



Διάλεξη 09: Αλγόριθμοι Ταξινόμησης I

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

- Οι αλγόριθμοι ταξινόμησης:

A. SelectionSort – Ταξινόμηση με Επιλογή

B. InsertionSort – Ταξινόμηση με Εισαγωγή

Γ. MergeSort – Ταξινόμηση με Συγχώνευση

Αλγόριθμοι ταξινόμησης

Δοθέντων μιας **συνάρτησης f (ordering function)** και ενός **συνόλου στοιχείων**

$$x_1, x_2, \dots, x_n$$

η ταξινόμηση συνίσταται στη **μετάθεση** των στοιχείων ώστε να μπουν σε μια σειρά

$$x_{k_1}, x_{k_2}, \dots, x_{k_n}$$

η οποία να ικανοποιεί

$$f(x_{k_1}) \leq f(x_{k_2}) \leq \dots \leq f(x_{k_n}) \text{ **αύξουσα σειρά**}$$

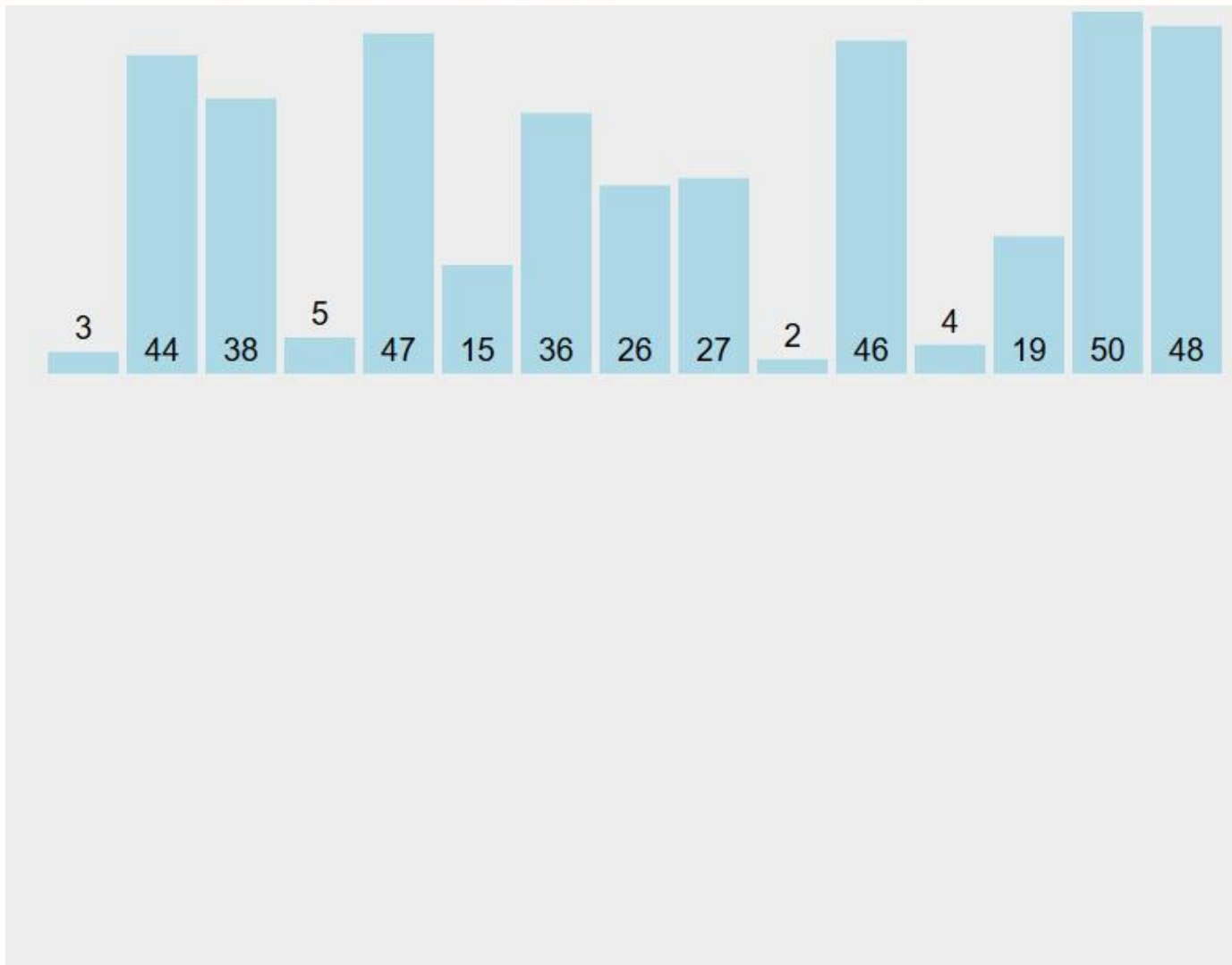
ή

$$f(x_{k_1}) \geq f(x_{k_2}) \geq \dots \geq f(x_{k_n}) \text{ **φθίνουσα σειρά**}$$

- Ταξινόμηση Ονομάτων: $f(\text{"Maria"}) < f(\text{"Michalis"})$, δηλαδή η συνάρτηση f συγκρίνει τις δυο λέξεις αλφαριθμητικά.

Θα εξετάσουμε αλγόριθμους ταξινόμησης με κύριο γνώμονα την αποδοτικότητά τους (χρόνος εκτέλεσης, χρήση μνήμης).

