Εισαγωγή στον Προγραμματισμό

Μέρος 3ο: Πίνακες - Ταξινόμηση - Αναζήτηση

Εξάμηνο Σπουδών: 3ο

Κωδικός Μαθήματος: 343

Τμήμα Μαθηματικών
Πανεπιστήμιο Ιωαννίνων

Μιχάλης Α. Μπέκος
bekos@uoi.gr

Εισαγωγή

- Πίνακες:
  - Συλλογή δεδομένων ίδιου τύπου
  - Στατική δομή - η πληθικότητα της δεν τροποποιείται

- Τύποι πινάκων:
  - Στην C: πίνακες βασιζόμενοι σε δείκτες
  - Στην C++: πίνακες αντικείμενα

- Παραδοχές:
  - πίνακας $\rightarrow$ πίνακας της C
    $\rightarrow$ μονοδιάστατος πίνακας
  - ...εκτός εάν ορίζεται διαφορετικά

Μέρος 1°:
Εισαγωγικές έννοιες

Πίνακες

- **Ορισμός:** Συλλογή δεδομένων ίδιου τύπου
- Η πρώτη σύνθετη δομή τύπων:
  - σύνθετη: “ομαδοποίηση”
  - απλές δομές τύπων:
    `int, float, double, char, bool`
- Ομαδοποιεί κοινές μεταβλητές σε μια “δομή”
  - Βαθμολογίες, θερμοκρασίες, ονόματα, κ.τ.λ
  - Αποφεύγουμε να δηλώσουμε πολλές απλές μεταβλητές
  - Μπορεί να χειριστεί τη “δομή” ως μια οντότητα

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

int main() {
  int day, month, total_days;
  int monthdays[12] = {31, 28, 31, 30,
                       31, 30, 31, 31,
                       30, 31, 30, 31};
  // get date
  cout << "Enter month(1 to 12): ";
  cin >> month;
  cout << "Enter day(1 to 31): ";
  cin >> day;

  // find the no of days from year start
  total_days = day;
  for (int j = 0; j < month - 1; j++) {
    total_days += monthdays[j];
  }

  cout << "Total days from year start:"
       << total_days << endl;
  return 0;
}
```

Part03/days.cpp

Δήλωση Πινάκων

- Σύνταξη δήλωσης πίνακα:

  όνομα_τύπου όνομα_πίνακα [μέγεθος];

- Τα στοιχεία του πίνακα όνομα_πίνακα είναι:

  όνομα_πίνακα [0],...,όνομα_πίνακα [μέγεθος-1];

- Κάθε στοιχείο είναι μια μεταβλητή τύπου όνομα_τύπου

- Παραδείγματα:
  - `double numbers[100];`
    - `numbers[0],...,numbers[99]` → όλα τύπου `double`
  - `int grades[10];`
    - `grades[0],...,grades[9]` → όλα τύπου `int`

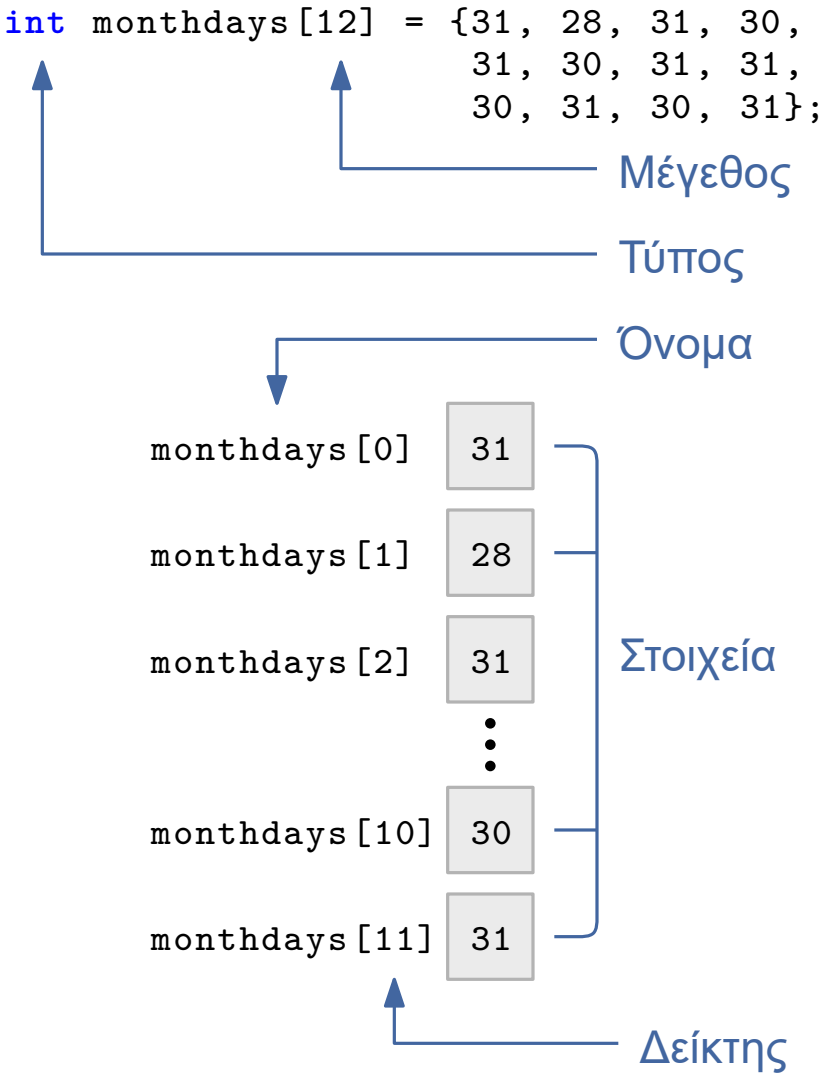
Δήλωση Πινάκων

- Ανοίγει κατάλληλες θέσεις στη μνήμη
  - 12 μεταβλητές τύπου `int`:

    `int monthdays[12];`
  - Οι 12 μεταβλητές θα είναι:

    `monthdays[0], ..., monthdays[11]`

- Συστατικά ενός πίνακα:
  - Όνομα
  - Τύπος
  - Μέγεθος
  - Στοιχεία
  - Δείκτης



Το μέγεθος του πίνακα

- Το μέγεθος ενός πίνακα πρέπει να είναι μια ακέραια θετική σταθερά
  - Δεν επιτρέπεται δήλωση πίνακα μεταβλητού μεγέθους
- Χρήση καθορισμένης σταθεράς για το μέγεθος του πίνακα:
  - Αντί να χρησιμοποιούμε έναν αριθμό (π.χ., 5) για το μέγεθος του πίνακα μπορούμε να κάνουμε χρήση προκαθορισμένης σταθεράς,
  - η οποία μπορεί να χρησιμοποιηθεί και για την προσπέλαση των στοιχείων του πίνακα
- Η C++ σας επιτρέπει να αναφερθείτε σε στοιχεία εκτός ορίου
  - Τα αποτελέσματα είναι ασαφή
  - Ο μεταφραστής δεν αναγνωρίζει τέτοια λάθη

```
int arr1[0];
int arr2[-1];
```

```
int size = 3;
int arr3[size];
```

```
cout << "Specify a number:";
cin >> number;
int arr4[number];
```

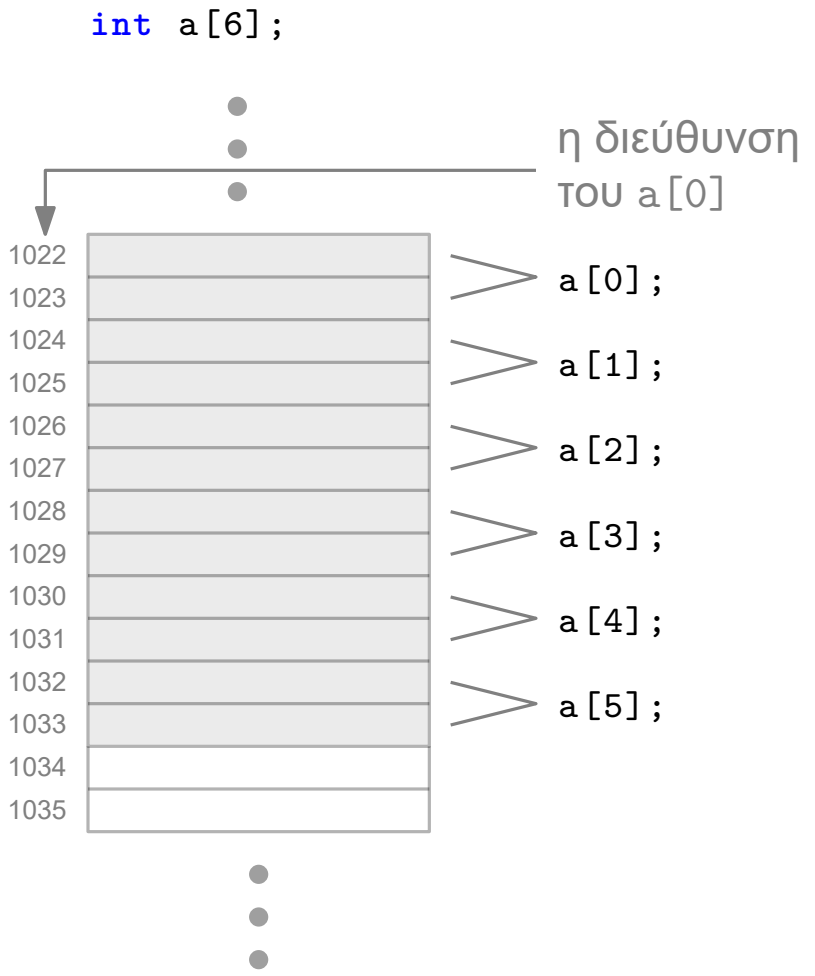
```
const int NO_OF_STUDENTS = 3;
int grades[NO_OF_STUDENTS];
...
for (int i=0; i<NO_OF_STUDENTS; i++) {
  cout << grades[i];
}
```

