

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ  
ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

Email: [Dionys@ceid.upatras.gr](mailto:Dionys@ceid.upatras.gr)

Γλώσσα :C++



ΠΑΝΕΠΙΣΤΗΜΙΟ  
ΠΑΤΡΩΝ  
UNIVERSITY OF PATRAS



Δομές Δεδομένων

Project 2014 - 2015

Ονοματεπώνυμο: Φώτιος Διονυσόπουλος

AM: 5753

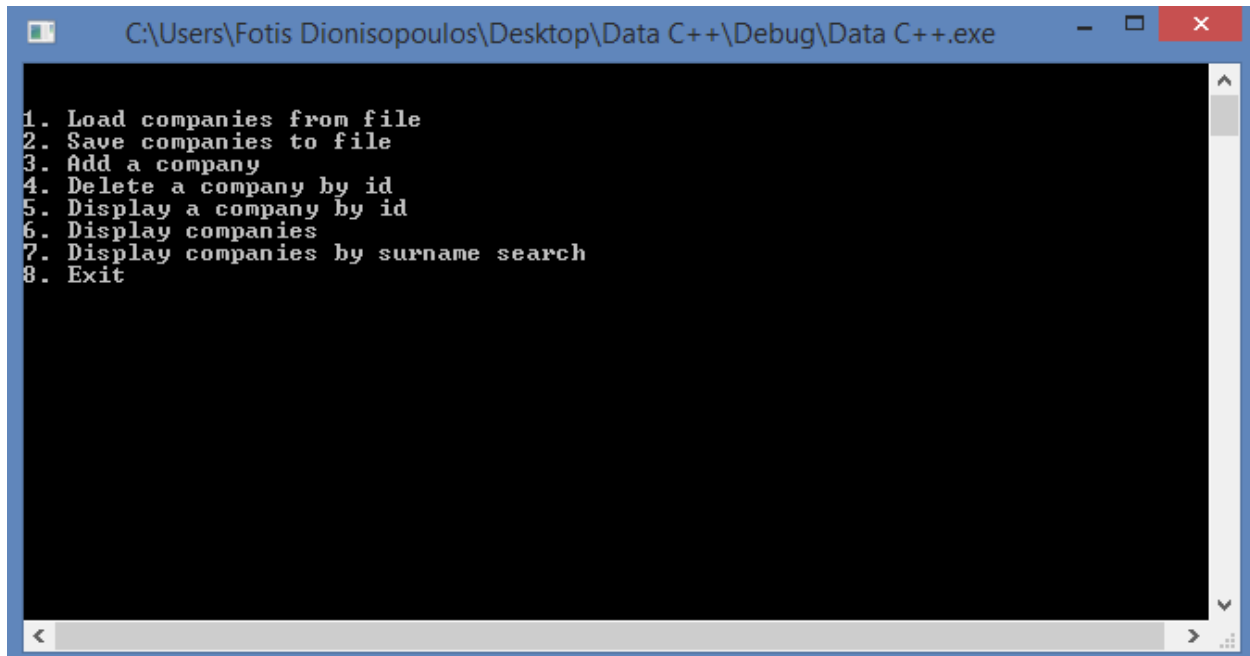
### Μέρος Α

Σκοπός αυτού του μέρους ήταν η υλοποίηση του βασικού μηχανισμού αυτής της εργασίας. Δηλαδή, η δημιουργία λογισμικού το οποίο να αποθηκεύει όλη την πληροφορία σε ειδικά διαμορφωμένες δομές δεδομένων. Το λογισμικό αυτό, είναι γραμμένο στη γλώσσα C++, η βασική δομή που επιλέχθηκε ήταν vectors.

Με την έναρξη του λογισμικού ο χρήστης δίνει συγκεκριμένο path στο οποίο βρίσκεται ένα csv αρχείο. Αρχικά, όλες οι εγγραφές από το αρχείο μεταφορτώνονται στην κύρια μνήμη όπου ο χρήστης θα μπορεί να επεξεργάζεται την συλλογή

Ενώ με τη λήξη του προγράμματος ή όταν το επιλέγει ο χρήστης, τα δεδομένα από την μνήμη θα αποθηκεύονται στο αρχείο.

Υλοποιήθηκαν οι κατάλληλες συναρτήσεις ώστε να πραγματοποιηθούν οι ακόλουθες πράξεις:



```
1. Load companies from file
2. Save companies to file
3. Add a company
4. Delete a company by id
5. Display a company by id
6. Display companies
7. Display companies by surname search
8. Exit
```

Σε αυτό το πρώτο μέρος οι πράξεις της αναζήτησης έγιναν με γραμμική αναζήτηση.

