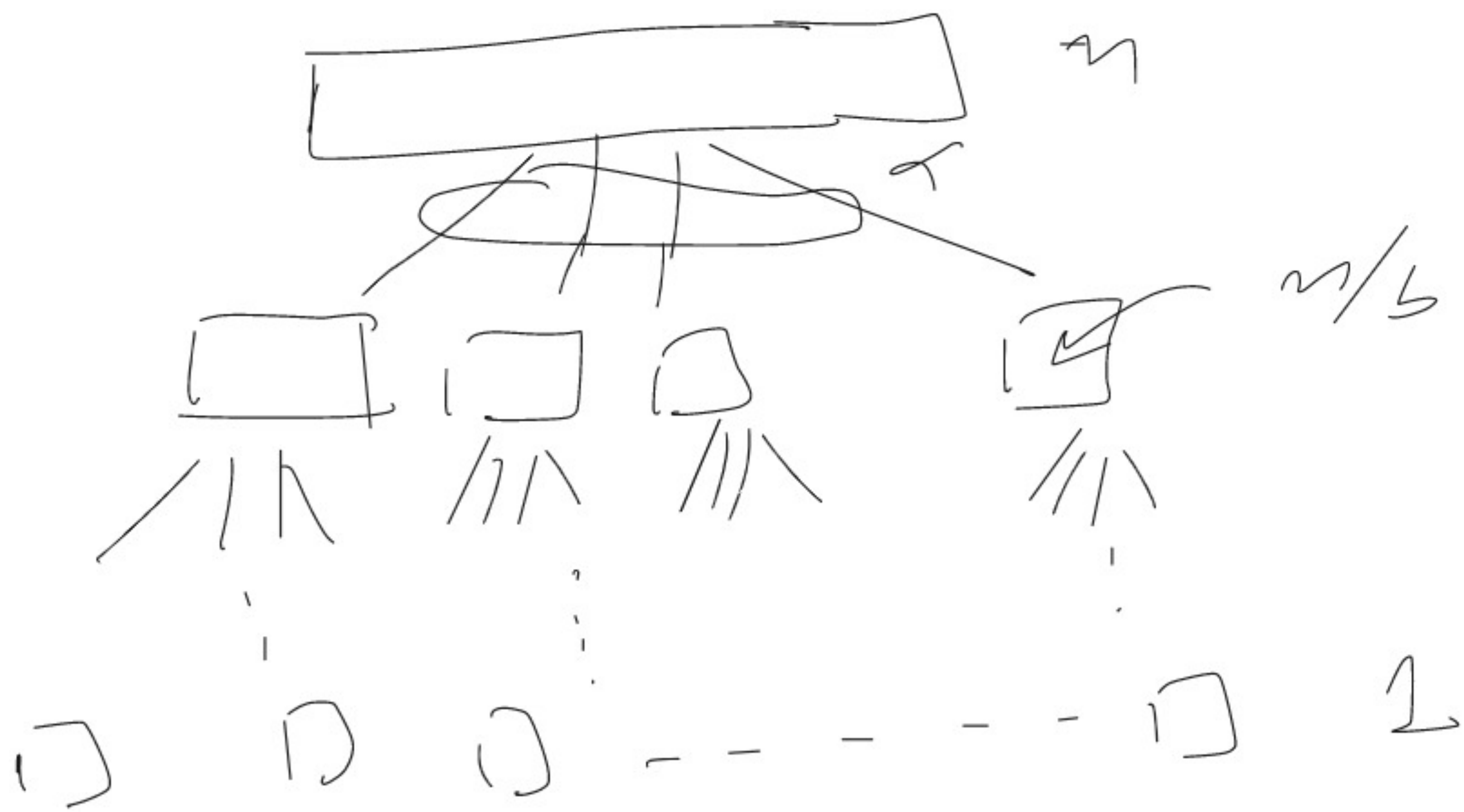
Διαίρει και Βασίλευε

1ο στάδιο: Διάσπαση του προβλήματος σε υποπροβλήματα

2ο στάδιο: Αναδρομική λύση των υποπροβλημάτων

3ο στάδιο: Κατάλληλος συνδυασμός των απαντήσεων.