```
double temperatures[24];
temperatures[24] = 5;
```

Οι πίνακες στη μνήμη

- Μια απλή μεταβλητή περιγράφεται από:
  - Μία διεύθυνση μνήμης (αρχή του πρώτου byte)
  - και ένα τύπος μεταβλητής (πόσα byte καταλαμβάνει)

- Πίνακες:
  - Όλα τα στοιχεία τοποθετούνται το ένα δίπλα στο άλλο
  - Ο πίνακας “θυμάται” τη διεύθυνση του πρώτου του στοιχείου (και κανενός άλλου)
  - Π.χ., η δήλωση `int a[6]` δεσμεύει 6 θέσεις για μεταβλητές τύπου `int`



Αρχικοποίηση και προβολή των στοιχείων

- Ο λάθος τρόπος:

```
int grades[5];
cin >> grades;
```

```
int grades[5] = {4, 8, 6, 7, 5};
cout << grades;
```

- Ο σωστός τρόπος:

```
int grades[5];

// reading one variable at a time:
for (int i=0; i<5; i++) {
  cout << "Specify grade " << i << ": ";
  cin >> grades[i];
}
```

```
int grades[5] = {4, 8, 6, 7, 5};

// displaying all variables:
cout << "The grades are: ";
for (int i=0; i<5; i++) {
  cout << grades[i];
}
```

Επισημάνσεις

- Για την πρόσβαση στα στοιχεία ενός πίνακα, συνήθως χρησιμοποιείται εντολή επανάληψης `for`

- Οι δείκτες των πινάκων αρχίζουν από το 0
  - αρχίζουν πάντοτε από το 0
  - αρχίζουν πάντοτε από το 0
  - αρχίζουν πάντοτε από το 0
  - ...
  - τελειώνουν με τον ακέραιο που είναι μικρότερος κατά 1 από το μέγεθος του πίνακα

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

const int NO_OF_STUDENTS = 4;

int main() {
  int grades[NO_OF_STUDENTS] = {7,5,4,8};

  // upgrade all grades by 1
  for (int i=0; i<NO_OF_STUDENTS; i++) {
    grades[i] += 1;
  }

  // print array elements
  cout << "The grades are: ";
  for (int i=0; i<NO_OF_STUDENTS; i++) {
    cout << grades[i] << " ";
  }
  cout << endl;

  return 0;
}
```

Part03/grades.cpp

Μερικώς συμπληρωμένοι πίνακες

- Πολλές φορές δεν γνωρίζουμε από πριν το ακριβές μέγεθος του πίνακα
  - Παράδειγμα: Είσοδος αριθμών από τον χρήστη
- Στην περίπτωση αυτή πρέπει να:
  - Δημιουργήσουμε ένα πίνακα με μέγεθος μεγαλύτερο από τον αριθμό των στοιχείων που αναμένουμε
  - Χρησιμοποιήσουμε μια επιπλέον μεταβλητή για την καταγραφή του πλήθους των στοιχείων

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

int main() {
  int numbers[1000];
  int input = 0, size = 0;

  cout << "Enter numbers (-1 to stop):";

  for (int i = 0; i<1000; i++) {
    cin >> input;

    if (input == -1) break;

    numbers[i] = input;
    size++;
  }

  cout << "The given numbers are: ";
  for (int i = 0; i<size; i++) {
    cout << numbers[i] << " ";
  }
  return 0;
}
```

Part03/partially_filled.cpp

Μέρος 2°:

Πίνακες σε συναρτήσεις

Πίνακες σε συναρτήσεις

- Ως παράμετροι σε συναρτήσεις:
  - Μεμονωμένα στοιχεία του πίνακα
  - Ολόκληρο τον πίνακα

- Ως επιστρεφόμενες τιμές από συναρτήσεις:
  - Δεν θα το δούμε με απλούς πίνακες
  - Εύκολο με πίνακες αντικείμενα (αργότερα)

Στοιχεία πινάκων ως παράμετροι

- Χρησιμοποιούνται όπως ακριβώς οι απλές μεταβλητές
- **Παράδειγμα:** Η εκτύπωση των στοχείων ενός πίνακα

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

const int NO_OF_STUDENTS = 4;

// prints the specified element
// to the standard output
void print(int element);

int main() {
  int grades[NO_OF_STUDENTS] = {7,5,4,8};

  // print array elements
  cout << "The grades are: ";
  for (int i=0; i<NO_OF_STUDENTS; i++) {
    print(grades[i]);
  }

  return 0;
}

void print(int element) {
  cout << element << endl;
}
```

Part03/grades.cpp

Ολόκληροι πίνακες ως παράμετροι

- Αν θέλουμε να περάσουμε ως παράμετρο ολόκληρο τον πίνακα:
  - Χρησιμοποιούμε το όνομα του πίνακα
  - Ο τύπος της παραμέτρου είναι πίνακας
- Στέλνουμε και το μέγεθος του πίνακα ως παράμετρο:
  - Ως μια τυπική παράμετρο τύπου int
- Στην κλήση της συνάρτησης στέλνουμε ολόκληρο τον πίνακα, χωρίς [ ]
- Παράμετρος πίνακα: **με αναφορά**

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

const int NO_OF_ELEMENTS = 4;

// it fills up the input array
void fillUp(int a[], int n);

int main() {
  int elements[NO_OF_ELEMENTS];

  fillUp(elements, NO_OF_ELEMENTS);

  cout << "The given elements are: ";
  for (int i=0; i<NO_OF_ELEMENTS; i++) {
    cout << elements[i] << endl;
  }
}

void fillUp(int a[], int n) {
  cout << "Enter " << n << " elements";
  for (int i = 0; i < n; i++)
    cin >> a[i];
}
```

Part03/listing.cpp

Ολόκληροι πίνακες ως παράμετροι

- Αν θέλουμε να περάσουμε ως παράμετρο ολόκληρο τον πίνακα:
  - Χρησιμοποιούμε το όνομα του πίνακα
  - Ο τύπος της παραμέτρου είναι πίνακας
- Στέλνουμε και το μέγεθος του πίνακα ως παράμετρο:
  - Ως μια τυπική παράμετρο τύπου int
- Στην κλήση της συνάρτησης στέλνουμε ολόκληρο τον πίνακα, χωρίς [ ]
- Παράμετρος πίνακα: **με αναφορά**

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

const int NO_OF_ELEMENTS = 4;

// it fills up the input array
void fillUp(int a[], int n);

int main() {
  int elements[NO_OF_ELEMENTS];

  fillUp(elements, NO_OF_ELEMENTS);

  cout << "The given elements are: ";
  for (int i=0; i<NO_OF_ELEMENTS; i++) {
    cout << elements[i] << endl;
  }
}

void fillUp(int a[], int n) {
  cout << "Enter " << n << " elements";
  for (int i = 0; i < n; i++)
    cin >> a[i];
}
```

Part03/listing.cpp

Μπορούμε να χρησιμοποιούμε την ΙΔΙΑ συνάρτηση για να γεμίσουμε πίνακες διαφορετικού μεγέθους

Μέρος 3°:
Παραδείγματα

Αρχικοποίηση πίνακα από τον χρήστη

- Για την αρχικοποίηση ενός πίνακα με στοιχεία που θα δώσει ο χρήστης, χρησιμοποιούμε εντολή επανάληψης `for`
  - Σε κάθε επανάληψη, αρχικοποιούμε ένα στοιχείο του πίνακα στη δοθείσα τιμή.

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

/**
 * Initialize all n elements of the
 * input array a
 */
void initialize(int a[], const int& n);

int main() {
  const int SIZE = 5;
  int a[SIZE];
  initialize(a, SIZE);
}

void initialize(int a[], const int& n) {
  for (int i=0; i<n; i++) {
    cout << "a[" << i << "]= ";
    cin >> a[i];
  }
  cout << endl;
}
```

Part03/stdoperations.cpp

Προβολή των στοιχείων ενός πίνακα

- Για την προβολή των στοιχείων ενός πίνακα στο τερματικό, χρησιμοποιούμε ξανά εντολή επανάληψης `for`
  - Σε κάθε επανάληψη, δίνεται στην έξοδο ένα στοιχείο του πίνακα.
  - Από τη στιγμή που τα στοιχεία του πίνακα δεν τροποποιούνται, μια καλή ιδέα είναι να ορίσετε τον πίνακα ως `const` στη συνάρτηση σας.

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

/**
 * Output all n elements of the
 * input array a
 */
void display(const int a[], const int& n);

int main() {
  const int SIZE = 5;
  int a[SIZE];
  initialize(a, SIZE);
  display(a, SIZE);
}

void display(const int a[], const int& n)
{
  for (int i=0; i<n; i++) {
    cout << a[i] << " ";
  }
  cout << endl;
}
```

