343 – Εισαγωγή στον Προγραμματισμό

Τμήμα Μαθηματικών
Πανεπιστήμιο Ιωαννίνων

Ακαδημαϊκό Έτος 2015-2016

Χάρης Παπαδόπουλος
207δ, Β' όροφος
e-mail: charis@cs.uoi.gr

Ώρες Γραφείου:
Πέμπτη 11-13

Θ: διάλεξη (θεωρία)

Ε: Εργαστήριο

Q: Τεστ quiz

Ημερολόγιο Μαθήματος

Οκτώβριος 2015

Δ	Τ	Τ	Π	Π
			1	2
5	6	7	8	**9 Θ**
12	13	14	15	**16 Θ**
19 Ε	20	21	22	**23 Θ**
26	27	28	29	**30 Θ**

Νοέμβριος 2015

Δ	Τ	Τ	Π	Π
2 Ε	3	4	5	**6 Θ**
9 Ε	10	11	12	**13 Θ**
16 Q	17	18	19	**20 Θ**
23 Ε	24	25	26	**27 Θ**
30 Ε				

Δεκέμβριος 2015

Δ	Τ	Τ	Π	Π
	1	2	3	**4 Θ**
7 Ε	8	9	10	**11 Θ**
14 Q	15	16	17	**18 Θ**

Ιανουάριος 2016

Δ	Τ	Τ	Π	Π
4	5	6	7	8
11	12	13	14	**15 Θ**

Εβδομάδα	Θέματα	Ύλη βιβλιογραφίας
Πα, 9 Οκτωβρίου	**Εισαγωγικά μαθήματος & Δυαδική αναπαράσταση**	[1]: 1.1, Παράρτημα 3 [2]: Κεφ. 1, Β, Δ
Πα, 16 Οκτωβρίου	**Είσοδος/Έξοδος δεδομένων, τύποι δεδομένων & μεταβλητών**	[1]: 1.2, 1.3, 1.4, 1.5, Παράρτημα 1 [2]: Κεφ. 2, Γ
Δε, 19 Οκτ	1ο Εργαστήριο	
Πα, 23 Οκτωβρίου	**Προεπεξεργαστής, αριθμητικοί και λογικοί τελεστές**	[1]: 2.1, Παράρτημα 2 [2]: 4.11, 4.12, Α, ΣΤ
Πα, 30 Οκτωβρίου	**Ροή ελέγχου: if/else, switch, for, while, do-while και ροή ελέγχου if/else**	[1]: 2.2, 2.3 [2]: Κεφ. 4, Κεφ. 5
Δε, 2 Νοε	2ο Εργαστήριο	
Πα, 6 Νοεμβρίου	**Συναρτήσεις, εμβέλεια μεταβλητών και αναδρομή**	[1]: 3.1, 3.2, 3.3, 4.1, 4.2, 13.1, 13.2 [2]: Κεφ. 6
Δε, 9 Νοε	3ο Εργαστήριο	
Πα, 13 Νοεμβρίου	**Επανάληψη Εργαστηρίων**	
Δε, 16 Νοε	**1ο Quiz**	
Πα, 20 Νοεμβρίου	**Επανάληψη με Παραδείγματα**	
Δε, 23 Νοε	4ο Εργαστήριο	
Πα, 27 Νοεμβρίου	**Πίνακες (μονοδιάστατοι και πολυδιάστατοι)**	[1]: 5.1, 5.2, 5.4 [2]: Κεφ. 7
Δε, 30 Νοε	5ο Εργαστήριο	
Πα, 4 Δεκεμβρίου	**Εφαρμογές σε ταξινομήσεις και αναζήτηση στοιχείων**	[1]: Παράρτημα 4, 9.1, 9.2, 9.3 [2]: 6.7, 6.8, Κεφ. 18
Δε, 7 Δεκ	6ο Εργαστήριο	
Πα, 11 Δεκεμβρίου	**Αλφαριθμητικά και Συμβολοσειρές**	[1]: 6.1, 12.1, 12.2, 12.4 [2]: Κεφ. 21, 17.1-17.10
Δε, 14 Δεκ	**2ο Quiz**	
Πα, 18 Δεκεμβρίου	**Εγγραφές, δομές και χρήση αρχείων**	[1]: 5.3, 13.3 [2]: 7.7, 7.8, 8.6, Κεφ. 19
Πα, 15 Ιανουαρίου	**Επανάληψη**	

Θ: διάλεξη (θεωρία)

