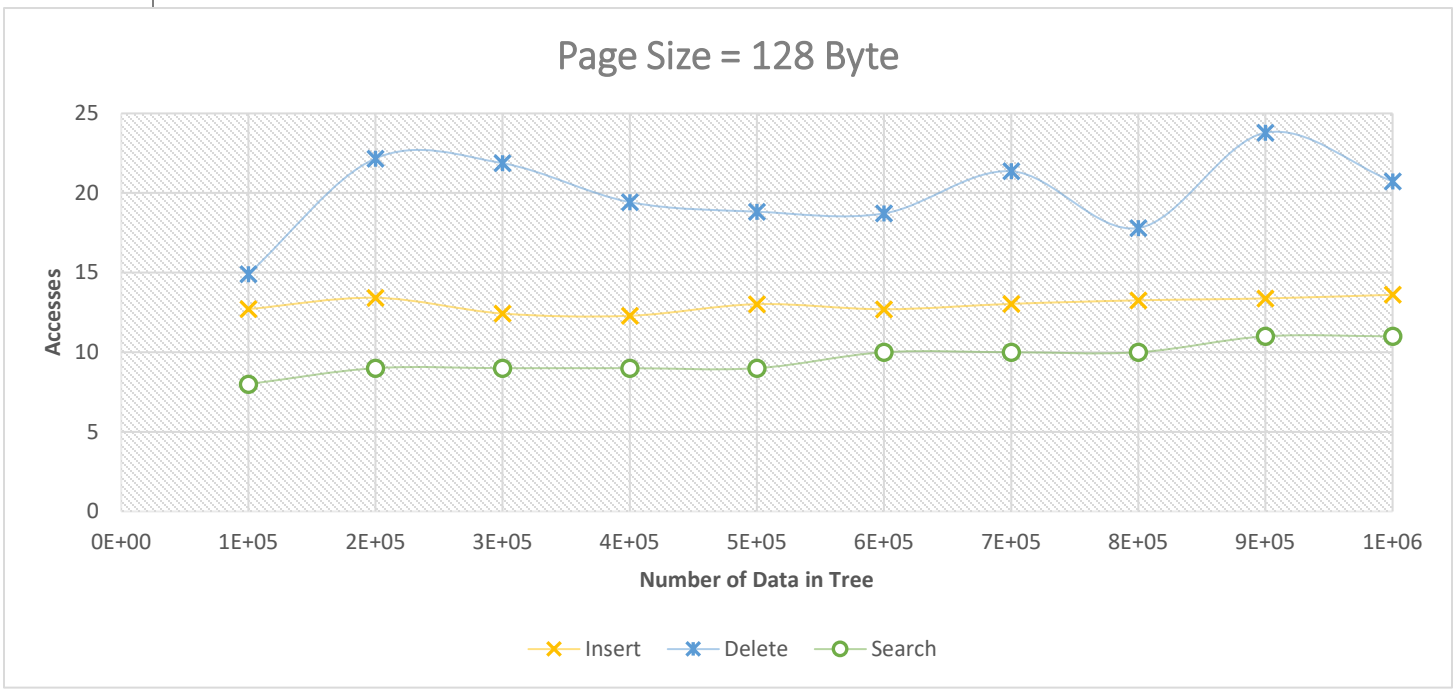
Δομές Δεδομένων - Άσκηση 3

Για το πρώτο μέρος οι μετρήσεις ήταν οι εξής :