Η μορφή του αρχείου ήταν η εξής :

id;	title;	summary;	numberOfEmployees;	employee1;	employee2;	employee3;	....;	employee7
-----	--------	----------	--------------------	------------	------------	------------	-------	-----------

## Μέρος Β

Σε αυτό το μέρος υλοποιήθηκε αναζήτηση με βάση το ID με χρήση δυαδικής αναζήτησης και binary interpolation search. Αυτό έχει ως αποτέλεσμα να πρέπει να διατηρείται η συλλογή ταξινομημένη με βάση το id.

```

1. Load companies from file
2. Save companies to file
3. Add a company
4. Delete a company by id
5. Display a company by id
6. Display companies
7. Display companies by surname search
8. Exit
5
Give the ID
12
For lenear search press '1',for binary search press '2'
for binary interpolation search press '3' ,for AVL tree press '4'

```

Μόλις ο χρήστης δώσει το ID που θέλει να αναζητήσει ,εμφανίζεται ένα ακόμη μενού.Στο μενού αυτό ο χρήστης δίνει έναν αριθμό ανάλογα με την αναζήτηση που θέλει.Οι διαθέσιμες επιλογές ήταν η γραμμική αναζήτηση ,δυαδική ,δυαδική με interpolation και τέλος, αναζήτηση(αφού πρώτα είχε γίνει ένθεση όλων των στοιχείων) με AVL tree.

## Μέρος Γ

Στο τρίτο μέρος σκοπός ήταν η δημιουργία AVL tree για την αναζήτηση του ID.

Δημιουργήθηκε κλάση **AVL** στην οποία βρίσκονται οι συναρτήσεις **insert**, **del** και **find** για την ένθεση , τη διαγραφή και την εύρεση αντίστοιχα.

Επίσης , υπάρχουν συναρτήσεις για τις περιστροφές(**srl**,**srr**,**drr**,**drl**) ,συνάρτηση εύρεσης ύψους.

```

Give the ID
12
For lenear delete press '1',for AVL delete press '2'
2
Element deleted successfully

```

## Μέρος Δ

Οργανώθηκε η αναζήτηση με βάση το επίθετο με τη χρήση Digital Tries.

Για τις υλοποίηση του Digital Trie δημιουργήθηκε η κλάση **Digital\_Tries**. Σε αυτή τη κλάση βρίσκονται η συνάρτηση **load** η οποία χρησιμοποιεί την συνάρτηση **addWord** η οποία προσθέτει την λέξη στο δέντρο.

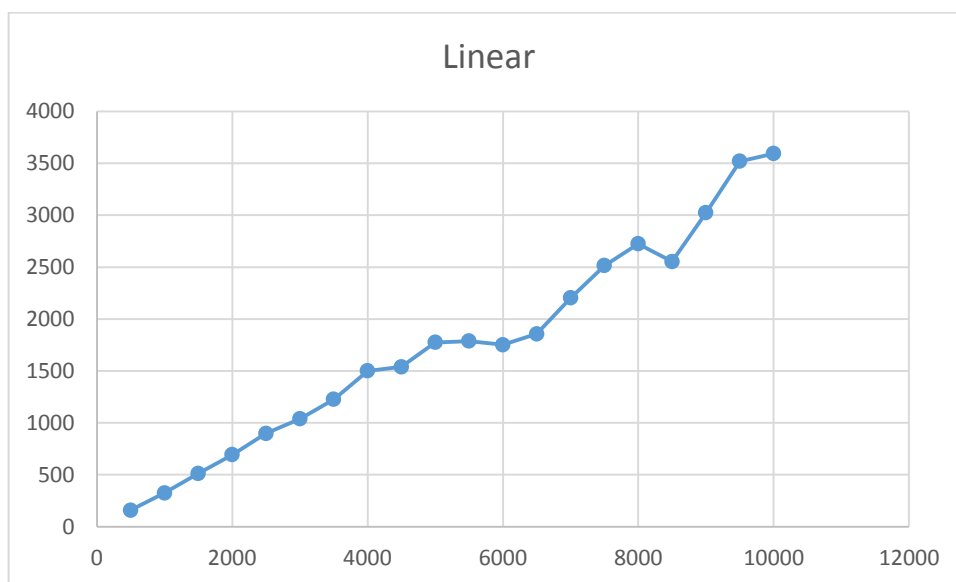
Επίσης, υπάρχει η συνάρτηση **searchWord** η οποία αναζητεί το δοθέν στοιχείο.Υπάρχει και η κλάση **Node** η οποία περιέχει δεδομένα που χρησιμοποιεί η κλάση **Digital\_Tries**. σδτέλος, υπάρχει η **findChild** η οποία παίρνει σαν όρισμα έναν χαρακτήρα και βρίσκει το παιδί.