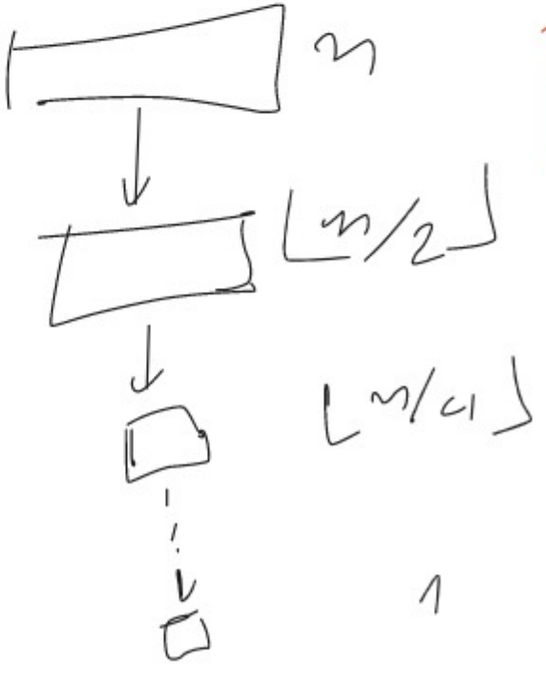
Binary search (Z, n, k):

Είσοδος: Ταξινομημένος πίνακας Z με n-πολλά (διαφορετικούς μεταξύ τους) και αριθμό k

Έξοδος: Η θέση του k στον Z (αν υπάρχει)

```
L = 1
R = n
while L ≤ R
    M = ⌊(L+R)/2⌋
    if Z[M] < k : L = M+1
    elseif Z[M] > k : R = M-1
    else return M
return "Δεν βρέθηκε"
```

$O(1)$ $O(1)$ $O(1)$ $O(\log n)$ επαναλ.



$\log n$

χείρως $O(\log n)$

Binary search (Z, L, R, k):

Είσοδος: Ταξιν. πίνακας Z, L, k ακέραιοι (με νόημα μεταξύ των δεικτών του πίνακα) και αριθμός k

Έξοδος: Η θέση του k στον Z από τη θέση L ως τη θέση R (αν υπάρχει)

```
if L > R
    return "Δεν βρέθηκε"
M = ⌊(L+R)/2⌋
if Z[M] < k : return BinarySearch(Z, M+1, R, k)
elseif Z[M] > k : return BinarySearch(Z, L, M-1, k)
else return M.
```

ο χρόνος του αλγορίθμου προκύπτει από την αναδρομική σχέση:

$$\begin{cases} T(n) = T(\lceil n/2 \rceil) + c = (*) \\ T(0) = c \end{cases}$$

$$(*) = T(\lceil n/4 \rceil) + c + c = \dots = T\left(\left\lceil \frac{n}{2^{\log n}} \right\rceil\right) + \underbrace{c + \dots + c}_{\log n} =$$

$$= (\log n + 1) \cdot c = O(\log n)$$

MergeSort (α[1, .., n])

Είσοδος: Μία ακολουθία αριθμών α[1 .. n]

Έξοδος: Μία ταξινομημένη εκδοχή της ακολουθίας

```
if n>1:
    return merge(mergesort(α[1 .. ⌊n/2⌋]), mergesort(α[⌊n/2⌋+1 ... n]))
else: return α
```

μέγεθος εισόδου n:

a $O(n^d)$

$$\begin{cases} T(n) = 2\,T(n/2) + O(n) \\ T(1) = O(1) \end{cases}$$

n/b

$$O(n \log n)$$

μέγεθος εισόδου k+l

merge (X[1 ... k], Y[1 ... l]):

```
if k=0 : return Y[1 ... l]
if l=0 : return X[1 .. k]
if X[1] ≤ Y[1] :
        return X[1] ∘ merge(X[2 ... k], Y[1 ... l])
else :  return Y[1] ∘ merge(X[1 ... k], Y[2 ... l])
```

