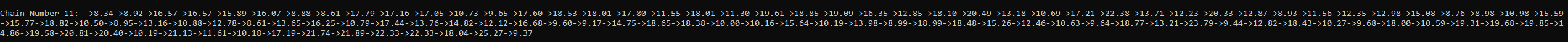


**Βρισκεί την ημερομηνία και δίνει την επιλογή στον χρήστη άμα θέλει να την τροποποιήσει ή όχι.**

**Μετά την τροποποίηση δίνεται η επιλογή για την εκτύπωση του hashing chain όπου μπορούμε να δούμε την αλλαγή.**







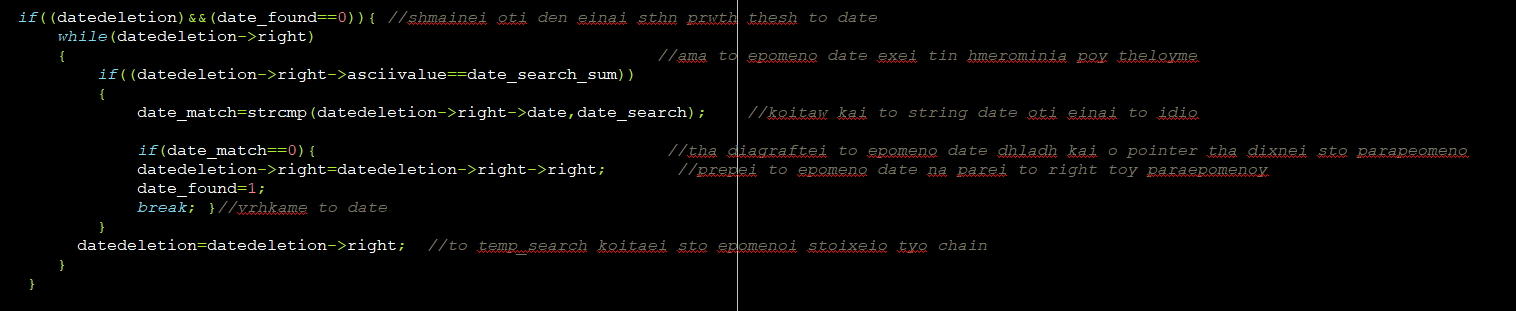
**Στην περίπτωση που δεν θέλει ο χρήστης να γίνει τροποποίηση το πρόγραμμα κλείνει .**

(3) Διαγραφή θερμοκρασίας μέσω ημερομηνίας:

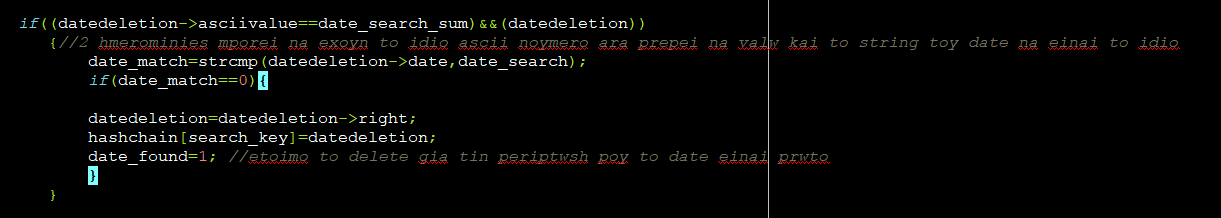
Παρόλο που η αναζήτηση και η τροποποίηση είναι ήταν παρόμοιες διαδικασίες η διαγραφή είναι διαφορετική. Καταρχάς, επειδή έχουμε υλοποιήσει τα days με στατικό τρόπο δεν καθίσταται δυνατή η διαγραφή διαφόρων days όπως άμα είχαμε malloc με free. Αυτό που μπορούμε να κάνουμε είναι ο δείκτης του next να δείχνει στο παραεπόμενο στοιχείο, «διαγράφοντας» το έτσι από το chain που ανήκει.

Έτσι,αφού βρούμε την ημερομηνία που ζητείται,πρέπει να γίνει η παραπάνω διαδικασία που περιέγραψα.

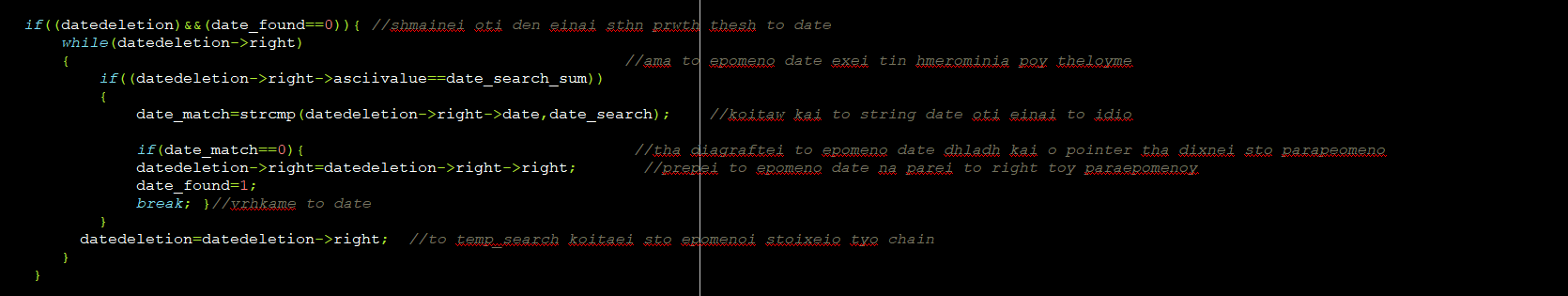
Αυτό επιτυγχάνεται με τον ακόλουθο κώδικα:



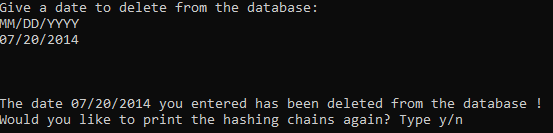
Υπάρχουν δύο περιπτώσεις να είναι το πρώτο στοιχείο στην αλυσίδα,κάτι που θα πρέπει να διαχειριστουμε ξεχωριστά καθώς δεν ψάχνουμε στο next αλλά στο πρώτο.

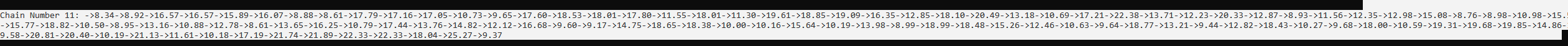


Στην περίπτωση που δεν είναι στο πρώτο στοιχείο,τότε προχωράμε κατα μήκος της αλυσίδας μέσω του datedeletion->right μέχρι να βρούμε το string date που πρέπει να διαγραφτεί.



Αφού το βρούμε το στοιχείο που βρισκόμαστε το datedeletion->right παίρνει την τιμή datedeletion->right=datedeletion->right->right ,δηλαδή του παραεπόμενου. Έτσι,δεν υπάρχει πλέον δείκτης που να δείχνει στο στοιχείο που διαγράφτηκε.

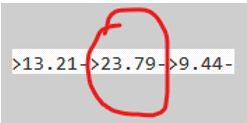








**Ενώ,πριν ηταν**



**Άρα έχουμε επιτυχή διαγραφή.**

(BIBLIOGRAFIA)

<https://www.geeksforgeeks.org/recursive-insertion-sort/>

<https://www.youtube.com/watch?v=Hoixgm4-P4M>

<https://dev.to/christinamcmahon/linear-binary-and-interpolation-search-algorithms-explained-55ni>

<https://www.youtube.com/watch?v=MFhxShGxHWc>