Bubble Sort Animation



<https://visualgo.net/bn/sorting>

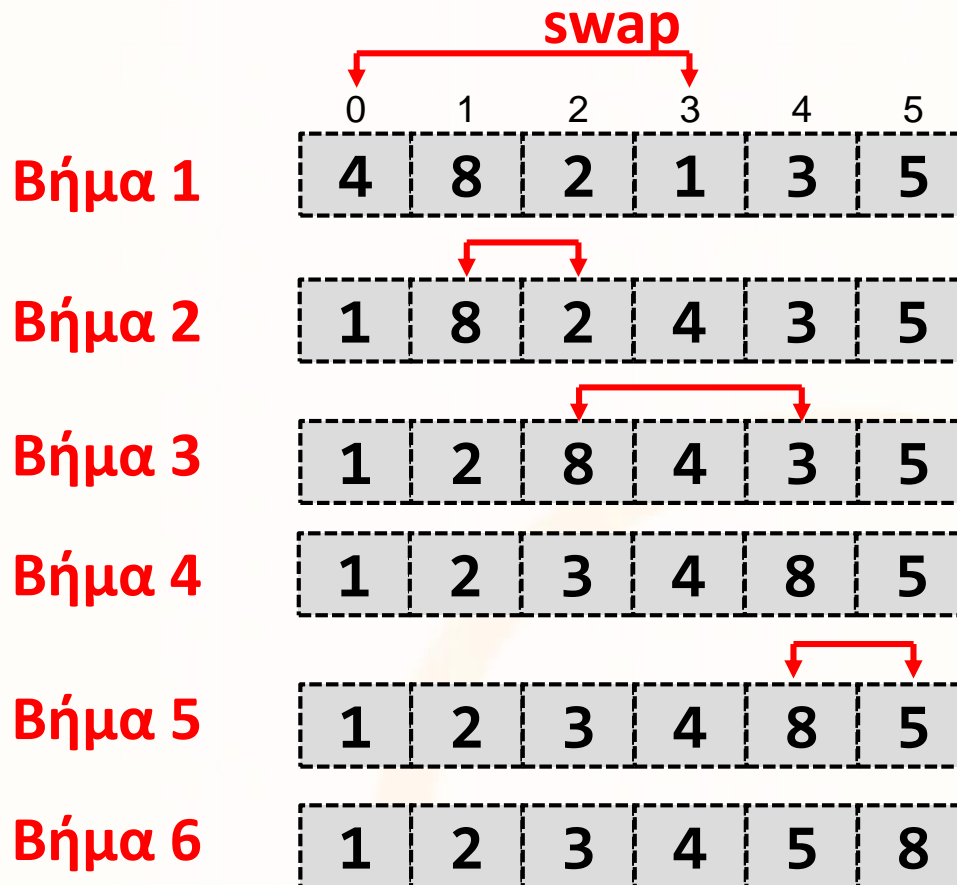
A. Ταξινόμηση με Επιλογή - Selection Sort

- Η Ταξινόμηση με Επιλογή (Selection Sort) βασίζεται στα ακόλουθα τρία βήματα:
 1. επιλογή του ελάχιστου στοιχείου
 2. ανταλλαγή με το i -οστό στοιχείο (i είναι μια μεταβλητή που αυξάνεται κατά ένα).
 3. επανάληψη των βημάτων 1 και 2 για τα υπόλοιπα στοιχεία.
- Το ελάχιστο μεταξύ i στοιχείων μπορεί να βρεθεί με τη χρήση ενός `while-loop`, σε χρόνο $O(i)$.
- Άρα ο χρόνος εκτέλεσης του Selection Sort είναι:

$$\sum_{i=1}^{n-1} i = O(n^2)$$

Ταξινόμηση με επιλογή: SelectionSort

- SelectionSort: ο αλγόριθμος αυτός ταξινομεί μία λίστα με στοιχεία. Σε κάθε βήμα i βρίσκει το i -οστό πιο μικρό στοιχείο και το τοποθετεί στην θέση i .
- Παράδειγμα: `int x = {4, 8, 2, 1, 3, 5};`



Ταξινόμηση με επιλογή: SelectionSort (συν.)

Πρότυπο Συνάρτησης

```
void selectionSort(int x[],           //ο πίνακας  
                  int length)      //το μέγεθος του x
```

Αλγόριθμος

- Αρχικοποίησε μία μεταβλητή `min_index` η οποία θα αποθηκεύει τη θέση του ελάχιστου αριθμού
- Με ένα βρόγχο `for` που θα ελέγχει ένα ένα τα στοιχεία του πίνακα ($i < \text{length}$) κάνε τα εξής:
 - Θέσε σαν `min_index` το `i`
 - Ψάξε ένα-ένα τα στοιχεία με ένα δεύτερο εσωτερικό βρόγχο αρχίζοντας από `j=i`
 - Αν το στοιχείο που βρίσκεται στη θέση `j` είναι μικρότερο από το στοιχείο που βρίσκεται στη θέση `min_index` τότε θέσε `min_index = j`
 - Με την έξοδο από το βρόγχο, αντάλλαξε τα στοιχεία που βρίσκονται στη θέση `i` με το στοιχείο που βρίσκεται στη θέση `min_index`

