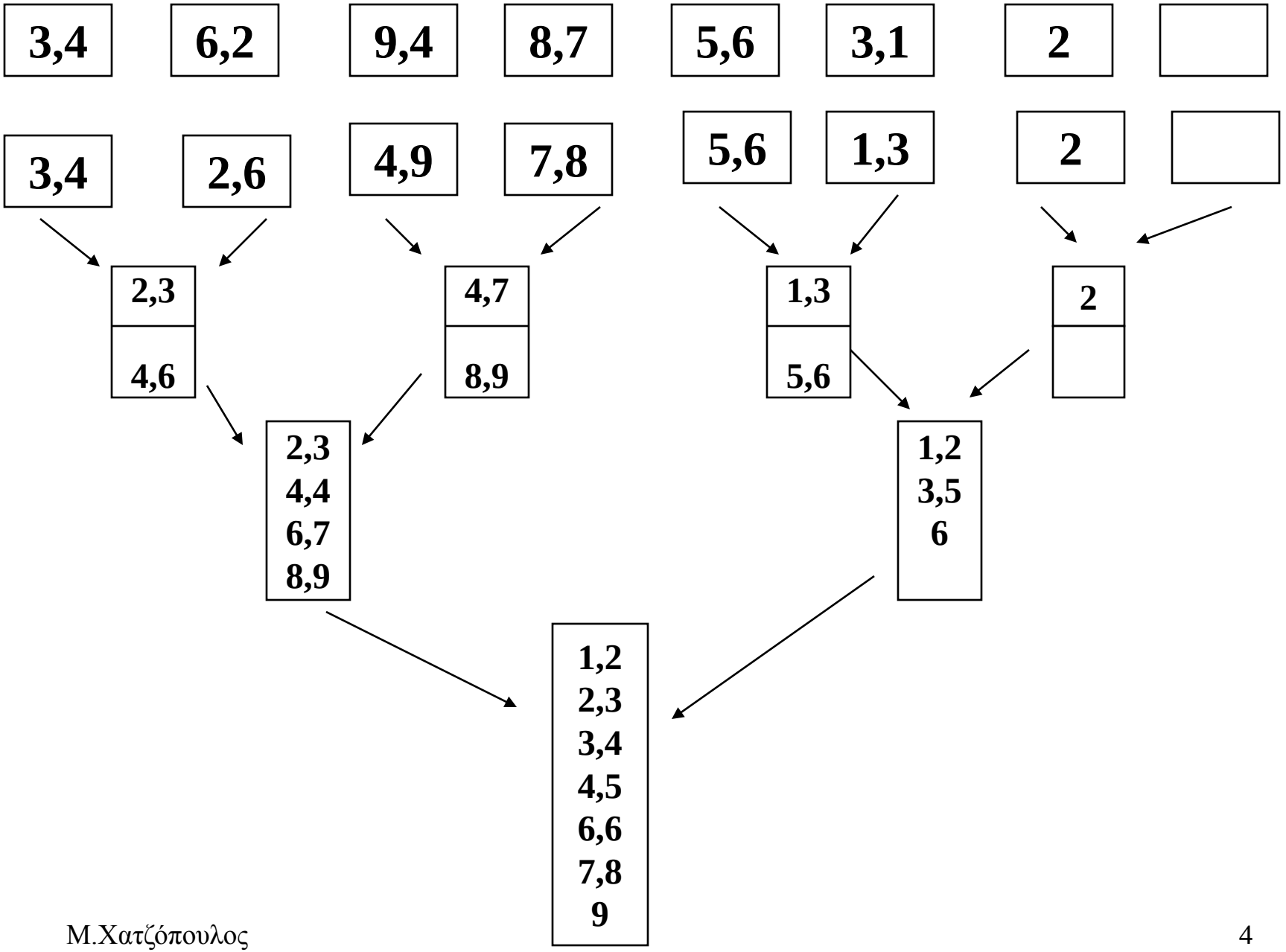
Εξωτερική Ταξινόμηση

Γιατί είναι απαραίτητη;

- Κλασσικό Πρόβλημα της Πληροφορικής
  - Πολλές φορές θέλουμε να παρουσιάσουμε δεδομένα σε ταξινομημένη μορφή
- Είναι σημαντική για την απαλοιφή διπλοτύπων
- Πράξεις στις βάσεις δεδομένων απαιτούν ταξινόμηση
- Σε πολλές οργανώσεις αρχείων αποτελεί το πρώτο βήμα στο αρχικό φόρτωμα του αρχείου.
- Σημαντικό πρόβλημα: τα αρχεία είναι μεγάλα και εφόσον δεν χωράνε στη μνήμη δεν μπορώ να χρησιμοποιήσω μια μέθοδο εσωτερικής ταξινόμησης
  - Θα μπορούσα να χρησιμοποιήσω εικονική μνήμη;

Η εξωτερική ταξινόμηση αναφέρεται σε αλγόριθμους ταξινόμησης που είναι κατάλληλοι για μεγάλα αρχεία εγγραφών αποθηκευμένα στο δίσκο που δεν μπορούν να χωρέσουν ολόκληρα στην κύρια μνήμη.

Ο τυπικός αλγόριθμος εξωτερικής ταξινόμησης χρησιμοποιεί μια στρατηγική ταξινόμησης-συγχώνευσης, που ξεκινά με την ταξινόμηση μικρών υποαρχείων – ονομάζονται σειρές (runs)- του κυρίως αρχείου και στη συνέχεια συγχώνευση των ταξινομημένων σειρών, δημιουργώντας μεγαλύτερα ταξινομημένα υποαρχεία που γίνεται με την σειρά συγχώνευση και αυτών.


3,4
6,2
9,4
8,7
5,6
3,1
2
3,4
2,6
4,9
7,8
5,6
1,3
2
2,3
4,6
4,7
8,9
1,3
5,6
2
2,3
4,4
6,7
8,9
1,2
3,5
6
1,2
2,3
3,4
4,5
6,6
7,8
9

function rmerge(X, l, m, n);
// (X_l,.. X_m) και (X_{m+1},…X_n) είναι ταξινομημένα (με κλειδιά τα x_i) και
// η συγχώνευση παράγει στο Z στις θέχσεις ($Z_ι$,…Z_n)
{ k ← l; i ← l; j ← m+1;
while ((i<=m) && (j<=n)) {
if x_i<= x_j {Z_k=X_i; i++}
else {Z_k=X_j; j++}
k++ }
if i>m { (Z_k,…Z_n)←(X_j,…X_n) }
else {(Z_k,…Z_n)←(X_j,…X_m) }

(X_l,…X_n)←(Z_l,…Z_n)
}

```
function mpass(X, Y, n. l);
// συγχωνεύει διαδοχικά υποαρχεία μεγέθους l από το αρχείο X στο
// αρχείο Y. n είναι το πλήθος των εγγραφών του X
{ i=1;
while (i<=(n-2*l+1)) do {
                        rmerge(X, i, i+l-1, i+2*l-1, Y);
                        i=i+2*l }
if ((i+l-1)<n) {rmerge(X, i, i+l-1, n, Y) }
          else ({(Yi,…Yn)←(Xj,…Xn) }
}
```

```
function msort(X, n)
declare Y(n)
{l=1;  /* το μέγεθος των υποαρχείων  */
 while (l<n) {mpass(X,Y,n, l);
                  l=2*l;
                 mpass(Y,X,n.l);
                l=2*l }
}
```

Φάση Ταξινόμησης

set i←1;

j←b; {το μέγεθος του αρχείου σε μπλοκ}

k←n_b; {το μέγεθος του μπαφερ σε μπλοκ}

m←⌈(j/k)⌉;

{φάση της ταξινόμησης}

while (i<=m)

do {

διάβασε τα επόμενα k μπλοκ του αρχείου στο μπαφερ ή αν απομένουν λιγότερα από k μπλοκ διάβασε τα μπλοκ που απομένουν

ταξινόμησε τις εγγραφές στο μπαφερ και να γραφούν σαν προσωρινό υποαρχείο;

i←i+1;

}

Φάση Συγχώνευσης

set i←1;

p←⌈$\log_{k-1}$m⌉;

j←m;

while (i<=p)

do {

n←1;

q←⌈(j/(k-1))⌉; {πλήθος υποαρχείων που θα γραφούν στο πέρασμα αυτό}

while (n<=q)

do {

διάβασε τα επόμενα k-1 υποαρχεία (από το προηγούμενο πέρασμα) ένα μπλοκ τη φορά;

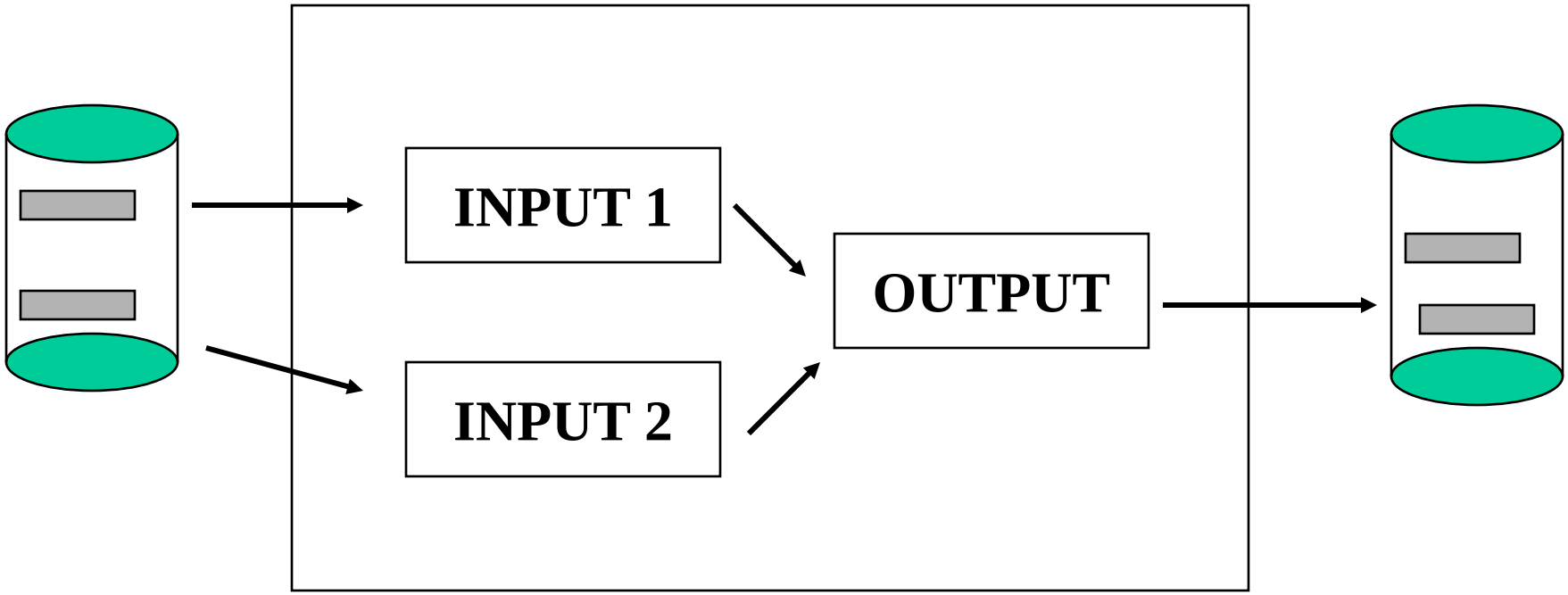
συνένωση και γράψιμο σαν νέο υποαρχείο;

n←n+1;

} j←q;

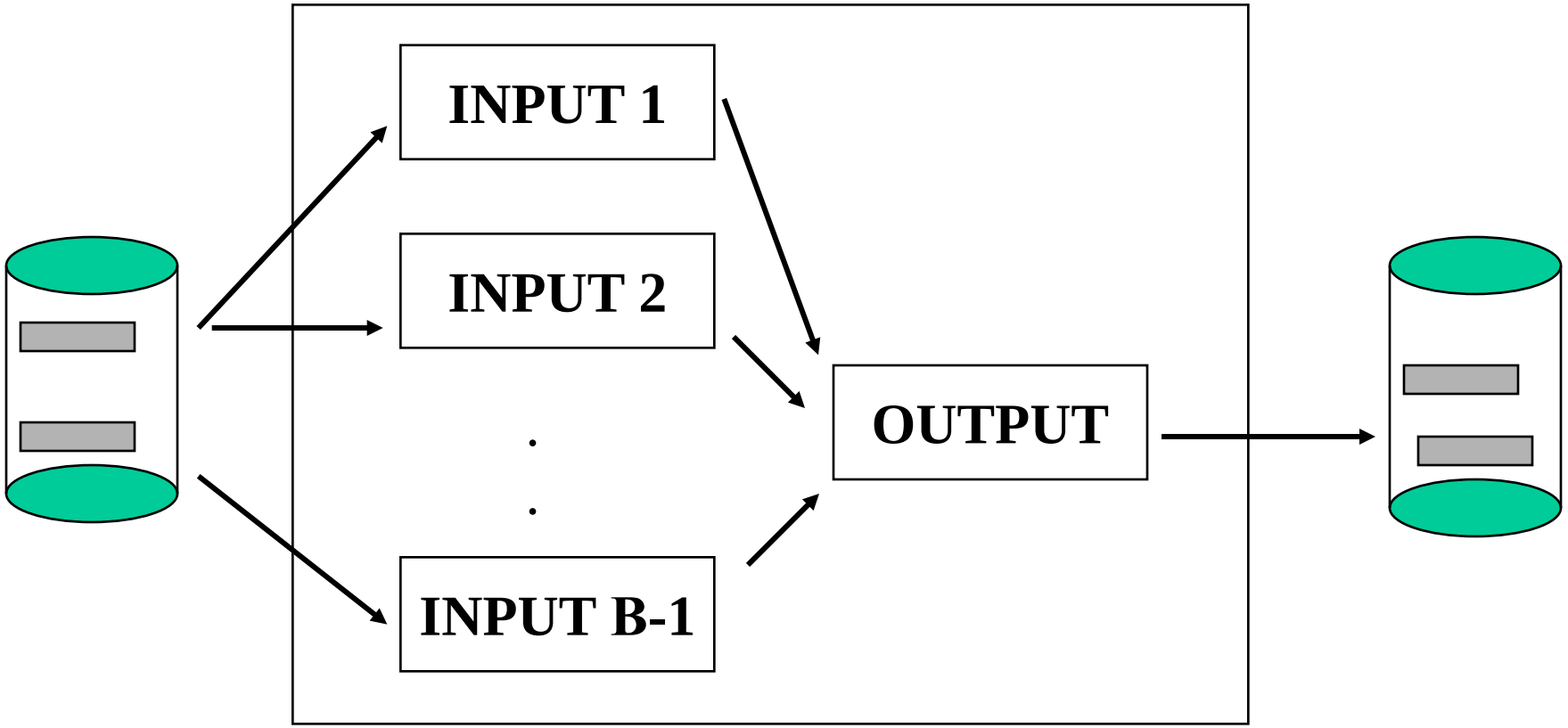
i←i+1;

}


INPUT 1
INPUT 2
OUTPUT

Εξωτερική ταξινόμηση

- Αν μπορούμε να έχουμε στη μνήμη περισσότερες από 2 σελίδες μπορούμε να κάνουμε κάτι καλύτερο
- Αν έχουμε n_B σελίδες μνήμης
  - Στο πέρασμα 0 ταξινομούμε n_B σελίδες (θα δημιουργηθούν $\lceil b/ n_B \rceil$ ταξινομημένες σειρές
  - Μετά σε κάθε πέρασμα θα γίνεται συγχώνευση n_B-1 ταξινομημένων σειρών


INPUT 1
INPUT 2
.
.
INPUT B-1
OUTPUT

Φάση Ταξινόμησης

Έστω b το πλήθος των μπλοκ του αρχείου και έστω ότι μπορώ να έχω στη μνήμη n_B μπλοκ. Τότε οι πρώτες ταξινομημένες σειρές που μπορώ να δημιουργήσω είναι:

$$n_R = \lceil (b/ n_B) \rceil$$

Φάση της Συγχώνευσης

Κάθε φάση της συγχώνευσης απαιτεί μια ανάγνωση και μια εγγραφή ολόκληρου του αρχείου. Επομένως το κόστος θα εξαρτάται από το πλήθος των συγχωνεύσεων. Αν γίνει συγχώνευση βαθμού d τότε το πλήθος των συγχωνεύσεων θα είναι:

$$\lceil (\log_d(n_R)) \rceil$$

Και στην περίπτωση δυαδικής συγχώνευσης:

$$\lceil (\log_2(n_R)) \rceil$$

Συνολικό κόστος

Το κόστος για την δημιουργία των ταξινομημένων σειρών θα είναι:

$$2*b$$

Για την συγχώνευση:

$$2*b*\lceil(\log_d(n_R))\rceil$$

Επομένως συνολικά:

$$2*b+2*b*\lceil(\log_d(n_R))\rceil$$

Πλήθος συγχωνεύσεων

b	n_B=3	n_B=5	n_B=9	n_B=17	n_B=129	n_B=257
100	7	4	3	2	1	1
1000	10	5	4	3	2	2
10000	13	7	5	4	2	2
100000	17	9	6	5	3	3
1000000	20	10	7	5	3	3
10000000	23	12	8	6	4	3
100000000	26	14	9	7	4	4
1000000000	30	15	10	8	5	4

Μπορώ να κάνω κάτι καλλίτερο;

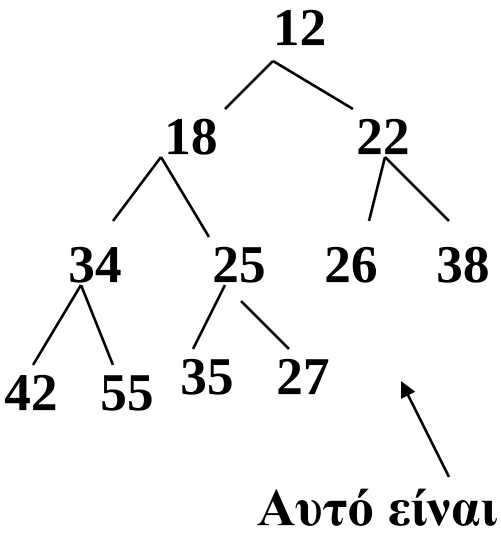
Από τον προηγούμενο τύπο για να έχω λιγότερες σαρώσεις του αρχείου πρέπει να συμβεί:

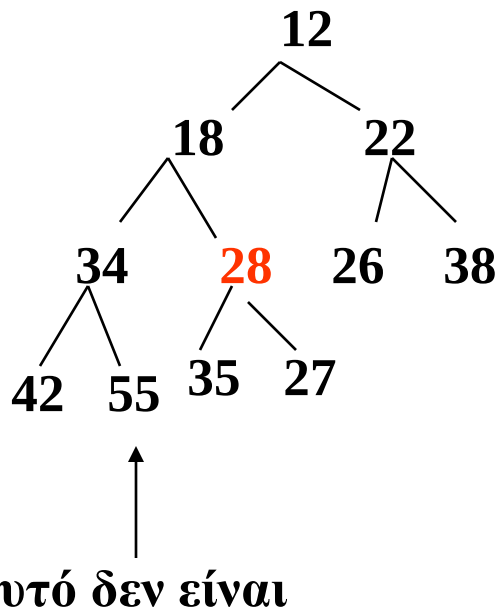
1)Όσο το δυνατόν μεγαλύτερο βαθμό συνένωσης. Όμως το d δεν μπορεί να ξεπεράσεις το n_b-1. Ένα για την έξοδο και όλα τα υπόλοιπα για συγχώνευση.