## Μέρος Ε

Στο τελευταίο μέρος ζητήθηκε να εκτελέσουμε ένα μεγάλο αριθμό απο αναζητήσεις με βάση το ID και τα επίθετα και να μετρηθεί ο χρόνος για την εμφάνιση των αποτελεσμάτων. Μετρήθηκε ο μέσος χρόνος αναζήτησης σε συγκρίσεις και εκτελέσιμο χρόνο σε κάθε ένα από τα προηγούμενα μέρη.

Μετρήσεις:

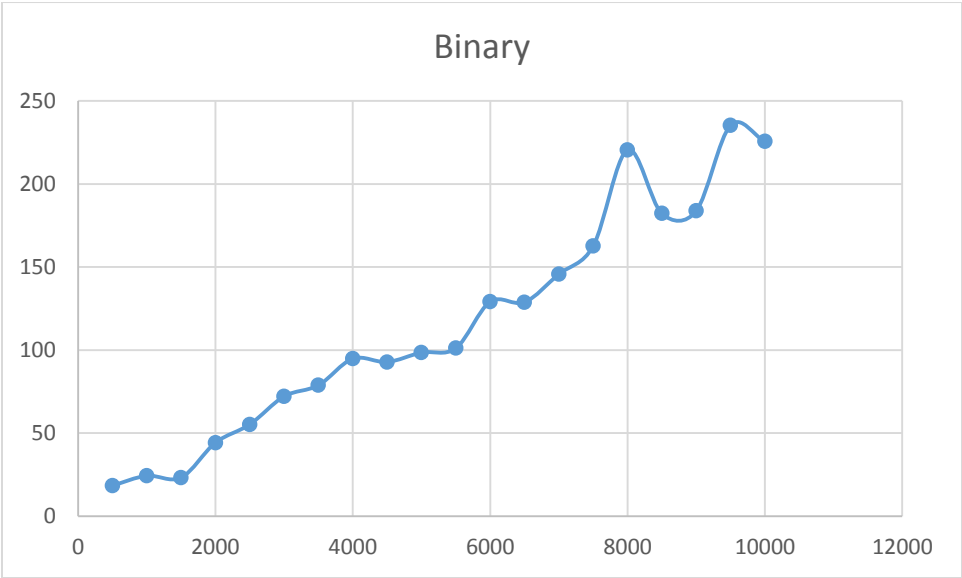
Linear M.O.= 1578.66



500	158.4
1000	325.5
1500	512.7
2000	692.8
2500	898.6
3000	1038.2
3500	1226.1
4000	1500.7
4500	1540
5000	1775.3
5500	1787.7
6000	1752.2
6500	1856.6

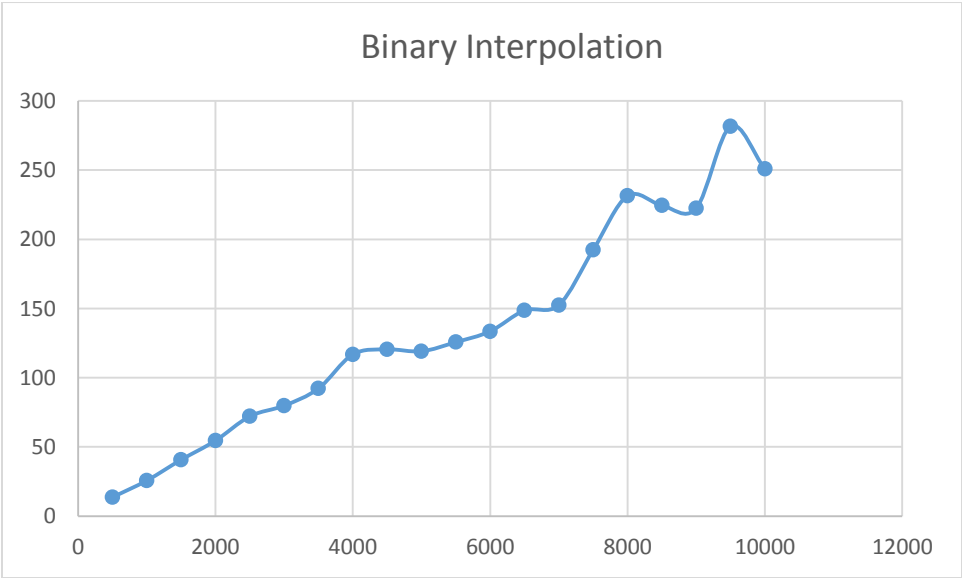
7000	2204
7500	2514.6
8000	2724.2
8500	2553.6
9000	3023.1
9500	3520.3
10000	3594.8

