# Διάλεξη 13: Αλγόριθμοι Ταξινόμησης

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

*Οι αλγόριθμοι ταξινόμησης*

*SelectionSort, InsertionSort,*

*Στις ερχόμενες διαλέξεις θα δούμε τους αλγόριθμους*

*Mergesort, QuickSort,*

*BucketSort*

Διδάσκων: Δημήτρης Ζεϊναλιπούρ

# Αλγόριθμοι ταξινόμησης

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

$$x_1, x_2, \ldots, x_n$$

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

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

η οποία να ικανοποιεί

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

ή

$$f(x_{k_1}) \geq f(x_{k_2}) \geq \ldots \geq f(x_{k_n}).$$ **φθίνουσα σειρά**

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

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

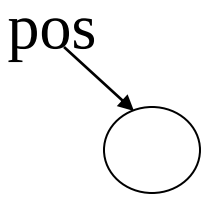
# 1/5) Ταξινόμηση με Επιλογή - **Selection Sort**

- 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.

# Παράδειγμα Selection Sort

| **Θέση** | **0** | **1** | **2** | **3** | **4** | **5** |
|---|---|---|---|---|---|---|
| *Αρχικός Πίνακας* | *34* | *8* | *64* | *51* | *32* | *33* |
| Με i=0 | 8 | 34 | 64 | 51 | 32 | 33 |
| Με i=1 | 8 | 32 | 64 | 51 | 34 | 33 |
| Με i=2 | 8 | 32 | 33 | 51 | 34 | 64 |
| Με i=3 | 8 | 32 | 33 | 34 | 51 | 64 |
| Με i=4 | 8 | 32 | 33 | 34 | 51 | 64 |




Στοιχείο που θα εισαχθεί στην θέση i

# Αλγόριθμος Selection Sort

```
void SelectionSort(int A[],int n){
   for (int i=0; i<n-1; i++){
        // θέση επόμενου μικρότερου στοιχείου - αρχικοποίηση
        pos=i;

        // βρες το μικρότερο στοιχείο
        for (j = i+1; j < n, j++){
               if A[j]<A[pos]
                   pos=j;    // θέση μικρότερου
        }
        if (pos != i) {
               // αντάλλαξε το A[i] με το  A[pos]
               temp = A[i];
               A[i] = A[pos];
               A[pos] = temp;
        }
   }
}
```

| 34 | 8 | 64 | 51 | 32 | 33 |
|---|---|---|---|---|---|

# Εκτέλεση Selection Sort

**BEFORE:**

**[8,4,8,43,3,5,2,1,10,]**

**Swapping 8 <-> 1**

**[1,4,8,43,3,5,2,8,10,]**

**Swapping 4 <-> 2**

**[1,2,8,43,3,5,4,8,10,]**

**Swapping 8 <-> 3**

**[1,2,3,43,8,5,4,8,10,]**

**Swapping 43 <-> 4**

**[1,2,3,4,8,5,43,8,10,]**

**Swapping 8 <-> 5**

**[1,2,3,4,5,8,43,8,10,]**

**[1,2,3,4,5,8,43,8,10,]**

**Swapping 43 <-> 8**

**[1,2,3,4,5,8,8,43,10,]**

**Swapping 43 <-> 10**

**[1,2,3,4,5,8,8,10,43,]**

**AFTER:**

**[1,2,3,4,5,8,8,10,43,]**

# Ανάλυση Χρόνου Εκτέλεσης **Selection Sort**

```
void SelectionSort(int A[],int n){
    for (int i=0; i<n-1; i++){
        // θέση επόμενης ανταλλαγής
        pos=i;                                          O(1)

        // βρες το μικρότερο στοιχείο
        for (j = i+1; j < n, j++){                      |
                if A[j]<A[pos]                          |
                        pos=j;                          |  O(i)
        }                                               |

        if (pos != i) {
                // αντάλλαξε το A[i] με το  A[pos]
                temp = A[i];                                    O(1)
                A[i] = A[pos];                                  O(1)
                A[pos] = temp;                                  O(1)
        }
    }
}
```