2)Όσο το δυνατόν μικρότερο n_R. Δηλαδή να φτιάξω αρχικές ταξινομημένες σειρές μεγαλύτερου μεγέθους.

Μπορώ όμως να φτιάξω αρχικές ταξινομημένες σειρές με μέγεθος μεγαλύτερο από αυτό που χωράει η μνήμη;

Σωρός ταξινόμησης είναι ένα πλήρες δυαδικό δένδρο με την ιδιότητα το στοιχείο της ρίζας να είναι μικρότερο από το στοιχείο της ρίζας των υποδένδρων του και κάθε υποδένδρο έχει την ιδιότητα να είναι σωρός ταξινόμησης

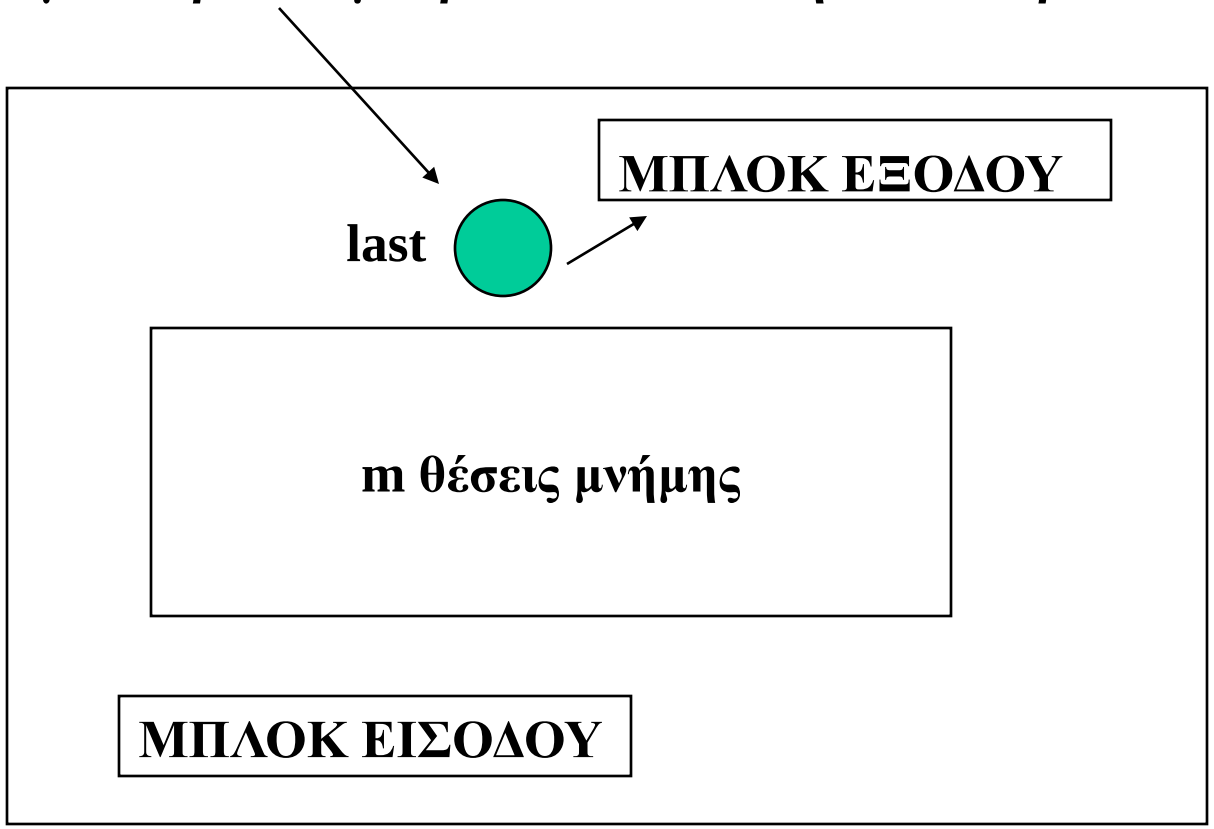




Έστω ο πίνακας Α που έχει την ιδιότητα του σωρού ταξινόμησης από το στοιχείο i+1 μέχρι το τέλος m

```
procedure pilesort(A, i, n)
{
if i<=⌊n/2⌋ then
{
j:=2*i; if j<n then
                    {if A[j+1]<A[j] then j:=j+1}
          if A[i]>A[j] then {A[i]↔]A[j]; pilesort(A,j,n)}
}
else exit
}
```

Το πιο πρόσφατο στοιχείο της εξόδου. Αν το επόμενο που θα διαβασθεί Είναι μεγαλύτερο του μπορούν να πάνε στην ίδια σειρά



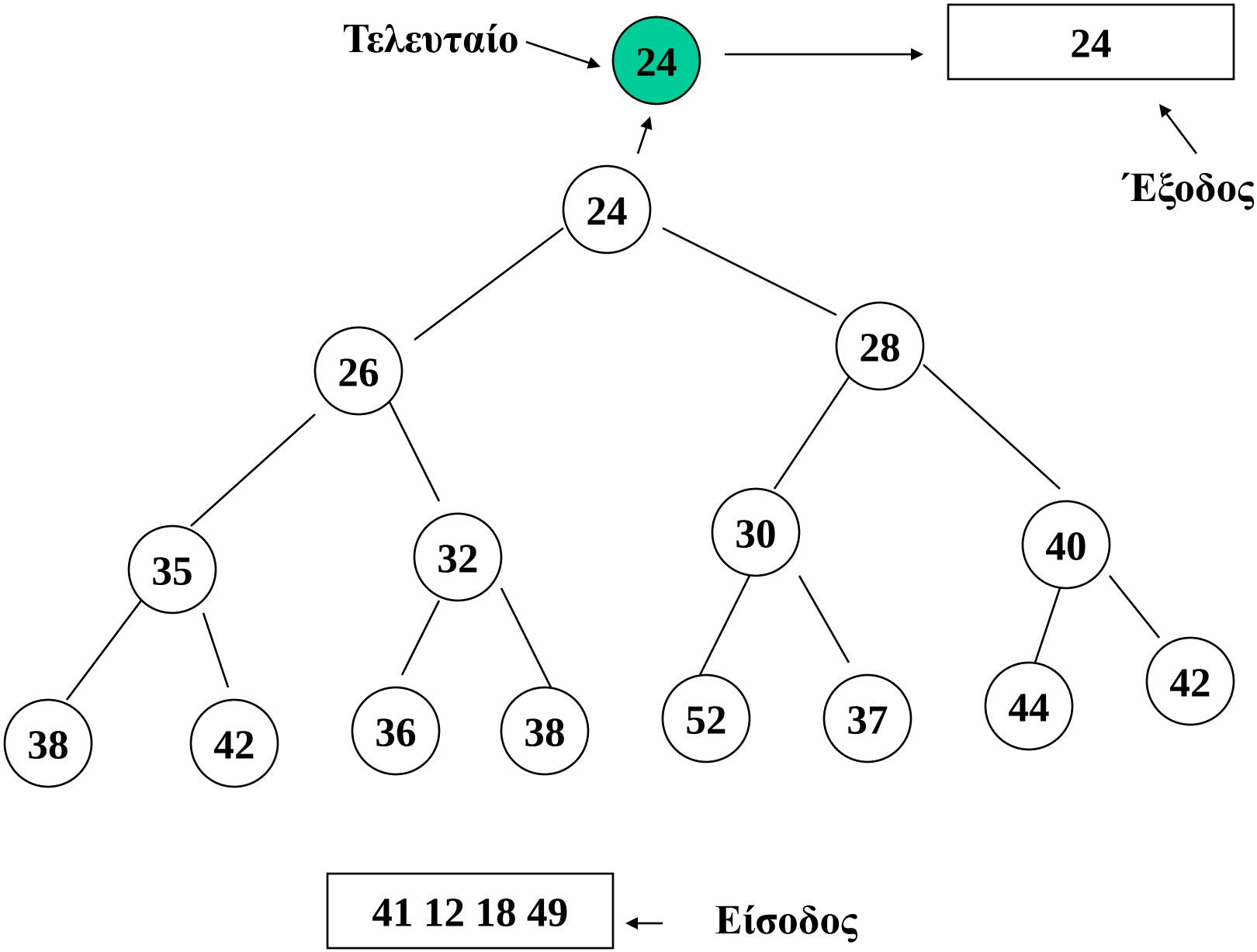
Αρχική εισαγωγή στοιχείων (δημιουργία αρχικού σωρού ταξινόμησης

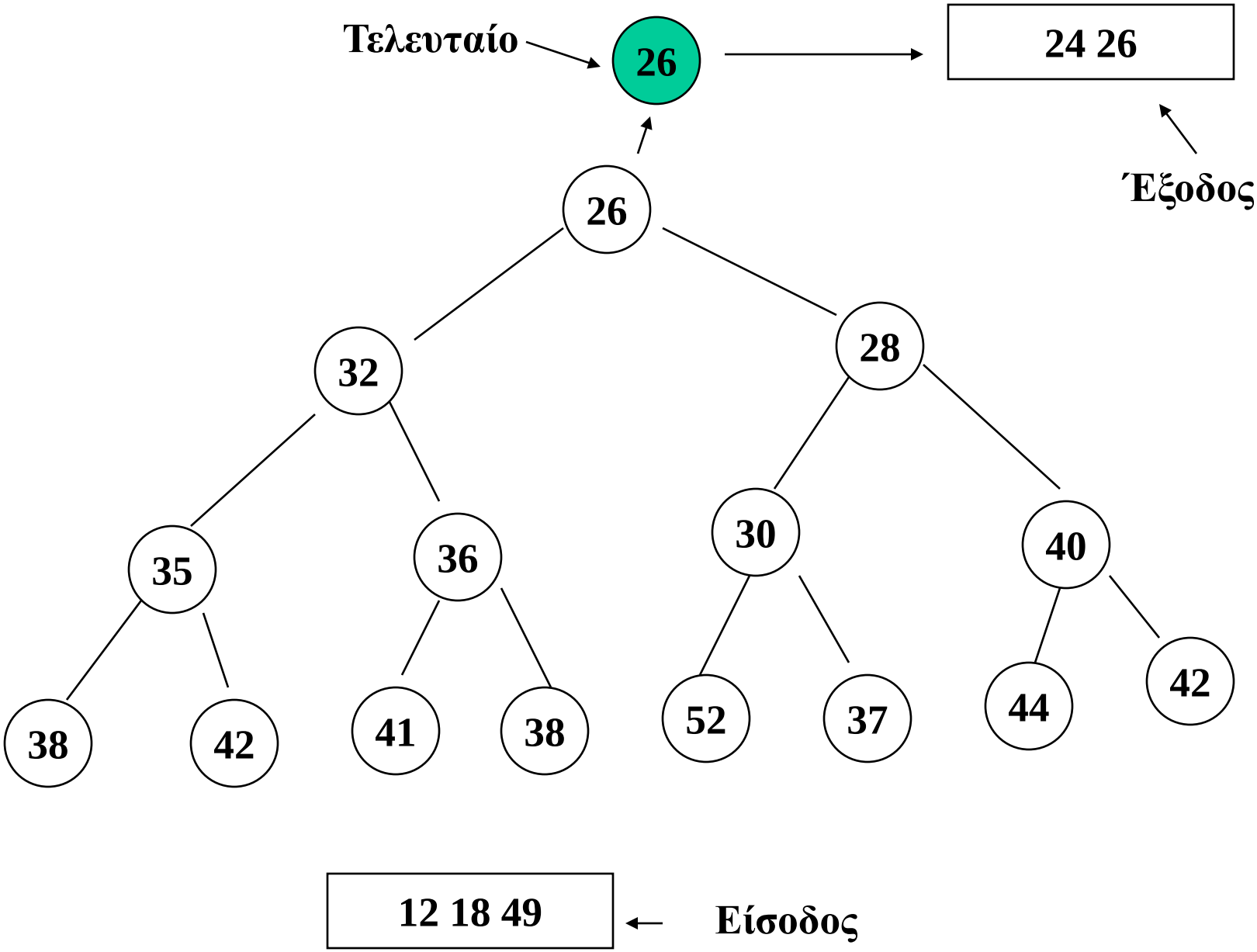
```
for i:=m downto ⌊m/2⌋+1 do read(in,A[i])
for i:= ⌊m/2⌋ downto 1 do
                    {
                    read(in,A[i]);
                     pilesort(A,I,m)
                     }
```

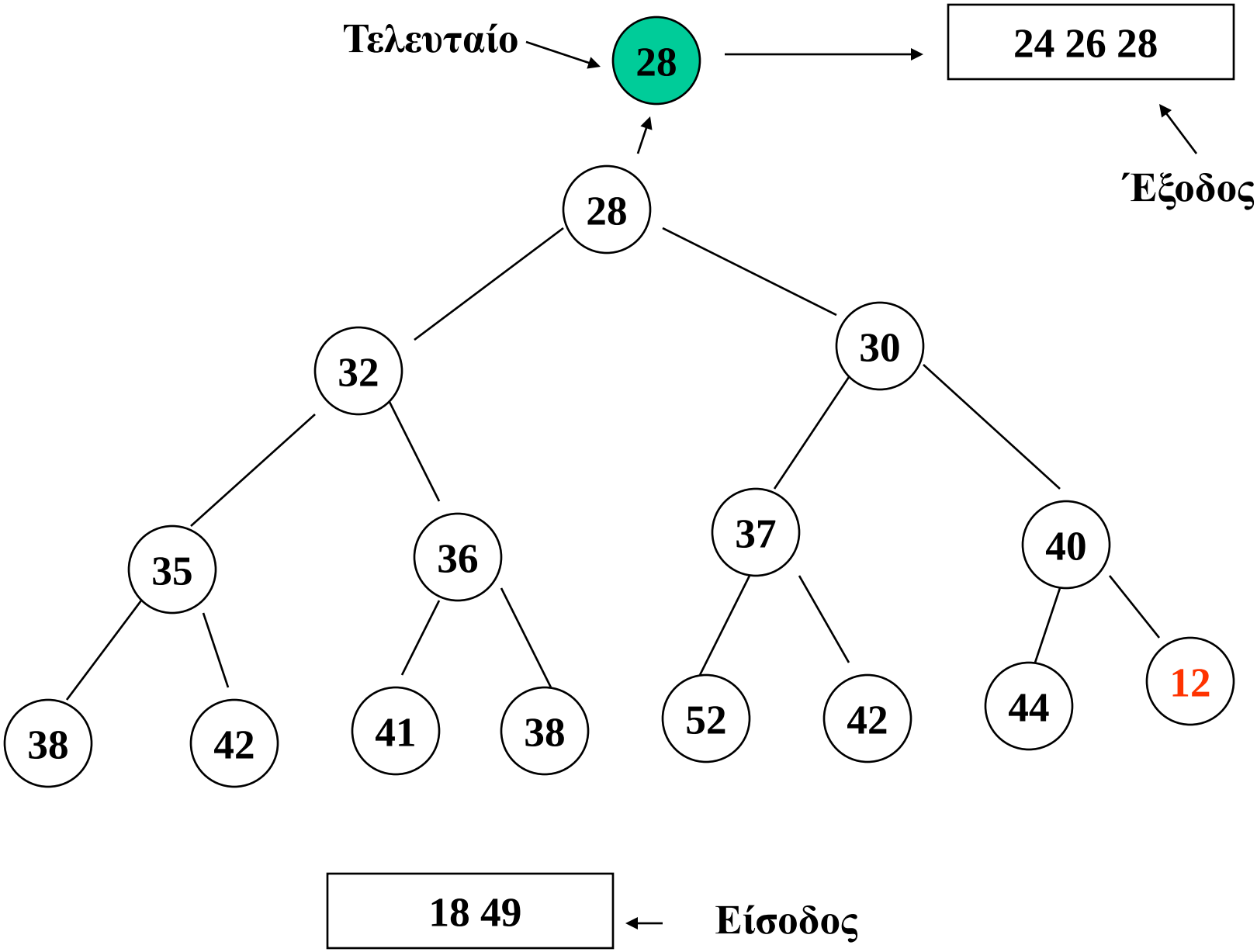
Κάθε φορά κατασκευάζονται δύο σειρές, η τρέχουσα και η επόμενη