SelectionSort: Παράδειγμα Εκτέλεσης

Θέση

0

1

2

3

4

5

Αρχικός Πίνακας

34

8

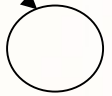
64

51

32

33

min_index



Στοιχείο που θα
εισαχθεί στην θέση i

i=1

8

34

64

51

32

33

i=2

8

32

64

51

34

33

i=3

8

32

33

51

34

64

i=4

8

32

33

34

51

64

i=5

8

32

33

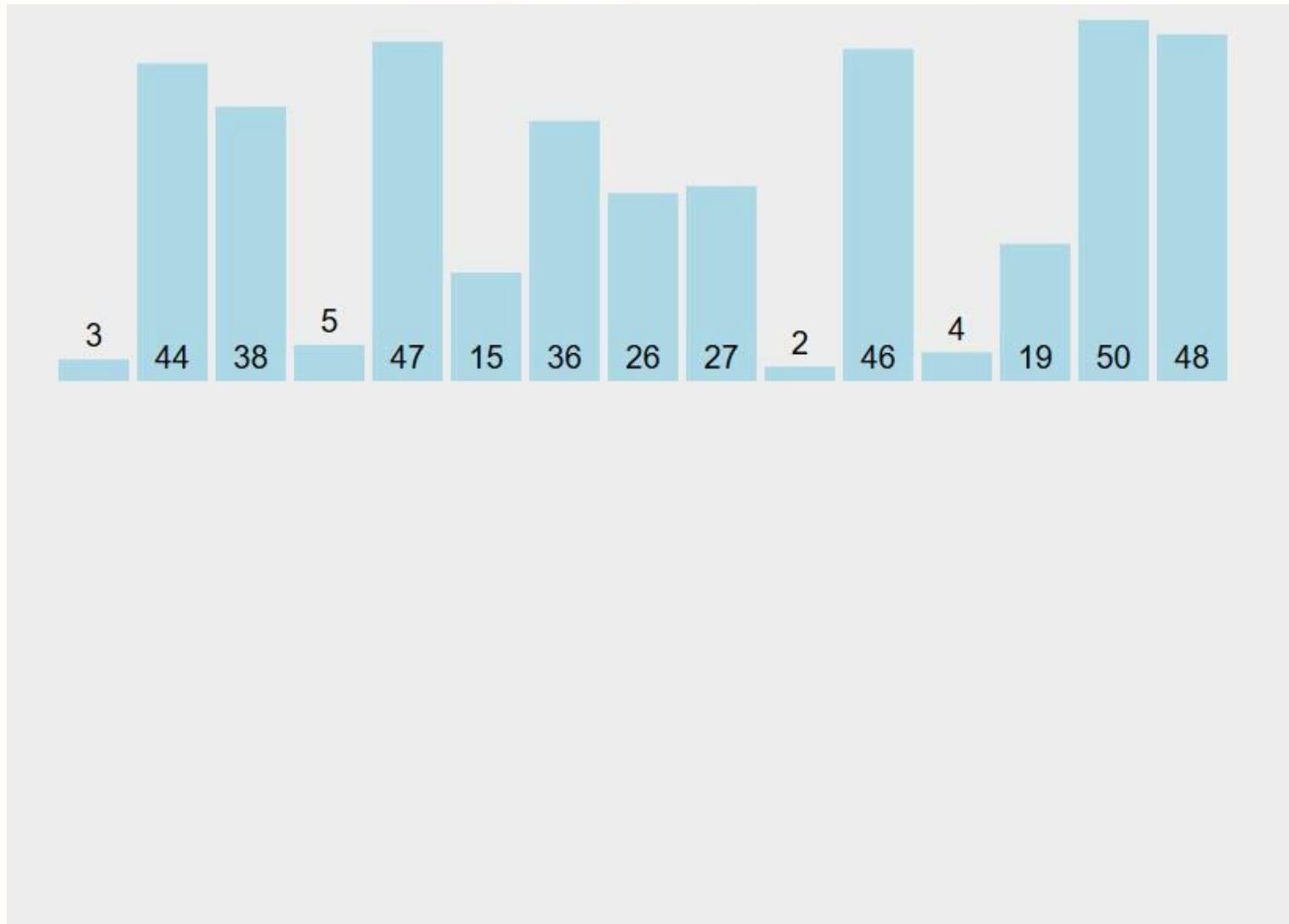
34

51

64

Δεν είναι
υποχρεωτικό να
γίνει αυτό το βήμα

Selection Sort Animation



<https://visualgo.net/bn/sorting>

SelectionSort: Υλοποίηση

```
void SelectionSort(int A[]) {  
    int n = A.length;  
    int min_index;  
    int tmp;  
    for(int i=0; i<n-1; i++){  
        min_index = i;  
  
        for(int j=i+1; j<n; j++){  
            if(A[j]<A[min_index])  
                min_index=j;  
        }  
  
        tmp = A[i];  
        A[i] = A[min_index];  
        A[min_index]=tmp;  
    }  
}
```

Αρχικοποίηση θέσης
μικρότερου στοιχείου

Βρες το μικρότερο
στοιχείο

Αντάλλαξε το
μικρότερο στοιχείο
με το τρέχον στοιχείο
στη θέση [i]

SelectionSort: Ανάλυση Χρόνου Εκτέλεσης

```
void SelectionSort(){  
    ...  
    for (i=0; i<n-1; i++){  
        for (j=i+1; j<n; j++){  
            sum++;  
        }  
    }  
}
```

Εσωτερικός Βρόγχος

Εξωτερικός Βρόγχος

• **Εσωτερικός Βρόγχος:** $IL = \sum_{j=i+1}^n 1 = n - (i + 1) + 1 = n - i$

• **Εξωτερικός Βρόγχος:** $OL = \sum_{i=0}^{n-1} IL = \sum_{i=0}^{n-1} (n - i)$

$$= n \sum_{i=0}^{n-1} 1 - \sum_{i=0}^{n-1} i = n^2 - \frac{n(n-1)}{2} \in \Theta(n^2)$$

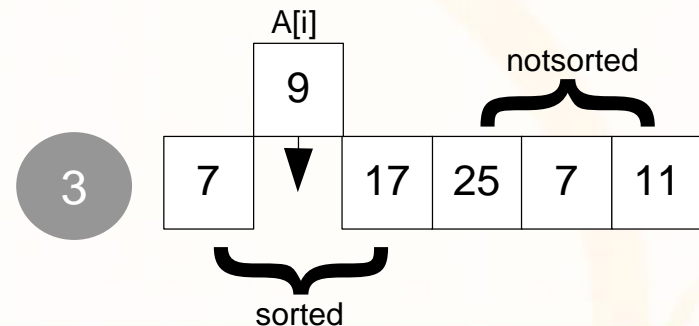
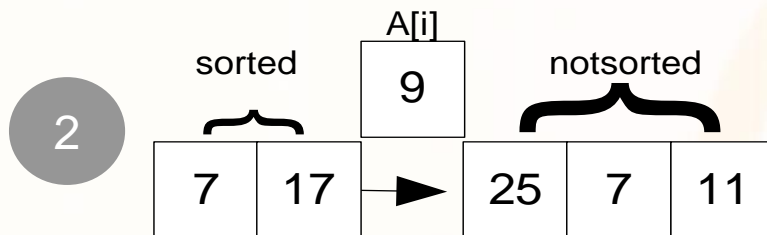
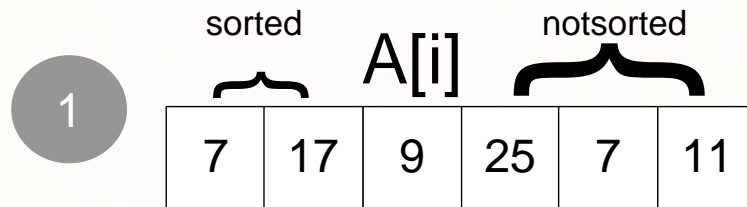
B. Ταξινόμηση με Εισαγωγή - Insertion Sort