} $O(1)$

παράθεση (∘)

$$T(k+l) = T(k+l-1) + O(1) = O(k+l)$$

miro

Iterative-mergesort(A[1...n]):

Είσοδος: Συλλογή Α με η στοιχεία;

Έξοδος: Η Α ταξινομημένη

Q = [] (κενή ουρά)

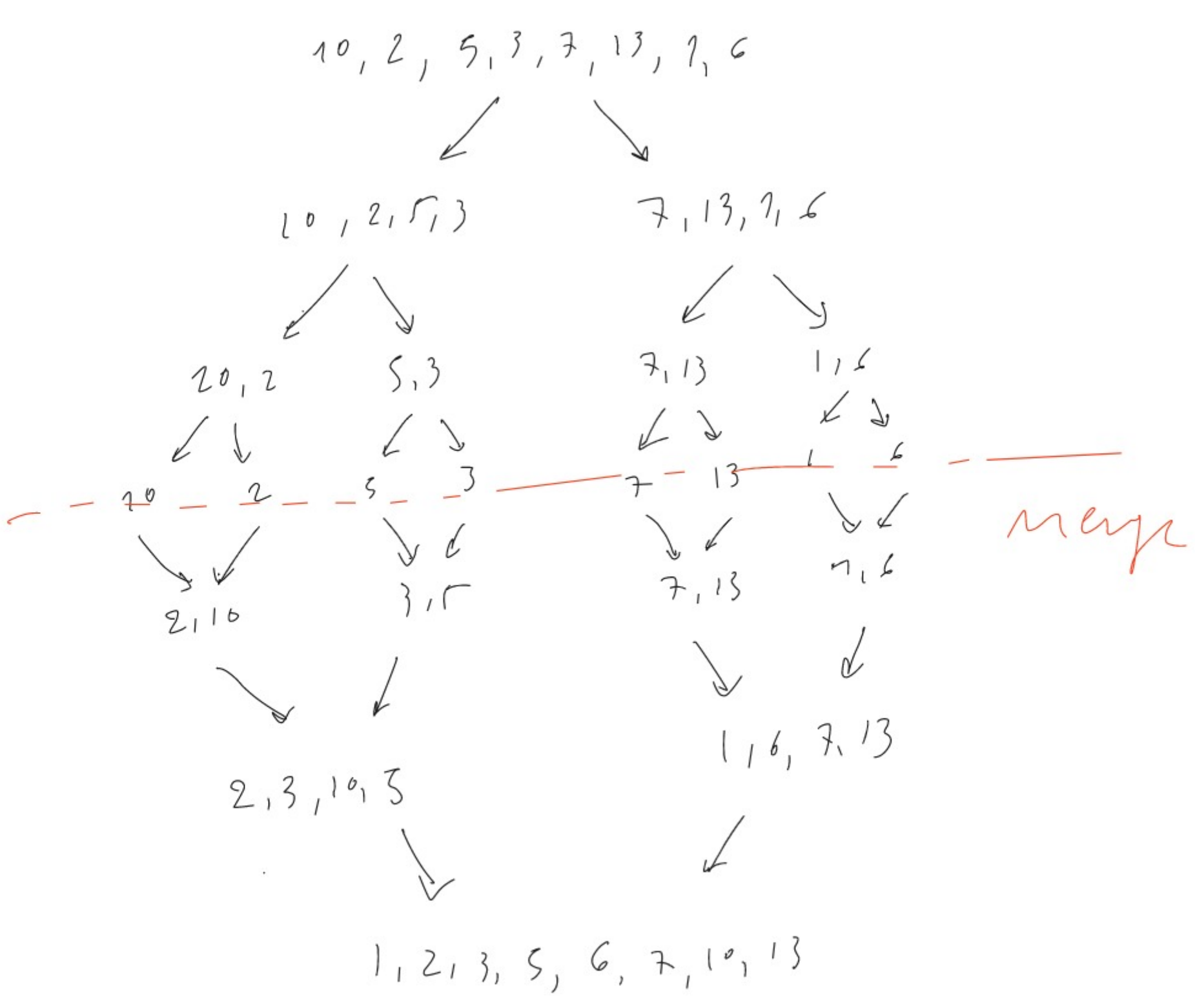
for i = 1 to n:

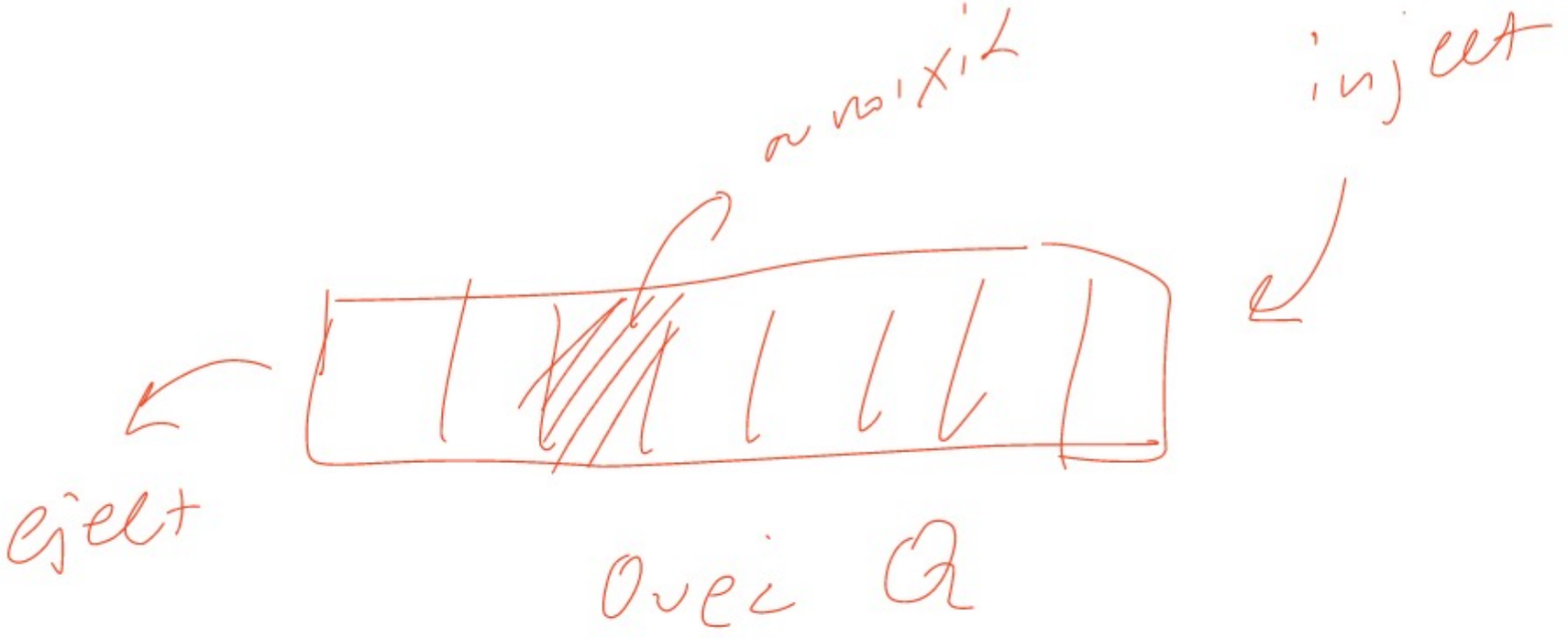
    inject(Q, [A[i]])

while |Q| > 1:

    inject(Q, merge(eject(Q), eject(Q)))

return eject(Q).