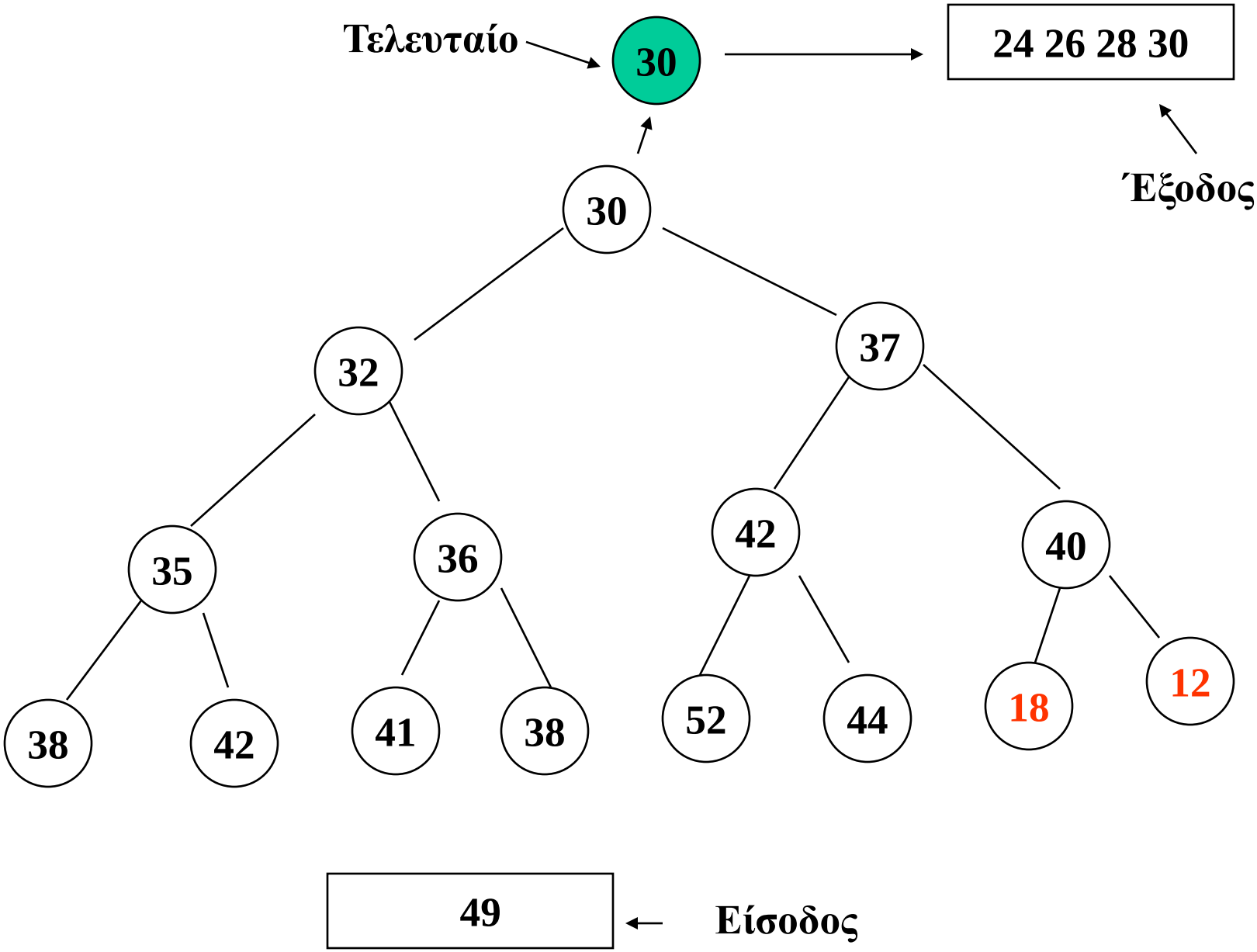
```
last:=A[1]; write(out, last);
k:=m; no_of_runs:=1;
while not eof(in) do{
read(in, A[1]); if A[1]>=last then pilesort(A,1,k)
        else{ if k>1 then{A[1]↔A[k];
                        pilesort(A,1,k-1);
                         pilesort(A, k, m);
                          k:=k-1}
                    else {write(out, end_of_run);
                      no_of_runs:= no_of_runs+1;
                          k:=m; pilesort(A, 1,k)}
                }
write(out,A[1])
}
```

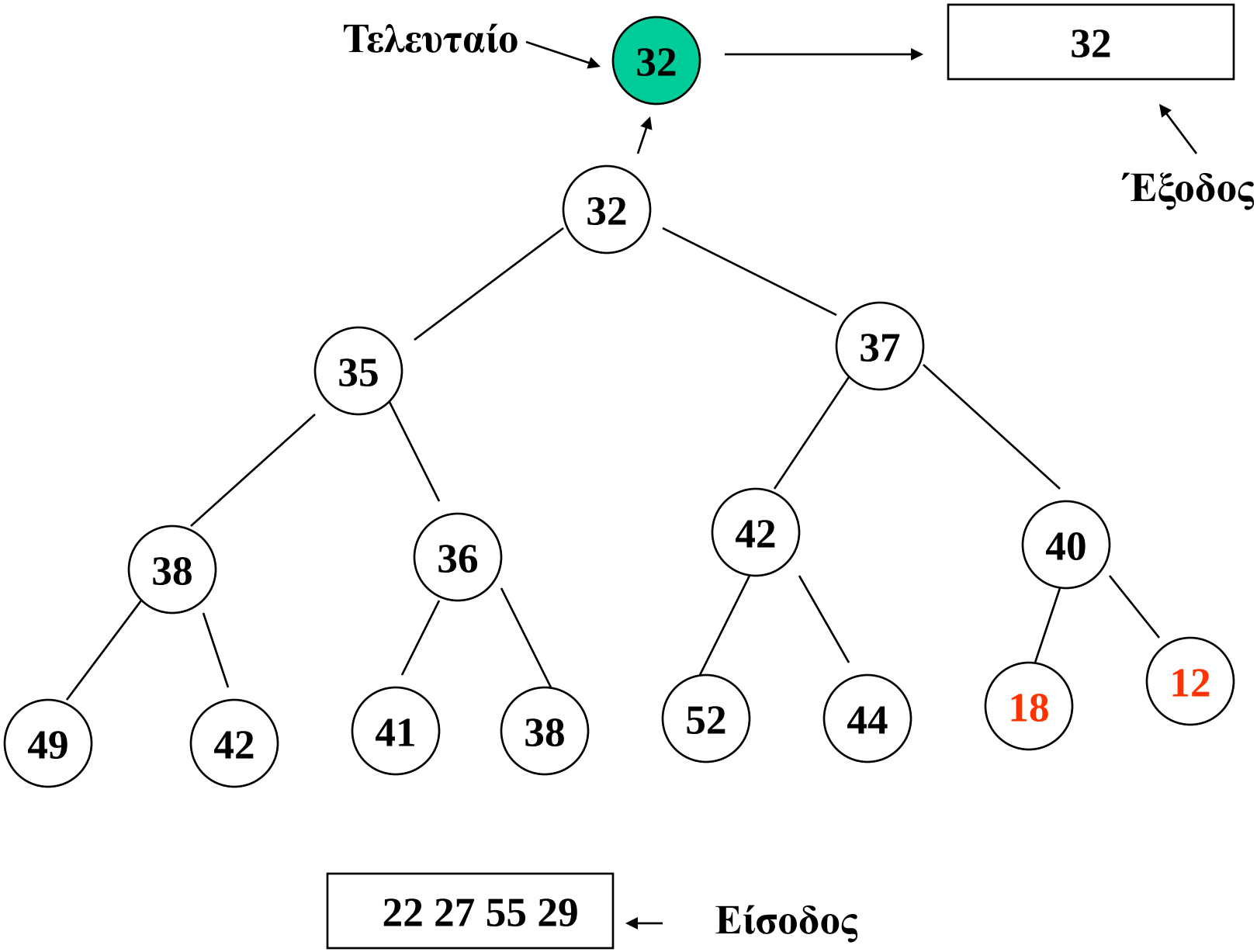
```
if k<m then {
for i:=k downto 2 do{write(out, A[1]); A[1]:=A[i];
                                pilesort(A, 1, i-1);
                                A[i]:=A[m-k+i]; pilesort(A, i, m-k+i-1]
                            };
                        write(out, end_of_run); k:=m-k;
                        no_of_runs:= no_of_runs+1;
                       }
          else {for i:=k downto 1 do{write(out, A[1]); A[1]:=A[i];
                                pilesort(A, 1, i-1); }
```

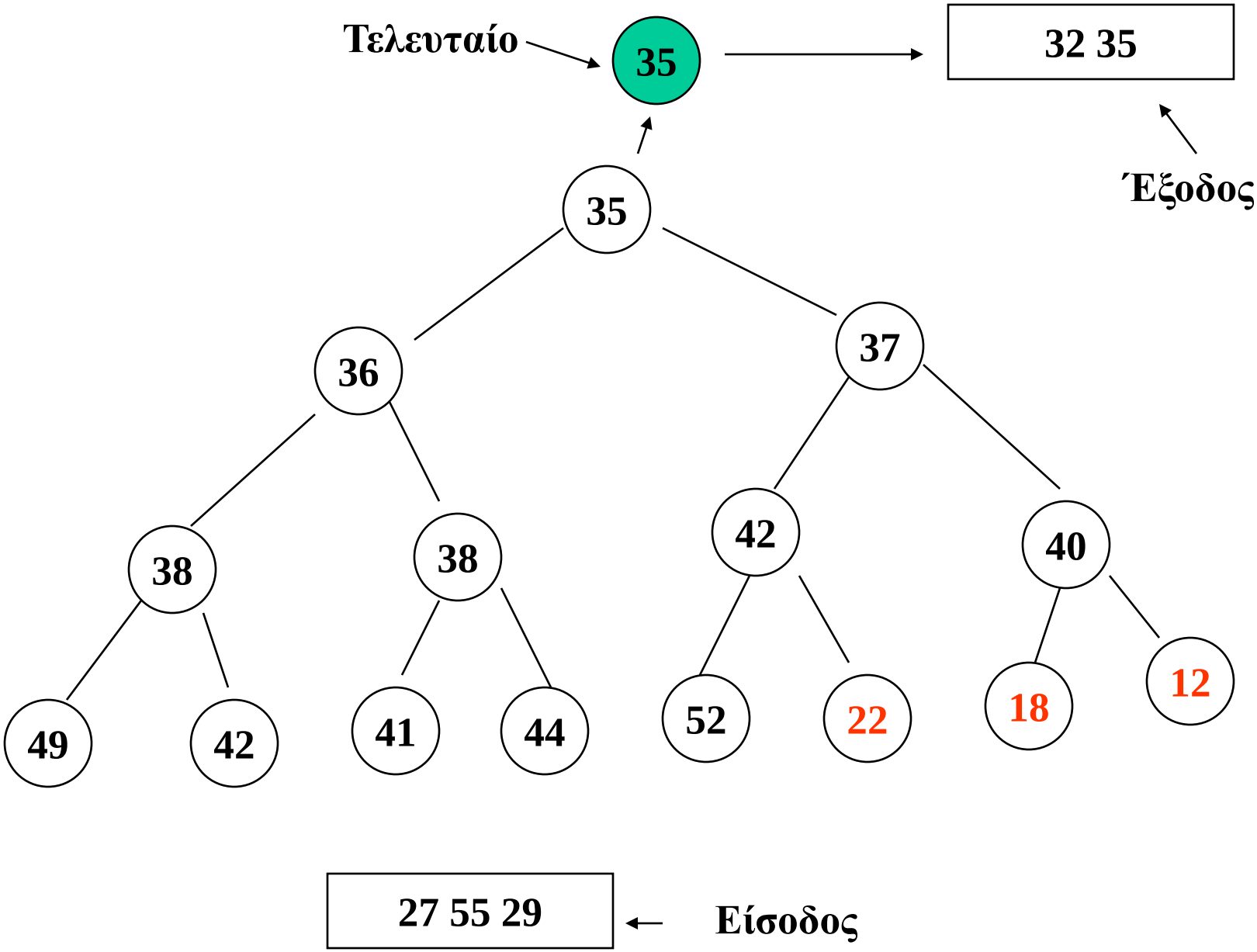

Τελευταίο
24
24
Έξοδος
24
26
28
35
32
30
40
38
42
36
38
52
37
44
42
41 12 18 49
Είσοδος

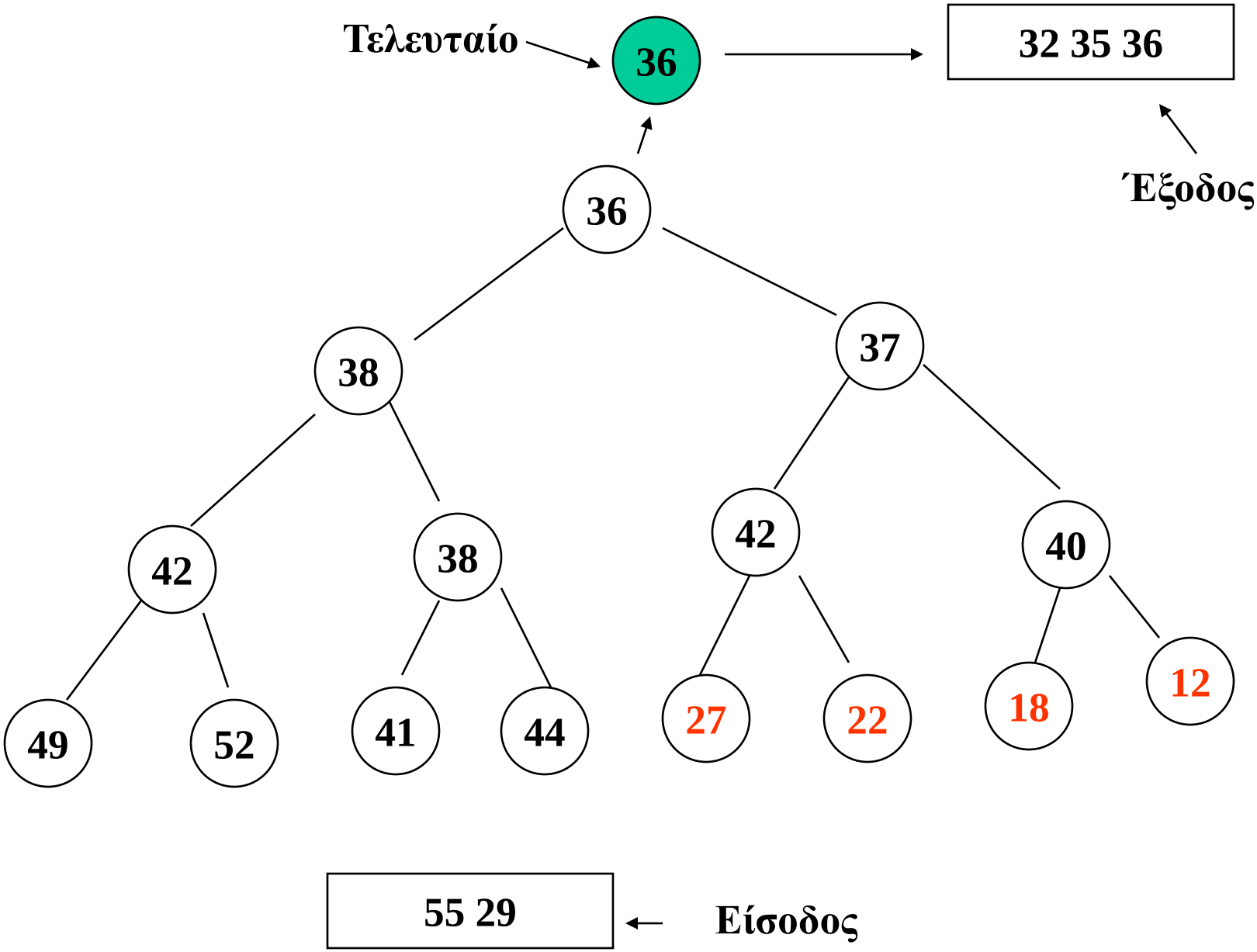

Τελευταίο
26
24 26
Έξοδος
26
32
28
35
36
30
40
38
42
41
38
52
37
44
42
12 18 49
Είσοδος

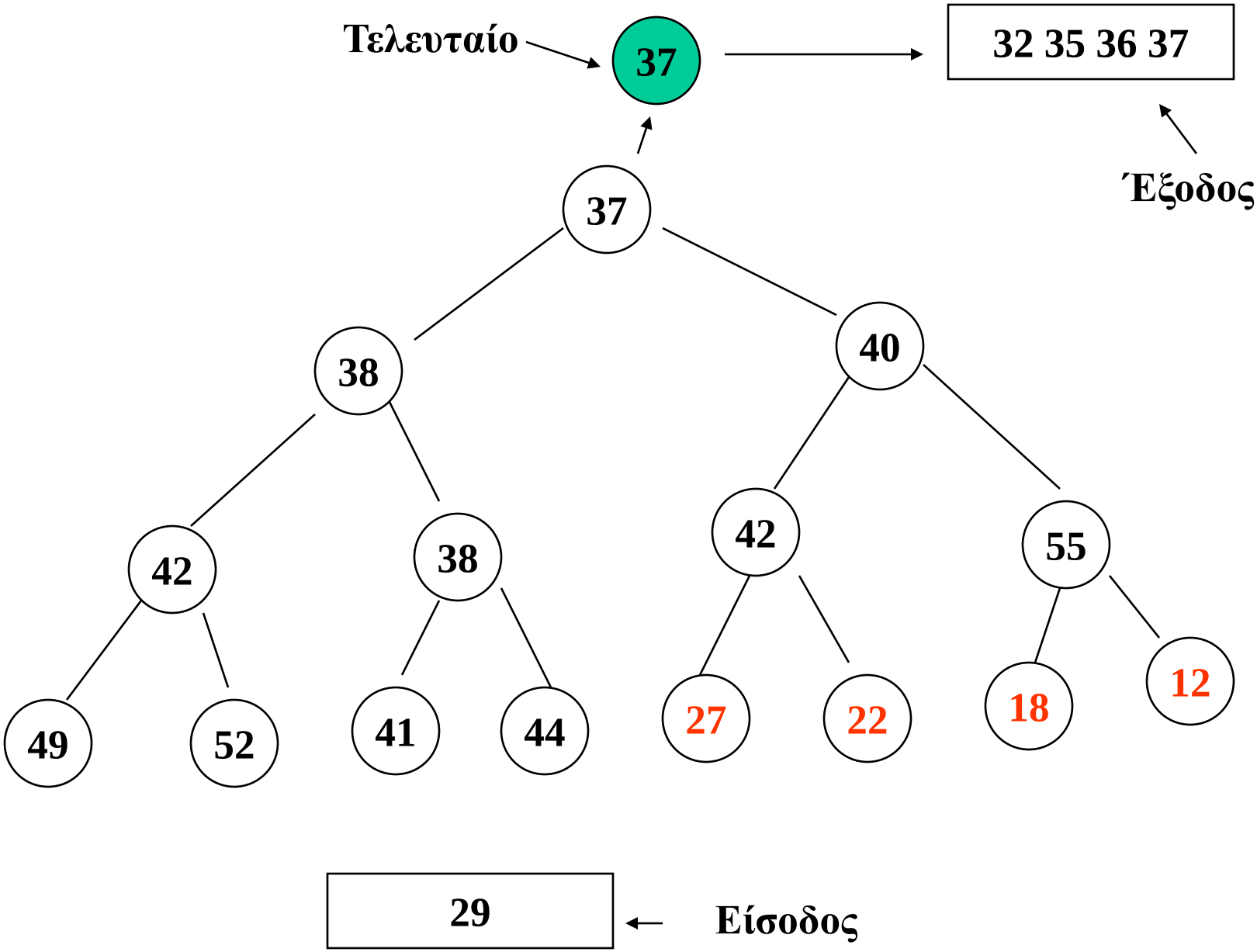

Τελευταίο
28
24 26 28
Έξοδος
28
32
30
35
36
37
40
38
42
41
38
52
42
44
12
18 49
Είσοδος