Ε: Εργαστήριο

Q: Τεστ quiz

Ημερολόγιο Μαθήματος

Οκτώβριος 2015

Δ	Τ	Τ	Π	Π
			1	2
5	6	7	8	**9 Θ**
12	13	14	15	**16 Θ**
19 Ε	20	21	22	**23 Θ**
26	27	28	29	**30 Θ**

Νοέμβριος 2015

Δ	Τ	Τ	Π	Π
2 Ε	3	4	5	**6 Θ**
9 Ε	10	11	12	**13 Θ**
16 Q	17	18	19	**20 Θ**
23 Ε	24	25	26	**27 Θ**
30 Ε				

Δεκέμβριος 2015

Δ	Τ	Τ	Π	Π
	1	2	3	**4 Θ**
7 Ε	8	9	10	**11 Θ**
14 Q	15	16	17	**18 Θ**

Ιανουάριος 2016

Δ	Τ	Τ	Π	Π
4	5	6	7	8
11	12	13	14	**15 Θ**

Εβδομάδα	Θέματα	Ύλη βιβλιογραφίας
Πα, 9 Οκτωβρίου	**Εισαγωγικά μαθήματος & Δυαδική αναπαράσταση**	[1]: 1.1, Παράρτημα 3 [2]: Κεφ. 1, Β, Δ
Πα, 16 Οκτωβρίου	**Είσοδος/Έξοδος δεδομένων, τύποι δεδομένων & μεταβλητών**	[1]: 1.2, 1.3, 1.4, 1.5, Παράρτημα 1 [2]: Κεφ. 2, Γ
Δε, 19 Οκτ	1ο Εργαστήριο	
Πα, 23 Οκτωβρίου	**Προεπεξεργαστής, αριθμητικοί και λογικοί τελεστές**	[1]: 2.1, Παράρτημα 2 [2]: 4.11, 4.12, Α, ΣΤ
Πα, 30 Οκτωβρίου	**Ροή ελέγχου: if/else, switch, for, while, do-while και ροή ελέγχου if/else**	[1]: 2.2, 2.3 [2]: Κεφ. 4, Κεφ. 5
Δε, 2 Νοε	2ο Εργαστήριο	
Πα, 6 Νοεμβρίου	**Συναρτήσεις, εμβέλεια μεταβλητών και αναδρομή**	[1]: 3.1, 3.2, 3.3, 4.1, 4.2, 13.1, 13.2 [2]: Κεφ. 6
Δε, 9 Νοε	3ο Εργαστήριο	
Πα, 13 Νοεμβρίου	**Επανάληψη Εργαστηρίων**	
Δε, 16 Νοε	**1ο Quiz**	
Πα, 20 Νοεμβρίου	**Επανάληψη με Παραδείγματα**	
Δε, 23 Νοε	4ο Εργαστήριο	
Πα, 27 Νοεμβρίου	**Πίνακες (μονοδιάστατοι και πολυδιάστατοι)**	[1]: 5.1, 5.2, 5.4 [2]: Κεφ. 7
Δε, 30 Νοε	5ο Εργαστήριο	
Πα, 4 Δεκεμβρίου	**Εφαρμογές σε ταξινομήσεις και αναζήτηση στοιχείων**	[1]: Παράρτημα 4, 9.1, 9.2, 9.3 [2]: 6.7, 6.8, Κεφ. 18
Δε, 7 Δεκ	6ο Εργαστήριο	
Πα, 11 Δεκεμβρίου	**Αλφαριθμητικά και Συμβολοσειρές**	[1]: 6.1, 12.1, 12.2, 12.4 [2]: Κεφ. 21, 17.1-17.10
Δε, 14 Δεκ	**2ο Quiz**	
Πα, 18 Δεκεμβρίου	**Εγγραφές, δομές και χρήση αρχείων**	[1]: 5.3, 13.3 [2]: 7.7, 7.8, 8.6, Κεφ. 19
Πα, 15 Ιανουαρίου	**Επανάληψη**	

Ενότητα 17

ΑΛΓΟΡΙΘΜΟΙ ΤΑΞΙΝΟΜΗΣΗΣ

Αποτελεσματικότητα Αλγορίθμου

- Θα εξετάσουμε διάφορους αλγορίθμους που επιλύουν ίδια προβλήματα.
- Όταν σχεδιάζουμε έναν αλγόριθμο μας ενδιαφέρει η *αποτελεσματικότητά* του (ταχύτητα)
- Ένας γνωστός τρόπος για να ποσοτικοποιούμε την αποτελεσματικότητα ενός αλγορίθμου (πρόγραμμα) γίνεται με την εύρεση του

  πλήθος των συγκρίσεων που χρησιμοποιεί.
- Το πλήθος των συγκρίσεων μπορεί να εξαρτάται από την είσοδο. Περιπτώσεις:

  χειρότερη,

  μέση,

  καλύτερη

Παραδείγματα

- ```
  if ( i < 5)
     i++;
  ```
- ```
  if (( i < 5 ) && ( i >0 ))
     i++;
  ```
- Γίνονται 1 και 2 συγκρίσεις, αντίστοιχα.
- Στο ακόλουθο κομμάτι πόσες συγκρίσεις γίνονται;

  ```
  for(i=0; i<n; i++)
     cout << " i =  " <<  i << endl;
  ```

- Στο ακόλουθο (δοκιμάστε με αρχικές τιμές `i=0` και `i=4`):

  ```
  while(i<5){
     cout << " i =  " <<  i << endl;
     i+=2;
   }
  ```