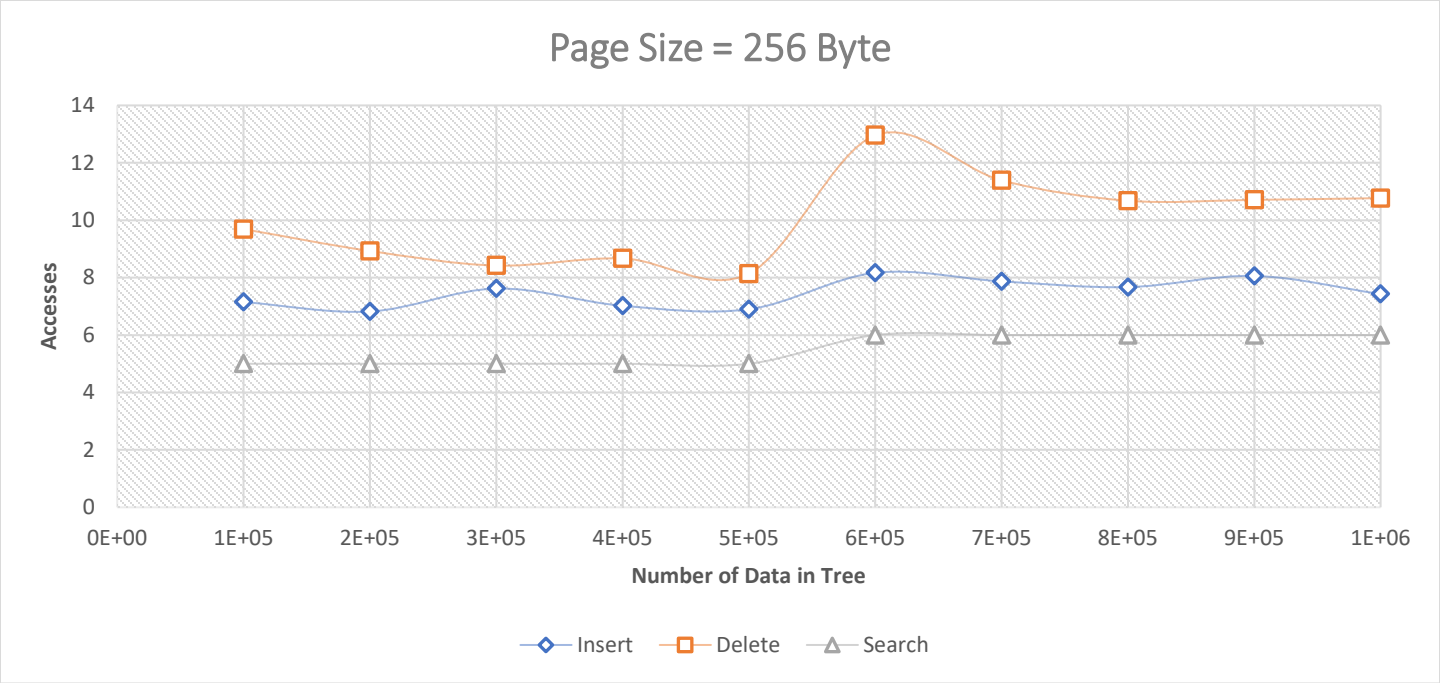
Page Size = 128b

Data	$1 \cdot 10^5$	$2 \cdot 10^5$	$3 \cdot 10^5$	$4 \cdot 10^5$	$5 \cdot 10^5$	$6 \cdot 10^5$	$7 \cdot 10^5$	$8 \cdot 10^5$	$9 \cdot 10^5$	$10 \cdot 10^5$
Insert	12.71	13.42	12.43	12.29	13.02	12.7	13.04	13.26	13.38	13.61
Delete	14.91	22.17	21.87	19.43	18.83	18.72	21.37	17.8	23.79	20.73
Search	8.00	9.00	9.00	9.00	9.00	10.00	10.00	10.00	11.00	11.00



Page Size = 256b

Data	$1 \cdot 10^5$	$2 \cdot 10^5$	$3 \cdot 10^5$	$4 \cdot 10^5$	$5 \cdot 10^5$	$6 \cdot 10^5$	$7 \cdot 10^5$	$8 \cdot 10^5$	$9 \cdot 10^5$	$10 \cdot 10^5$
Insert	7.15	6.82	7.62	7.02	6.9	8.17	7.87	7.67	8.06	7.43
Delete	9.69	8.93	8.42	8.67	8.14	12.96	11.39	10.68	10.71	10.77
Search	5.00	5.00	5.00	5.00	5.00	6.00	6.00	6.00	6.00	6.00