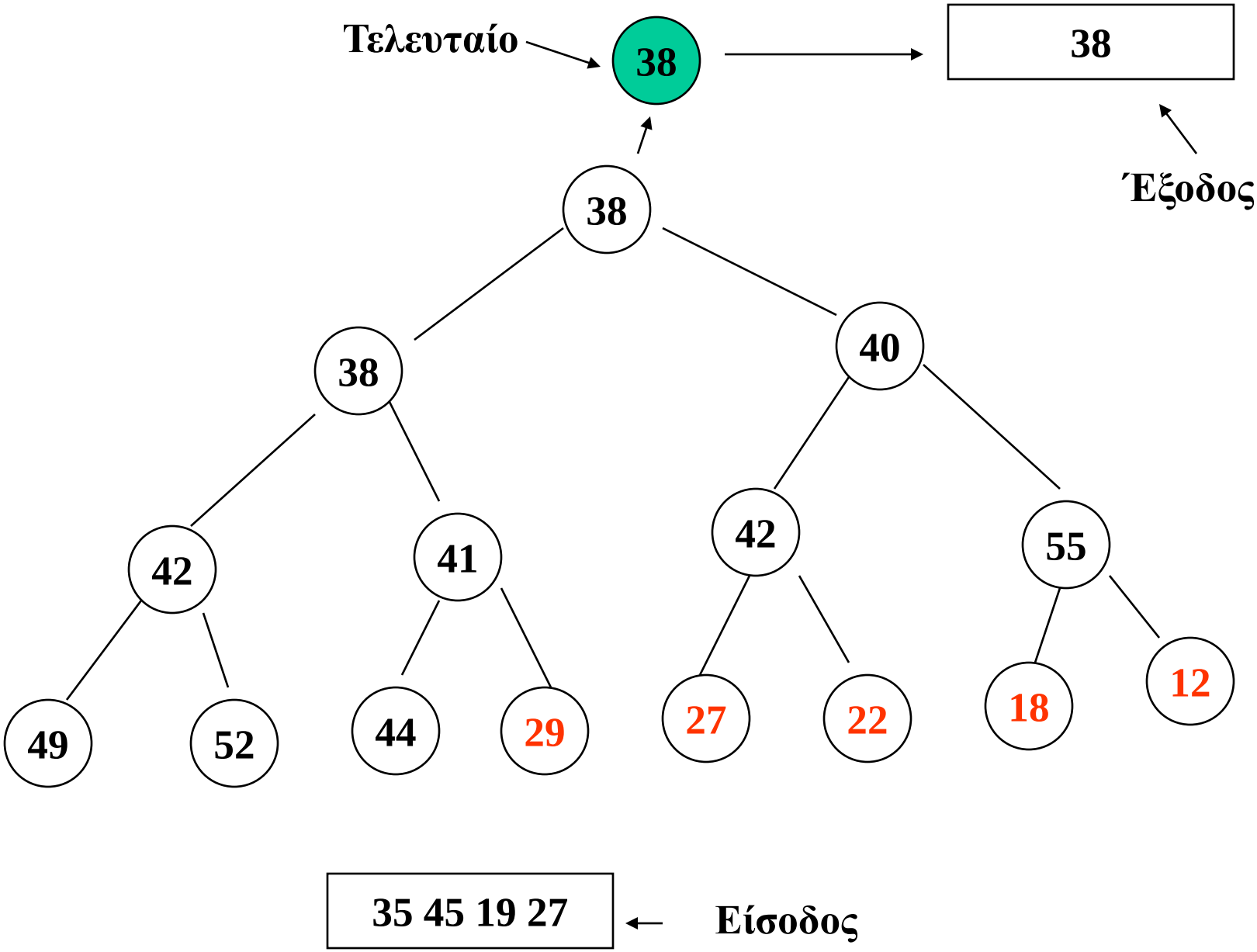

Τελευταίο
30
24 26 28 30
Έξοδος
30
32
37
35
36
42
40
38
42
41
38
52
44
18
12
49
Είσοδος

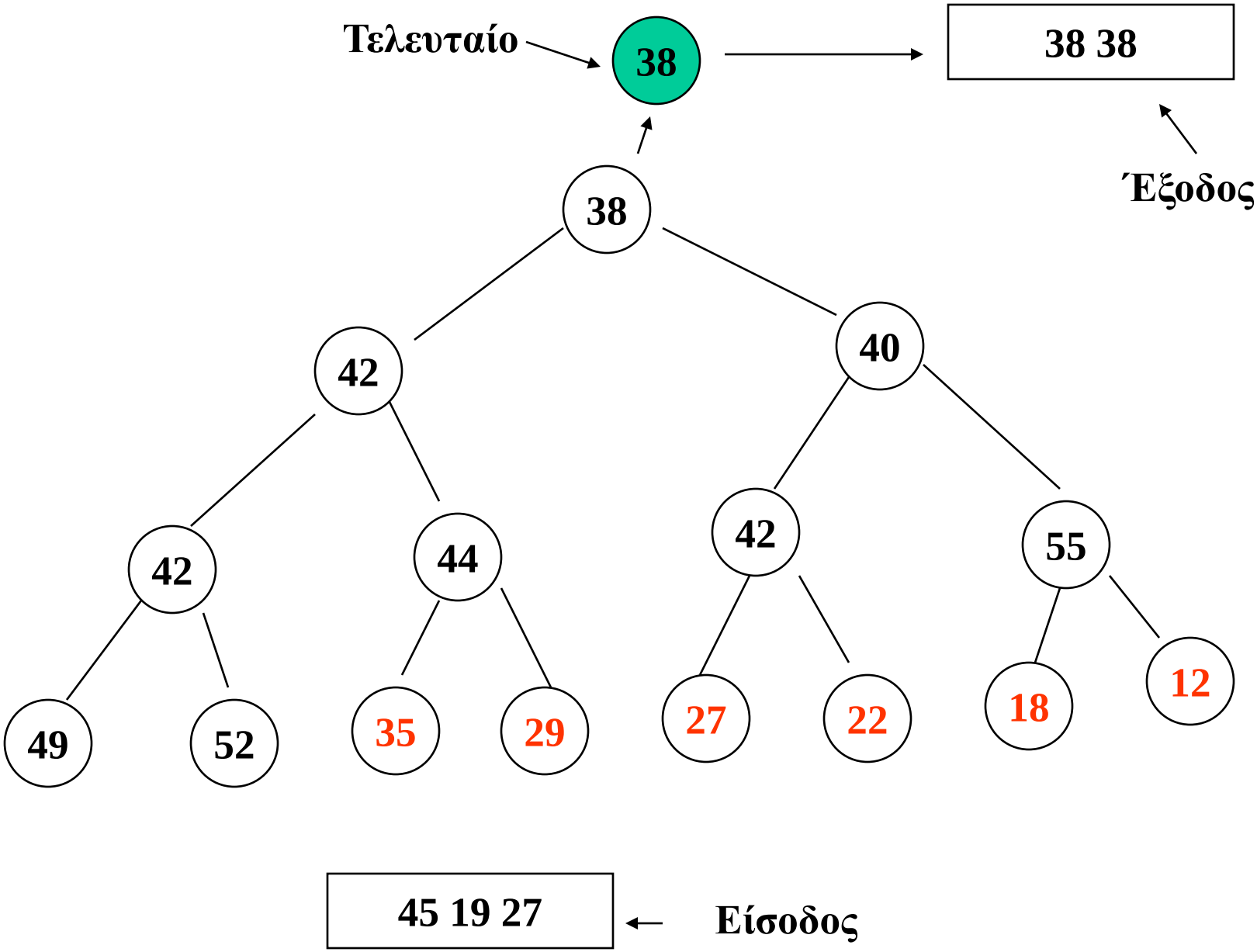

Τελευταίο
32
32
Έξοδος
32
35
37
38
36
42
40
49
42
41
38
52
44
18
12
22 27 55 29
Είσοδος

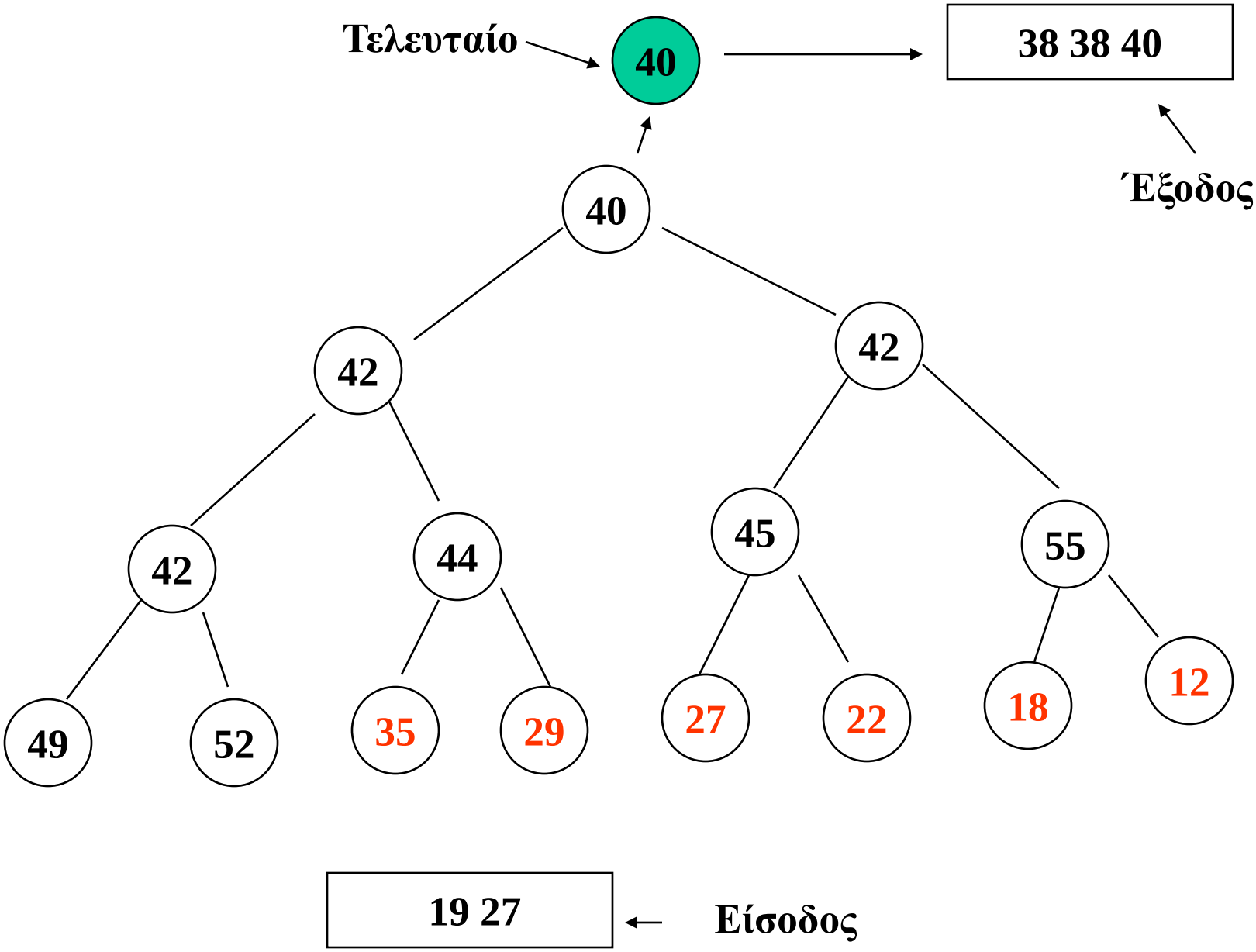

Τελευταίο
35
32 35
Έξοδος
35
36
37
38
38
42
40
49
42
41
44
52
22
18
12
27 55 29
Είσοδος

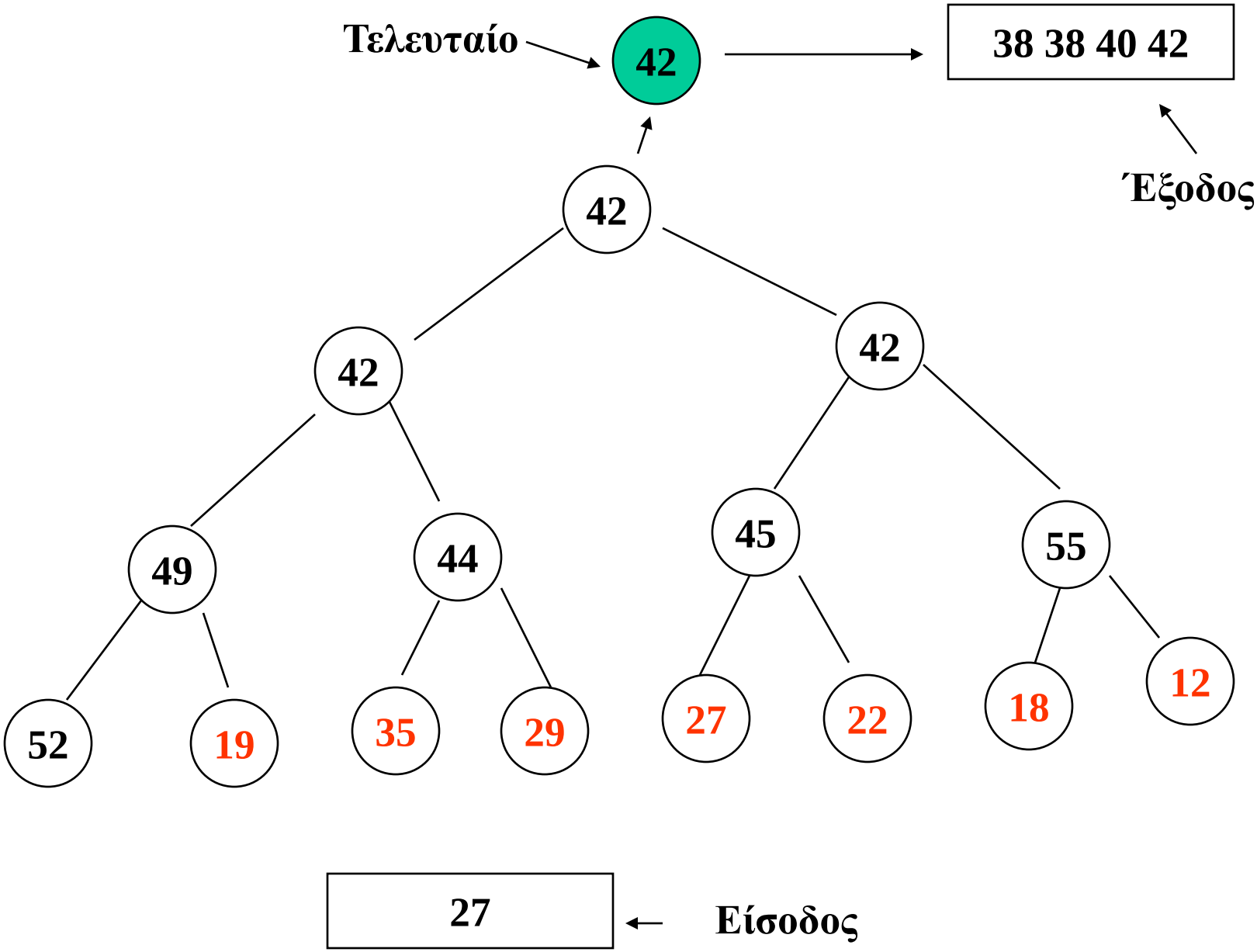

Τελευταίο
36
32 35 36
Έξοδος
36
38
37
42
38
42
40
49
52
41
44
27
22
18
12
55 29
Είσοδος