Ταξινόμηση Πινάκων

- Ταξινόμηση σε αύξουσα τάξη:

  $a[0] \leq a[1] \leq \ldots \leq a[99]$

- Ταξινόμηση σε φθίνουσα τάξη:

  $a[0] \geq a[1] \geq \ldots \geq a[99]$

- Πολύ σπουδαία εφαρμογή
  - Σχεδόν κάθε οργανισμός πρέπει να ταξινομεί κάποια δεδομένα
  - Συνήθως οι επιχειρήσεις πρέπει να ταξινομούν ή να κρατάνε ταξινομημένα μεγάλο όγκο δεδομένων

- Ταξινομούμε ένα πίνακα με *n* (< 100) στοιχεία
  - Θεωρούμε τον πίνακα ως μερικώς συμπληρωμένο πίνακα

Χρήση συναρτήσεων

- Σε όλους τους αλγορίθμους θα χρησιμοποιήσουμε τις ακόλουθες συναρτήσεις:

Εναλλαγή στοιχείων

```
void swapValues(int& v1, int& v2)
{
    int temp;
    temp = v1;
    v1 = v2;
    v2 = temp;
}
```

Θα μπορούσε και με αυτή την δήλωση

```
void swap(int a[], int i, int j)
{
    int temp;
    temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}
```

Χρήση συναρτήσεων

- Σε όλους τους αλγορίθμους θα χρησιμοποιήσουμε τις ακόλουθες συναρτήσεις:

Διάβασμα

```
void fillArray(int a[], int size, int& numberUsed)
{
    cout << "Δώσε θετικούς αριθμούς (αρνητικό για τέλος).\n";

    int next, index = 0;
    cin >> next;
    while ((next >= 0) && (index < size))
    {
        a[index] = next;
        index++;
        cin >> next;
    }

    numberUsed = index;
}
```

Χρήση συναρτήσεων

- Σε όλους τους αλγορίθμους θα χρησιμοποιήσουμε τις ακόλουθες συναρτήσεις:

Εκτύπωση

```
void printArray(int a[], int size)
{
    for(int i = 0; i < size; i++)
       cout << a[i] << " ";
    cout << endl;
}
```

```
...
printArray(a, numberUsed)
...
```

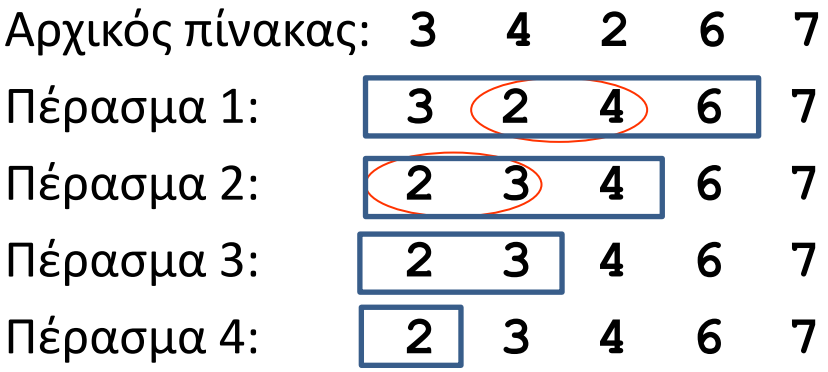
```
#include <iostream>
using namespace std;

const int MAX_SIZE = 100;

void fillArray(int a[], int size, int& numberUsed);

void printArray(int a[], int size);

void swapValues(int& v1, int& v2);

void sort(int a[], int size);

int main( )
{
    int a[MAX_SIZE], numberUsed;

    fillArray(a, MAX_SIZE, numberUsed);

    sort(a, numberUsed);

    printArray(a, numberUsed);

    return 0;
}
```

Ταξινόμηση Πινάκων - Bubble Sort

- Μέθοδος της πέτρας (παραλλαγή: μέθοδος φυσαλίδας)
  - Κάνει n-1 περάσματα πάνω από τον πίνακα: p=1,2,…,n-1
  - Κατά το p πέρασμα, διαδοχικά ζεύγη στοιχείων στις θέσεις i=0,1,…,n-p συγκρίνονται
    - Εάν είναι σε αύξουσα τάξη (ή ίσα), τα αφήνουμε έτσι
    - Εάν δεν είναι σε αύξουσα τάξη, τα ανταλλάσσουμε
  - Σε κάθε πέρασμα το επόμενο μεγαλύτερο στοιχείο βυθίζεται στην σωστή θέση
- Επαναλαμβάνουμε μέχρι πλήρους ταξινόμησης
- Εύκολο πρόγραμμα αλλά αργό
- Αρχίζοντας τις συγκρίσεις από το τέλος του πίνακα προς την αρχή έχουμε τον αλγόριθμο της φυσαλίδας, όπου σε κάθε πέρασμα το επόμενο μικρότερο στοιχείο ανεβαίνει στην σωστή θέση