Part03/stdoperations.cpp

Εύρεση ελαχίστου

- Για την εύρεση του ελαχίστου στοιχείου ενός πίνακα, διατρέχουμε τα στοιχεία του πίνακα ένα προς ένα.
- Διατηρούμε μια μεταβλητή, η οποία αντιστοιχεί στο ελάχιστο των στοιχείων που έχουμε επισκεφτεί.
  - Η μεταβλητή αυτή αρχικοποιείται στο πρώτο στοιχείο του πίνακα
  - Σε κάθε επανάληψη η τιμή της επαναπροσδιορίζεται
- Αντίστοιχα υλοποιείται η εύρεση μεγίστου

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

// Return the minimum of the n elements
// of the input array a
int min(const int a[], const int& n);

int main() {
  const int SIZE = 5;
  int a[SIZE];
  initialize(a, SIZE);
  cout << "Min: " << min(a, SIZE);
}

int min(const int a[], const int& n) {
  int min = a[0];
  for (int i=1; i<n; i++) {
    if (a[i] < min) {
      min = a[i];
    }
  }
  return min;
}
```

Part03/stdoperations.cpp

Εύρεση του δείκτη του ελαχίστου στοιχείου

- Για την εύρεση του δείκτη του ελαχίστου στοιχείου ενός πίνακα, διατρέχουμε τα στοιχεία του πίνακα ένα προς ένα.
- Εντοπίζουμε το ελάχιστο στοιχείο.
- Χρησιμοποιώντας ακόμη μια βοηθητική μεταβλητή, η οποία διατηρεί το δείκτη του τρέχοντος ελαχίστου σε κάθε επανάληψη, επιστρέφουμε τη θέση του.

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

// Return the index of the minimum of the
// n elements of the input array a
int minidx(const int a[], const int& n);

int main() {
  const int SIZE = 5;
  int a[SIZE];
  initialize(a, SIZE);
  cout << "minidx: " << minidx(a, SIZE);
}

int minidx(const int a[], const int& n) {
  int index = 0;
  int min = a[0];
  for (int i=1; i<n; i++) {
    if (a[i] < min) {
      min = a[i]; index = i;
    }
  }
  return index;
}
```

Part03/stdoperations.cpp

Εύρεση του αθροίσματος των στοιχείων πίνακα

- Για την εύρεση του αθροίσματος των στοιχείων ενός πίνακα, διατρέχουμε τα στοιχεία του πίνακα ένα προς ένα.
- Σε κάθε επανάληψη διατηρούμε μια μεταβλητή, η οποία αντιστοιχεί στο άθροισμα των στοιχείων που έχουμε ήδη επισκεφτεί.
- Όταν ολοκληρωθεί η διαπέραση, επιστρέφουμε την τιμή της μεταβλητής.

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

// Return the sum of the n elements
// of the input array a
int sum(const int a[], const int& n);

int main() {
  const int SIZE = 5;
  int a[SIZE];
  initialize(a, SIZE);
  cout << "Sum: " << sum(a, SIZE);
}

int sum(const int a[], const int& n) {
  int sum = 0;
  for (int i=0; i<n; i++)
  {
    sum += a[i];
  }
  return sum;
}
```

Part03/stdoperations.cpp

Εύρεση του εύρους και του μ.ο. των στοιχείων πίνακα

- Το εύρος των στοιχείων ενός πίνακα αντιστοιχεί στη διαφορά μεταξύ του μέγιστου και του ελάχιστου στοιχείου του.

- Ο μέσος όρος των στοιχείων ενός πίνακα αντιστοιχεί στο πηλίκο του αθροίσματους προς το πλήθος τους.

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

// Return the range and the mean of the
// n elements of the input array a
int range(const int a[], const int& n);
double mean(const int a[], const int& n);

int main() {
  const int SIZE = 5;
  int a[SIZE];
  initialize(a, SIZE);
  cout << "Range: " << range(a, SIZE);
  cout << "Mean: " << mean(a, SIZE);
}

int range(const int a[], const int& n) {
  return max(a,n) - min(a,n);
}

double mean(const int a[], const int& n)
{
  return (double) sum(a,n)/n;
}
```

Part03/stdoperations.cpp

Εύρεση της τυπικής απόκλισης των στοιχείων πίνακα

- Η τυπική απόκλιση βρίσκεται λαμβάνοντας την τετραγωνική ρίζα του μέσου όρου των τετραγώνων των αποκλίσεων των τιμών από τη μέση τιμή τους.
  - Π.χ. η τυπική απόκληση των οχτώ στοιχείων 2, 4, 4, 4, 5, 5, 7, 9, των οποίων ο μ.ο. είναι 5, είναι:

$$\sqrt{\frac{(2-5)^2+(4-5)^2+\ldots+(9-5)^2}{8}} = 2$$

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

// Return the standard deviation of the
// n elements of the input array a
double stdiv(const int a[], const int& n);

int main() {
  const int SIZE = 5;
  int a[SIZE];
  initialize(a, SIZE);
  cout << "Stdiv: " << stdiv(a, SIZE);
}

double stdiv(const int a[], const int& n)
{
  double stdiv = 0.0;
  double amean = mean(a,n);
  for (int i=0; i<n; i++) {
    stdiv += pow(a[i]-amean,2);
  }
  return sqrt(stdiv/n);
}
```

Part03/stdoperations.cpp

Μέρος 4°:

Πίνακες και Αντικείμενα

Πίνακες με στοιχεία αντικείμενα

- Δηλώνονται κανονικά όπως οι βασικοί τύποι δεδομένων:
  - Π.χ. `Fraction rationals[3] = {Fraction(), Fraction(2,4), Fraction(5,2)};`
- Σε κάθε θέση του πίνακα υπάρχει ένα αντικείμενο, το οποίο υποστηρίζει αντίστοιχες μεθόδους:
  - Π.χ. `dist[i].showDistance();`

```
#include <iostream>
using namespace std;
class Distance {
private:
  unsigned int meters, centimeters;
public:
  ...
  void getDistance() {
    cout << "Meters: "; cin >> meters;
    cout << "CMs: "; cin >> centimeters;
  }
  void showDistance() {
    cout << meters << ","
         << (centimeters <10 ? "0" : "")
         << centimeters << endl;
  }
};
int main() {
  Distance dist[5];
  for (int i=0; i<5; i++)
    dist[i].getDistance();
  for (int i=0; i<5; i++)
    dist[i].showDistance();
}
```

Part03/Distance.cpp

Πίνακες ως πεδία κλάσεων

- Οι πίνακες μπορεί να χρησιμοποιηθούν ως μεταβλητές-μέλη (πεδία) κλάσεων

- **Παράδειγμα:** Στοίβα
  - Οι εισαγωγές και οι διαγραφές ακολουθούν το σχήμα “last-in first-out”
  - Εφαρμογές:

    Αναιρέσεις (undo) ενός προγράμματος

    Βοηθητική δομή δεδομένων για αλγόριθμους

    Συστατικό στοιχείο άλλων δομών δεδομένων

    Ιστορικό επισκέψεων ιστοσελίδων web browser

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

const int STACK_MAX_SIZE = 100;

class Stack {
private:
  int elements[STACK_MAX_SIZE];
  int index; // <-- index to latest entry

public:
  Stack(): index(-1) {}

  void push(int element) {
    elements[++index] = element;
  }
  void pop() {
    index--;
  }
  int top() {
    return elements[index];
  }
  ...
};
```

Part03/Stack.cpp

Πίνακες ως πεδία κλάσεων

- Οι πίνακες μπορεί να χρησιμοποιηθούν ως μεταβλητές-μέλη (πεδία) κλάσεων

- **Παράδειγμα:** Στοίβα
  - Οι εισαγωγές και οι διαγραφές ακολουθούν το σχήμα “last-in first-out”

`empty()`	ελέγχει αν η στοίβα είναι κενή
`size()`	επιστρέφει το πλήθος των στοιχείων της
`push(e)`	ενθέτει το στοιχείο e στη στοίβα
`pop()`	διαγράφει το τελευταίο εισαχθέν στοιχείο
`top()`	επιστρέφει το τελευταίο εισαχθέν στοιχείο (χωρίς να το διαγράφει)

