1. Υπολογισμός πολυπλοκότητας για το αρχείο sort.c

- Να δείξετε με απλά αθροίσματα όπως αυτά των διαλέξεων την πολυπλοκότητα της sort και της main
- Σκεφτείτε: Πιθανώς να βοηθήσει να θέσετε n = r - l
- Η compexch είναι ΜΙΑ πράξη και δεν προσθέτει παραπάνω πολυπλοκότητα στον αλγόριθμο. Παρ' όλα αυτά καλό θα ήταν να προσπαθήσετε να καταλάβετε τι κάνουν τα defined macros στην αρχή.
- Hint: Η main έχει 3 σημεία με ξεχωριστή πολυπλοκότητα το καθένα. Ποιο από αυτά υπερισχύει ως προς το Big O notation?
- Η δέσμευση μνήμης, η εκτύπωση και το διάβασμα εισόδου χρήστη γίνονται σε μία πράξη

2. Ταξινόμηση αλγορίθμων με βάση την πολυπλοκότητα

- Να αποδειχθεί με απλές ανισότητες η σειρά πολυπλοκότητας των αλγορίθμων.
- Σε κάποιες παραστάσεις πιθανόν να χρειαστεί απλοποίηση ώστε να γίνει η σύγκριση.

Π.χ. δύο αλγόριθμοι πολυπλοκότητας 500·2^n και 100·2^n + n

Σύγκριση: 498·2^n > 99·2^n

και 2^n + 2^n > 2^n + n

Προσθέτοντας κατα μέλη προκύπτει 500·2^n > 100·2^n + n

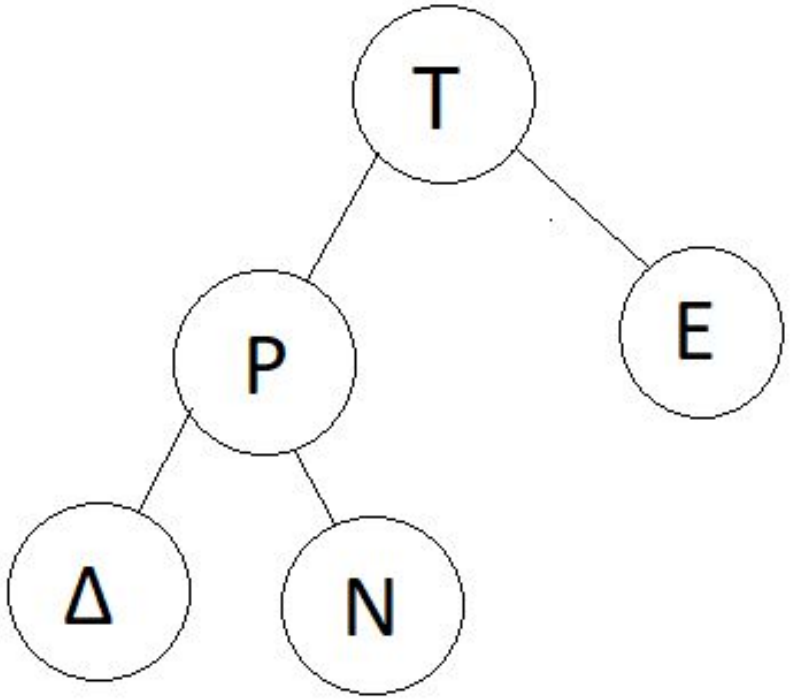
Υπενθύμιση από διαφάνειες:

$$O(1) < O(\log n) < O(n) < O(n^2) < O(n^3) < O(2^n) < O(10^n)$$

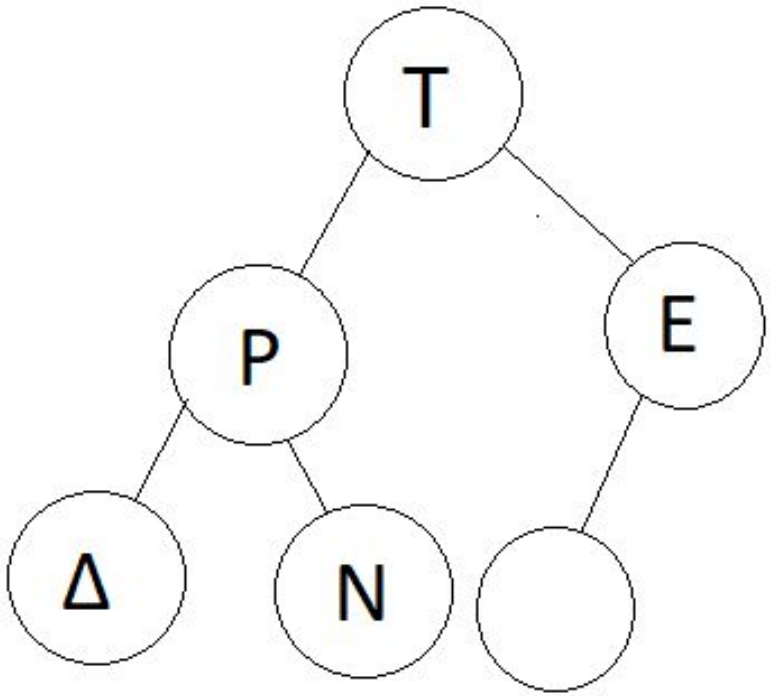
3. Εισαγωγή της λέξης `ΚΟΛΟΚΟΤΡΩΝΗΣ’ σε άδειο Σωρό.

- Σωρός: Πλήρες Δυαδικό δέντρο όπου η τιμή κάθε γονέα είναι μεγαλύτερη ή ίση των τιμών των παιδιών του.
- Η σύγκριση για τα γράμματα του αλφαβήτου ακολουθεί τη λεξικογραφική σειρά, π.χ. `Ο’ > `Κ’.

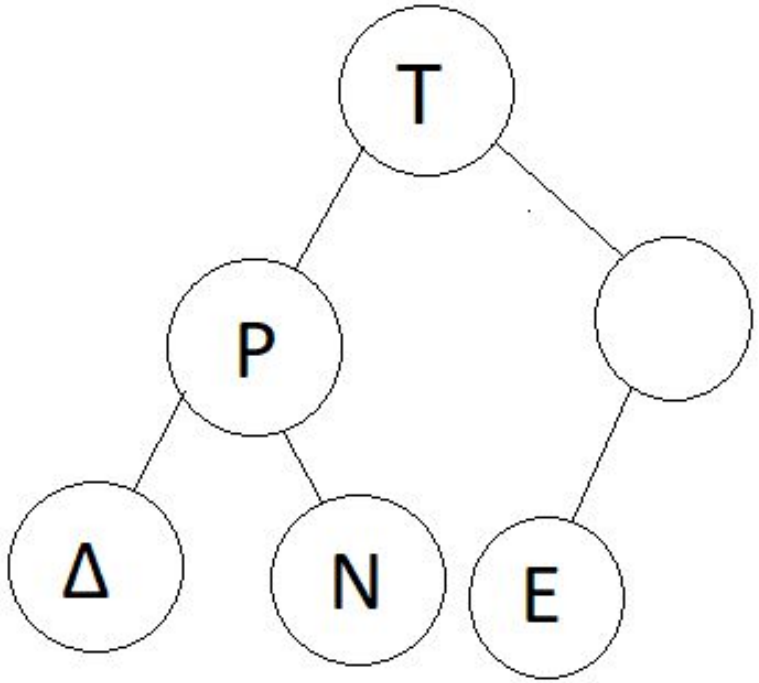
- Παράδειγμα: Έστω ότι έχουν ήδη συμπληρωθεί όλα τα γράμματα της λέξης ΔΕΝΤΡΟ στο σωρό εκτός από το τελευταίο.