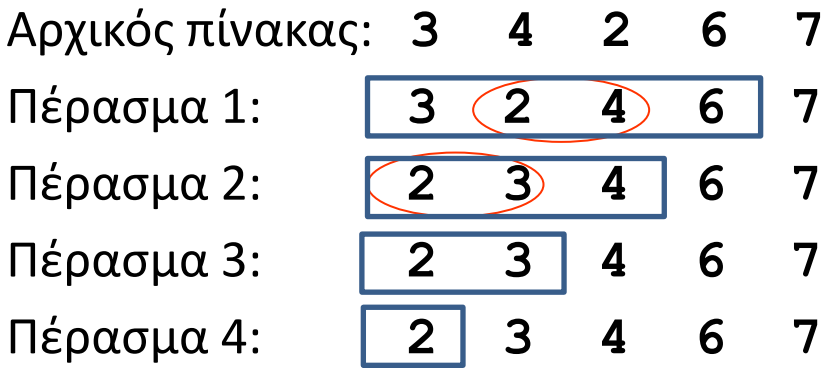
Ταξινόμηση Πινάκων - Bubble Sort

- Παράδειγμα



Ταξινόμηση Πινάκων - Bubble Sort

- Παράδειγμα




```
void sort(int a[], int size)
{
    for(int pass = 1; pass < size; pass++)
      for(int i = 0; i < size – pass; i++)
        if( a[i] > a[i+1])
           swapValues(a[i], a[i+1]);
}
```

Ταξινόμηση Πινάκων – Selection Sort

- Μέθοδος της επιλογής του επόμενου μικρότερου στοιχείου
  - Κάνει n-1 περάσματα πάνω από τον πίνακα: p=0,1,2,3,…,n-2
  - Στο p πέρασμα επιλέγουμε το μικρότερο στοιχείο μεταξύ των στοιχείων στις θέσεις i=p,…,n-1 και το ανταλλάσσουμε με αυτό στην p θέση του πίνακα
- Στο πρώτο πέρασμα επιλέγουμε το μικρότερο στοιχείο και το βάζουμε στην πρώτη θέση του πίνακα,
  - μετά επιλέγουμε το επόμενο μικρότερο στοιχείο και το ανταλλάσσουμε με αυτό στην δεύτερη θέση, κ.ο.κ
- Επαναλαμβάνουμε μέχρι πλήρους ταξινόμησης
- Εύκολο πρόγραμμα αλλά αργό

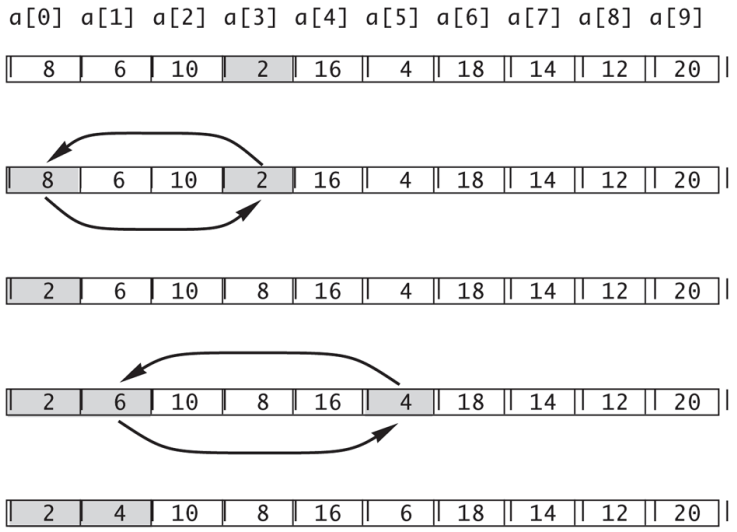
Ταξινόμηση Πινάκων – Selection Sort

- Παράδειγμα

Αρχικός πίνακας:	6	5	2	4	3
Πέρασμα 1:	2	5	6	4	3
Πέρασμα 2:	2	3	6	4	5
Πέρασμα 3:	2	3	4	6	5
Πέρασμα 4:	2	3	4	5	6

- Δυο εκδοχές
  - Μπορούμε να δουλέψουμε και με maximum

Display 5.7 **Selection Sort**



Ταξινόμηση Πινάκων – Selection Sort

```
void sort(int a[], int size)
{
    int min;

    for(int pass = 0; pass < size-1; pass++)
    {

      min = pass;
      for(int i = pass+1; i < size; i++)
        if( a[i] < a[min])
          min = i;

      swapValues(a[pass], a[min]);
    }

}
```

Ταξινόμηση Πινάκων – Insertion Sort

- Μέθοδος της εισαγωγής του επόμενου στοιχείου στην κατάλληλη θέση
  - Κάνει n-1 περάσματα πάνω από τον πίνακα p=1,2,3,…,n-1
  - Στο p πέρασμα το p στοιχείο εισάγεται στην σωστή θέση συγκρινόμενο με τα στοιχεία στις θέσεις i=0,1,…,p-1.
  - Θεωρούμε ότι τα στοιχεία στις θέσεις i=0,1,…,p-1 είναι ήδη ταξινομημένα
- Σε κάθε πέρασμα ένα στοιχείο συγκρίνεται με τα στοιχεία πριν από αυτό και εισάγεται στην σωστή θέση
  - γίνονται ανταλλαγές με τα μεγαλύτερα από αυτό στοιχεία
- Επαναλαμβάνουμε μέχρι πλήρους ταξινόμησης
- Εύκολο πρόγραμμα αλλά αργό