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

const int STACK_MAX_SIZE = 100;

class Stack {
private:
  int elements[STACK_MAX_SIZE];
  int index; // <-- index to latest entry

public:
  Stack(): index(-1) {}

  void push(int element) {
    elements[++index] = element;
  }
  void pop() {
    index--;
  }
  int top() {
    return elements[index];
  }
  ...
};
```

Part03/Stack.cpp

Μέρος 5°:

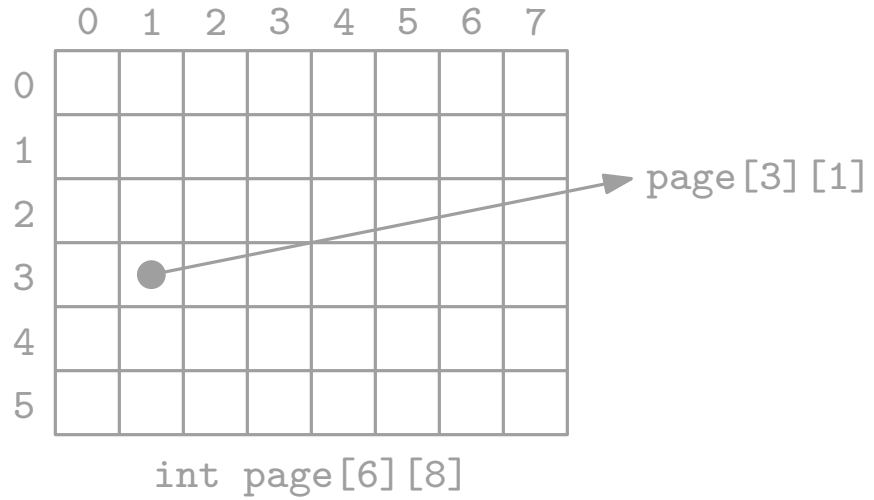
Πολυδιάστατοι πίνακες

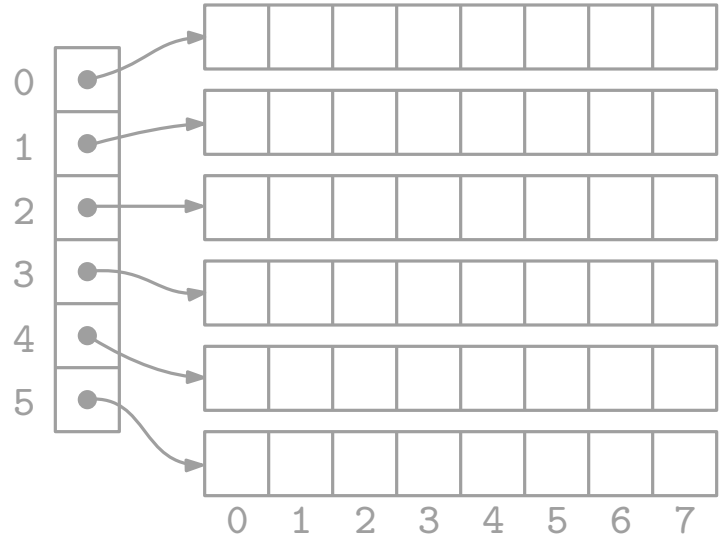
Πολυδιάστατοι πίνακες

- Πίνακες με παραπάνω από ένα δείκτη
  - Π.χ. `int page[6][8]`
- Οπτικοποίηση:

```
page[0][0], page[0][1],..., page[0][7]
page[1][0], page[1][1],..., page[1][7]
...
page[5][0], page[5][1],..., page[5][7]
```

- Η C++ επιτρέπει οποιοδήποτε αριθμό από δείκτες
  - Δεν θα ασχοληθούμε με παραπάνω από δύο
- Μπορούν να οριστούν και ως πίνακες από πίνακες





Αρχικοποίηση

- Πρώτη μέθοδος:

```
int x[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

- Δεύτερη μέθοδος:

```
int x[3][4] = {{0,1,2,3},
               {4,5,6,7},
               {8,9,10,11}};
```

- Τρίτη μέθοδος:

```
int x[3][4];
for(int i = 0; i < 3; i++){
  for(int j = 0; j < 4; j++){
    cin >> x[i][j];
  }
}
```

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

// C++ program to print the elements of
// a 2-dimensional array
int main()
{
  // an array with 3 rows and 2 columns.
  int x[3][2] = {{0,1}, {2,3}, {4,5}};

  // output each array element's value
  for (int i = 0; i < 3; i++)
  {
    for (int j = 0; j < 2; j++)
    {
      cout << "Element at x[" << i
           << "][" << j << "]: ";
      cout << x[i][j] << endl;
    }
  }

  return 0;
}
```

Πολυδιάστατοι πίνακες ως παράμετροι συναρτήσεων

- Παρόμοια με μιας διάστασης πίνακα
- Με σημαντικές διαφορές:
  - Η 1η διάσταση δεν δίνεται απαραίτητα
  - Η 2η διάσταση ΔΙΝΕΤΑΙ
- Η χρήση αριθμητικών σταθερών για τον προσδιορισμό του μεγέθους των πινάκων είναι πολύτιμη.

```
#include <iostream>
using namespace std;
const int N = 3;

// Stores in C the addition of A and B
void add(int A[][N],int B[][N],int C[][N])
{
  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
      C[i][j] = A[i][j] + B[i][j];
}
int main() {
  int A[N][N]={{1,1,1},{2,2,2},{3,3,3}};
  int B[N][N]={{4,4,4},{4,5,5},{6,6,6}};

  int C[N][N];
  add(A, B, C);

  cout << "Result matrix is " << endl;
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++)
      cout << C[i][j] << " ";
    cout << endl;
  }
}
```

Part03/matrices.cpp

Μέρος 6°:

std::array (πίνακες ως αντικείμενα στην C++)

std::array

- Μια wrapper κλάση για τους standard πίνακες της C όπως τους έχουμε δει ως τώρα
  - Για την αρχικοποίηση τους απαιτείται ο τύπος και το μέγεθος τους
  - Το μέγεθος δεν είναι απαραίτητα σταθερά
  - `#include <array>`
- Μέθοδοι:
  - `size()`: πλήθος στοιχείων
  - `empty()`: ελέγχει αν ο πίνακας είναι κενός
  - `at(int )`: πρόσβαση μέσω δείκτη
  - `front()`: πρώτο στοιχείο
  - `back()`: τελευταίο στοιχείο
  - `fill()`: γεμίζει τον πίνακα με την δοθείσα τιμή

```cpp
#include <iostream>
#include <array>

int main () {
  // instantiate an array of integers
  // of size 10
  std::array<int,10> myarray;

  // the at() method procides access to
  // the elements of the array, e.g., to
  // initiate them
  for (int i=0; i<myarray.size(); i++) {
    myarray.at(i) = i+1;
  }

  // using at() method one can also
  // output the elements of the array
  std::cout << "Contents of the array:";
  for (int i=0; i<myarray.size(); i++) {
    std::cout << ' ' << myarray.at(i);
  }
  std::cout << std::endl;

  return 0;
}
```

Part03/stdarray.cpp

Δήλωση - Αρχικοποίηση

- **Δήλωση:**
  `std::array<τύπος, μέγεθος> όνομα;`
- Το μέγεθος τους δεν αλλάζει μετά την αρχικοποίηση τους
- Διάφοροι τρόποι αρχικοποίησης τους
- Οι πίνακες τύπου std::array είναι παράμετροι με τιμή σε συναρτήσεις
  - Υπάρχει τρόπος να μην ορίσουμε τον τύπο και το μέγεθος τους στη συνάρτηση (θα το δούμε αργότερα).

```cpp
#include <iostream>
#include <array>

void printArray(std::array<int, 5>& A) {
  // Printing array
  for (int i=0; i<A.size(); i++)
    std::cout << A.at(i) << " , ";
  std::cout << std::endl;
}
int main() {
  // Uninitialized array (garbage values)
  std::array<int, 5> a;
  printArray(a);
  // Initialized array
  std::array<int, 5> b = {1,2,3,4,5};
  printArray(b);
  // First 2 values given; others -> 0.
  std::array<int, 5> c = {1,2};
  printArray(c);
  // Fill all elements with same value
  std::array<int, 5> d;
  d.fill(4);
  printArray(d);
  return 0;
}
```

Part03/stdarrayinit.cpp

Χρήση του τελεστή `[]` αντί της μεθόδου `at()`

- Τα στοιχεία ενός πίνακα `std::array` μπορούν να προσπελαθούν χρησιμοποιώντας τον τελεστή `[]` αντί της μεθόδου `at()`
- Με τον τρόπο αυτό, η χρήση τους γίνεται πανομοιότυπη με εκείνη των κλασικών πινάκων της C.
- Πλεονέκτημα: Πρόσβαση στο μέγεθος τους (και άλλα πολλά που θα δούμε αργότερα).

```cpp
#include <iostream>
#include <array>

using namespace std;

int main()
{
  array<int, 5> n;

  //taking values of elements from user
  for(int i=0; i<n.size(); i++)
  {
    cout << "Enter value of n[" << i
         << "]: ";
    cin >> n[i];
  }

