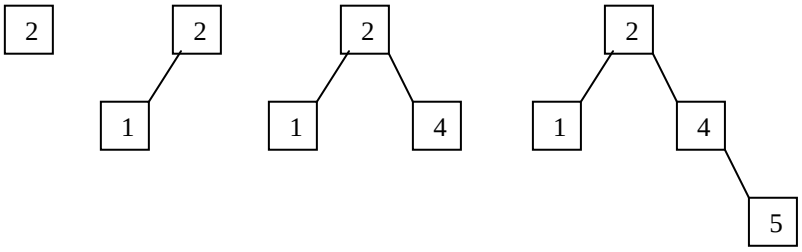
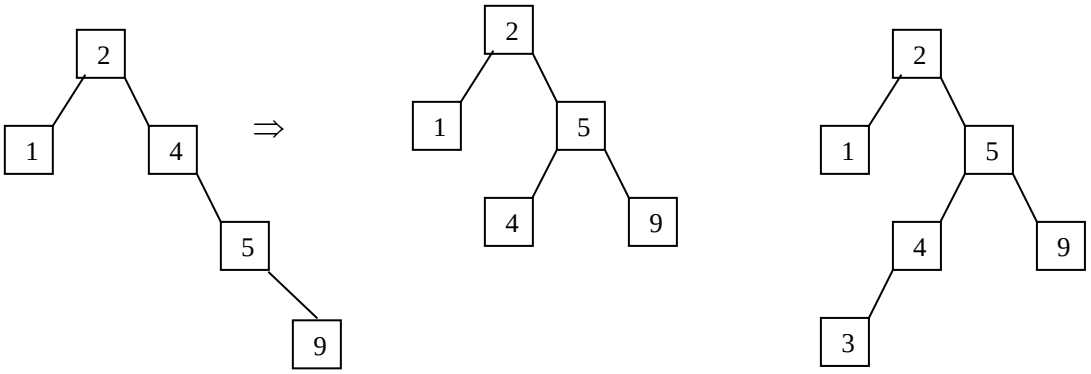
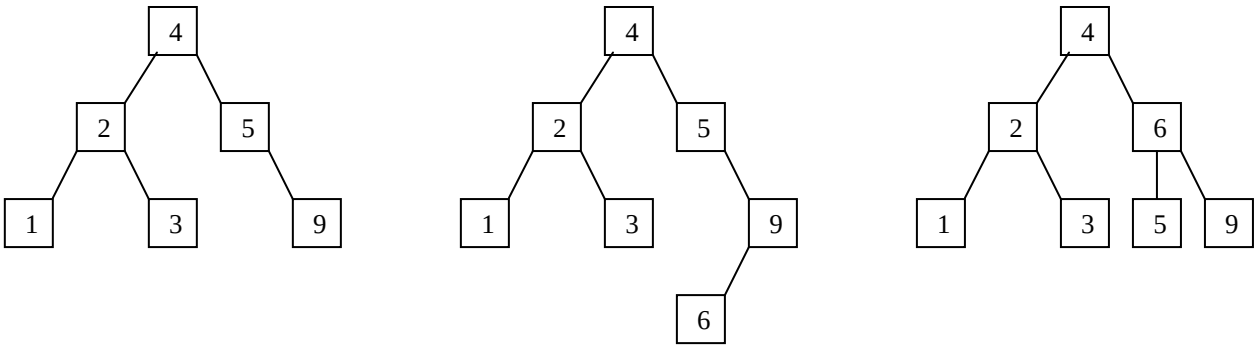
Φροντιστήριο 6 – Σκελετοί Λύσεων