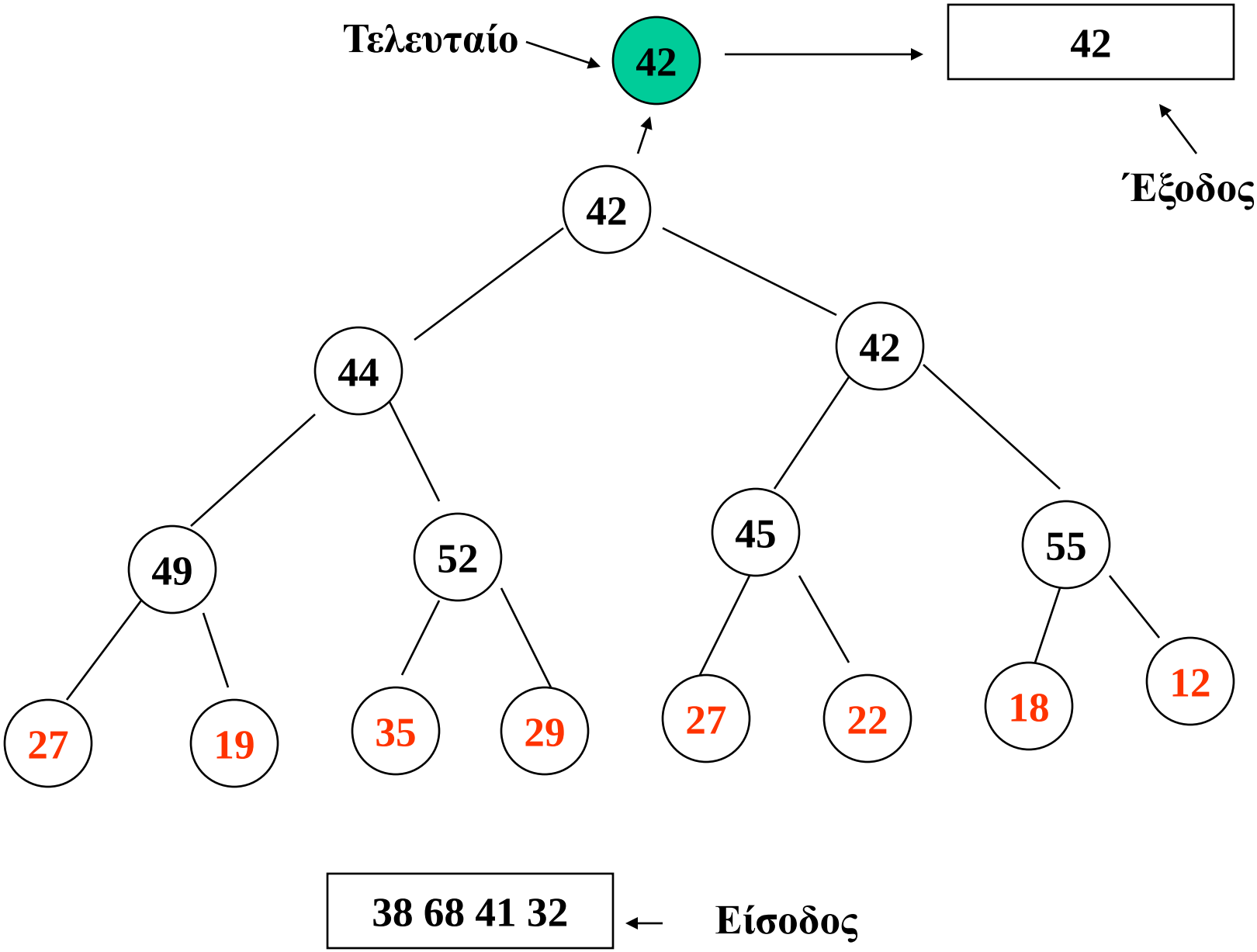

Τελευταίο
37
32 35 36 37
Έξοδος
37
38
40
42
38
42
55
49
52
41
44
27
22
18
12
29
Είσοδος

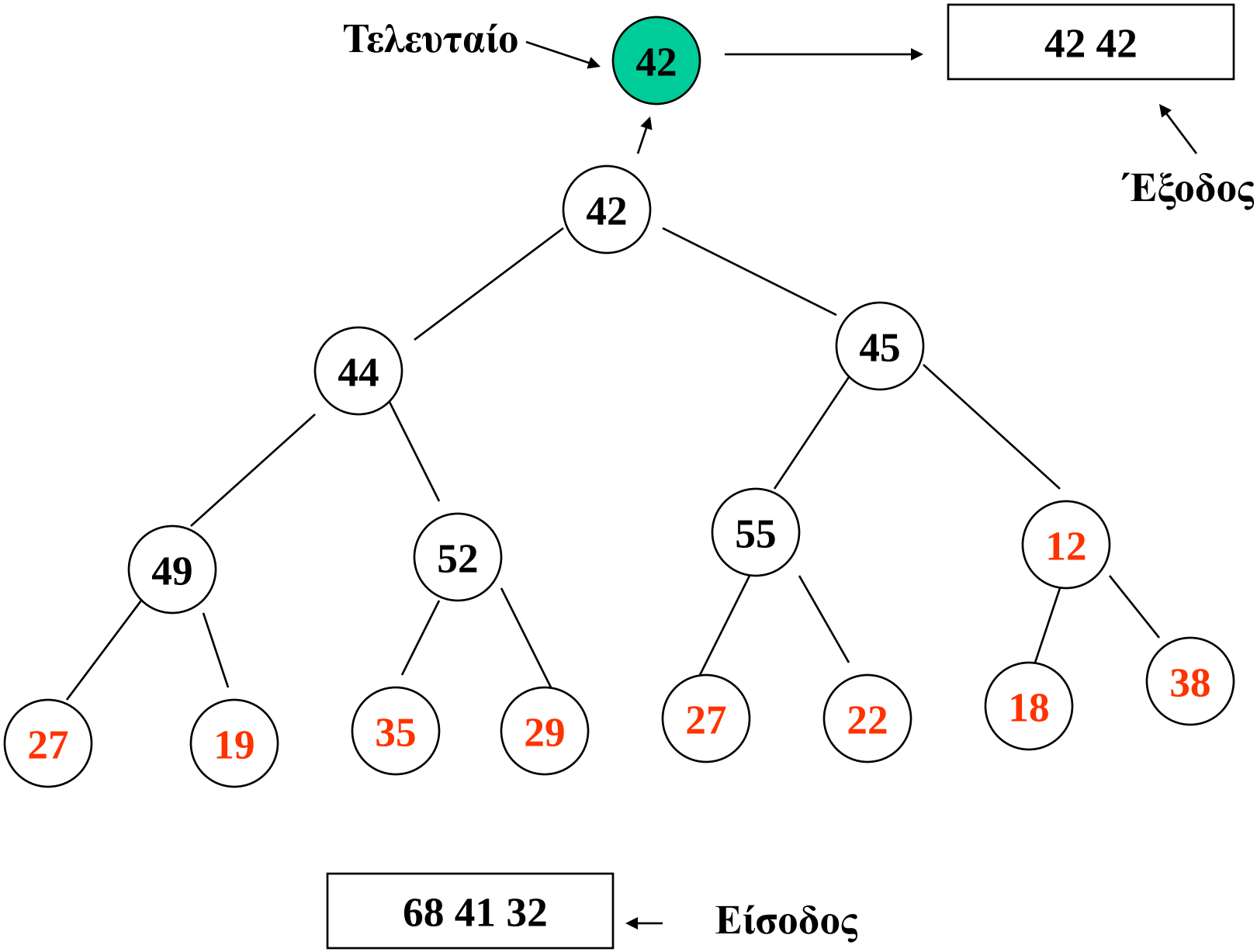

Τελευταίο
38
38
Έξοδος
38
38
40
42
41
42
55
49
52
44
29
27
22
18
12
35 45 19 27
Είσοδος

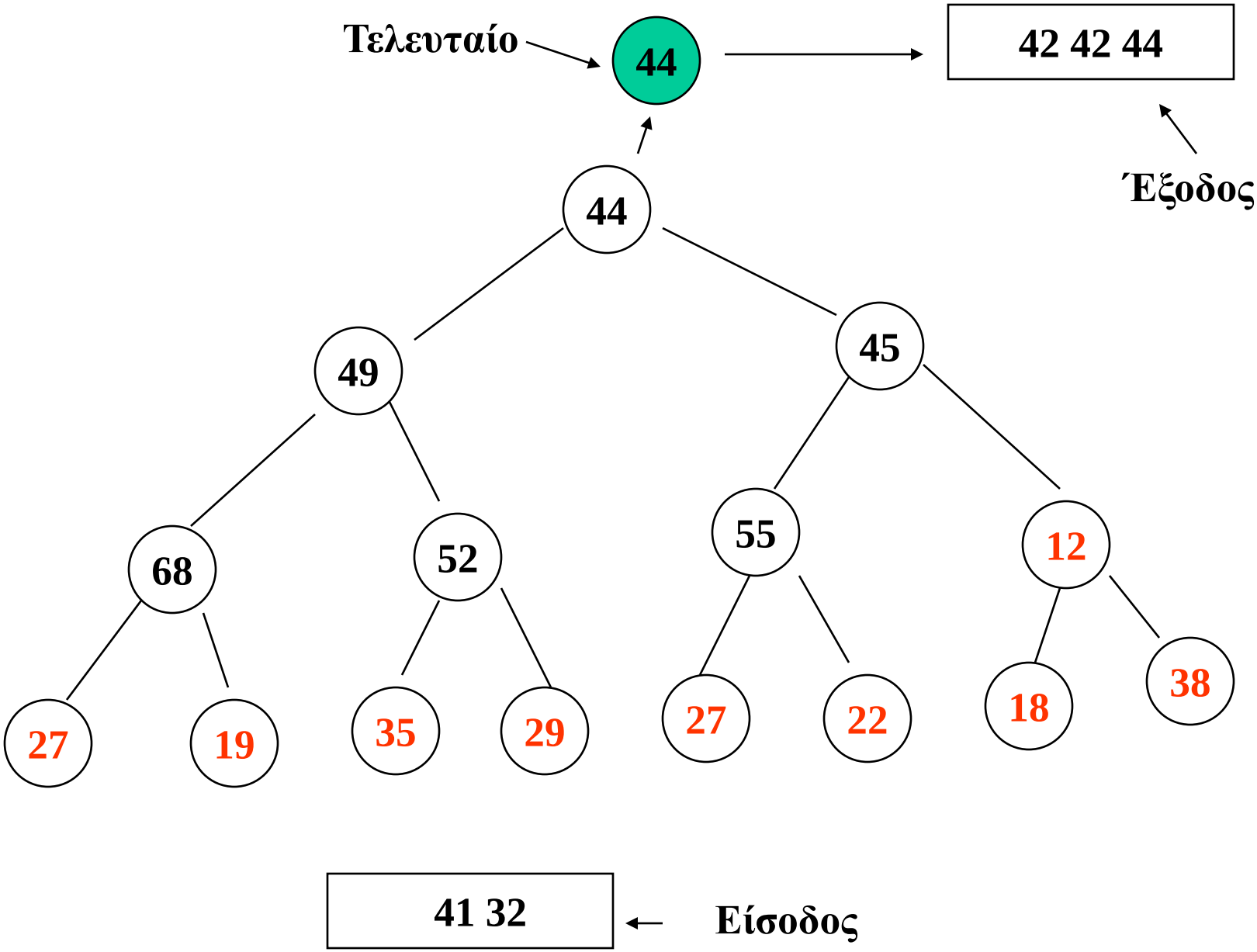

Τελευταίο
38
38 38
Έξοδος
38
42
40
42
44
42
55
49
52
35
29
27
22
18
12
45 19 27
Είσοδος

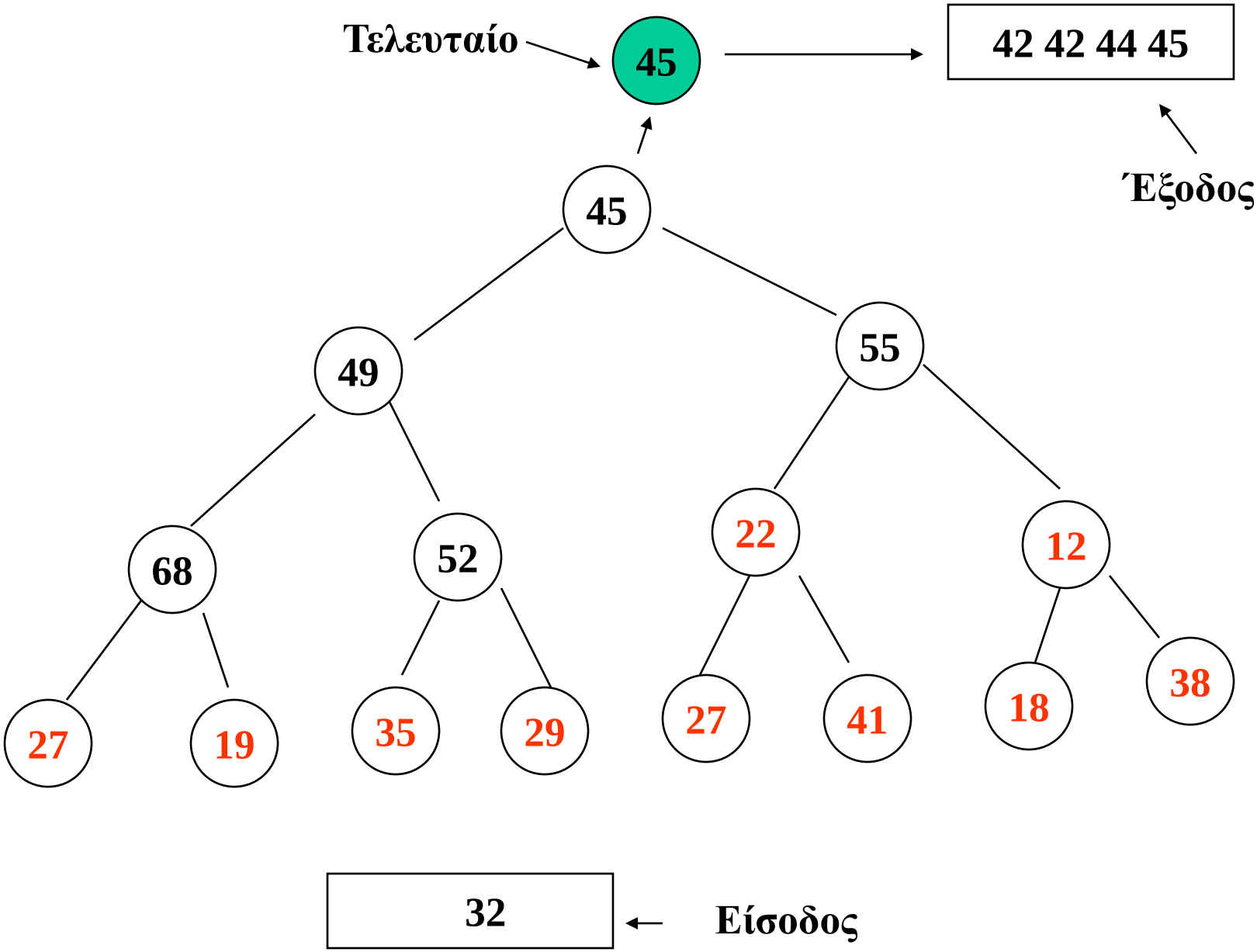

Τελευταίο
40
38 38 40
Έξοδος
40
42
42
42
44
45
55
49
52
35
29
27
22
18
12
19 27
Είσοδος

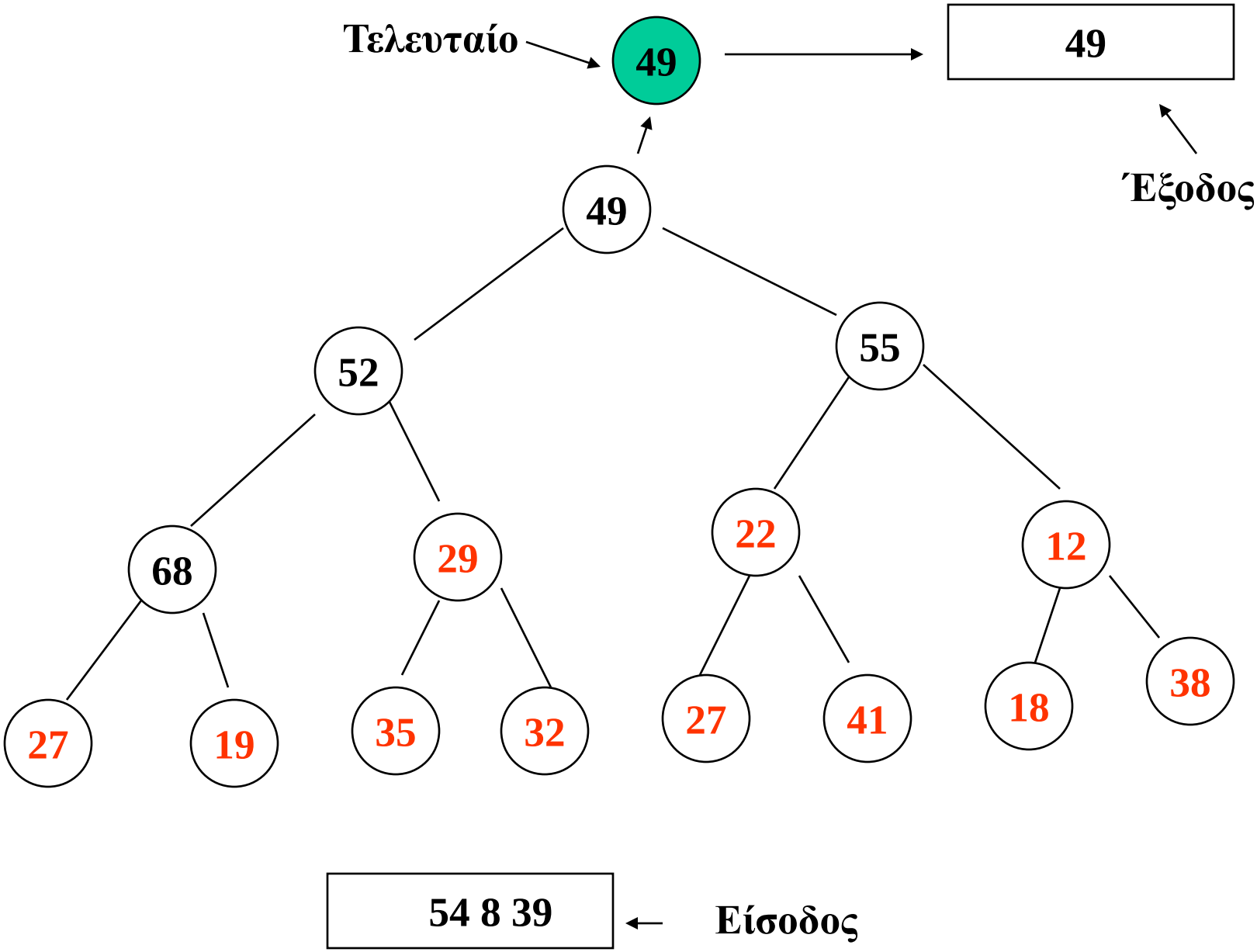

Τελευταίο
42
38 38 40 42
Έξοδος
42
42
42
49
44
45
55
52
19
35
29
27
22
18
12
27
Είσοδος