Ταξινόμηση Πινάκων – Selection Sort

- Παράδειγμα

Αρχικός πίνακας: 6 5 2 4 3

Πέρασμα 1: 6 5 2 4 3

Πέρασμα 1: 5 6 2 4 3

Πέρασμα 2: 5 6 2 4 3

Πέρασμα 2: 2 5 6 4 3

Πέρασμα 3: 2 5 6 4 3

Πέρασμα 3: 2 4 5 6 3

Πέρασμα 4: 2 5 6 4 3

Πέρασμα 4: 2 4 5 6 3

Ταξινόμηση Πινάκων – Selection Sort

- Παράδειγμα


```
...
temp = a[i];
j = i;
while( temp < a[ j - 1 ] )
{
    a[ j ] = a[ j - 1 ];
    j--;
}
...
```

Ταξινόμηση Πινάκων – Insertion Sort

```
void sort(int a[], int size)
{
    int i, j, temp;

    for( int i = 1; i < size; i++ )
    {
        temp = a[ i ];

        for(j = i; j > 0 && temp < a[ j - 1 ]; j-- )
            a[ j ] = a[ j - 1 ];

        a[ j ] = temp;
    }

}
```

Δεν έχει βρεθεί ακόμα η σωστή θέση

Μετακίνηση όλων των στοιχείων μια θέση δεξιά

Ενότητα 18

ΑΝΑΖΗΤΗΣΗ ΣΤΟΙΧΕΙΟΥ

Αναζήτηση στοιχείου σε πίνακα

- Αναζήτηση κάποιας τιμής κλειδί σε ένα μονοδιάστατο πίνακα. Υπάρχουν δύο βασικές μέθοδοι.

- Γραμμική αναζήτηση
  - Απλή
  - Συγκρίνουμε κάθε στοιχείο με την τιμή κλειδί που αναζητάμε
  - Χρήσιμη για μικρούς μη ταξινομημένους πίνακες
  - Πολύ αργή, είναι δυνατόν να κάνει *n* συγκρίσεις σε ένα πίνακα με *n* στοιχεία

Παράδειγμα γραμμικής αναζήτησης

- Κλειδί αναζήτησης:

  `key = 7`

- Ελέγχουμε `A[i] == key`

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
• A =	[1,	3,	3,	6,	8,	9,	4]

`i=0`

Παράδειγμα γραμμικής αναζήτησης

- Κλειδί αναζήτησης:

  `key = 7`

- Ελέγχουμε `A[i] == key`

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
• A =	[1,	3,	3,	6,	8,	9,	4]

`i=1`

Παράδειγμα γραμμικής αναζήτησης

- Κλειδί αναζήτησης:

  `key = 7`

- Ελέγχουμε `A[i] == key`

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
A =	[1,	3,	3,	6,	8,	9,	4]

`i=2`

Παράδειγμα γραμμικής αναζήτησης

- Κλειδί αναζήτησης:

  `key = 7`

- Ελέγχουμε `A[i] == key`

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
• A =	[1,	3,	3,	6,	8,	9,	4]

`i=3`

Παράδειγμα γραμμικής αναζήτησης

- Κλειδί αναζήτησης:

  `key = 7`

- Ελέγχουμε `A[i] == key`

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
• A =	[1,	3,	3,	6,	8,	9,	4]

`i=4`

Παράδειγμα γραμμικής αναζήτησης

- Κλειδί αναζήτησης:

  `key = 7`

- Ελέγχουμε `A[i] == key`

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
• A =	[1,	3,	3,	6,	8,	9,	4]

`i=5`

Παράδειγμα γραμμικής αναζήτησης

- Κλειδί αναζήτησης:

  `key = 7`

- Ελέγχουμε `A[i] == key`

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
• A =	[1,	3,	3,	6,	8,	9,	4]

`i=6`

Παράδειγμα γραμμικής αναζήτησης

- Κλειδί αναζήτησης:

  `key = 7`

- Ελέγχουμε `A[i] == key`

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
A =	[1,	3,	3,	6,	8,	9,	4]

`i=6`

- Αν δεν βρέθηκε (δεν επιστρέψαμε την θέση `i` του πίνακα) τότε επιστρέφουμε `-1` (η θέση `-1` σε πίνακα δεν υπάρχει)

```
int search(int a[ ], int numberUsed, int target)
{
    int index = 0;

    bool found = false;

    while ((!found) && (index < numberUsed))
    {

        if (target == a[index])

                found = true;

        else

                index++;

    }


    if (found)

        return index;

    else

        return -1;

   return -1;
}
```