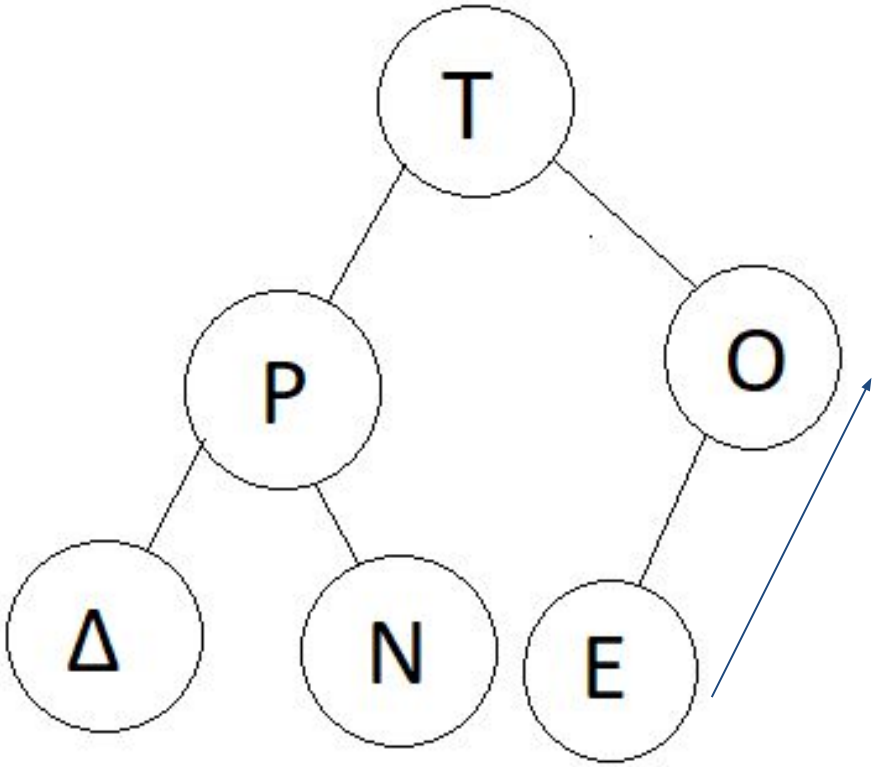
- Παράδειγμα: Έστω ότι έχουν ήδη συμπληρωθεί όλα τα γράμματα της λέξης ΔΕΝΤΡΟ στο σωρό εκτός από το τελευταίο.



- Παράδειγμα: Έστω ότι έχουν ήδη συμπληρωθεί όλα τα γράμματα της λέξης ΔΕΝΤΡΟ στο σωρό εκτός από το τελευταίο.

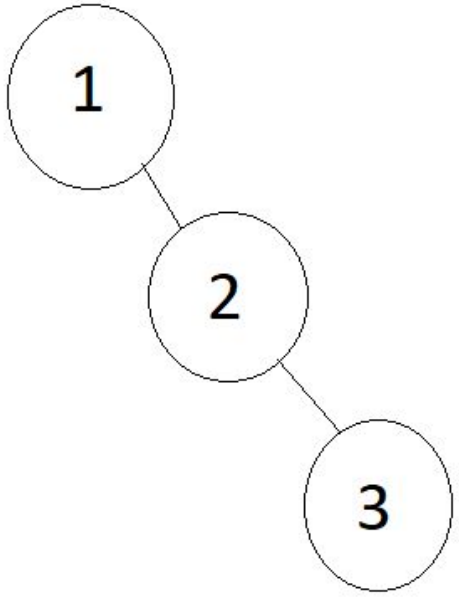


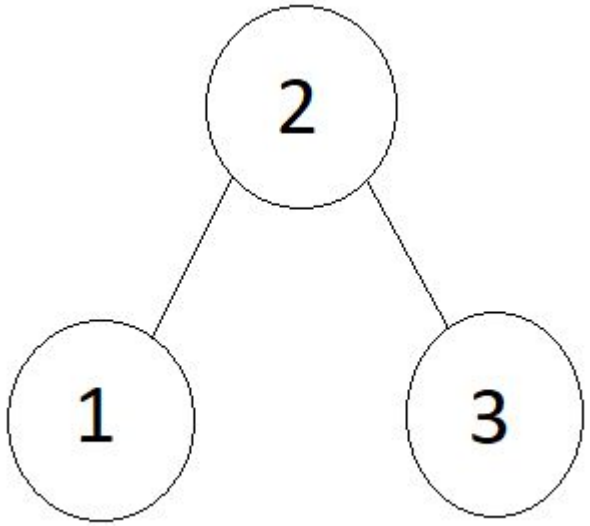
- Παράδειγμα: Έστω ότι έχουν ήδη συμπληρωθεί όλα τα γράμματα της λέξης ΔΕΝΤΡΟ στο σωρό εκτός από το τελευταίο.



4. Πλήθος διαφορετικών δυαδικών δέντρων αναζήτησης με κλειδιά: `1’, `2’, `3’ και `4’.