Binary M.O.= 115.85



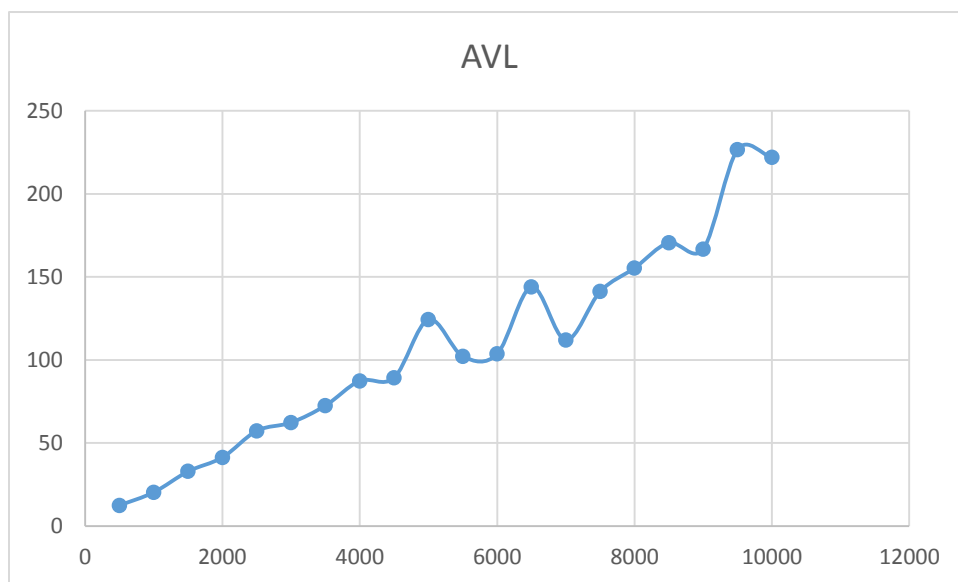
500	18.4
1000	24.3
1500	23.2
2000	44.2
2500	55.2
3000	72.1
3500	78.8
4000	94.8
4500	92.8
5000	98.5
5500	101.1
6000	129.1
6500	128.8
7000	145.7
7500	162.6
8000	220.5
8500	182.2
9000	183.8
9500	235.3
10000	225.6

Binary Interpolation: M.O.= 134.92



1000	25.7
1500	40.7
2000	54.5
2500	72.1
3000	79.8
3500	92.2
4000	116.8
4500	120.5
5000	119.1
5500	125.7
6000	133.5
6500	148.6
7000	152.5
7500	192.4
8000	231.4
8500	224.5
9000	222.4
9500	281.6
10000	250.8