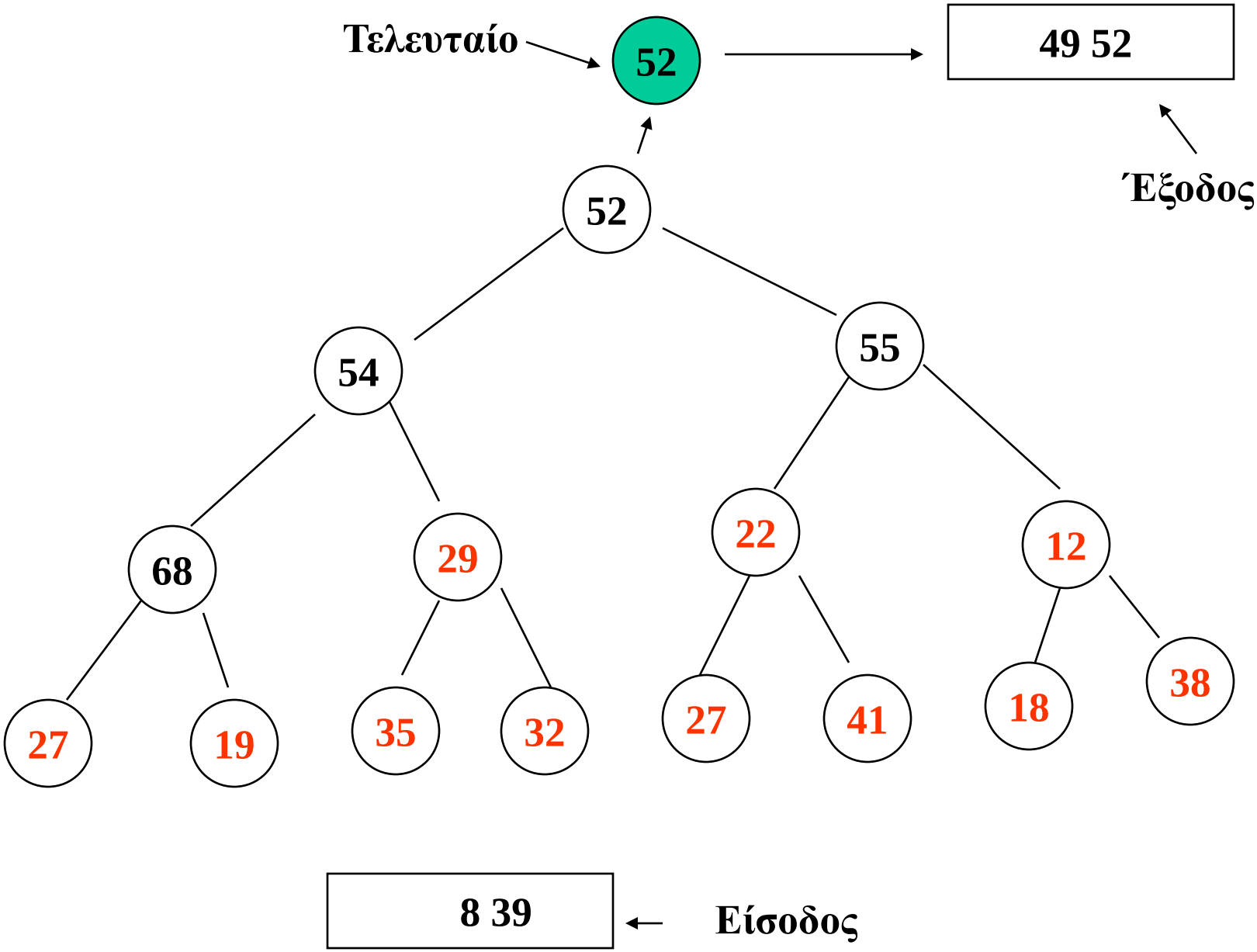

Τελευταίο
42
42
Έξοδος
42
44
42
49
52
45
55
27
19
35
29
27
22
18
12
38 68 41 32
Είσοδος

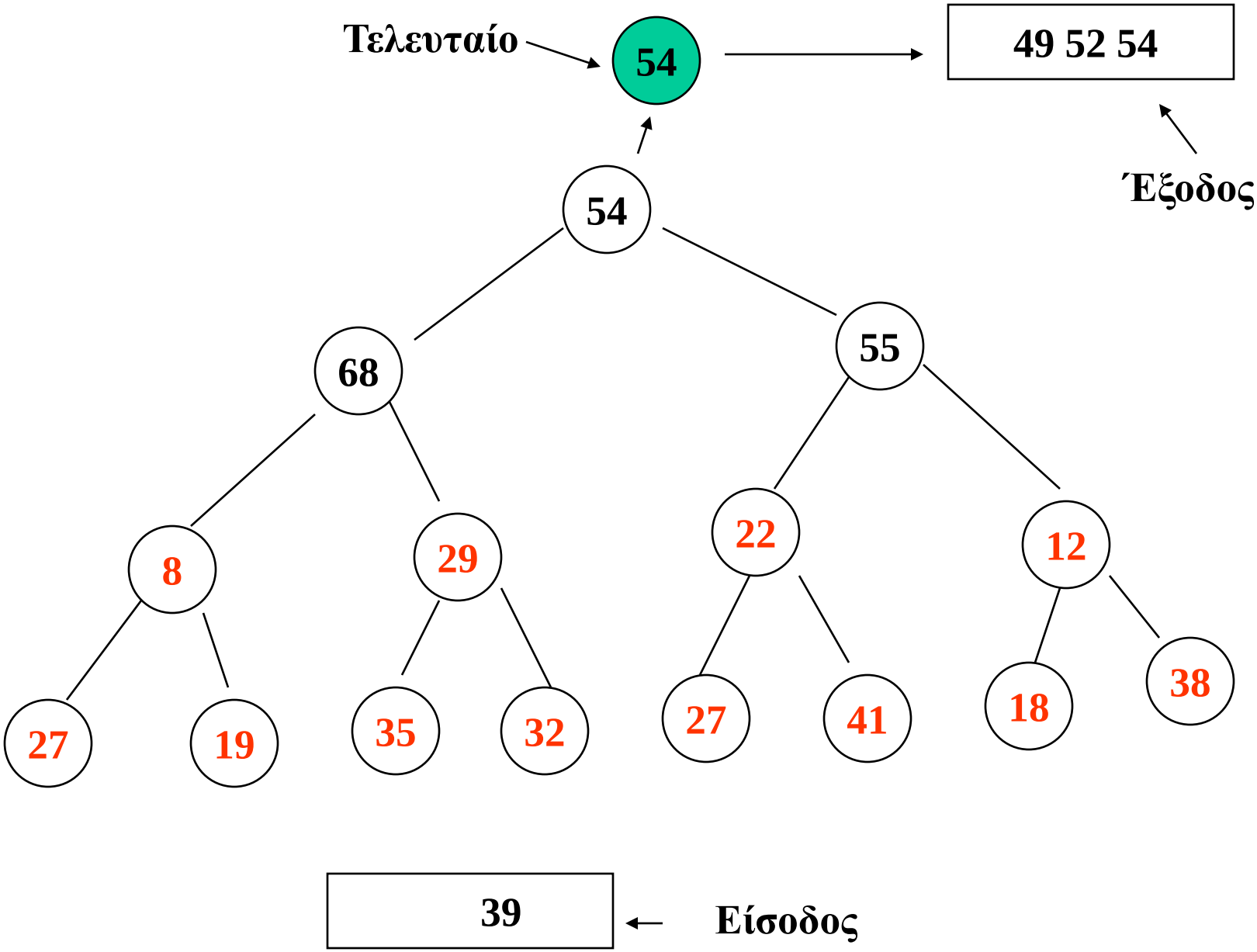

Τελευταίο
42
42 42
Έξοδος
42
44
45
49
52
55
12
27
19
35
29
27
22
18
38
68 41 32
Είσοδος

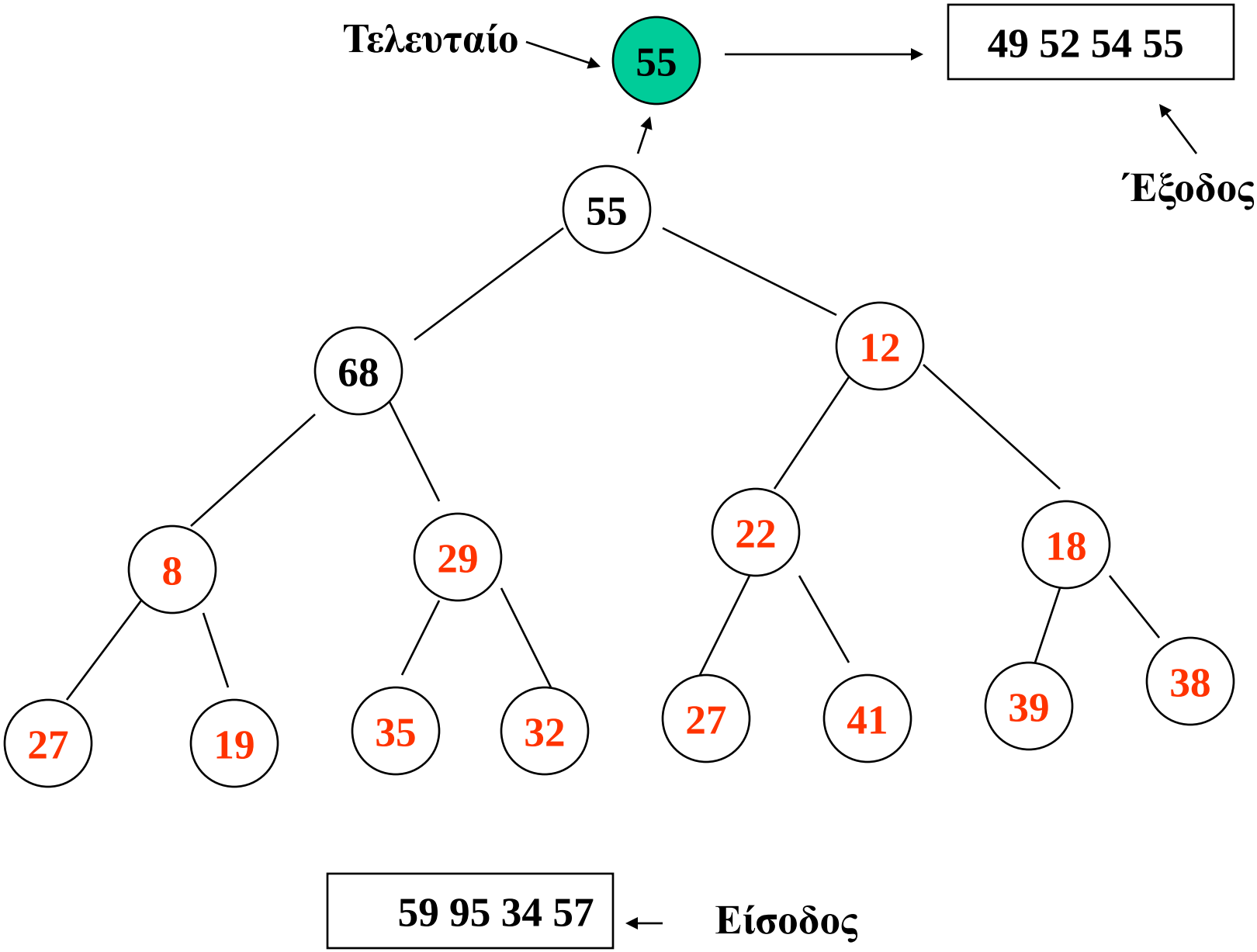

Τελευταίο
44
42 42 44
Έξοδος
44
49
45
68
52
55
12
27
19
35
29
27
22
18
38
41 32
Είσοδος

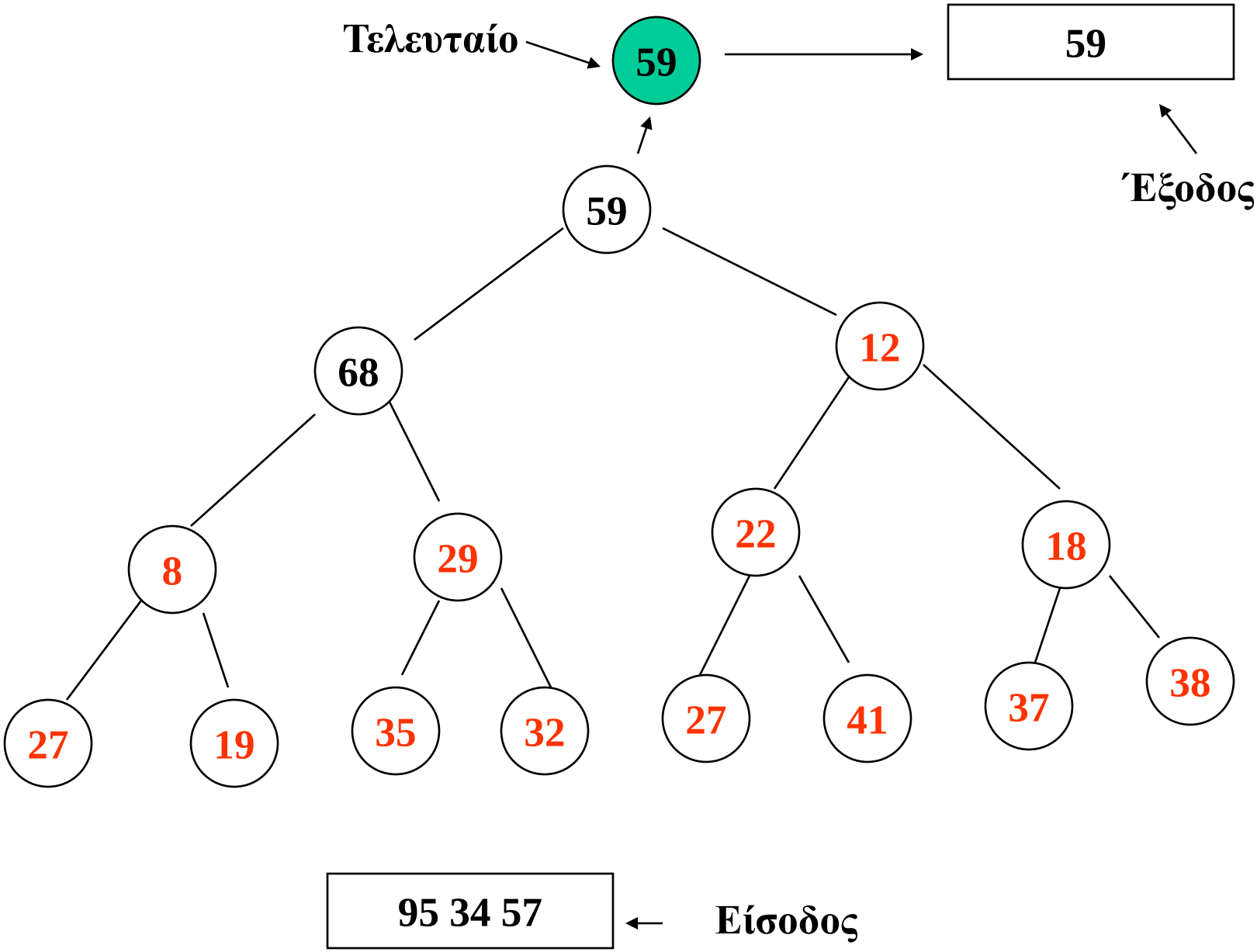

Τελευταίο
45
42 42 44 45
Έξοδος
45
49
55
68
52
22
12
27
19
35
29
27
41
18
38
32
Είσοδος