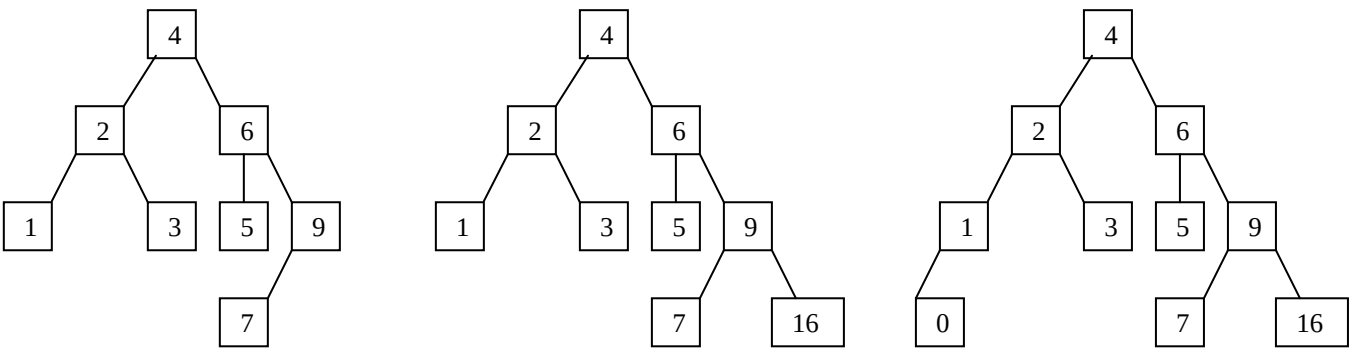
1. i) Ακολουθούν οι εισαγωγές σε AVL δένδρο.

κενό δένδρο ⇒

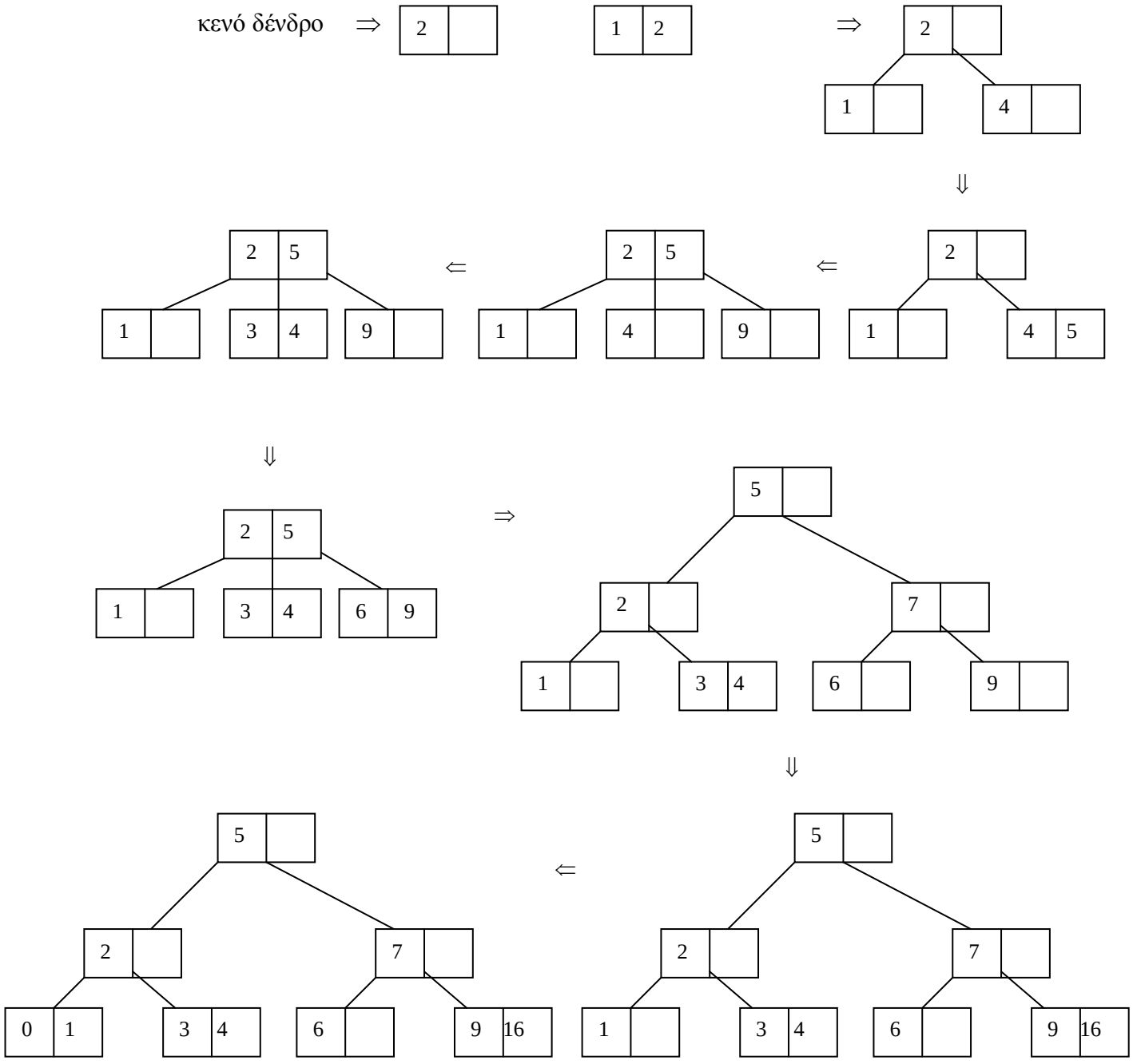








ii) Ακολουθούν οι εισαγωγές σε 2-3 δένδρο.