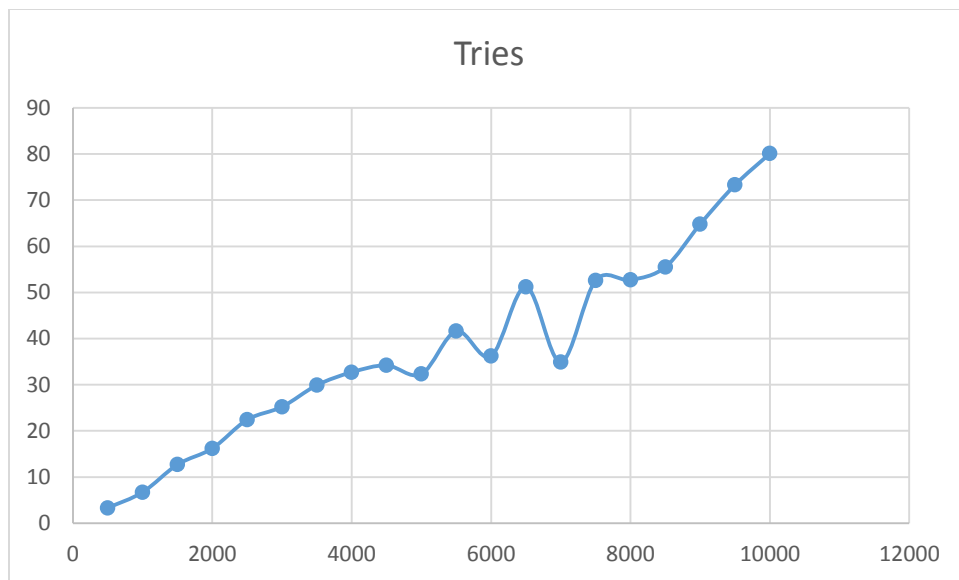
AVL M.O.= 107.2



500	12.4
1000	20.3
1500	32.9
2000	41.3
2500	57.3
3000	62.3
3500	72.5
4000	87.4
4500	89.3
5000	124.3
5500	102.2
6000	103.6
6500	143.9
7000	112
7500	141.2
8000	155.3
8500	170.6
9000	166.7
9500	226.6
10000	221.9

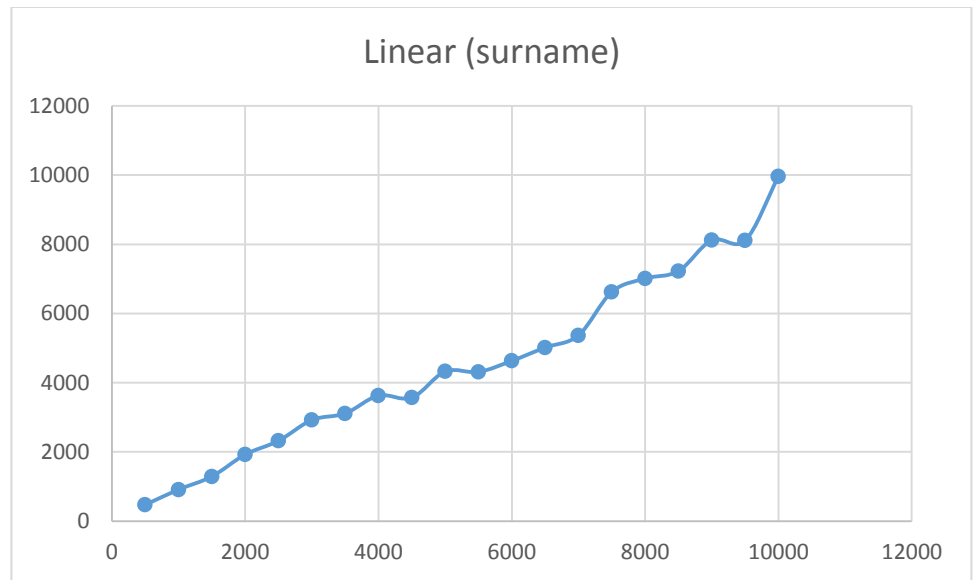
Tries M.O.= 37.925





500	3.3
1000	6.7
1500	12.7
2000	16.2
2500	22.4
3000	25.2
3500	29.9
4000	32.7
4500	34.2
5000	32.3
5500	41.6
6000	36.2
6500	51.2
7000	34.9
7500	52.6
8000	52.7
8500	55.5
9000	64.8
9500	73.3
10000	80.1

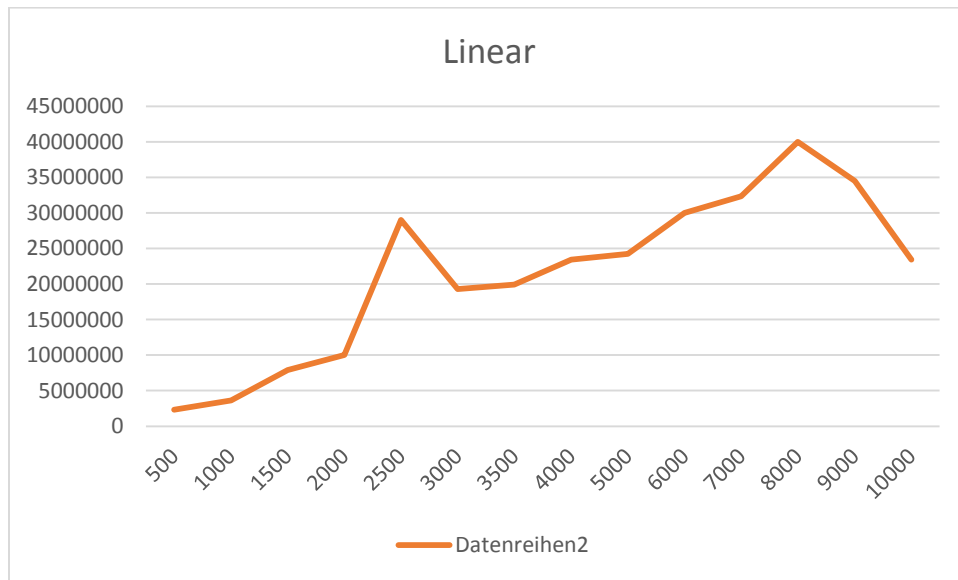
Linear search of surname M.O.= 4542.93



500	471.5
1000	911.8
1500	1290.4
2000	1925.3
2500	2319.9
3000	2922.2
3500	3112.5
4000	3628.8
4500	3567.3
5000	4325.5
5500	4312.2
6000	4632.4
6500	5017.2
7000	5369.3
7500	6622.8
8000	7012.2
8500	7225.4
9000	8122.1
9500	8110.1
10000	9959.7