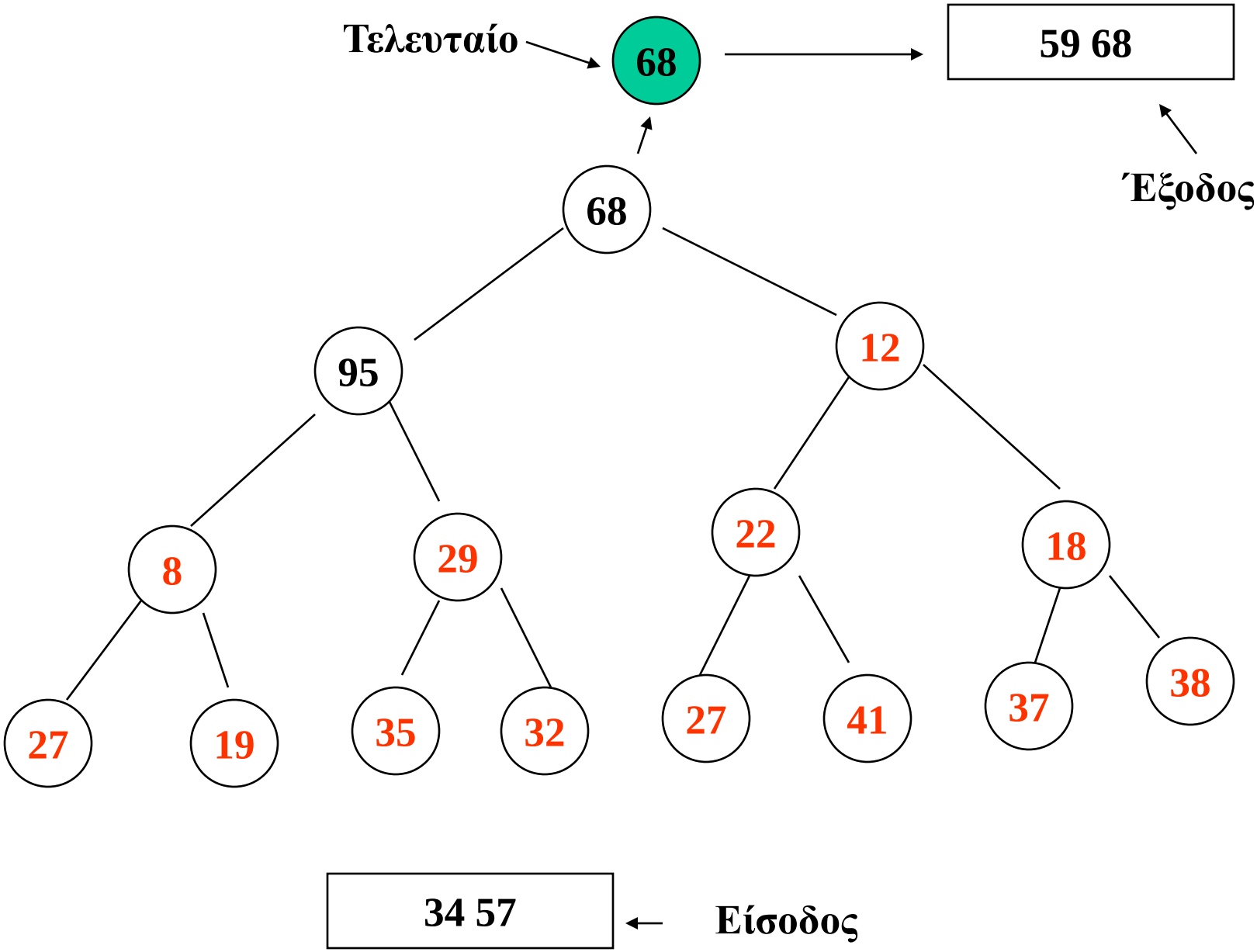

Τελευταίο
49
49
Έξοδος
49
52
55
68
29
22
12
27
19
35
32
27
41
18
38
54 8 39
Είσοδος

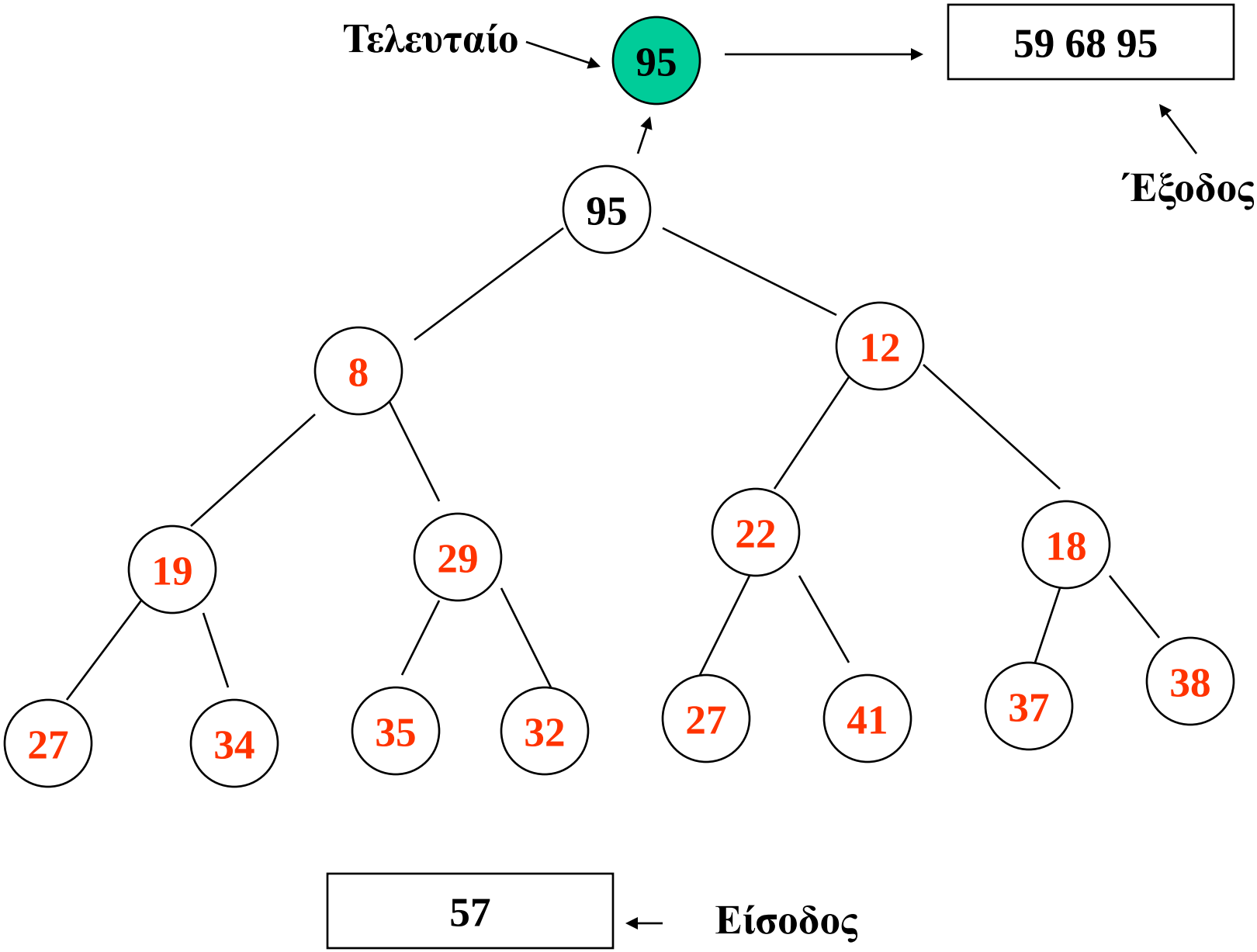

Τελευταίο
52
49 52
Έξοδος
52
54
55
68
29
22
12
27
19
35
32
27
41
18
38
8 39
Είσοδος

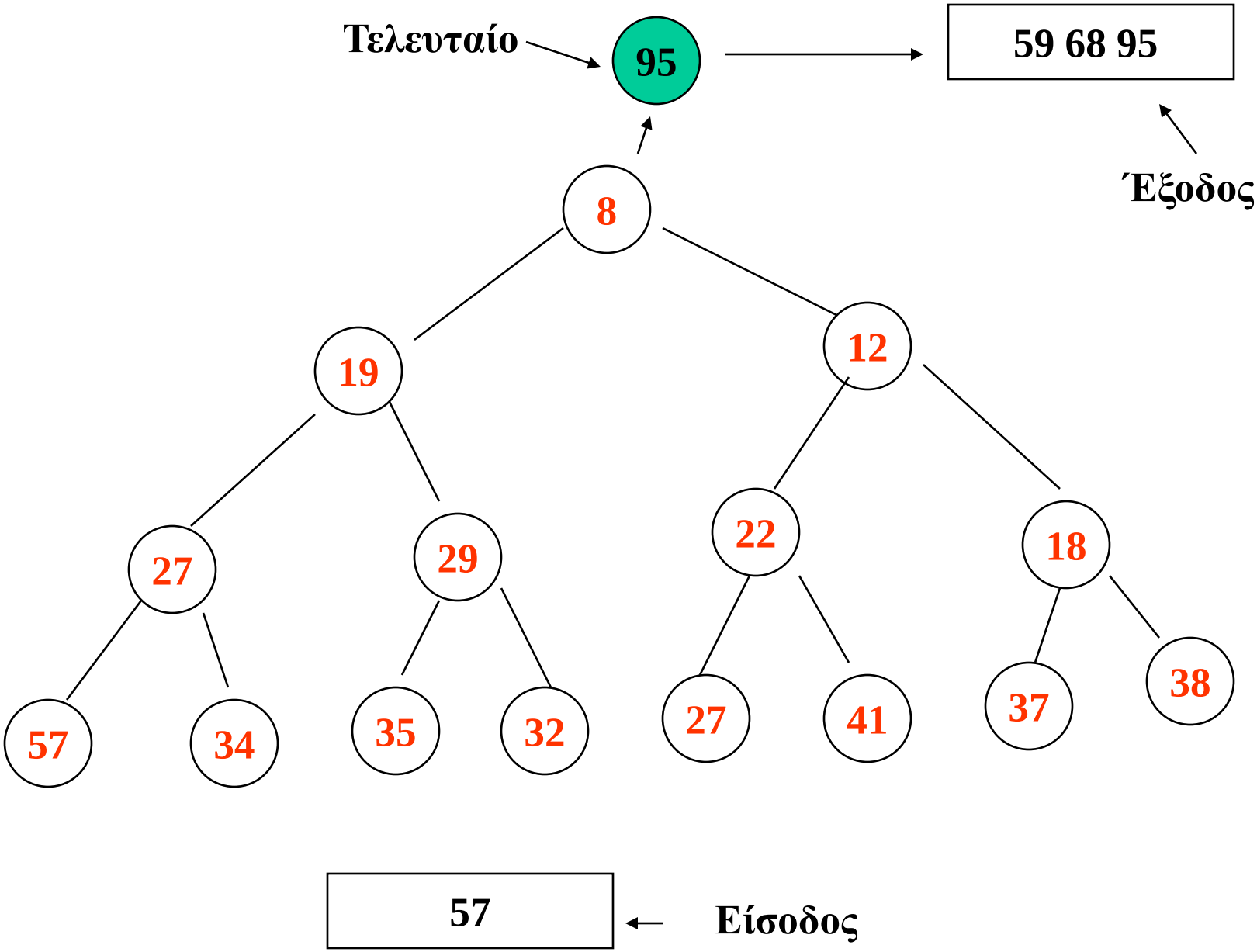

Τελευταίο
54
49 52 54
Έξοδος
54
68
55
8
29
22
12
27
19
35
32
27
41
18
38
39
Είσοδος

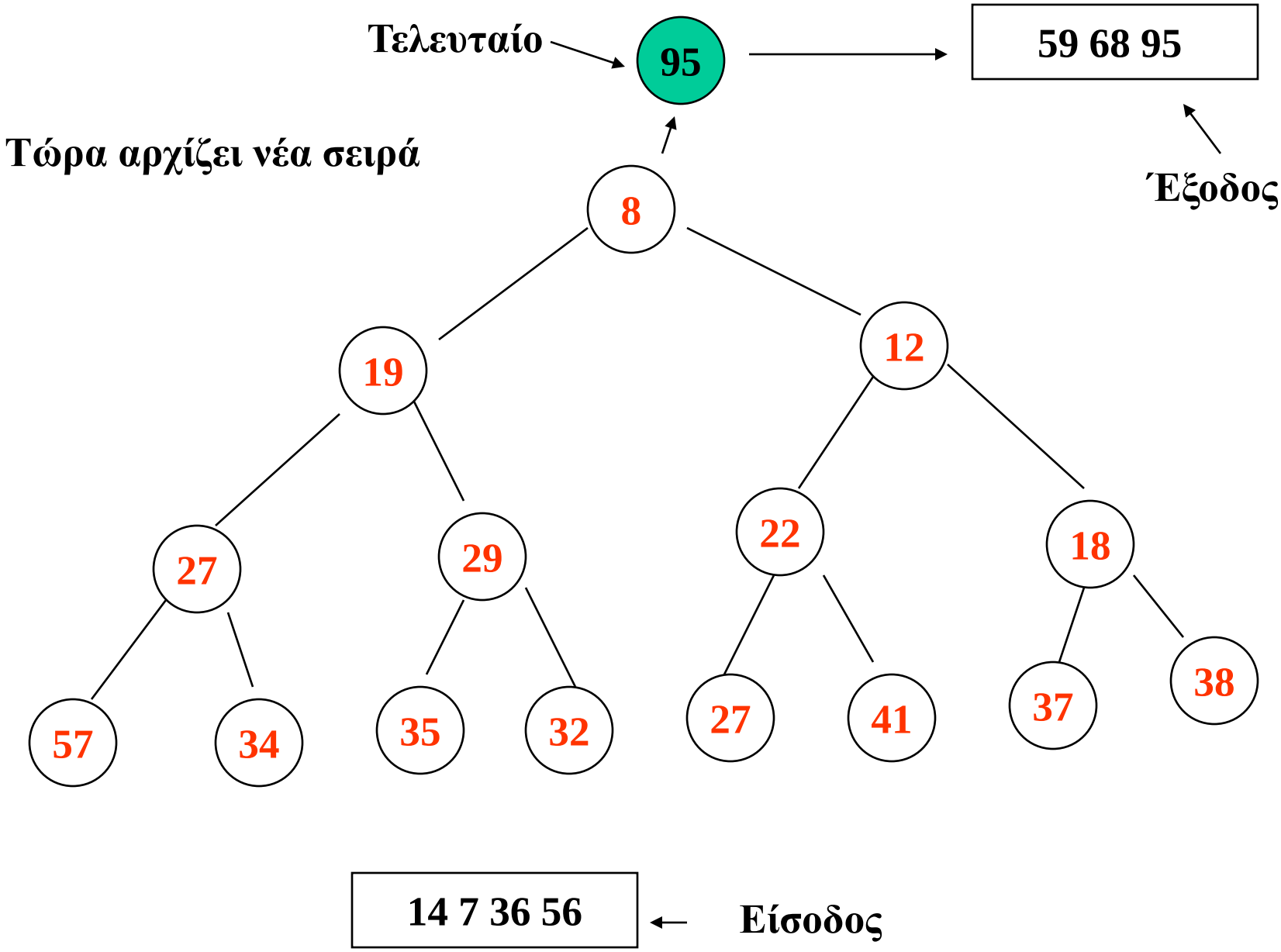

Τελευταίο
55
49 52 54 55
Έξοδος
55
68
12
8
29
22
18
27
19
35
32
27
41
39
38
59 95 34 57
Είσοδος

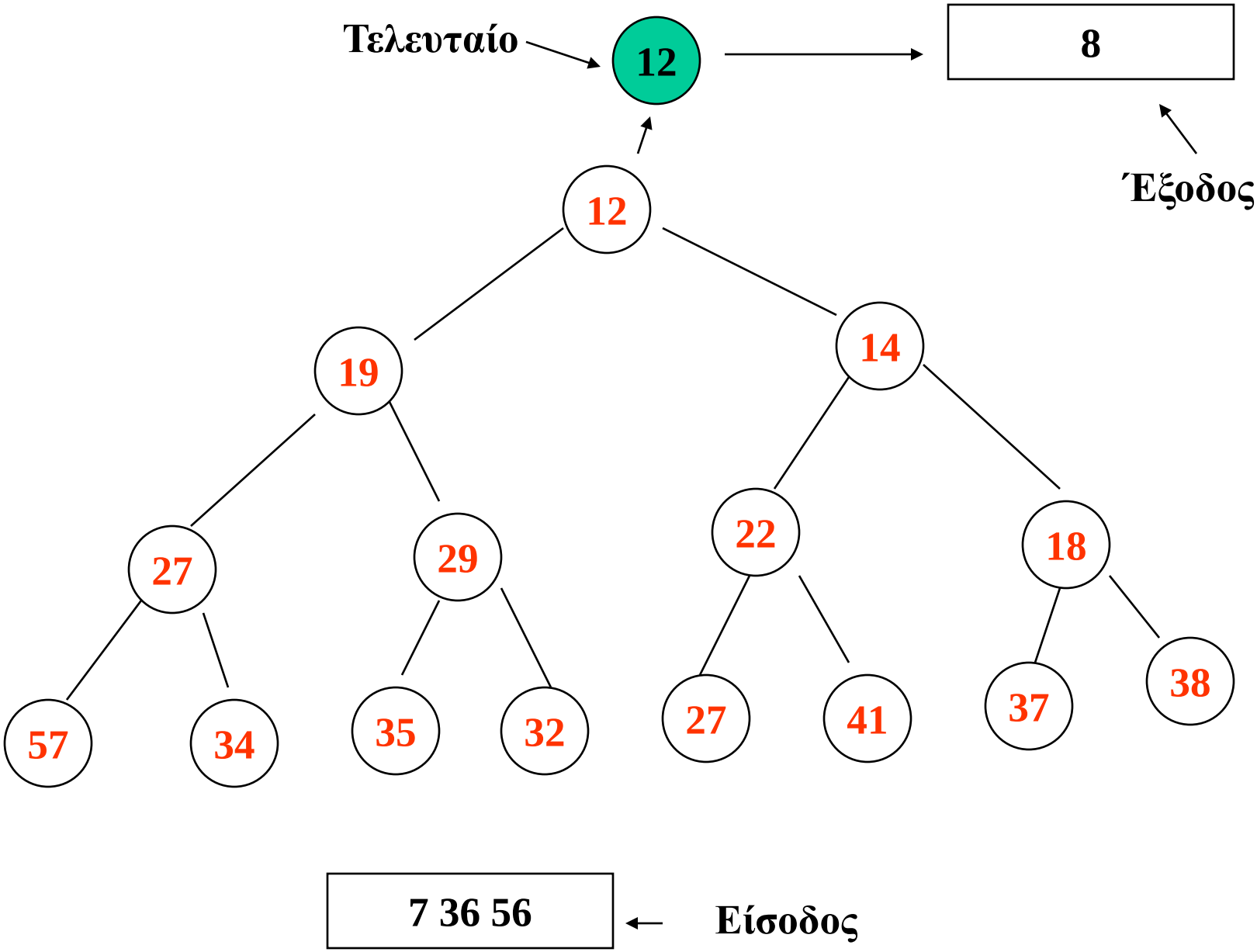

Τελευταίο
59
59
Έξοδος
59
68
12
8
29
22
18
27
19
35
32
27
41
37
38
95 34 57
Είσοδος