Δυαδική Αναζήτηση

- Δυαδική Αναζήτηση
  - Για ταξινομημένους πίνακες
- Συγκρίνουμε το μεσαίο στοιχείο (middle) του πίνακα με το κλειδί (key) που αναζητάμε
  - If key = middle, τότε βρέθηκε
  - If key < middle, τότε η αναζήτηση γίνεται στο αριστερό ήμισυ του πίνακα
  - If key > middle, τότε η αναζήτηση γίνεται στο δεξιό ήμισυ του πίνακα
  - Επαναλαμβάνουμε την δυαδική αναζήτηση στο ήμισυ που επιλέξαμε
- Πολύ γρήγορη, κάνει το πολύ *log(n)* συγκρίσεις σε ένα πίνακα με *n* στοιχεία
  - Για *n* = 1024 κάνει το πολύ 10 συγκρίσεις

Παράδειγμα

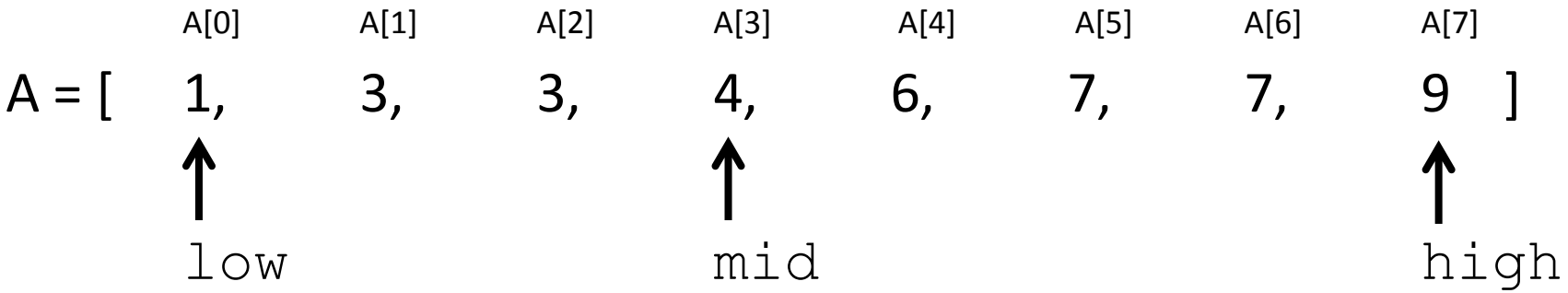
- Κλειδί αναζήτησης:

  `key = 8` `mid = (low+high)/2`

- Ελέγχουμε `A[mid] == key` ⇒ `return mid`

  `A[mid] < key` ⇒ `low=mid+1`

  `A[mid] > key` ⇒ `high=mid-1`



Παράδειγμα

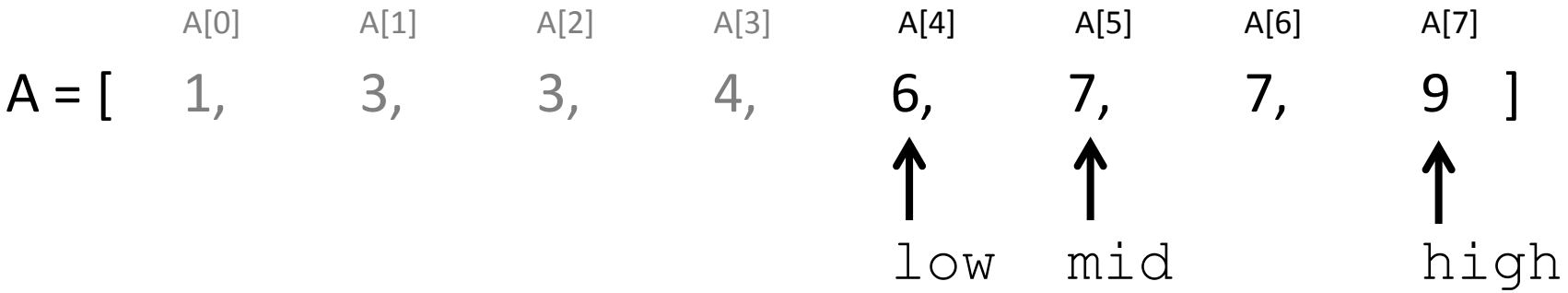
- Κλειδί αναζήτησης:

  `key = 8` `mid = (low+high)/2`

- Ελέγχουμε `A[mid] == key` ⇒ `return mid`

  `A[mid] < key` ⇒ `low=mid+1`

  `A[mid] > key` ⇒ `high=mid-1`



Παράδειγμα

- Κλειδί αναζήτησης:

  `key = 8` `mid = (low+high)/2`

- Ελέγχουμε `A[mid] == key` ⇒ `return mid`

  `A[mid] < key` ⇒ `low=mid+1`

  `A[mid] > key` ⇒ `high=mid-1`

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	
A = [	1,	3,	3,	4,	6,	7,	7,	9	]
							↑	↑	
							low	high	
							mid		

Παράδειγμα

- Κλειδί αναζήτησης:

  `key = 8` `mid = (low+high)/2`

- Ελέγχουμε `A[mid] == key` ⇒ `return mid`

  `A[mid] < key` ⇒ `low=mid+1`

  `A[mid] > key` ⇒ `high=mid-1`

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
A = [	1,	3,	3,	4,	6,	7,	7,	9]

high

low

mid

Παράδειγμα

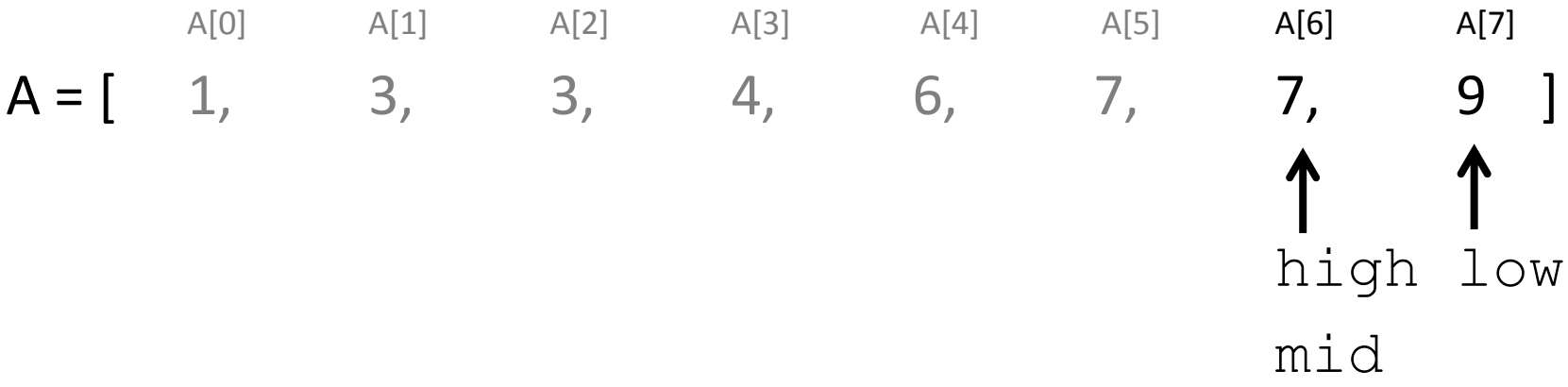
- Κλειδί αναζήτησης:

  `key = 8` `mid = (low+high)/2`

- Ελέγχουμε `A[mid] == key` ⇒ `return mid`

  `A[mid] < key` ⇒ `low=mid+1`

  `A[mid] > key` ⇒ `high=mid-1`



Παράδειγμα

- Κλειδί αναζήτησης:

  `key = 8` `mid = (low+high)/2`

- Ελέγχουμε `A[mid] == key` ⇒ `return mid`

  `A[mid] < key` ⇒ `low=mid+1`

  `A[mid] > key` ⇒ `high=mid-1`

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	
A = [	1,	3,	3,	4,	6,	7,	7,	9	]

high low

mid

high < low:
δεν βρέθηκε!

```
bool binarysearch(int a[ ], int low, int high, int key)
{
    int mid;
    if (low > high)
       return false;
    else
    {
        mid = (low + high)/2;
        if (key == a[mid])
            return true;
        else if (key < a[mid])
            return binarysearch(a, low, mid - 1, key);
        else if (key > a[mid])
            return binarysearch(a, mid + 1, high, key);
    }
}
```

```
bool binarysearch(int a[ ], int low, int high, int key)
{
    int mid;
    if (low > high)
       return false;
    else
    {
        mid = (low + high)/2;
        if (key == a[mid])
            return true;
        else if (key < a[mid])
            return binarysearch(a, low, mid - 1, key);
        else if (key > a[mid])
            return binarysearch(a, mid + 1, high, key);
    }
}
```

Αν θέλουμε και την θέση που βρίσκεται το key;

Ενότητα 19

ΕΛΑΧΙΣΤΑ & ΜΕΓΙΣΤΑ ΣΤΟΙΧΕΙΑ

Εύρεση Μεγίστου Ελαχίστου ταυτόχρονα

- Αν θέλουμε το μέγιστο και το ελάχιστο στοιχείο ενός πίνακα:

```
void minmax1(int a[], int size, int& min, int& max)
{
    max = 0;
    min = 0;
    for(int i=0; i < size; i++)
    {
       if(a[i] > a[max])
         max = i;
       if(a[i] < a[min])
         min = i;
    }
}
```