10, 2, 5, 3, 7, 13, 1, 6
10, 2, 5, 3
7, 13, 1, 6
10, 2
5, 3
7, 13
1, 6
10
2
5
3
7
13
1
6
merge
2, 10
3, 5
7, 13
1, 6
2, 3, 10, 5
1, 6, 7, 13
1, 2, 3, 5, 6, 7, 10, 13



Κύριο Θεώρημα: Αν $T(n) = \alpha \cdot T(\lceil n/b \rceil) + O(n^d)$ με $\alpha > 0, b > 1$ και $d \geq 0$ τότε:

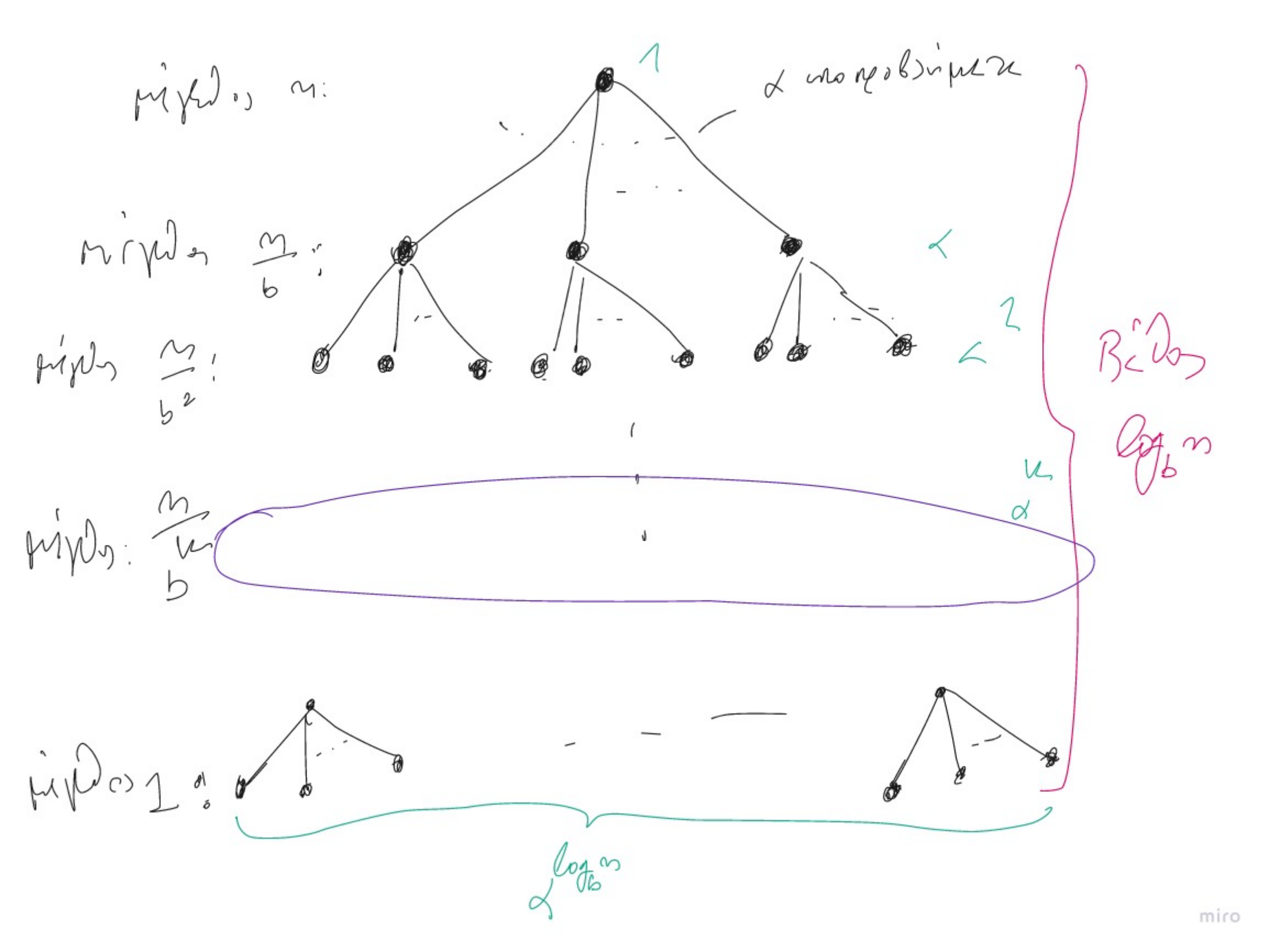
$$T(n) = \begin{cases} O(n^d) & , \text{αν } d > \log_b \alpha \\ O(n^d \log n) & , \text{αν } d = \log_b \alpha \\ O(n^{\log_b \alpha}) & , \text{αν } d < \log_b \alpha \end{cases}$$

Ανάλ.