- Η **Ταξινόμηση με Εισαγωγή (Insertion Sort)** εισάγει ένα-ένα τα στοιχεία του συνόλου που εξετάζεται, στη σωστή τους θέση.
- Στο βήμα i :
 1. υποθέτουμε πως ο πίνακας $A[0..(i-1)]$ είναι ταξινομημένος,
 2. εισάγουμε το στοιχείο $A[i]$ στην ακολουθία $A[0..(i-1)]$ στη σωστή θέση. Αυτό επιτυγχάνεται μετακινώντας όλα τα στοιχεία που είναι μεγαλύτερα του $A[i]$ μια θέση δεξιά.



B. Ταξινόμηση με Εισαγωγή - Insertion Sort (συν.)

- Στο βήμα i :
 1. υποθέτουμε πως ο πίνακας $A[0..(i-1)]$ είναι ταξινομημένος,
 2. εισάγουμε το στοιχείο $A[i]$ στην ακολουθία $A[0..(i-1)]$ στη σωστή θέση. Αυτό επιτυγχάνεται μετακινώντας όλα τα στοιχεία που είναι μεγαλύτερα του $A[i]$ μια θέση δεξιά.



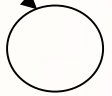
InsertionSort: Παράδειγμα Εκτέλεσης

Θέση 0 1 2 3 4 5

Αρχικός Πίνακας

34	8	64	51	32	21
----	---	----	----	----	----

min_index



Στοιχείο που θα
εισαχθεί στην θέση i

Με $i=1$ 8 34 64 51 32 21

Με $i=2$ 8 34 64 51 32 21

Με $i=3$ 8 34 51 64 32 21

Με $i=4$ 8 32 34 51 64 21

Με $i=5$ 8 21 32 34 51 64

Insertion Sort: Παράδειγμα Εκτέλεσης 2

BEFORE: [8,4,8,43,3,5,2,] (Το κόκκινο δείχνει τα ταξινομημένα στοιχεία)

i:1(index:4) >8 (“>” Δείχνει τις μετακινήσεις)

[4,8,8,43,3,5,2,] (Νέος Πίνακας)

i:2(index:8) (Τίποτα δεν μετακινείται)

[4,8,8,43,3,5,2,]

i:3(index:43) (Τίποτα δεν μετακινείται)

[4,8,8,43,3,5,2,]

i:4(index:3) >43>8>8>4

[3,4,8,8,43,5,2,]

i:5(index:5) >43>8>8

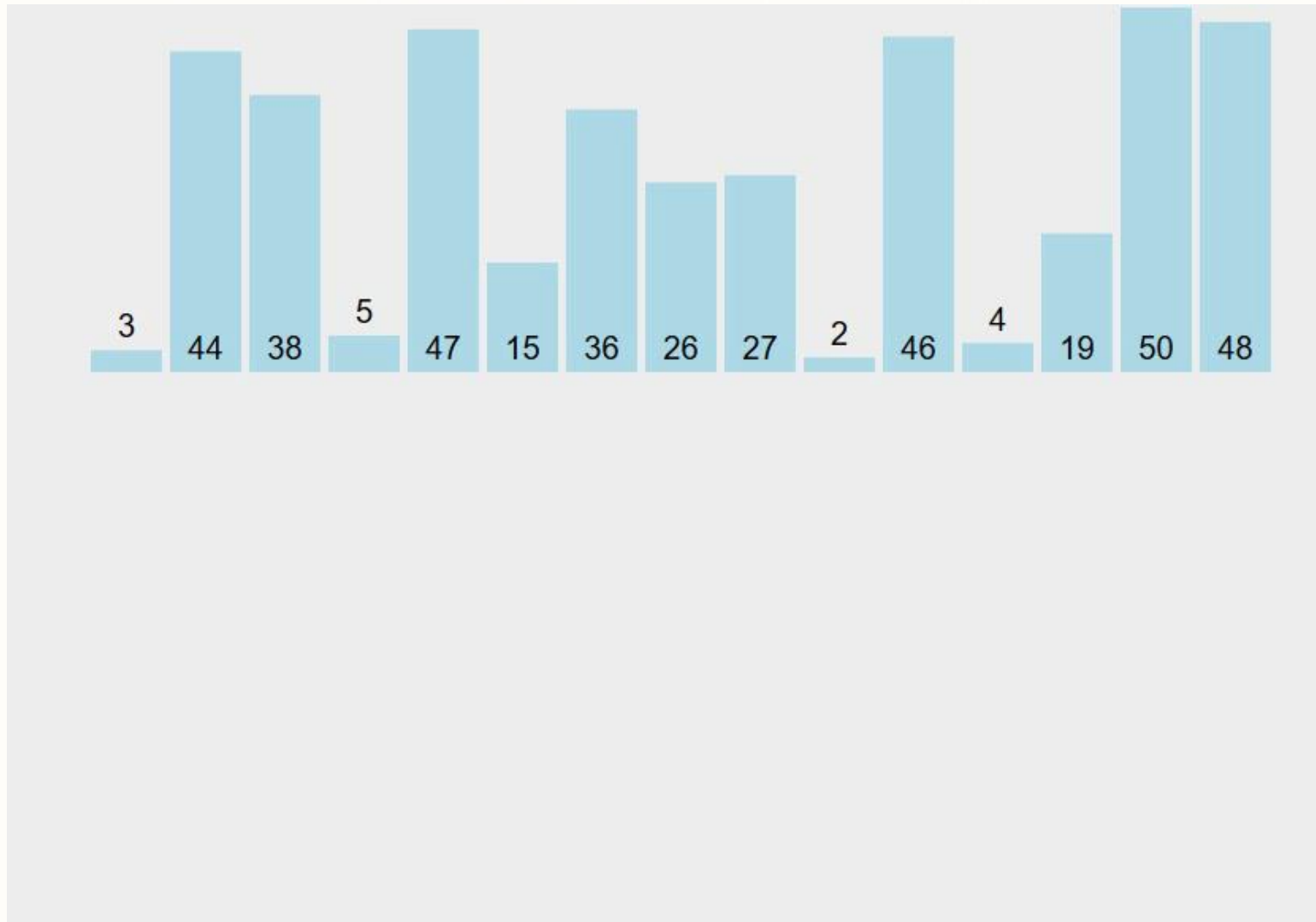
[3,4,5,8,8,43,2,]