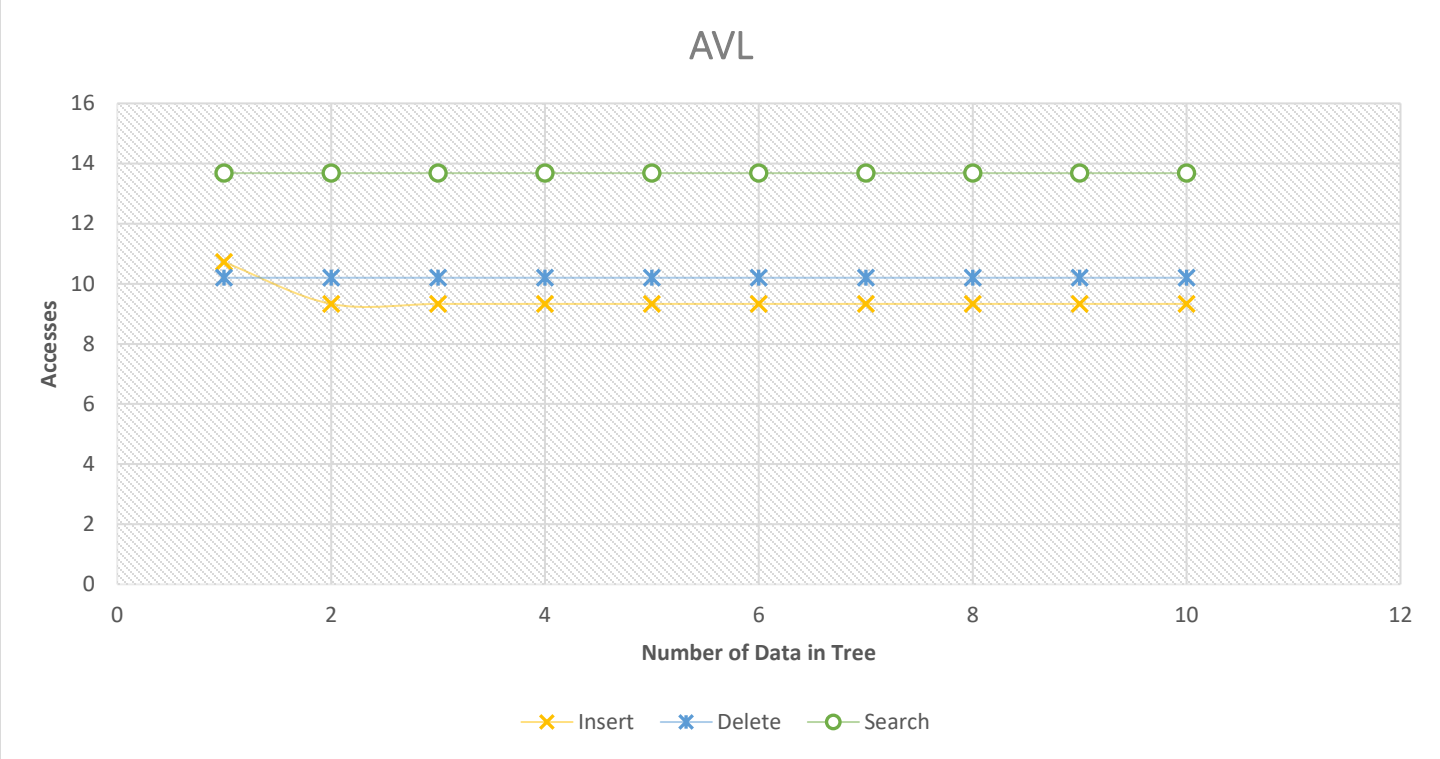
Παρατηρούμε, πως καθώς εισάγουμε δεδομένα οι μέθοδοι γίνονται *αποτελεσματικότεροι*, καθώς τα δεδομένα μοιράζονται στα data pages, και χρειάζοντια λιγότερες προσβάσεις.

Επίσης αυξάνοντας την χωριτικότητα της σελίδας, η κάθε σελίδα περιέχει περισσότερα δεδομένα, και έτσι μειώνουμε τις προσβάσεις στις υπόλοιπες σελίδες.

Μέρος Β

AVL

Data	$1 \cdot 10^5$	$2 \cdot 10^5$	$3 \cdot 10^5$	$4 \cdot 10^5$	$5 \cdot 10^5$	$6 \cdot 10^5$	$7 \cdot 10^5$	$8 \cdot 10^5$	$9 \cdot 10^5$	$10 \cdot 10^5$
Insert	10.72	9.33	9.33	9.33	9.33	9.33	9.33	9.33	9.33	9.33
Delete	10.2	10.2	10.2	10.2	10.2	10.2	10.2	10.2	10.2	10.2
Search	13.68	13.68	13.68	13.68	13.68	13.68	13.68	13.68	13.68	13.68



- Τα αποτελέσματα είναι καχύποπτα σταθερά, αλλά αυτό εξηγήτε από την *Σταθερότητα* της υλοποίησης AVL, και από το γεγονός πως μετράμε μονάχα πράξεις επάνω στα κλειδιά και όχι στα Nodes.
    - Αγνοώντας τις πράξεις στα nodes, αγνοούμε και τις μεθόδους εξισορρόπησης του δέντρου, οι οποίες θα είχαν μεγάλη βαρήτυτα στης μεθόδους Insert και Delete.
    - Αυτός είναι και ο κύριος λόγος που η Search (μία διεργασία που δεν μεταβάλει το δέντρο) φαίνεται να κάνει περισσότερες πράξεις από τις άλλες, πιο δραστικές μεθόδους.