MergeSort: $T(n) = 2 \cdot T(n/2) + O(n)$

$\left.\begin{array}{l} \alpha = 2 \\ b = 2 \\ d = 1 \end{array}\right\}$ $\log_2 2 = 1 = d$

$O(n \cdot \log n)$


μέγεθος n:
μέγεθος n/b:
μέγεθος n/b²:
μέγεθος: n/b^k
μέγεθος 1:
1
α υποπροβλήματα
α
α²
α^k
βάθος
log_b n
α^(log_b n)

$$T(n) = \alpha \cdot T\left(\lceil n/b \rceil\right) + O(n^d)$$

Έστω n θετικός ακέραιος και b.

(Τότε το διάστημα $[n, bn]$ περιέχει ακέραιο πολλαπλάσιο του b. Αποδείξτε με επαγωγή:

Βάση: $n = 1$ $[1, b]$ περιέχει τη δύναμη b^1

π.χ.: επαληθεύστε ότι το $[k, b \cdot k]$ περιέχει δύναμη του b για $k = b^2$.

Βήμα: Θ.δ.ο. το $[kn, b(kn)]$ περιέχει δύναμη του b.

- Αν $k = b^2$ τότε $b \cdot k = b^{2+1} \in [kn, b(kn)]$
- Αν $k < b^2$ τότε $b^2 \in [kn, b(kn)]$)

Στο επίπεδο το:

$$a^k \cdot O\left(\left(\frac{n}{b^k}\right)^d\right) = O(n^d) \cdot \left(\frac{a}{b^d}\right)^k$$

Συνολικό κόστος: $T(n) = O(n^d) \sum_{k=0}^{\log_b n} \left(\frac{a}{b^d}\right)^k$

Λήμμα: $c \in \mathbb{R}^+$ τότε $f(n) = 1 + c + c^2 + \cdots + c^n$ είναι:

α) $O(1)$ αν $c < 1$

β) $O(n)$ αν $c = 1$

γ) $O(c^n)$ αν $c > 1$

} Μπορούν και με Θ.

Ανάλυση περίπτωσης:

α) Αν $c<1$ τότε $g(n) \leq \sum_{i=0}^{\infty} c^i \overset{c<1}{=} \frac{1}{1-c} = O(1)$

β) Αν $c=1$ τότε $g(n) = \underbrace{1+1+\cdots+1}_{(n+1)-\text{φορές}} = n+1 =$

$= O(n)$

γ) Αν $c>1$ τότε $g(n) = \frac{1-c^{n+1}}{1-c} \leq$

$\leq \frac{c^n - c^{n+1}}{1-c} = \frac{(1-c)\cdot c^n}{1-c} =$

$= c^n = O(c^n).$

$\frac{a}{b^d} = c$

- $c < 1 \Rightarrow a < b^d \Rightarrow \log_b a < d$ ίσως:

$$T(n) = O(n^d) \cdot O(1) = O(n^d)$$

- $c = 1 \Rightarrow \log_b a = d$ ίσως

$$T(n) = O(n^d) \cdot O(\log n) = O(n^d \log n)$$

- $c > 1 \Rightarrow \log_b a > d$ ίσως:

$$T(n) = O(n^d) \cdot O\left(\left(\frac{a}{b^d}\right)^{\log_b n}\right) =$$

$$= O\left(n^d \cdot \left(\frac{a}{b^d}\right)^{\log_b n}\right) =$$

$$= O\left(n^d \frac{a^{\log_b n}}{b^{d \cdot \log_b n}}\right) = O\left(n^d \cdot \frac{a^{\log_b n}}{n^d}\right) =$$

$$= O\left(a^{\log_b n}\right) \cdot \frac{\log_a n = \frac{\log_b n}{\log_b a}}{}$$

$$= O\left(a^{\log_a n \cdot \log_b a}\right) = O\left(n^{\log_b a}\right)$$