i:6(index:2) >43>8>8>5>4>3

[2,3,4,5,8,8,43,]

AFTER: [2,3,4,5,8,8,43,]

Insertion Sort Animation



<https://visualgo.net/bn/sorting>

InsertionSort: Υλοποίηση

```
void InsertionSort(int A[]){  
    int n = x.length;  
    int index, i, j;  
    for (i=1; i < n; i++) {  
        index = A[i];  
  
        for (j=i; j>0; j--){  
  
            if (A[j-1] <= index) { break; }  
            A[j] = A[j-1];  
        }  
  
        A[j] = index;  
    }  
}
```

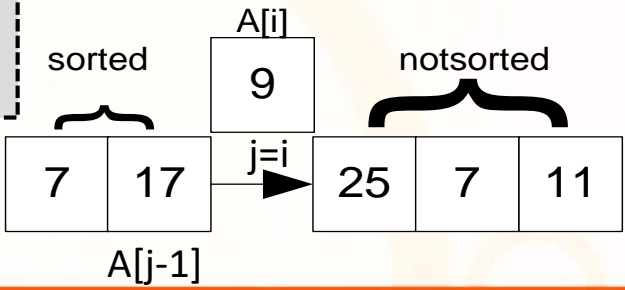
Τρέχον στοιχείο εισαγωγής

μετακίνηση στοιχείων προς δεξιά

θέλουμε να μετακινήσουμε μόνο τα $A[j-1] > index$

τοποθέτηση στοιχείου

index=9



InsertionSort: Ανάλυση Χρόνου Εκτέλεσης

```
void InsertionSort(){  
    ...  
    for (i=1; i<n; i++){  
        for (j=i; j>0; j--){  
            sum++;  
        }  
    }  
}
```

Εσωτερικός Βρόγχος

Εξωτερικός Βρόγχος

- **Εσωτερικός Βρόγχος:** $IL = \sum_{j=0}^i 1 = i$

- **Εξωτερικός Βρόγχος:** $OL = \sum_{i=1}^n IL = \sum_{i=1}^n i = \frac{n(n+1)}{2}$

$$\in \Theta(n^2)$$

Σύγκριση InsertionSort και SelectionSort

- Υπάρχουν δυο κριτήρια: **Βήματα** (μέχρι να βρω ένα στοιχείο) και **Μετακινήσεις** (όταν το βρω, πόσα swap κάνω) .
- Ο αλγόριθμος **Selection sort** απαιτεί πάντα **$O(n^2)$ βήματα** (δεν είναι δυνατή η γρήγορη έξοδος από τους βρόχους), έτσι η βέλτιστη περίπτωση είναι η ίδια με τη χειρίστη περίπτωση.
- Στον αλγόριθμο **Insertion Sort**, είναι δυνατό να βγούμε από το δεύτερο βρόχο γρήγορα. Στη βέλτιστη περίπτωση (ο πίνακας είναι ήδη ταξινομημένος), ο χρόνος εκτέλεσης είναι της τάξης **$\Omega(n)$ βήματα**.
- Παρά τούτου, το **Selection Sort** είναι πιο αποδοτικός αν κρίνουμε τους αλγόριθμους με βάση τον αριθμό των **μετακινήσεων (swaps)** που απαιτούν:
 - το **selection sort** απαιτεί **$O(n)$ μετακινήσεις**,
 - το **insertion sort**, απαιτεί **$O(n^2)$ μετακινήσεις** (στη χειρίστη περίπτωση όπου ο αρχικός πίνακας είναι ταξινομημένος σε φθίνουσα σειρά).

Γ. Ταξινόμηση με Συγχώνευση (Merge sort)

- Η **ταξινόμηση με συγχώνευση** είναι διαδικασία **διαίρει και βασίλευε** (**Divide and Conquer**: αναδρομική διαδικασία όπου το πρόβλημα μοιράζεται σε μέρη τα οποία λύνονται ξεχωριστά, και μετά οι λύσεις συνδυάζονται.)
- Περιγραφή του Mergesort
 1. **Διαίρεση**: Αναδρομικά μοιράζουμε τον πίνακα στα δύο μέχρι να φτάσουμε σε πίνακες μεγέθους ένα (**DIVIDE**)
 2. **Συγχώνευση**: Ταξινομούμε αναδρομικά τους πίνακες αυτούς με την συγχώνευση γειτονικών πινάκων (χρησιμοποιώντας βοηθητικό πίνακα). (**CONQUER**)