Επεξήγηση Κώδικα

- Πακέτο ***Files.***
  - Περιέχει τα αρχεία μαζί με την κλάση ***FileOps***.
  - Η κλάση ***FileOps*** ανοίγει και διαβάζει τα αρχεία
  - Η κλάση ***FileData*** παίρνει τα δεδομένα και τα προβιβάζει στις κλάσεις που την επεκτείνουν. Οι πίνακες δεδομένων είναι οι εξής :
    - **Data** – Array 10^6 θέσεων, όπου μπαίνουν τα στοιχεία του αρχείου *keys_1000000_BE.bin.*
    - **InsertData** – Array 10^2 θέσεων, όπου μπαίνουν τα στοιχεία του αρχείου *keys_insert_100_BE.bin.*
    - **DeleteData** – Array 10^2 θέσεων, όπου μπαίνουν τα στοιχεία του αρχείου *keys_delete_100_BE.bin.*
    - **SearchData** – Array 10^2 θέσεων, όπου μπαίνουν τα στοιχεία του αρχείου *keys_search_100_BE.bin.*
- Πακέτο ***Interfaces***
  - Περιέχει το πακέτο ***Statistics.***
    - Αρκεί μία μέθοδος να κάνει implement το ***StatisticsInterface*** για να έχει πρόσβαση στις παρακάτω μεθόδους :
      - Increment() : ανεβάζει το counter κατά 1
      - Increment(int increment) : ανεβάζει το counter κατά increment.
      - Reset() : αναθέτει στο counter την τιμή 0.
      - getCounter() : επιστρέφει το counter.
  - Περιέχει την διεπαφή ***TucTree.***
    - Η διεπαφή αυτή υπάρχει για ευκολεία και περιέχει τις παρακάτω μεθόδους
      - void insert(int key) : εισάγει ένα κλειδί στο δέντρο
      - void delete(int key) : διαγράφει ένα κλειδί από το δέντρο
      - boolean search(int key) : ψάχνει εάν υπάρχει το κλειδί στο δέντρο.
      - Object ResetTree() : φτιάχνει ένα καινούργιο δέντρο και το επιστρέφει.
- Πακέτο utils – Δεν επιρεάζουν την υλοποίηση του προγράμματος και μπορούν να αγνοηθούν.
  - Περιέχει την διεπαφή ProgressBar
    - Τρέχει σε άλλο Thread ώστε να μην επιβαρύνει την τρέχων διεργασία
    - Τυπώνει στην κονσόλα το ποσοστό ολοκλήρωσης της διεργασίας.
      - Μπαίνει σε sleep, ώστε να μην επιβαρίνει το Stream της κονσόλας.
  - Περιέχει την διεπαφή ConsoleHijacker – WORK IN PROGRESS
    - Υποκλέβει τα δεδομένα που κατευθήνοντε προς την κονσόλα.
      - Τα δρομολογεί προς ένα αρχείο – για data logging.
      - Τα δρομολογεί προς ένα PrintStream που τα αγνοεί.
    - Δεν δουλεύει ακόμη όπως πρέπει – η δρομολόγιση προς αρχείο είναι ασταθής και χάνεται εύκολα. Χρειάζεται Case-Mapping και debugging.
- Πακέτο Bplus, Bminus και AVLTree
  - Υλοποίηση δέντρων από τα link που μας δώθηκαν .

Για κάθε κομμάτι της άσκηση υπάρχει και το αντίστοιχο .java αρχείο.

- Για το μέρος Α, υπάρχει το Task_A.java.
- Για το μέρος Β, υπάρχει το Task_B.java.

Μέρος Α

Οι κλάσεις **Insert**, **Delete** και **Search** παίρνουν σαν παράμετρο ένα **BPlusTree**, και τα δεδομένα που πρέπει να εισάγουν, διαγράψουν ή ψάξουν. Επίσης θέτοντας την παράμετρο **Verbose** σε true, η μέθοδος τυπώνει στην κονσόλα τα αποτελέσματα

Για τα Tests υπάρχουν 2 ρουτίνες :

- **QuickTestRoutine**

Για το κέρδος χρόνου, αφού εισάγουμε το κάθε σετ από 100.000 στοιχεία, έπειτα από την κλήση της **Delete** καλούμε την **Insert** εισάγωντας πίσω τα στοιχεία που μόλις διαγράψαμε, και αντίστοιχα για την **Insert** καλούμε την **Delete**. **Το δέντρο δεν θα είναι το ίδιο** έπειτα από αυτό, αλλά θα περιέχει τον σωστό αριθμό στοιχείων.

- **TestRoutine**
  - Χτίζει από την αρχή τα δέντρα, και κάνει επάνω τους τα Test.

Στην γραμμή 33 του TaskA.java υπάρχει η boolean μεταβλητή **QuickTest**, η οποία εάν τεθεί σε true, κατά την εκτέλεση του προγράμματος θα πραγματοποιηθεί η **QuickTestRoutine.** Αν είναι false, θα πραγματοποιηθεί η **TestRoutine**.

Μέρος Β

Για το Β ακολουθούμε την ίδια μέθοδο (**TestRoutine**) με το μέρος Α, μόνο που για κάθε **test** φτιάχνουμε ένα καινούργιο δέντρο με τις μεθόδους :

- ResetTree() : καλεί την TucTree.ResetTree() και έπειτα γεμίζει το δέντρο με την FillTree().
- FillTree() : Παίρνει σαν παράμετρο το ***numberOfData*** και εισάγει στο δέντρο ***numberOfData*** στοιχεία.