- Κάνει 2*n*-2 συγκρίσεις σε ένα πίνακα με *n* στοιχεία

Εύρεση Μεγίστου Ελαχίστου ταυτόχρονα

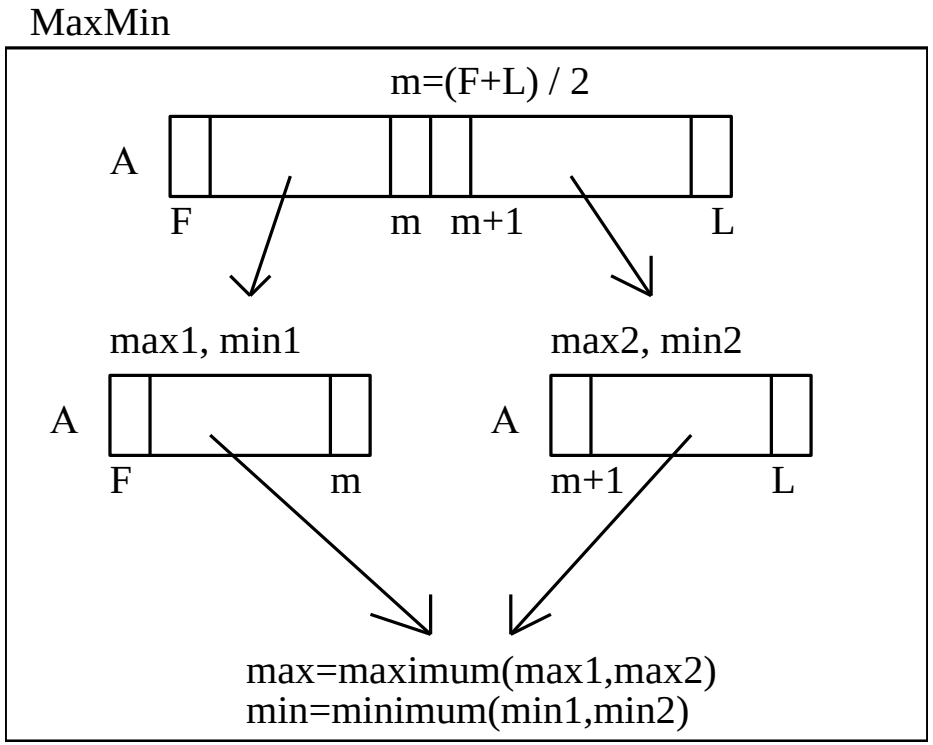
- Αν θέλουμε το μέγιστο και το ελάχιστο στοιχείο ενός πίνακα:

```
void minmax2(int a[], int size, int& min, int& max)
{
    max = 0;
    min = 0;
    for(int i=0; i < size; i++)
    {
       if(a[i] > a[max])
         max = i;
       else if(a[i] < a[min])
         min = i;
    }
}
```

- Κάνει κατά μέσο όρο n-1 + n/2 συγκρίσεις σε ένα πίνακα με n στοιχεία

Εύρεση Μεγίστου Ελαχίστου ταυτόχρονα

- Πιο γρήγορος: διαιρεί τον πίνακα σε δύο ήμισυ και βρίσκει αναδρομικά μέγιστο και ελάχιστο σε κάθε μισό
  - Το ολικό ελάχιστο είναι το ελάχιστο των δύο ελαχίστων και αντίστοιχα για το ολικό μέγιστο



- Κάνει το πολύ 3*n/2 - 2* συγκρίσεις

```
void minmax3(int a[], int low, int high, int& min, int& max)
{
    int min1, max1, min2, max2;
    int size;

    size = high – low + 1;

    if(size == 1)  // high = low
    {
       min = a[low];  max = a[high];
    }
    else if(size == 2) // a[] = {a[low], a[high]}
    {
       if(a[low] < a[high]) { min = a[low];  max = a[high]; }
       else                 { min = a[high];  max = a[low]; }
    }
    else
    {
       minmax3(a, low, (low + size/2 -1), min1, max1);
       minmax3(a, (low + size/2), high, min2, max2);
       min = (min1 < min2) ? min1 : min2 ;
       max = (max1 > max2) ? max1 : max2 ;
    }
}
```

Πίνακες (σύνοψη)

- Είναι μια συλλογή από μεταβλητές
- Βρόχοι for ταιριάζουν απόλυτα για τους πίνακες
- Είστε υπεύθυνοι για να μην βγείτε έξω από τα όρια του πίνακα
- Η παράμετρος Πίνακα είναι ένα "νέος" τύπος
  - Παρόμοια με το παράμετρο με αναφορά
- Τα στοιχεία του πίνακα αποθηκεύονται σειριακά
  - "Συνεχόμενο" κομμάτι στην μνήμη
  - Μόνο η διεύθυνση του 1ου στοιχείου περνάει σε συν/σεις
- Μερικώς συμπληρωμένοι πίνακες → περισσότερες μεταβλητές
- Πολυδιάστατοι πίνακες
  - "πίνακας από πίνακες"

Καλή Μελέτη

- **Βιβλιογραφία**

[1] W. Savitch, Πλήρης C++, Εκδόσεις Τζιόλα, 2011

[2] H. Deitel and P. Deitel, C++ Προγραμματισμός 6η Εκδοση, Εκδόσεις Μ. Γκιούρδας, 2013

Ύλη βιβλιογραφίας
[1]: 5.3, 13.3
[2]: 7.7, 7.8, 8.6, Κεφ. 19