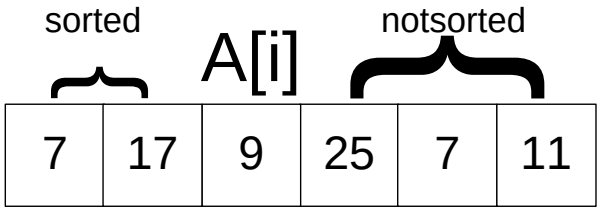
**Εσωτερικός Βρόχος (Ελάχιστο μεταξύ i):** Σε χρόνο O(i)
**Εξωτερικός Βρόχος (Επανάληψη εσωτερικού κόμβου N-1 φορές):**

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

# 2) Ταξινόμηση με Εισαγωγή - **Insertion Sort**

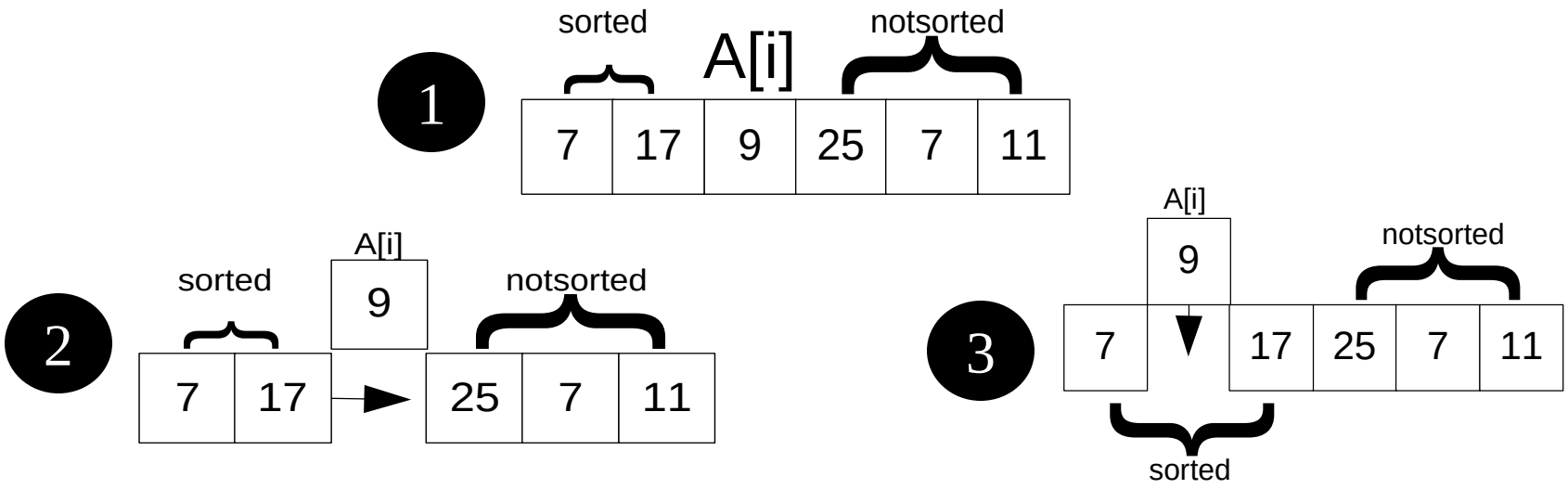
- Η ***Ταξινόμηση με Εισαγωγή*** **(Insert Sort)** εισάγει ένα-ένα τα στοιχεία του συνόλου που εξετάζεται, στη σωστή τους θέση.

- Στη φάση *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 επιτυγχάνετε μετακινώντας όλα τα στοιχεία που είναι μεγαλύτερα του *A*[*i*] μια θέση δεξιά.



# 2) Ταξινόμηση με Εισαγωγή - **Insertion Sort**

- Η ***ταξινόμηση με εισαγωγή*** **(Insertion Sort)** εισάγει ένα-ένα τα στοιχεία του συνόλου που εξετάζεται, στη σωστή τους θέση.
- Στη φάση *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 επιτυγχάνετε μετακινώντας όλα τα στοιχεία που είναι μεγαλύτερα του *A*[*i*] μια θέση δεξιά.