  // printing the values of elements
  for (int j=0; j<n.size(); j++)
  {
    cout << "n[" << j << "] = "
         << n[j] << endl;
  }
  return 0;
}
```

Part03/stdarrayelems.cpp

Επιπλέον υλικό

- Για επιπλέον υλικό και ολοκληρωμένα παραδείγματα ως προς τις μεθόδους που υποστηρίζονται:
  - https://cplusplus.com/reference/array/array/

Μέρος 7°:
Αλγόριθμοι ταξινόμησης

Ταξινόμηση Πινάκων

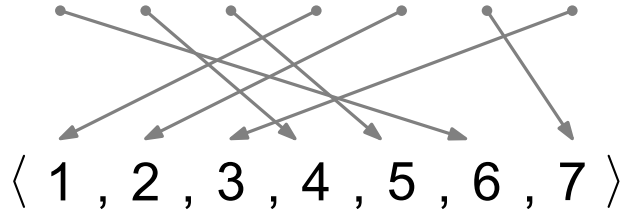
- Ταξινόμηση σε αύξουσα τάξη:

  $\texttt{a[0]} \leq \texttt{a[1]} \leq \ldots \leq \texttt{a[99]}$

- Ταξινόμηση σε φθίνουσα τάξη:

  $\texttt{a[0]} \geq \texttt{a[1]} \geq \ldots \geq \texttt{a[99]}$

- Παράδειγμα: $\langle 6, 4, 5, 1, 2, 7, 3 \rangle$



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

Αποτελεσματικότητα αλγορίθμου

- Θα εξετάσουμε διάφορους αλγορίθμους που επιλύουν το πρόβλημα της ταξινόμησης
- Όταν σχεδιάζουμε έναν αλγόριθμο μας ενδιαφέρει η αποτελεσματικότητά του (ταχύτητα)
- Ένας τρόπος για να ποσοτικοποιούμε την αποτελεσματικότητα ενός τέτοιου αλγορίθμου γίνεται με την εύρεση του πλήθος των συγκρίσεων που χρησιμοποιεί
- Το πλήθος των συγκρίσεων μπορεί να εξαρτάται από την είσοδο. Περιπτώσεις:
  - χειρότερη
  - μέση
  - καλύτερη

Χρήση βοηθητικών συναρτήσεων

- Σε όλους τους αλγορίθμους που ακολουθούν θα χρησιμοποιήσουμε τις ακόλουθες συναρτήσεις:
  - Αρχικοποίηση στοιχείων

    ```
    void initialize(int a[], const int& n);
    ```

  - Εκτύπωση στοιχείων

    ```
    void display(const int a[], const int& n);
    ```

  - Εναλλαγή στοιχείων

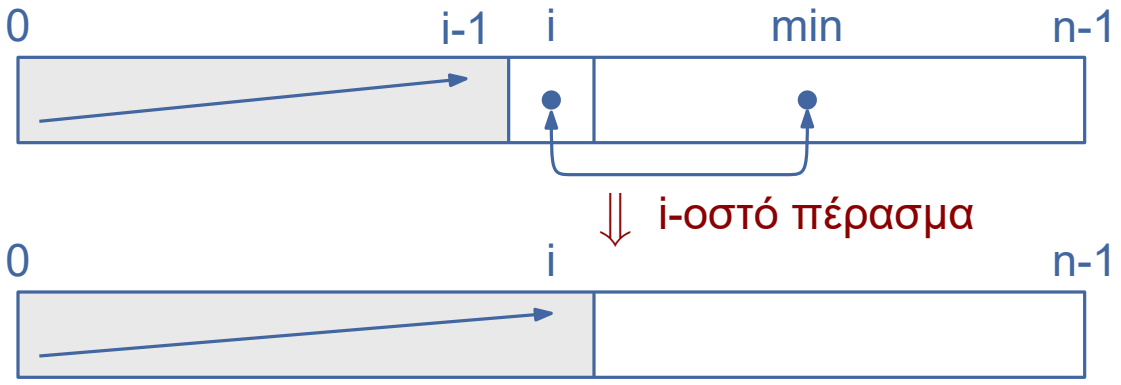
    ```
    void swap(int &x, int &y);
    ```

```
// Initialize all n elements of the input
// array a
void initialize(int a[], const int& n) {
  for (int i=0; i<n; i++) {
    cout << "a[" << i << "]= ";
    cin >> a[i];
  }
  cout << endl;
}

// Output all n elements of the input
// array a
void display(const int a[], const int& n) {
  for (int i=0; i<n; i++) {
    cout << a[i] << " ";
  }
  cout << endl;
}

// Swap the values of the two parameters.
void swap(int &x, int &y) {
  int temp = x;
  x = y;
  y = temp;
}
```

Part03/sorting/auxiliary.cpp

Ταξινόμηση επιλογής - Selection sort

- Κάνει $n-1$ περάσματα πάνω από τον πίνακα: $i = 0, 1, \ldots, n-2$
- Κατά το i-οστό πέρασμα, επιλέγουμε το μικρότερο στοιχείο μεταξύ των στοιχείων στις θέσεις $j = i, \ldots, n-1$ και το ανταλλάσσουμε με αυτό στην i θέση του πίνακα.
- Οπτικοποίηση:




```cpp
#include <iostream>
#include "auxiliary.cpp"
using namespace std;


void selectionSort(int a[], int n) {
  for (int i=0; i<n-1; i++) {
    // Find min in the unsorted part
    int min = i;
    for (int j=i+1; j<n; j++)
      if (a[j] < a[min])
        min = j;
    // Swap min with the i-th element
    swap(a[min], a[i]);
  }
}

int main() {
  const int SIZE = 10;
  int a[SIZE];
  initialize(a, SIZE);
  selectionSort(a, SIZE);
  cout << "After sorting: ";
  display(a, SIZE);
}
```

Part03/sorting/selectionsort.cpp

Ταξινόμηση επιλογής - Selection sort

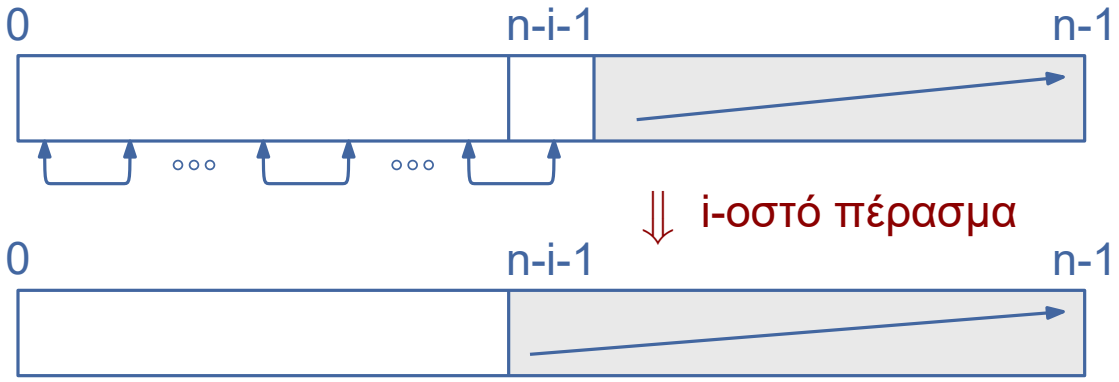
- Κάνει $n-1$ περάσματα πάνω από τον πίνακα: $i = 0, 1, \ldots, n-2$
- Κατά το i-οστό πέρασμα, επιλέγουμε το μικρότερο στοιχείο μεταξύ των στοιχείων στις θέσεις $j = i, \ldots, n-1$ και το ανταλλάσσουμε με αυτό στην i θέση του πίνακα.
- Η i-οστή επανάληψη απαιτεί $n-i-1$ συγκρίσεις.
  - Συνολικά:
    $(n-1) + \ldots + 1 = \frac{n(n-1)}{2} = \mathcal{O}(n^2)$ συγκρίσεις

```cpp
#include <iostream>
#include "auxiliary.cpp"
using namespace std;


void selectionSort(int a[], int n) {
  for (int i=0; i<n-1; i++) {
    // Find min in the unsorted part
    int min = i;
    for (int j=i+1; j<n; j++)
      if (a[j] < a[min])
        min = j;
    // Swap min with the i-th element
    swap(a[min], a[i]);
  }
}

int main() {
  const int SIZE = 10;
  int a[SIZE];
  initialize(a, SIZE);
  selectionSort(a, SIZE);
  cout << "After sorting: ";
  display(a, SIZE);
}
```

Part03/sorting/selectionsort.cpp

Ταξινόμηση φυσαλίδας - Bubble sort

- Κάνει $n-1$ περάσματα πάνω από τον πίνακα: $i = 0, 1, \ldots, n-2$
- Κατά το i-οστό πέρασμα, διαδοχικά ζεύγη στοιχείων στις θέσεις $j = 0, 1, \ldots, n-i-1$ συγκρίνονται.
  - Εάν είναι σε φθίνουσα τάξη, τα εναλλάσουμε.
- Οπτικοποίηση:




```cpp
#include <iostream>
#include "auxiliary.cpp"
using namespace std;


void bubbleSort(int a[], int n) {
  for (int i=0; i < n-1; i++) {
    // Last i elements are already sorted
    for (int j=0; j < n-i-1; j++) {
      if (a[j] > a[j+1])
        swap(a[j], a[j+1]);
    }
  }
}

int main() {
  const int SIZE = 10;
  int a[SIZE];
  initialize(a, SIZE);
  bubbleSort(a, SIZE);
  cout << "After sorting: ";
  display(a, SIZE);
}
```

Part03/sorting/bubblesort.cpp

Ταξινόμηση φυσαλίδας - Bubble sort

- Κάνει $n-1$ περάσματα πάνω από τον πίνακα: $i = 0, 1, \ldots, n-2$
- Κατά το i-οστό πέρασμα, διαδοχικά ζεύγη στοιχείων στις θέσεις $j = 0, 1, \ldots, n-i-1$ συγκρίνονται.
  - Εάν είναι σε φθίνουσα τάξη, τα εναλλάσουμε.
- Η i-οστή επανάληψη απαιτεί $n-i$ συγκρίσεις.
  - Συνολικά:
    $(n-1) + \ldots + 1 = \frac{n(n-1)}{2} = \mathcal{O}(n^2)$ συγκρίσεις