- Πώς μπορεί να διαφέρουν δύο δυαδικά δέντρα αναζήτησης που αποθηκεύουν τα ίδια κλειδιά;
- Hint: Παρατηρήστε παρακάτω την διαδοχική εισαγωγή των κλειδιών 1->2->3 και 2->1->3, αντίστοιχα.

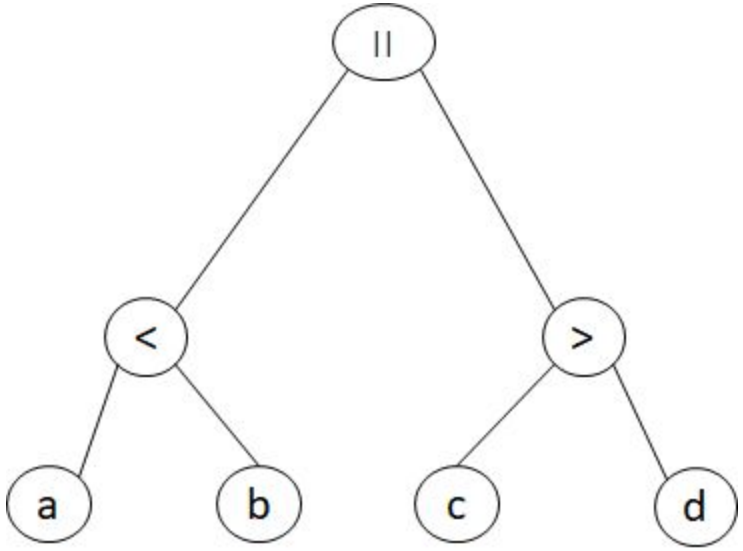




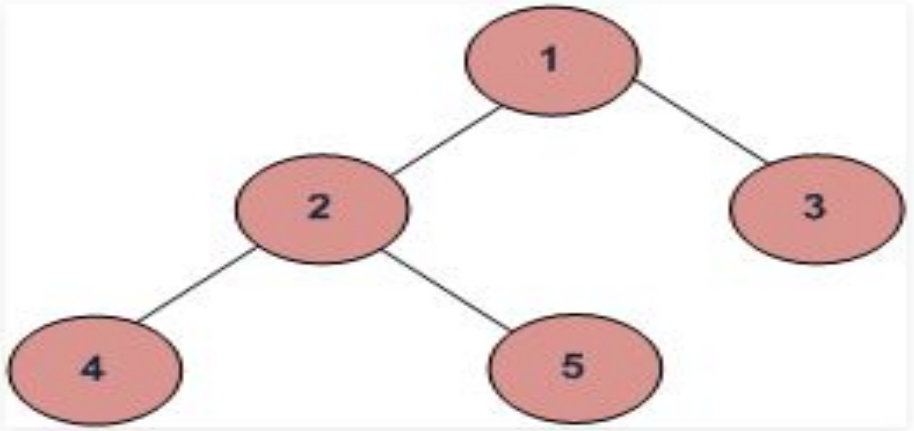
6. Δεντρο έκφρασης

- Φτιάχνω ένα δέντρο, όπου κάθε κόμβος θα είναι σύμβολο (γράμμα, πράξη κτλ) και λαμβάνω υπόψη την προσεταιριστικότητα της C.

Π.χ (a<b) || (c>d)



Παράδειγμα preorder, inorder, postorder



Example Tree

Depth First Traversals:

(a) Inorder (Left, Root, Right) : 4 2 5 1 3

(b) Preorder (Root, Left, Right) : 1 2 4 5 3

(c) Postorder (Left, Right, Root) : 4 5 2 3 1

7. Πράξη JOIN δυαδικών δέντρων

- Χτίστε το δέντρο βήμα-βήμα.
- Δείξτε με λεπτομέρεια την κατάσταση του δέντρου μετά από κάθε εισαγωγή.
- Θεωρείστε ότι κάθε γράμμα του αγγλικού αλφαβήτου έχει μία αύξουσα τιμή (όπως στο ASCII), δηλαδή A=1, B=2, …. Άρα A < B < C < …. < Y < Z
- Μελετήστε την πράξη join από τις διαφάνειες των διαλέξεων. Παρατηρήστε ότι η πράξη insert που χρησιμοποιεί εισάγει ένα στοιχείο στο δυαδικό δέντρο και με rotations το φέρνει στη ρίζα του δέντρου. Σκεφτείτε γιατί χρειάζεται αυτό αντί της απλής εισαγωγής.
- Δείξτε με βήματα την κατάσταση του αλγορίθμου σε κάθε σημείο. Πιθανώς να βοηθήσει με βελάκια να δείξετε τις περιστροφές.
- Λάβετε υπ’ όψιν ότι το πρώτο δυαδικό δέντρο εισάγεται στο πρώτο. Καθώς γίνεται αυτό, καταστρέφεται το πρώτο. Για κάθε εισαγωγή, δείξτε και τα αντίστοιχα free (με σχήματα, την κατάσταση των 2 δέντρων μετά από κάθε πράξη join)

8. Μη-αναδρομική υλοποίηση της STselect

```
Item selectR(link h, int k)
  {
    int t;
    if (h == z) return NULLitem;
    t = (h->l == z) ? 0 : h->l->N;
    if (t > k) return selectR(h->l, k);
    if (t < k) return selectR(h->r, k-t-1);
    return h->item;
  }

Item STselect(int k)
  { return selectR(head, k); }
```

- Η STselect καλεί την selectR το σώμα της οποίας περιέχει δύο αναδρομικές κλήσεις.
- Ο σκοπός είναι η τροποποίηση της συνάρτησης ώστε να μην περιέχει αναδρομή, χωρίς να αλλάζει η λειτουργία της.
- Hint: Παρατηρήστε ότι για κάθε εκτέλεση της selectR πραγματοποιείται το πολύ μία αναδρομική κλήση.

9. Μη-αναδρομική υλοποίηση της STdelete

```
link joinLR(link a, link b)
  {
     if (b == z) return a;
     b = partR(b, 0);
     b->l = a;
     return b;
   }

link deleteR(link h, Key v)
  {
     link x;
     Key t = key(h->item);
     if (h == z) return z;
     if (less(v, t)) h->l = deleteR(h->l, v);
     if (less(t, v)) h->r = deleteR(h->r, v);
     if (eq(v, t)) { x = h; h = joinLR(h->l, h->r); free(x); }
     return h;
   }

void STdelete(Key v)  { head = deleteR(head, v); }
```

10. Το πρόβλημα του ιστογράμματος

- Έχετε την ελευθερία να χρησιμοποιήσετε οποιοδήποτε κομμάτι κώδικα σας φανεί χρήσιμο από τις διαφάνειες (και προτείνεται)
- Ένα ιστόγραμμα παρουσιάζει τη συχνότητα εμφάνισης κάποιας τιμής σε κάποιο πείραμα. Πώς μπορούμε να το μοντελοποιήσουμε με ένα δυαδικό δέντρο; Πειράξτε κατάλληλα τον κώδικα των διαφανειών
- 2 είναι τα σημαντικά στοιχεία του ιστογράμματος: 1) Ποιο είναι το κλειδί; 2) Πόσες φορές το έχω συναντήσει στην είσοδο;
- Σε τι αντιστοιχούν αυτά τα δύο σε μία δομή δέντρου; Σκεφτείτε τι μπορείτε να προσθέσετε για να κρατήσετε και την πληροφορία του 2)
- Παραδώστε μία απλή main που θα δοκιμάζει το πρόγραμμά σας.
- Υπολογίστε με απλά αθροίσματα όπως στην άσκηση 1 την πολυπλοκότητα του προγράμματός σας. (διάβασμα των κλειδιών από την είσοδο, αποθήκευση των κλειδιών στον δέντρο, εκτύπωση του ιστογράμματος). Είναι τρία διαφορετικά κομμάτια, άρα θα βγάλετε αντίστοιχα και 3 πολυπλοκότητες για το καθένα. Ποια είναι η συνολική πολυπλοκότητα του προγράμματος;

11, 12.Κατασκευή AVL δέντρου

- Δέντρα δυαδικής αναζήτησης στα οποία κάθε κόμβος έχει την AVL ιδιότητα.
- Μετά από κάθε εισαγωγή ή διαγραφή, το δέντρο πραγματοποιεί περιστροφές όπου χρειάζεται για ώστε να πληροί πάλι τις προϋποθέσεις AVL δέντρου.