MergeSort: Βασική Ιδέα

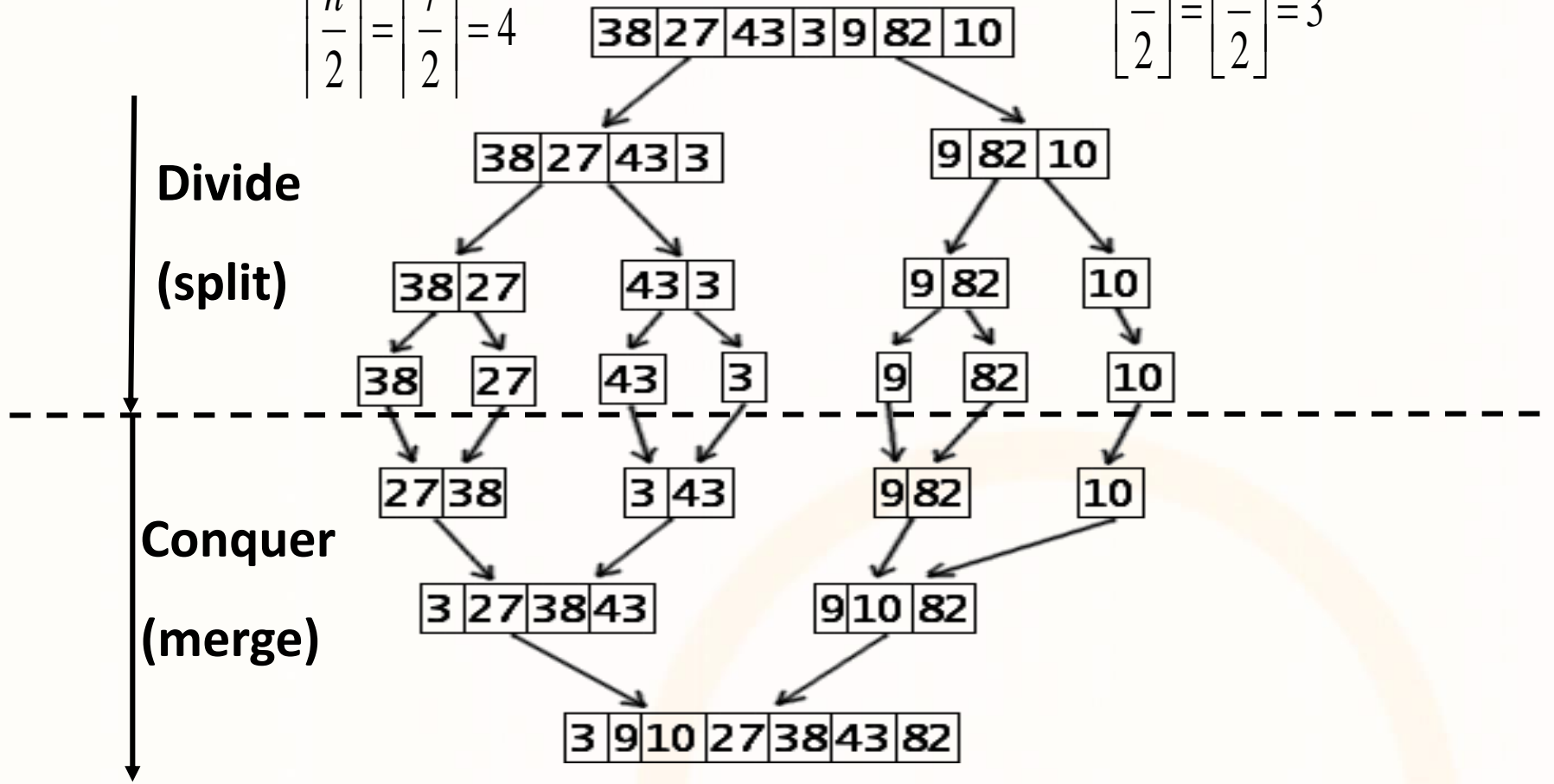
(#στοιχείων αριστερά)

$$\left\lfloor \frac{n}{2} \right\rfloor = \left\lfloor \frac{7}{2} \right\rfloor = 4$$

n=7

(#στοιχείων δεξιά)

$$\left\lceil \frac{n}{2} \right\rceil = \left\lceil \frac{7}{2} \right\rceil = 3$$



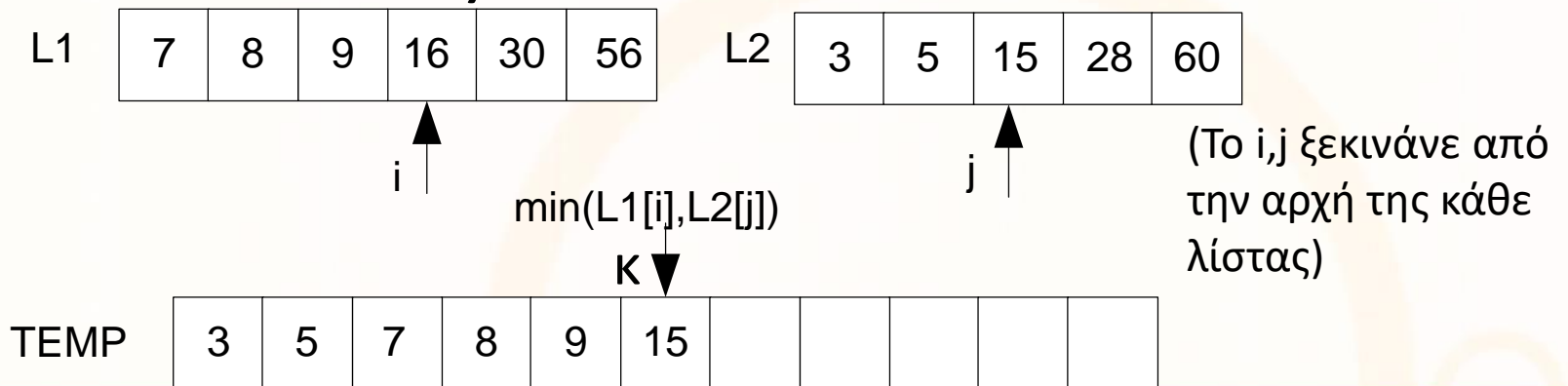
Συγχώνευση 2 Λιστών

Υποθέστε ότι θέλετε να συγχωνεύσετε 2 ταξινομημένες λίστες L1, L2 και να δημιουργήσετε μια νέα ταξινομημένη λίστα TEMP