Αριθμός Συγκρίσεων:

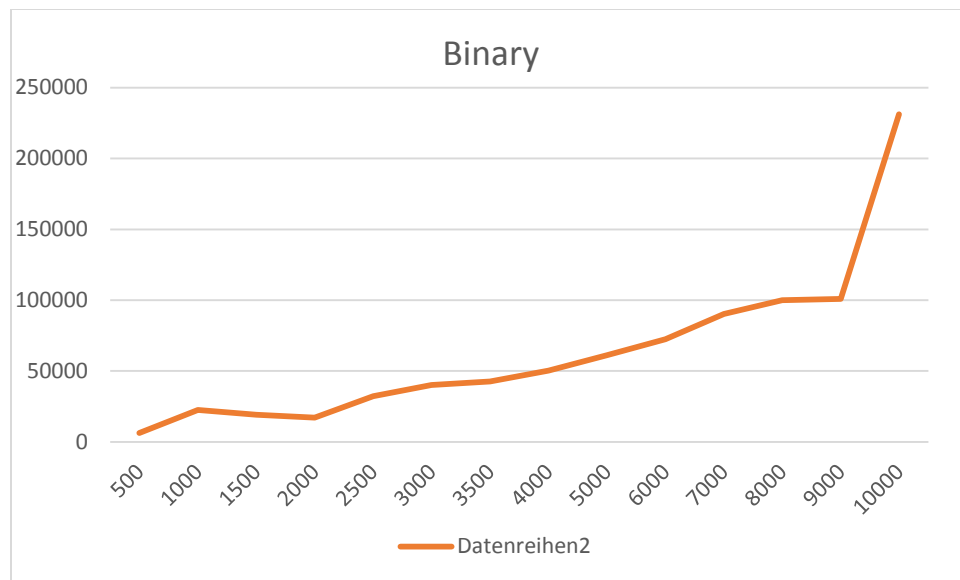
Linear M.O.= 21434798.57



500	1000	1500	2000	2500	3000	3500
2312447	3647753	7897631	10004683	29004553	19283390	19924331

4000	5000	6000	7000	8000	9000	10000
23431562	24251270	30007692	32351550	40016923	34502191	23451204