```cpp
#include <iostream>
#include "auxiliary.cpp"
using namespace std;


void bubbleSort(int a[], int n) {
  for (int i=0; i < n-1; i++) {
    // Last i elements are already sorted
    for (int j=0; j < n-i-1; j++) {
      if (a[j] > a[j+1])
        swap(a[j], a[j+1]);
    }
  }
}

int main() {
  const int SIZE = 10;
  int a[SIZE];
  initialize(a, SIZE);
  bubbleSort(a, SIZE);
  cout << "After sorting: ";
  display(a, SIZE);
}
```

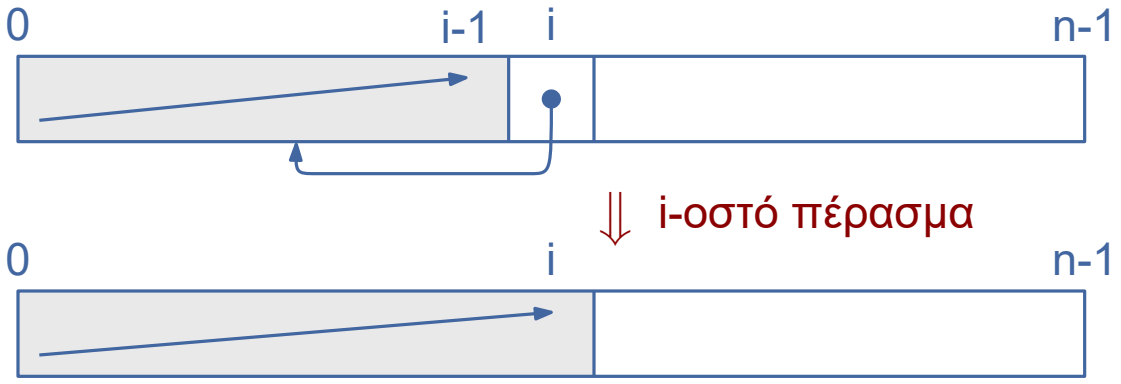
Part03/sorting/bubblesort.cpp

Ταξινόμηση εισαγωγής - Insertion sort

- Κάνει $n-1$ περάσματα πάνω από τον πίνακα: $i = 1, 2, \ldots, n-1$

- Κατά το i-οστό πέρασμα, το i-οστό στοιχείο εισάγεται στη σωστή θέση ως προς τα στοιχεία στις θέσεις $j = 0, 1, \ldots, i-1$.

- Οπτικοποίηση:




```
#include <iostream>
#include "auxiliary.cpp"
using namespace std;

void insertionSort(int a[], int n) {
  int i, j, key;
  for (i=1; i<n; i++) {
    key = a[i];
    for (j=i; j>0 && a[j-1] > key; j--) {
      // Shift to the right
      a[j] = a[j-1];
    }
    a[j] = key;
  }
}

int main() {
  const int SIZE = 10;
  int a[SIZE];
  initialize(a, SIZE);
  insertionSort(a, SIZE);
  cout << "After sorting: ";
  display(a, SIZE);
}
```

Part03/sorting/insertionsort.cpp

Ταξινόμηση εισαγωγής - Insertion sort

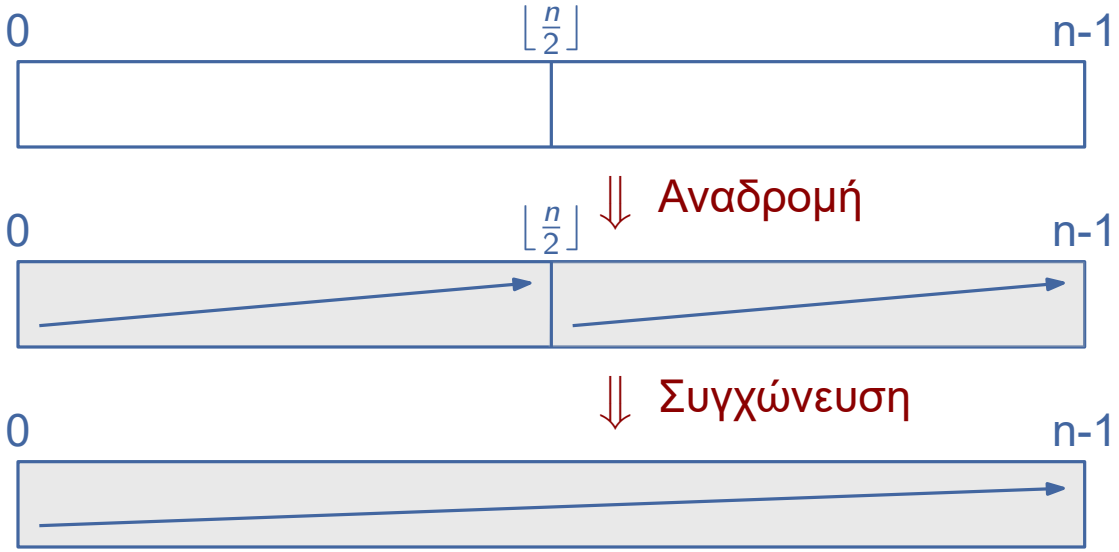
- Κάνει $n-1$ περάσματα πάνω από τον πίνακα: $i = 1, 2, \ldots, n-1$
- Κατά το i-οστό πέρασμα, το i-οστό στοιχείο εισάγεται στη σωστή θέση ως προς τα στοιχεία στις θέσεις $j = 0, 1, \ldots, i-1$.
- Το i-οστό στοιχείο συγκρίνεται με όλα τα $i-1$ στοιχεία που προηγούνται .
  - Συνολικά:
    $1 + 2 + \ldots + (n-1) = \frac{n(n-1)}{2} = \mathcal{O}(n^2)$ συγκρίσεις

```cpp
#include <iostream>
#include "auxiliary.cpp"
using namespace std;

void insertionSort(int a[], int n) {
  int i, j, key;
  for (i=1; i<n; i++) {
    key = a[i];
    for (j=i; j>0 && a[j-1] > key; j--) {
      // Shift to the right
      a[j] = a[j-1];
    }
    a[j] = key;
  }
}

int main() {
  const int SIZE = 10;
  int a[SIZE];
  initialize(a, SIZE);
  insertionSort(a, SIZE);
  cout << "After sorting: ";
  display(a, SIZE);
}
```

Part03/sorting/insertionsort.cpp

Ταξινόμηση μέσω συγχώνευσης - Merge sort

- Διαμερίζει τον πίνακα σε δύο (σχεδόν) ίσα μέρη διαδοχικών στοιχείων και αναδρομικά ταξινομεί καθένα από αυτά.
- Ακολούθως, συγχωνεύει τα δύο ταξινομημένα μέρη (`merge`).
- Οπτικοποίηση:

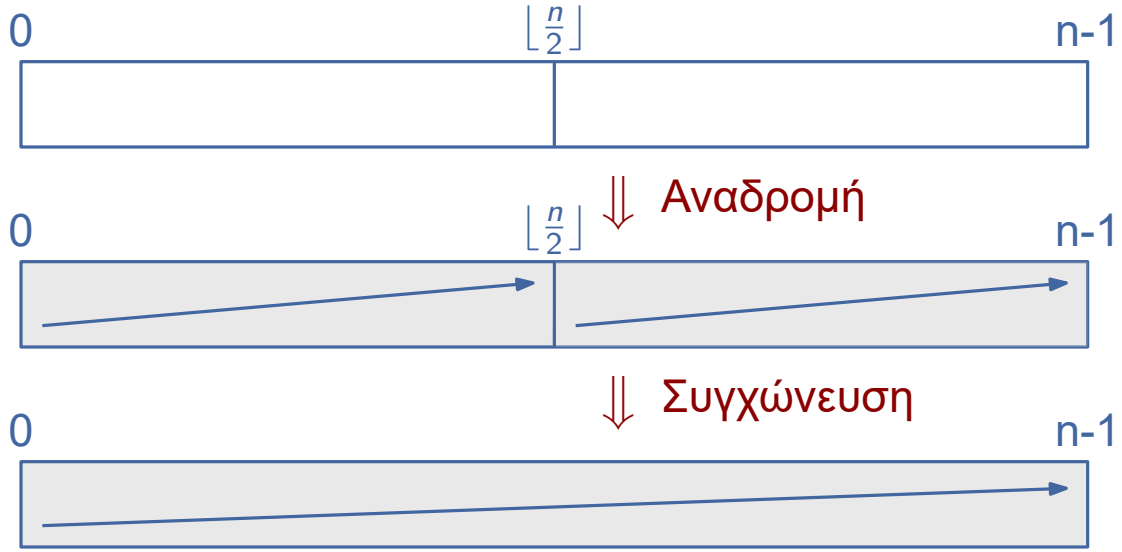



```
#include <iostream>
#include "auxiliary.cpp"
using namespace std;

void mergeSort(int a[], int left,
                        int right)
{
  if(left >= right){
    return;
  }
  int middle = left + (right-left)/2;
  mergeSort(a, left, middle);
  mergeSort(a, middle+1, right);
  merge(a, left, middle, right);
}