2. Χρησιμοποιούμε τη δομή

```
typedef struct BNode{
      int numkeys
      int key1;
      int key2;
      struct BNode *left;
      struct BNode *center;
      struct BNode *right;
} bnode;
```

και υποθέτουμε πως ένα 2-3 δένδρο είναι υλοποιημένο ως δείκτης στη ρίζα του δένδρου, δηλαδή, έχει τύπο *bnode.

(i) Για να υπολογίσουμε τα κλειδιά στα φύλλα ενός μη-κενού 2-3 δένδρου που δείχνεται από κάποιο δείκτη p, πρέπει να προσθέσουμε τα κλειδιά των φύλλων του αριστερού υποδένδρου, τα κλειδιά των φύλλων του μεσαίου υποδένδρου και τα κλειδιά των φύλλων του δεξιού υποδένδρου, αν υπάρχει.

Αυτή η ιδέα μεταφράζεται στην πιο κάτω αναδρομική διαδικασία:

```
int RecCountKeys(bnode *p){
     if (p == NULL)
          return 0;
     if (p->left == NULL)
          return p->numkeys;
     else
          x = RecCountKeys(p->left)
               +RecCountkeys(p->center);
          if (p-> numkeys == 2 )
               x =+ RecCountKeys(p->right);
          return x;
}
```

Η αναδρομική διαδικασία καλείται μια φορά σε κάθε κόμβο, επομένως ο χρόνος εκτέλεσής της είναι O(n) όπου n είναι ο αριθμός των κόμβων του δένδρου.

(ii) Η μη-αναδρομική διαδικασία απαιτεί τη χρήση στοίβας και έχει ως εξής:

I. Θέτουμε counter = 0 και p ίσο με το δείκτη στη ρίζα του δένδρου.
II. Εφόσον ο δείκτης δεν είναι NULL προχωρούμε στο βήμα III.
III Αν κόμβος είναι φύλλο αυξάνουμε την τιμή του counter κατά p->numkeys και
- Αν η στοίβα δεν είναι κενή συνεχίζουμε με τον κόμβο κορυφής της στοίβας και επαναλαμβάνουμε το βήμα.
- Διαφορετικά επιστρέφουμε την τιμή του counter.

IV. Αν ο κόμβος δεν είναι φύλλο τοποθετούμε στη στοίβα τον δείκτη (p->center) και, αν ο κόμβος έχει δύο κλειδιά, και τον δείκτη (p->right). Προχωρούμε προς τα αριστερά μέσω του p->left επαναλαμβάνοντας από το βήμα III.

```
int CountKeys(bnode *p){

   stack S;
   counter = 0;

   MakeEmpty(S);

   while (p != NULL){
          if (p->left == NULL) {
               counter += p->numkeys;
               if (!IsEmpty(S))
                    p = Pop(S);
               else
                    return counter;
          }
          else {
               if (p->numkeys == 2)
                    Push(p->right, S);
               Push(p->center, S);
               p = p->left;
          }
    }
}
```

3. Χρησιμοποιούμε τη δομή

```
typedef struct Node{
       int key;
       struct Node *left;
       struct Node *right;
} node;
```

και υποθέτουμε πως ένα δυαδικό δένδρο αναζήτησης είναι υλοποιημένο ως δείκτης στη ρίζα του δένδρου, δηλαδή, έχει τύπο *node.
Το πρόβλημα λύνεται με την πιο κάτω αναδρομική διαδικασία.

```
Between(node *p, int k1, intk2){
        if (p == NULL)
             return;

        if (p->key > k2)
             Between(p->left, k1, k2);

        if (p->key < k1)
             Between(p->right, k1, k2);

        if (k1 < p->key < k2) {
             Between(p->right, k1, k2);
             print p->key;
             Between(p->left, k1, k2);
        }
}
```