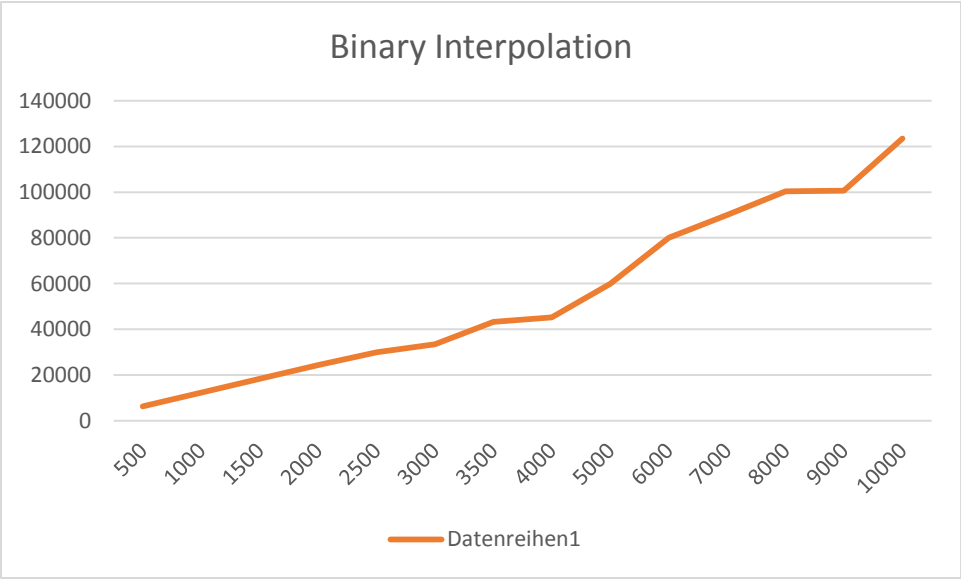
Binary M.O.= 63279



500	1000	1500	2000	2500	3000	3500
6153	22563	19026	17092	32332	40055	42553

4000	5000	6000	7000	8000	9000	10000
50245	61255	72339	90291	99990	100787	231225

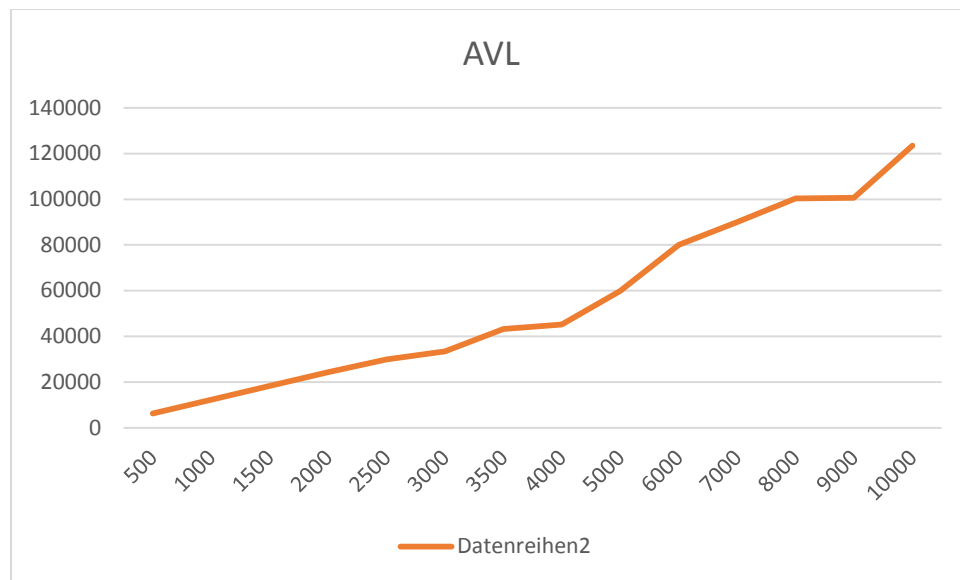
Binary Interpolation M.O.= 26180.21429



500	1000	1500	2000	2500	3000	3500
2924	6223	10190	12738	20089	23122	22352

4000	5000	6000	7000	8000	9000	10000
23435	32353	40447	33330	54530	52304	32486

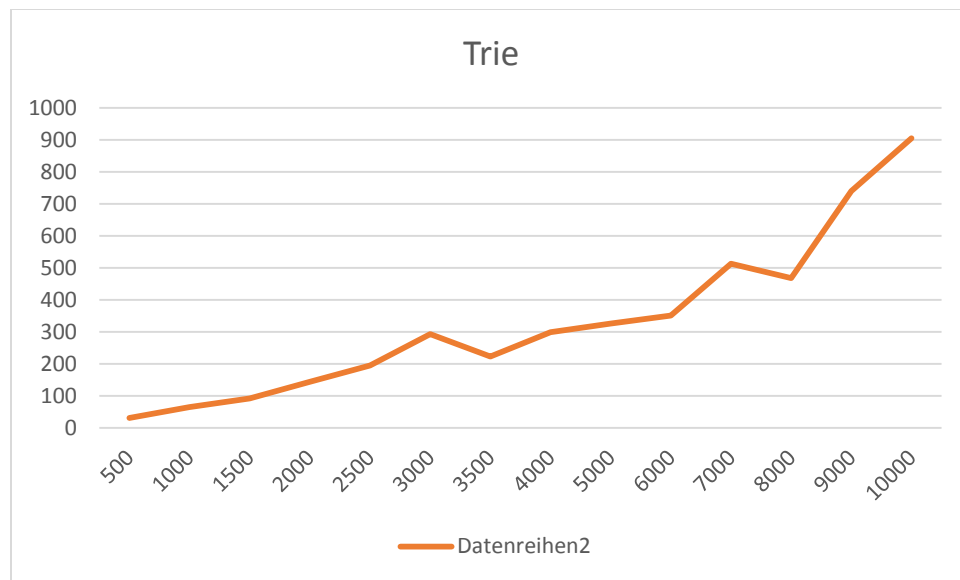
AVL M.O.= 54787.85714



500	1000	1500	2000	2500	3000	3500	4000
6171	12235	18232	24274	29962	33450	43221	45124