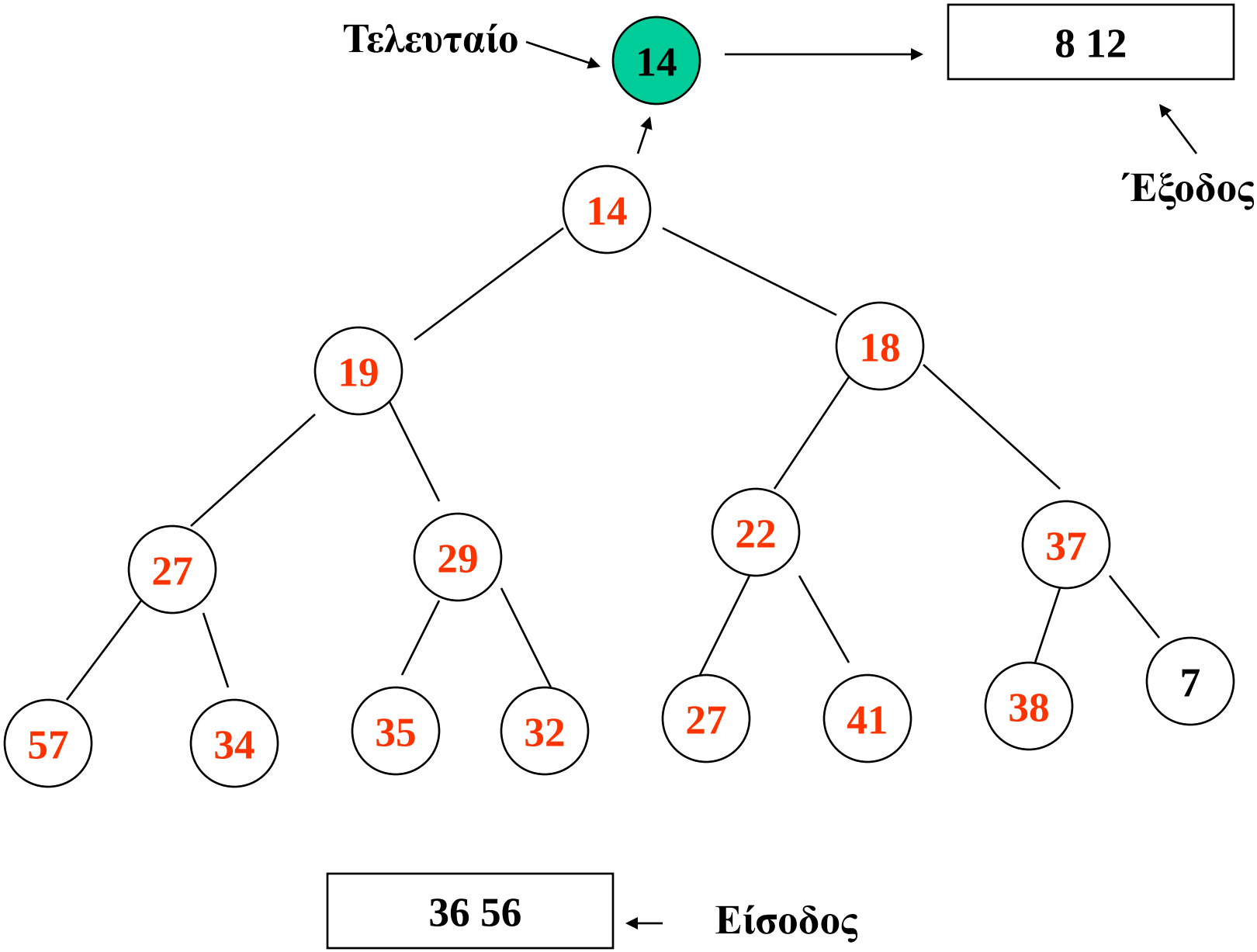

Τελευταίο
68
59 68
Έξοδος
68
95
12
8
29
22
18
27
19
35
32
27
41
37
38
34 57
Είσοδος


Τελευταίο
95
59 68 95
Έξοδος
95
8
12
19
29
22
18
27
34
35
32
27
41
37
38
57
Είσοδος

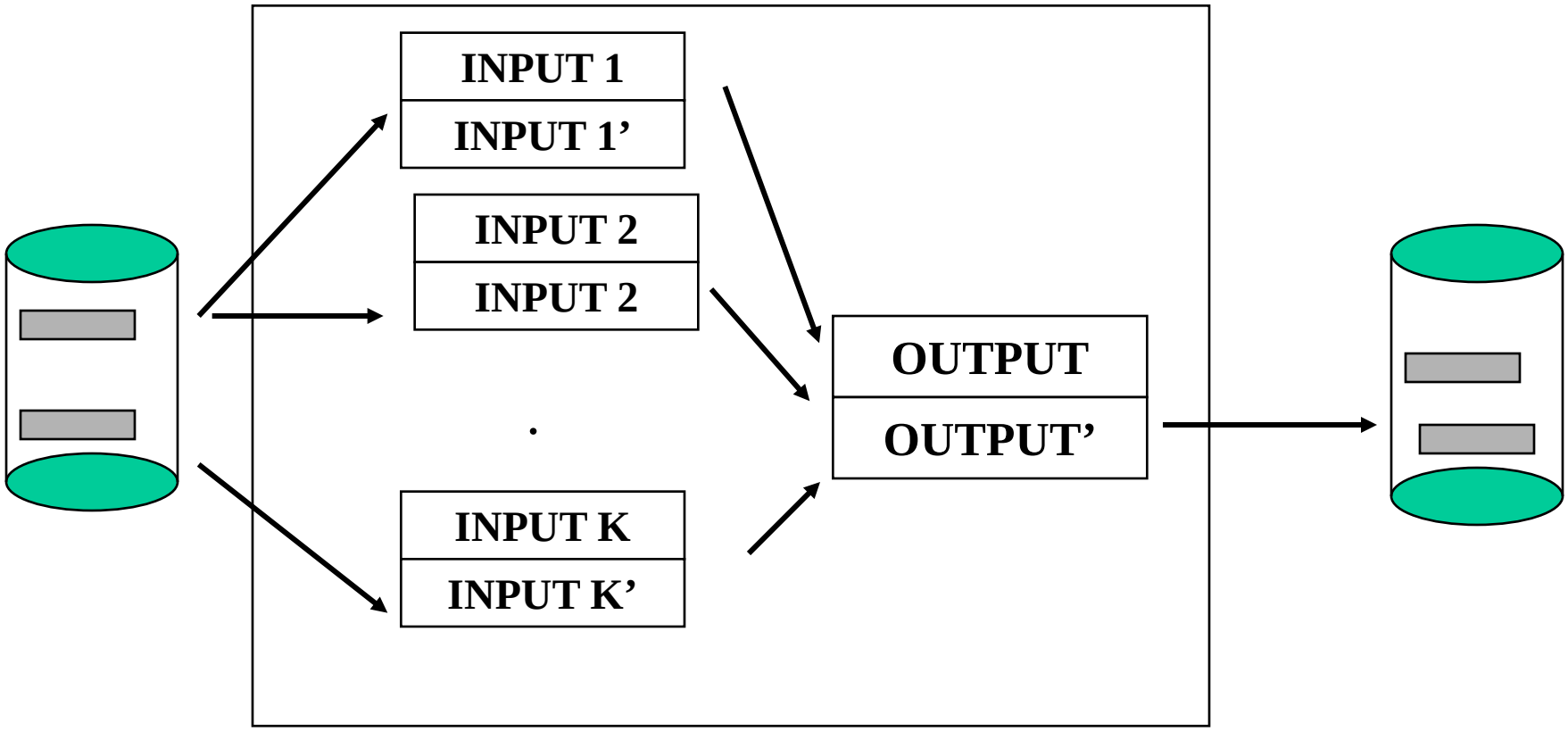

Τελευταίο
95
59 68 95
Έξοδος
8
19
12
27
29
22
18
57
34
35
32
27
41
37
38
57
Είσοδος


Τελευταίο
95
59 68 95
Τώρα αρχίζει νέα σειρά
Έξοδος
8
19
12
27
29
22
18
57
34
35
32
27
41
37
38
14 7 36 56
Είσοδος


Τελευταίο
12
8
Έξοδος
12
19
14
27
29
22
18
57
34
35
32
27
41
37
38
7 36 56
Είσοδος


Τελευταίο
14
8 12
Έξοδος
14
19
18
27
29
22
37
57
34
35
32
27
41
38
7
36 56
Είσοδος

- Όσο λιγότερες ταξινομημένες σειρές υπάρχουν τόσο είναι λιγότερες οι επαναλήψεις
- Για λιγότερες ταξινομημένες σειρές χρειαζόμαστε μεγάλες αρχικές ταξινομημένες σειρές
- Αποδεικνύεται ότι κατά μέσο όρο ο παραπάνω αλγόριθμος φτιάχνει αρχικές σειρές μεγέθους $2^* n_B$
- Άλλες τεχνικές για ελαχιστοποίηση του I/O χρόνου είναι η χρήση διπλού μπαφερ (που βέβαια μειώνει τον διαθέσιμο χώρο)

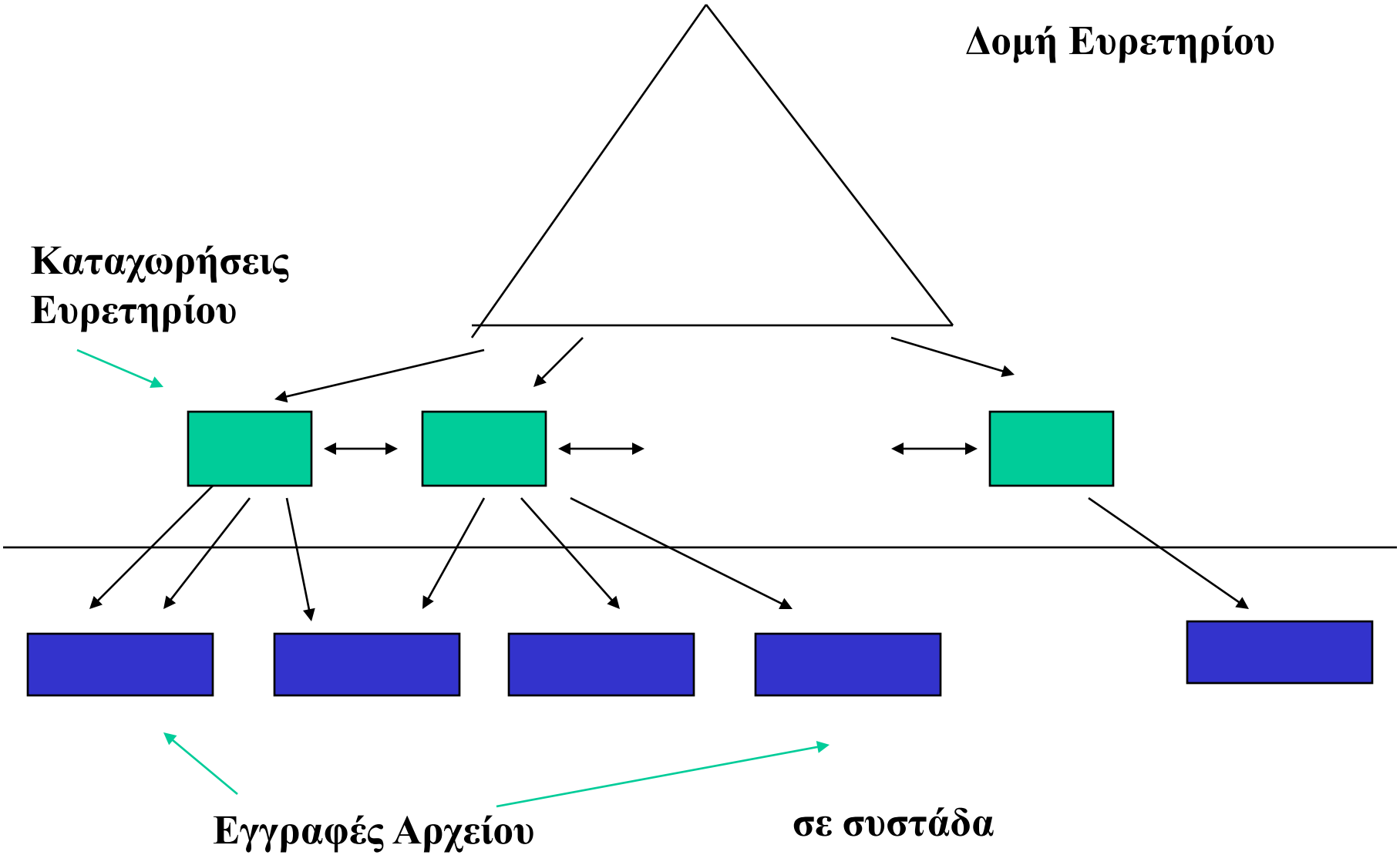

INPUT 1
INPUT 1'
INPUT 2
INPUT 2
.
INPUT K
INPUT K'
OUTPUT
OUTPUT'

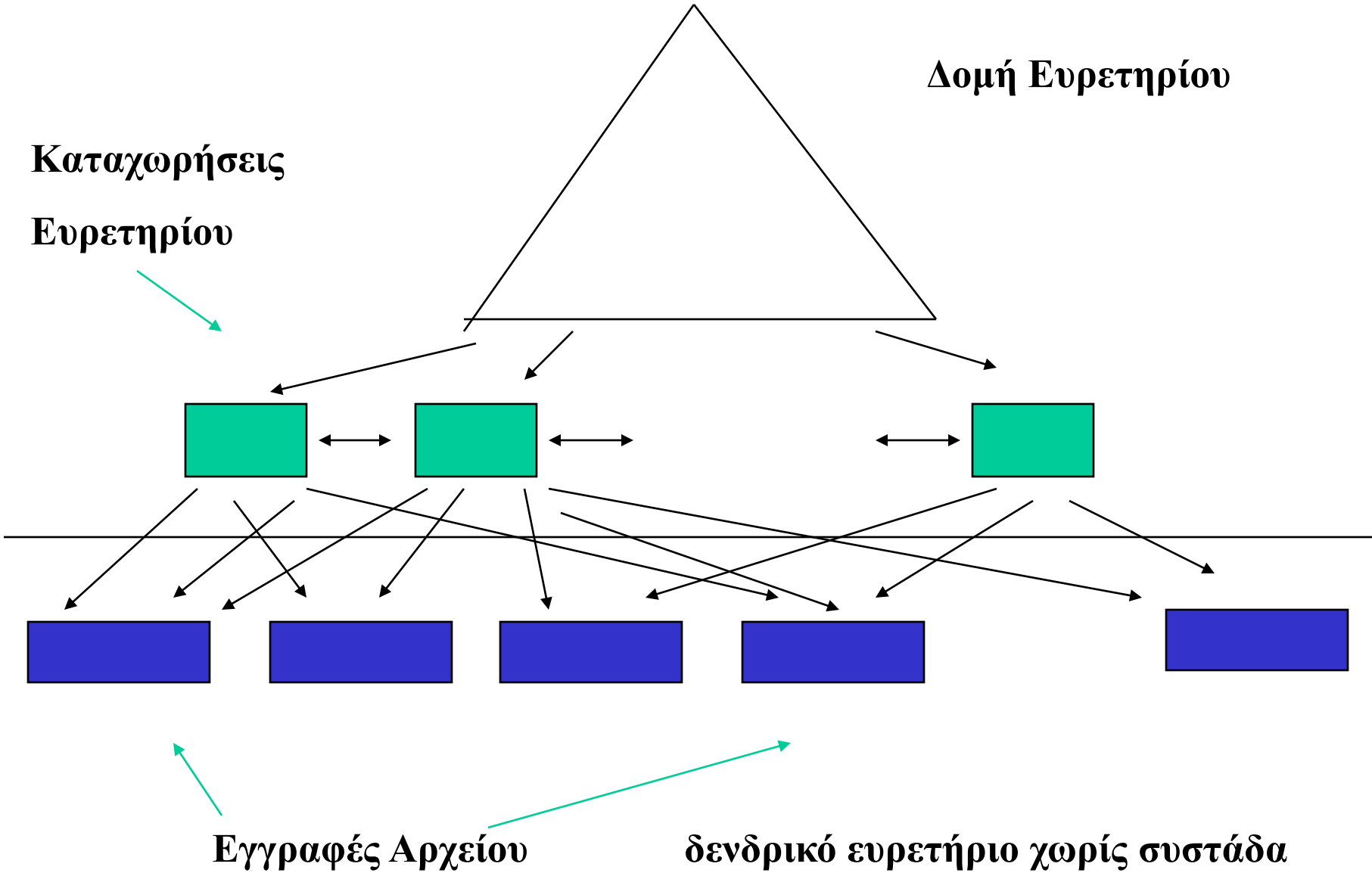
Η ταξινόμηση αρχείων γενικά κοστίζει

- Παλαιό πρόβλημα των απανταχού μηχανογράφων
  - Μια λύση η χρήση παραλληλίας
- Σε δοκιμές 1M εγγραφών μεγέθους 100 bytes
  - Τυπικά ΣΔΒΔ 15 λεπτά
  - 3,5 δευτερόλεπτα με 12 CPUs SGI μηχανή, 96 δίσκους, 2GB ram
- Νέοι τύποι δοκιμών
  - Πόσες εγγραφές μπορώ να ταξινομήσω στη μονάδα χρόνου
  - Πόσες εγγραφές μπορώ να ταξινομήσω με βάση το κόστος

Χρήση των B+ δένδρων για ταξινόμηση

- Το αρχείο που θέλουμε να ταξινομήσουμε έχει ευρετήριο B+ δένδρου στο πεδίο ταξινόμησης.
- Μπορούμε να ανακτήσουμε τις εγγραφές με σαρώνοντας στη σειρά τους κόμβους φύλλα
- Είναι καλή ιδέα?
- Θα θεωρήσουμε δύο περιπτώσεις:
  - Το B+ δένδρο είναι συστάδα (καλή ιδέα)
  - Το B+ δένδρο δεν είναι συστάδα (μπορεί να είναι πολύ κακή ιδέα)


Δομή Ευρετηρίου
Καταχωρήσεις Ευρετηρίου
Εγγραφές Αρχείου
σε συστάδα


Δομή Ευρετηρίου
Καταχωρήσεις
Ευρετηρίου
Εγγραφές Αρχείου
δενδρικό ευρετήριο χωρίς συστάδα