int main() {
  const int SIZE = 10;
  int a[SIZE];
  initialize(a, SIZE);
  mergeSort(a, 0, SIZE-1);
  cout << "After sorting: ";
  display(a, SIZE);
}
```

Part03/sorting/mergesort.cpp

Ταξινόμηση μέσω συγχώνευσης - Merge sort

- Διαμερίζει τον πίνακα σε δύο (σχεδόν) ίσα μέρη διαδοχικών στοιχείων και αναδρομικά ταξινομεί καθένα από αυτά.
- Ακολούθως, συγχωνεύει τα δύο ταξινομημένα μέρη (`merge`).
- Οπτικοποίηση:

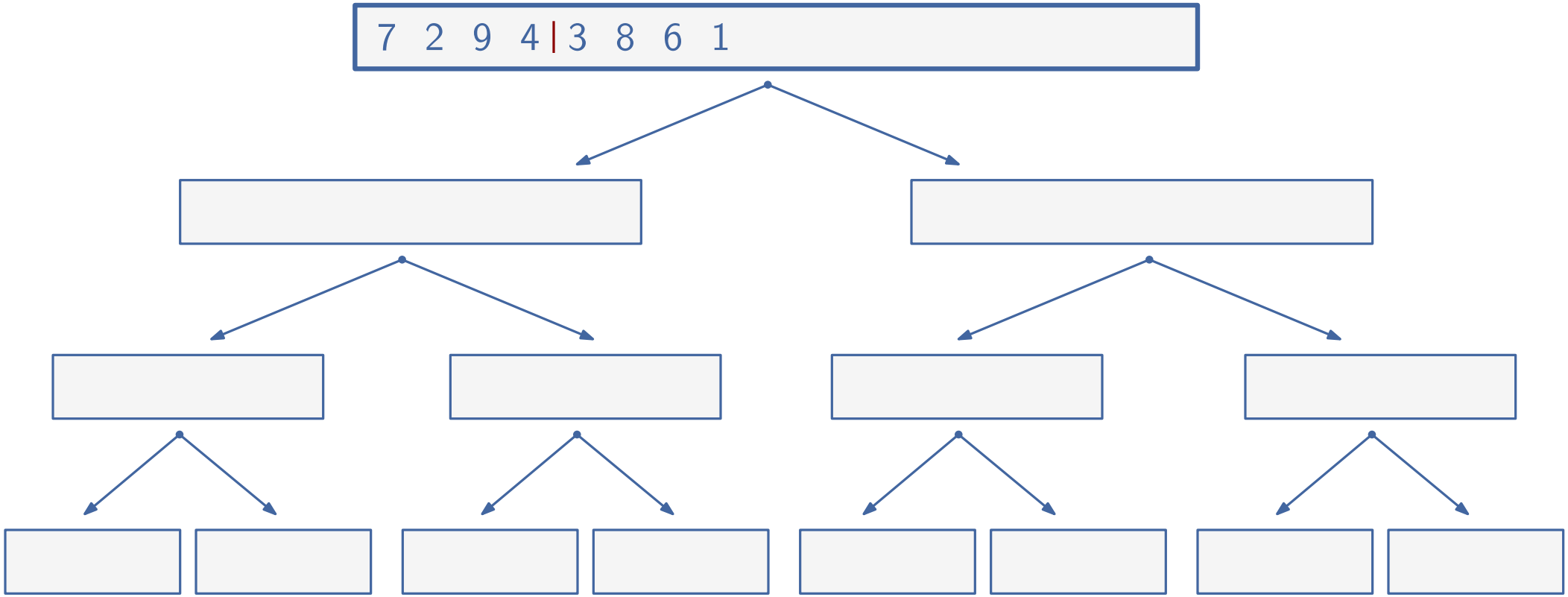



```
#include <iostream>
#include "auxiliary.cpp"
using namespace std;

void mergeSort(int a[], int left,
                        int right)
{
  if(left >= right){
    return;
  }
  int middle = left + (right-left)/2;
  mergeSort(a, left, middle);
  mergeSort(a, middle+1, right);
  merge(a, left, middle, right);
}