Διαδικασία

1. Τοποθέτησε τους δείκτες i , j στην κεφαλή κάθε λίστας.
2. Διάλεξε το μικρότερο από την λίστα L1 και L2 και τοποθέτησε τον στον πίνακα TEMP στην θέση k
3. Προχώρησε τον δείκτη i (αν το μικρότερο στοιχείο ήταν από την λίστα L1) ή τον δείκτη j στην αντίθετη περίπτωση.
4. Επανάλαβε τα βήματα 2-4 μέχρι να εισαχθούν όλα τα στοιχεία στον TEMP

Μετά από 6 εκτελέσεις:



MergeSort: Υλοποίηση

```
void MergeSort(int A[], int temp[], int l, int r){
```

```
// η συνθήκη τερματισμού της ανάδρομης
```

```
if (l==r) return;
```

```
int mid = (l+r)/2;
```

```
// για ελαχιστοποίηση overflow(για μεγάλα l,r)
```

```
// int mid = l + ((r - l) / 2);
```

```
// μοιράζουμε αναδρομικά τον πίνακα
```

```
MergeSort(A, temp, l, mid);
```

```
MergeSort(A, temp, mid+1, r);
```

DIVIDE

```
// συνεχίζεται
```

MergeSort: Υλοποίηση (συν.)

```
// Τώρα οι πίνακες [1..mid] και [mid+1..r] είναι  
// ταξινομημένοι → Η διαδικασία συγχώνευσης
```

```
int k=1, i=1, j=mid+1;
```

```
// συγχώνευση στον temp μέχρι μια από  
// τις λίστες αδειάσει
```

```
while ((i<=mid) && (j<=r)) {  
    if (A[i]<A[j]){  
        temp[k] = A[i]; i++;  
    }  
    else {  
        temp[k] = A[j]; j++;  
    }  
    k++;  
}
```

MERGE

```
// συνεχίζεται
```

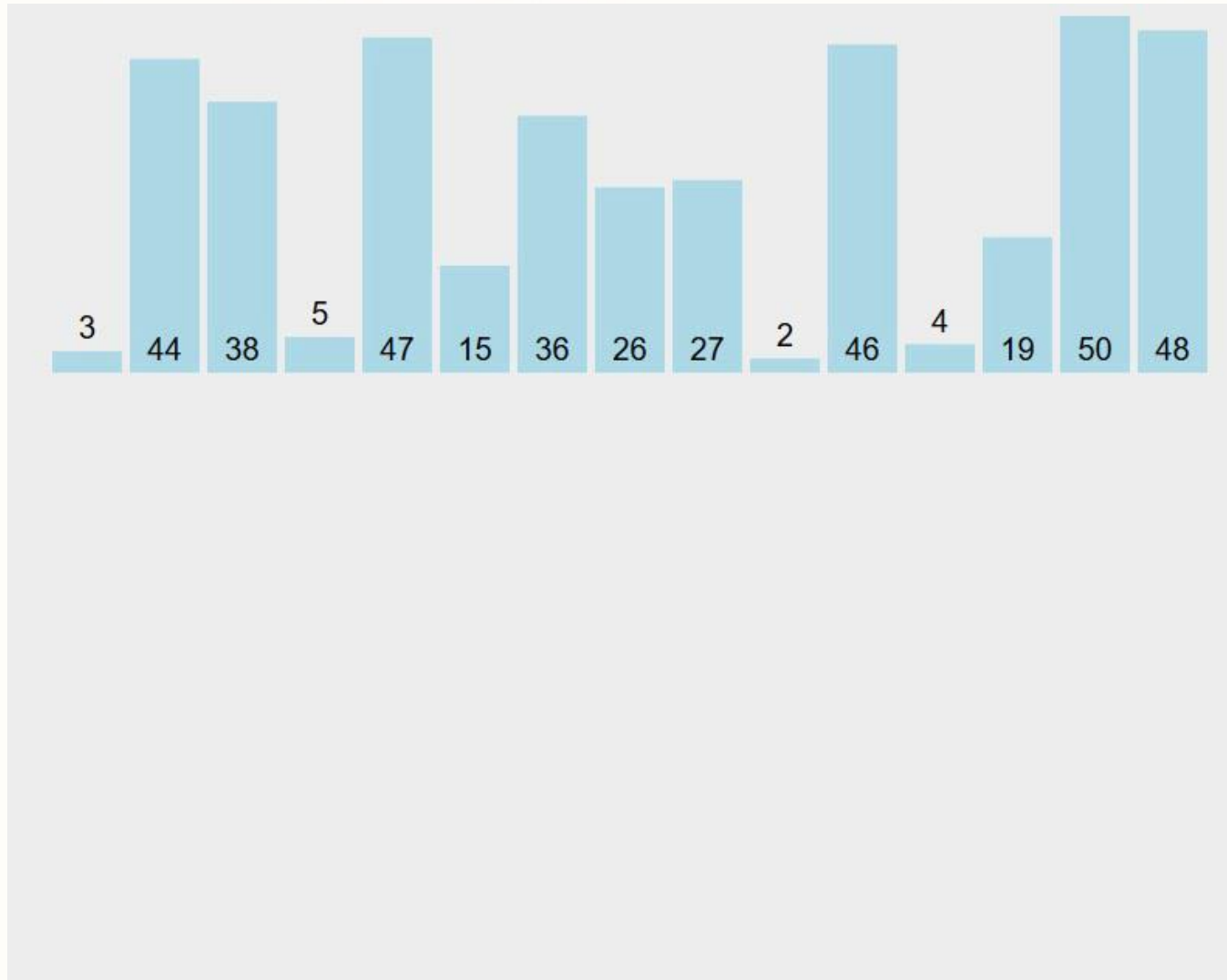
MergeSort: Υλοποίηση (συν.)

```
// copy όλων των υπόλοιπων στοιχείων λίστας L1
while (i<=mid) {
    temp[k] = A[i];
    k++;i++;
}

// copy όλων των υπόλοιπων στοιχείων λίστας L2
while (j<=r) {
    temp[k] = A[j];
    k++;j++;
}

// αντιγραφή όλων των στοιχείων από TEMP -> A
for (i=1; i<=r; i++) {
    A[i] = temp[i];
}
}
```