# Παράδειγμα Insertion Sort

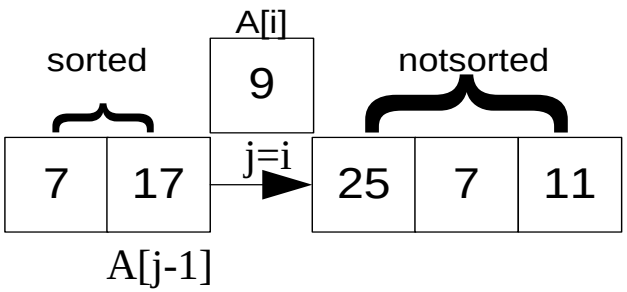
| **Θέση** | **0** | **1** | **2** | **3** | **4** | **5** |
|---|---|---|---|---|---|---|
| *Αρχικός Πίνακας* | *34* | *8* | *64* | *51* | *32* | *21* |
| Με i=1 | 8 | 34 | 64 | 51 | 32 | 21 |
| Με i=2 | 8 | 34 | 64 | 51 | 32 | 21 |
| Με i=3 | 8 | 34 | 51 | 64 | 32 | 21 |
| Με i=4 | 8 | 32 | 34 | 51 | 64 | 21 |
| Με i=5 | 8 | 21 | 32 | 34 | 51 | 64 |

Στοιχείο που θα εισαχθεί

Οι σκιασμένοι αριθμοί την στιγμή i, είναι ταξινομημένοι

# Αλγόριθμος Insertion Sort

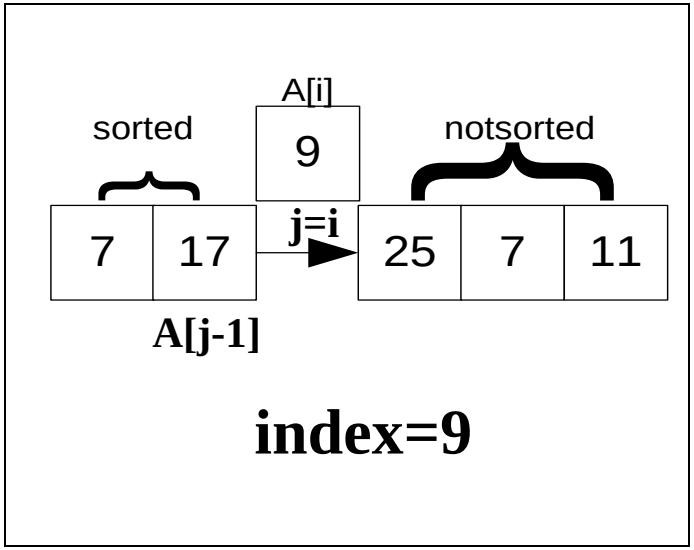
```
void InsertionSort(int A[], int n){
   for (i=1; i < n; i++) {
      // στοιχείο εισαγωγής
      index = A[i];

      // μετακίνηση στοιχείων δεξιά
      for (j=i; j>0; j--){
         // θέλουμε να μετακινήσουμε μόνο τα A[j-1] > index
         if (A[j-1] <= index) {
               break;
         }
         A[j] = A[j-1];
       }

      // τοποθέτηση στοιχείου
      A[j] = index;
   }
}
```




index=9

# Εκτέλεση Insertion Sort

```
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,]
```



# Ανάλυση Χρόνου Εκτέλεσης **Insertion Sort**

```
void InsertionSort(int A[], int n){
   for (i=1; i < n; i++) {
     // στοιχείο εισαγωγής
     index = A[i];

     // μετακίνηση στοιχείων δεξιά
     for (j=i; j>0; j--){
         if (A[j-1] <= index) {
               break;
         }
         A[j] = A[j-1];
      }

     // τοποθέτηση στοιχείου
     A[j] = index;
  }
}
```

O(i)

**Εσωτερικός Βρόχος (Ελάχιστο μεταξύ i):** Σε χρόνο O(i)
**Εξωτερικός Βρόχος (Επανάληψη εσωτερικού κόμβου N-1 φορές):**

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

# Σύγκριση Insertion Sort και Selection Sort

- 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 χείριστη 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 χείριστη περίπτωση όπου ο αρχικός πίνακας είναι ταξινομημένος σε φθίνουσα σειρά).