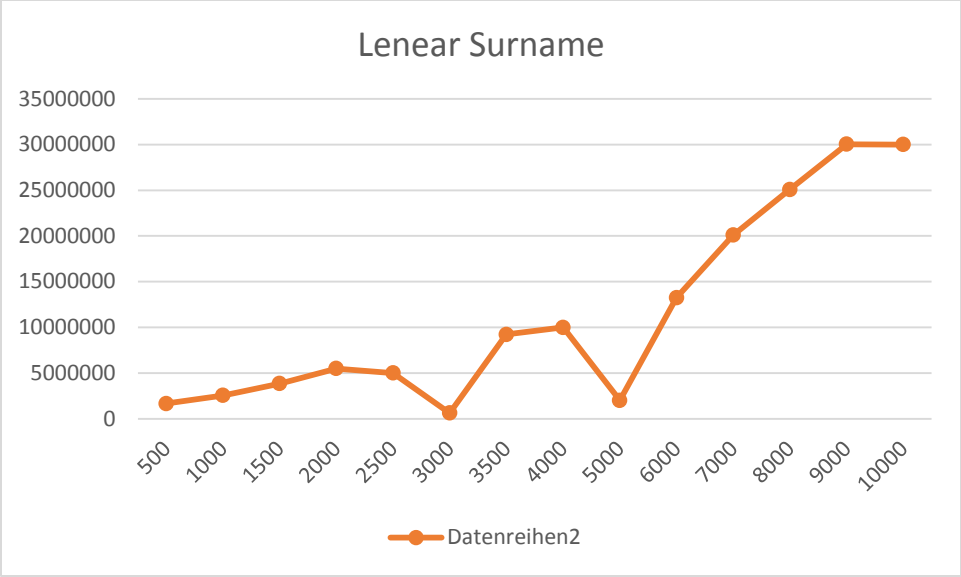
5000	6000	7000	8000	9000	10000
59938	79994	89977	100354	100650	123448

Αναζήτη ονόματος:Trie M.O.= 331.7857143



500	1000	1500	2000	2500	3000	3500	4000	5000	6000	7000	8000	9000	10000
31	65	92	144	195	293	223	299	326	351	513	468	740	905

Αναζήτη ονόματος:Linear M.O.= 11346175.21



500	1000	1500	2000	2500	3000	3500	4000
1656739	2548799	3837839	5483776	5004861	627066	9225679	9992742

5000	6000	7000	8000	9000	10000
1999549	13252939	20086233	25086239	30037626	30006366



Τα αποτελέσματα με βάση τη θεωρία(έστω για size=1000):

Στη γραμμική αναζήτηση:  $\text{size} \cdot O(n) = 1000 \cdot 10000 = 10000000 > 3647753$

Επομένως, φράσσεται από τη μέγιστη θεωρητική τιμή.

Στη Δυαδική:  $\text{size} \cdot O(\log n) = 1000 \cdot 13,2 = 13200 < 22563$

Binary Interpolation:  $\text{size} \cdot O(\log \log n) = 1000 \cdot 3,7 = 3700 < 6223$

Ένω στη χειρότερη περίπτωση  $\text{size} \cdot O(\sqrt{n}) = 1000 \cdot 31,62 = 31620$

AVL:  $\text{size} \cdot O(\log n) = 1000 \cdot 13,2 = 13200 < 22563$

Trie: Το αγγλικό λεξικό έχει 26 γράμματα άρα  $\log 26 = 4,8$ , το μέγεθος της λέξης είναι  $k$ , επομένως  $O(k)$  πράξεις και  $4,8k$  συγκρίσεις. Έστω size=10 και  $k=20$  το όνομα με μέγιστο μέγεθος:

Γραμμική αναζήτηση ονόματος:  $\text{size} \cdot O(mk) = 10 \cdot 40000 \cdot 20 = 8000000$

Αναζήτηση Trie:  $\text{size} \cdot O(k) = 10 \cdot 20 = 200$

Παρατηρήσεις: Οι χρόνοι του AVL και της δυαδικής αναζήτησης είναι περίπου ίσοι. Η δυαδική αναζήτηση με παρεμβολή παρουσιάζει τον καλύτερο χρόνο, κάτι το οποίο είναι αναμενόμενο. Ενώ, η γραμμική αναζήτηση για μεγάλο Size είναι η χειρότερη σε απόδοση.

Τα παραπάνω αποτελέσματα συμφωνούν αρκετά με την θεωρία. Τυχόν αποκλίσεις οφείλονται κυρίως στο γεγονός ότι ο κώδικας χρειάζεται βελτιστοποίηση.