Merge Sort Animation



<https://visualgo.net/bn/sorting>

Παράδειγμα Εκτέλεσης Merge Sort

BEFORE: [8,4,8,43,3,5,2,1,10]
Index: 0 1 2 3 4 5 6 7 8

0,8: [8,4,8,43,3 | 5,2,1,10]

0,4: [8,4,8 | 43,3]

0,2: [8,4 | 8]

0,1: [8 | 4]

0,0: [8]

1,1: [4]

Merging: [A0,A0] [A1,A1] => [4,8,]

2,2: [8]

Merging: [A0,A1] [A2,A2] => [4,8,8,]

3,4: [43,3]

3,3: [43]

4,4: [3]

Merging: [A3,A3] [A4,A4] => [3,43]

Merging: [A0,A2] [A3,A4] => [3,4,8,8,43]

↓ divide

↓ divide

↓ divide

5,8: [5,2 | 1,10]

5,6: [5 | 2]

5,5: [5]

6,6: [2]

Merging: [A5,A5] [A6,A6] => [2,5]

7,8: [1 | 10]

7,7: [1]

8,8: [10]

Merging: [A7,A7] [A8,A8] => [1,10]

Merging: [A5,A6] [A7,A8] => [1,2,5,10]

Merging: [A0,A4] [A5,A8] => [1,2,3,4,5,8,8,10,43]

↓ divide

↓ divide

AFTER: [1,2,3,4,5,8,8,10,43]

MergeSort: Ανάλυση Χρ. Εκτέλ. - Αναδρομή

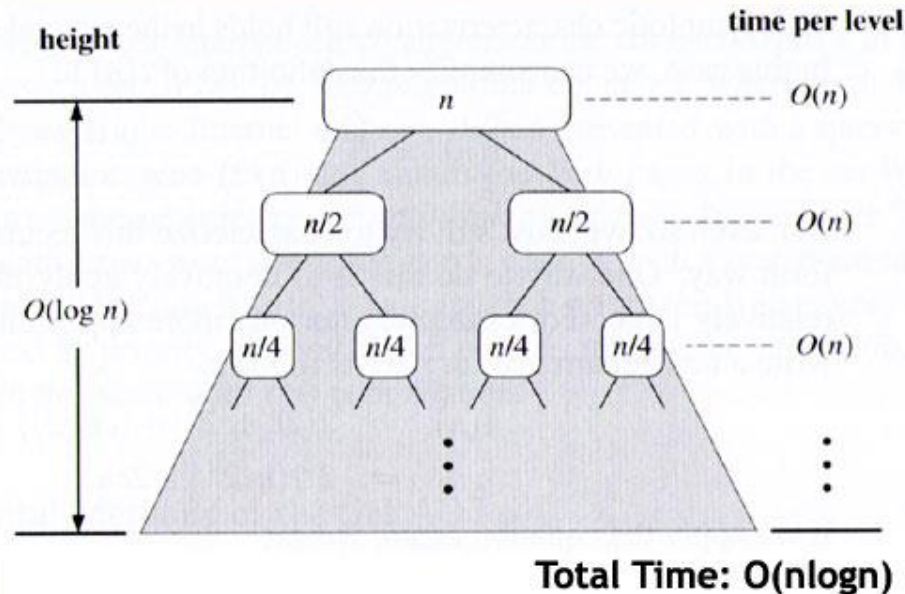
- Το πρόβλημα μοιράζει αναδρομικά σε δυο μέρη την λίστα που θέλουμε να ταξινομήσουμε.
- Όταν φτάσουμε στην λίστα που έχει μέγεθος 1 τότε σταματά η αναδρομή και το πρόγραμμα αρχίζει να συνδυάζει (merge) τις επιμέρους λίστες.
- Παρατηρούμε ότι πάνω σε μια λίστα μεγέθους N η αναδρομή εκτελείται $2\log_2 n$ φορές. Δηλαδή ο πίνακας μοιράζεται ως εξής: $n, n/2, n/4, \dots, 2, 1$

```
void MergeSort(int A[], int temp[], int l, int r){  
    if (l==r) return;  
    int mid = (l+r)/2;  
  
    MergeSort(A, temp, l, mid);  
    MergeSort(A, temp, mid+1, r);  
}
```

$\Theta(\log n)$

MergeSort: Ανάλυση Χρ. Εκτέλ. - Συγχώνευση

- Σε κάθε επίπεδο της ανάδρομης περνάμε μια φορά από το κάθε στοιχείο.
- Επομένως η συγχώνευση των στοιχείων σε κάθε επίπεδο της εκτέλεσης χρειάζεται **γραμμικό χρόνο $\Theta(n)$** .
- Σημειώστε ότι η διαδικασία απαιτεί τη χρήση βοηθητικού πίνακα.
- Μπορούμε να χρησιμοποιούμε τον ίδιο βοηθητικό πίνακα temp για όλες τις (αναδρομικές) κλήσεις του MergeSort.



MergeSort: Ανάλυση Χρ. Εκτέλ. - Συνολικός

- Η αντιγραφή και η συγχώνευση παίρνουν χρόνο $\Theta(n)$ και η αναδρομή παίρνει χρόνο $\Theta(\log n)$. Συνολικά $\Theta(n \log n)$.
- Ο χρόνος εκτέλεσης εκφράζεται και από την αναδρομική εξίσωση

$$T(0) = T(1) = 1$$

$$T(n) = 2 \cdot T(n/2) + n$$

- ... η οποία μπορεί να λυθεί με το Master Theorem ή με την μέθοδο της αντικατάστασης.
- **Πλεονέκτημα MergeSort:**
Ο συνολικός χρόνος εκτέλεσης είναι $\Theta(n \log n)$
(σε αντίθεση με το SelectionSort ($\Theta(n^2)$) και το InsertionSort ($O(n^2)$)
- **Μειονέκτημα:**
Απαιτεί τη χρήση βοηθητικού πίνακα (δηλαδή χρειάζεται διπλάσιο χώρο αποθήκευσης για την εκτέλεση του). Αυτό δεν καθιστά την μέθοδο πολύ εύχρηστη.