int main() {
  const int SIZE = 10;
  int a[SIZE];
  initialize(a, SIZE);
  mergeSort(a, 0, SIZE-1);
  cout << "After sorting: ";
  display(a, SIZE);
}
```

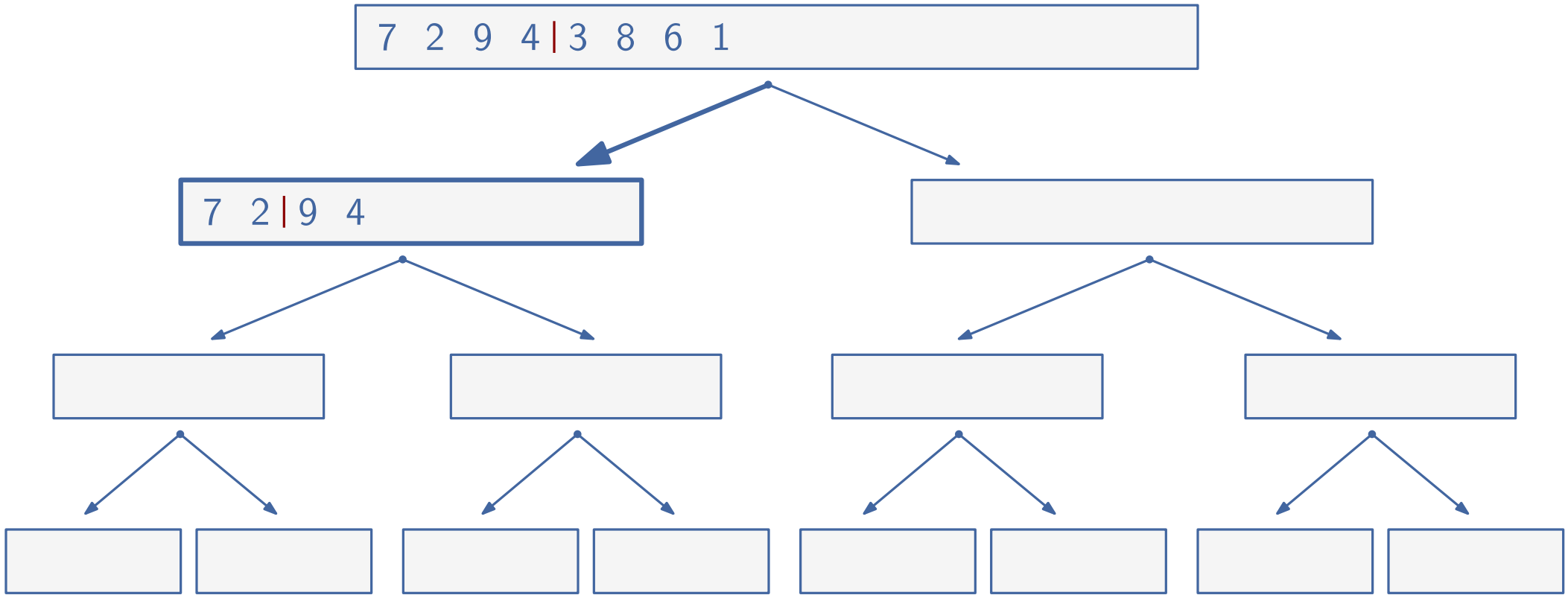
Part03/sorting/mergesort.cpp

Θα δούμε την υλοποίηση της σε λίγο

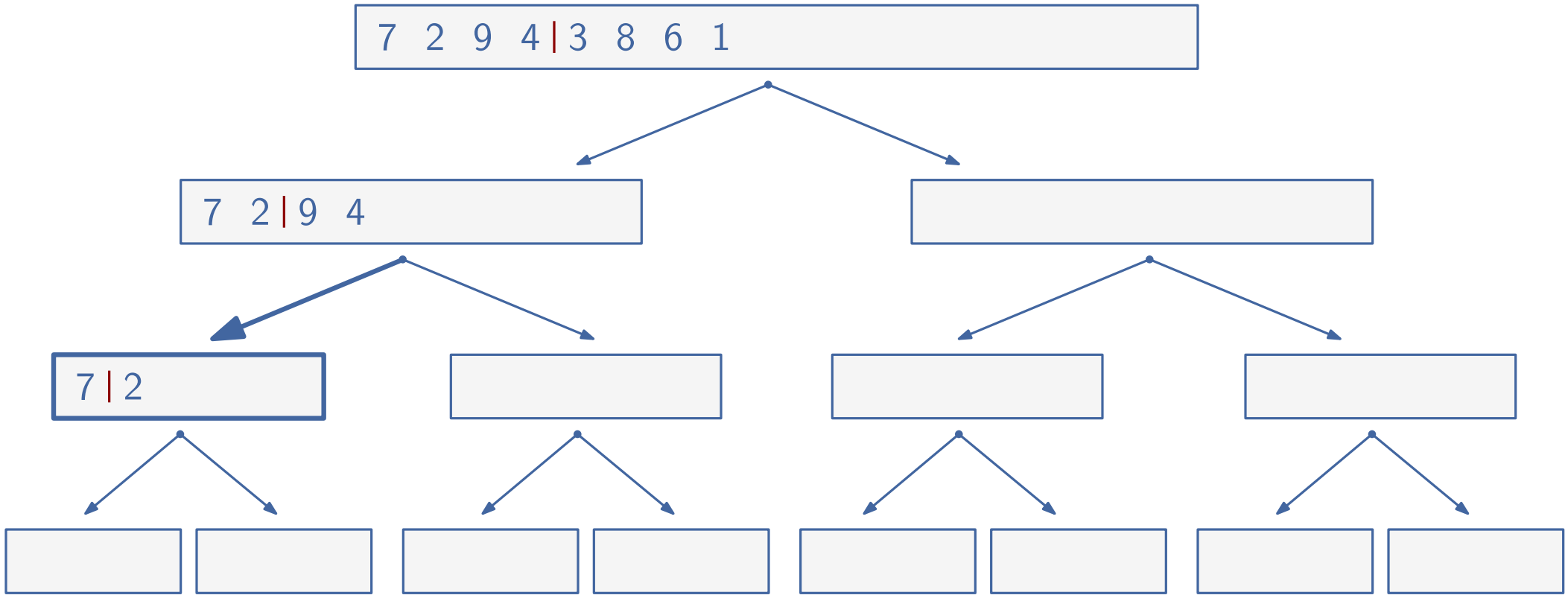
Ταξινόμηση μέσω συγχώνευσης: Παράδειγμα εκτέλεσης



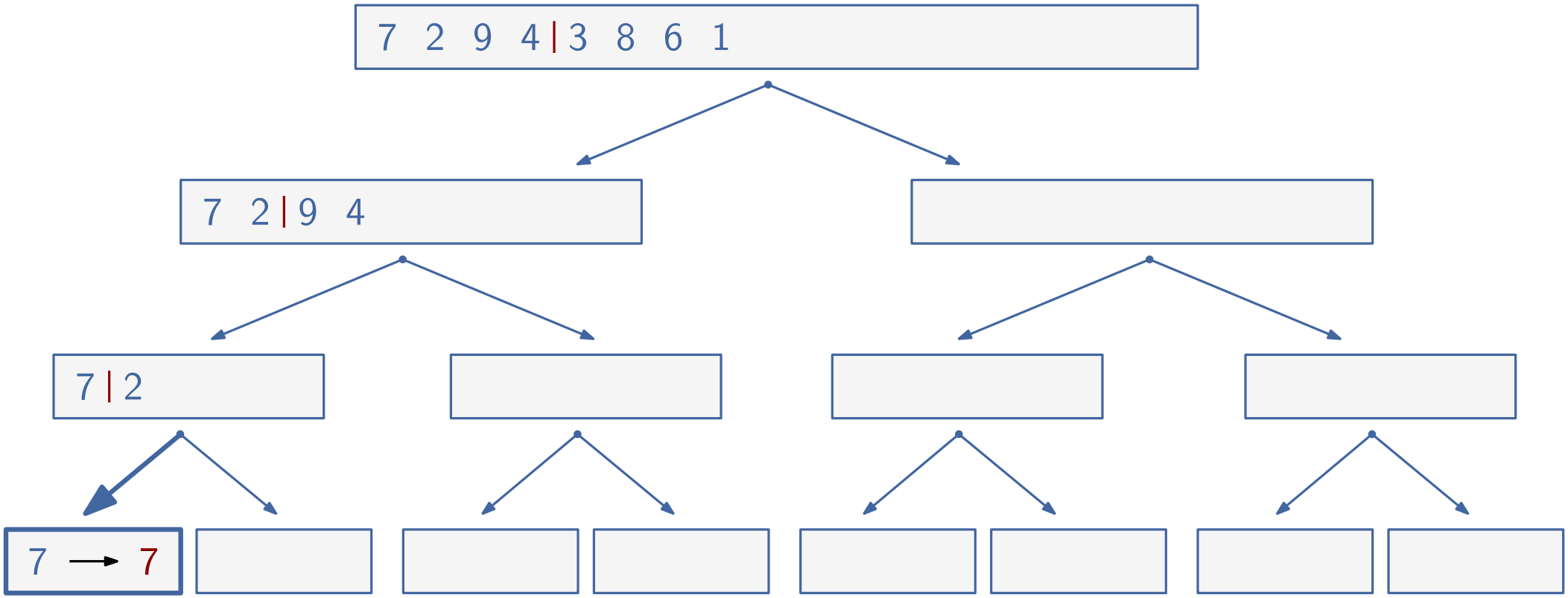
Ταξινόμηση μέσω συγχώνευσης: Παράδειγμα εκτέλεσης



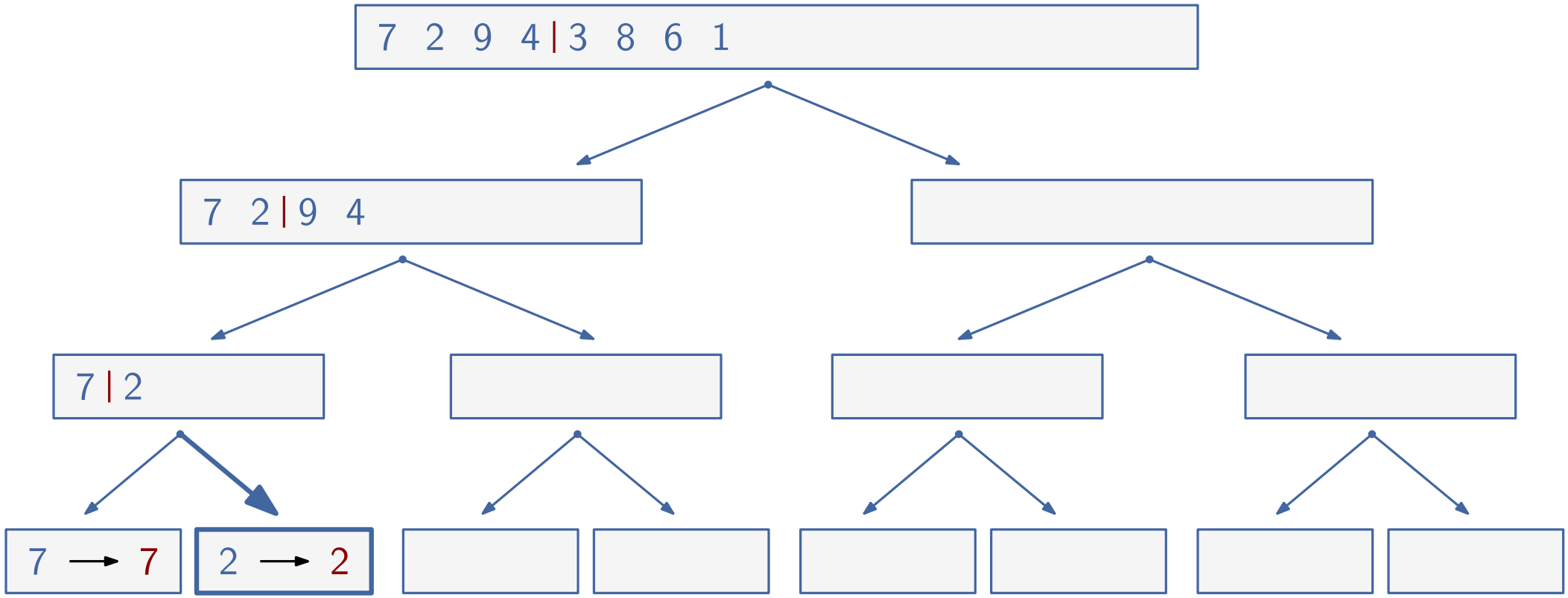
Ταξινόμηση μέσω συγχώνευσης: Παράδειγμα εκτέλεσης



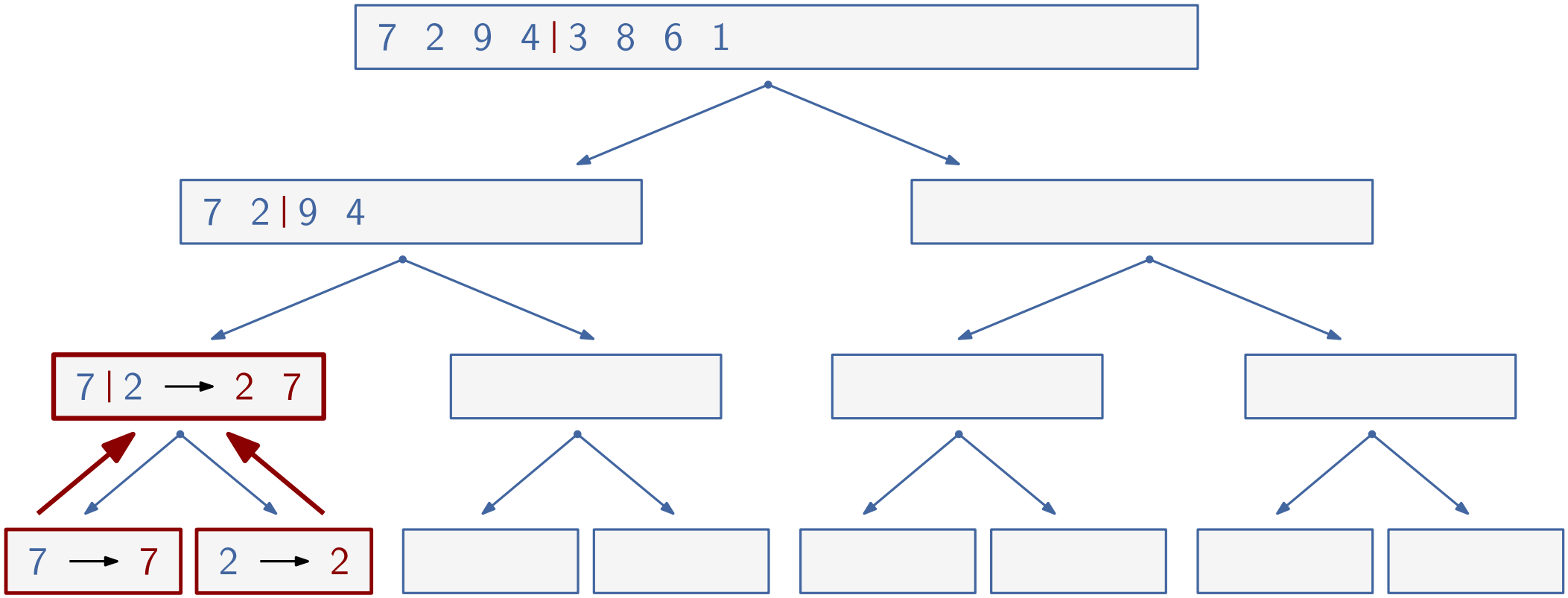
Ταξινόμηση μέσω συγχώνευσης: Παράδειγμα εκτέλεσης



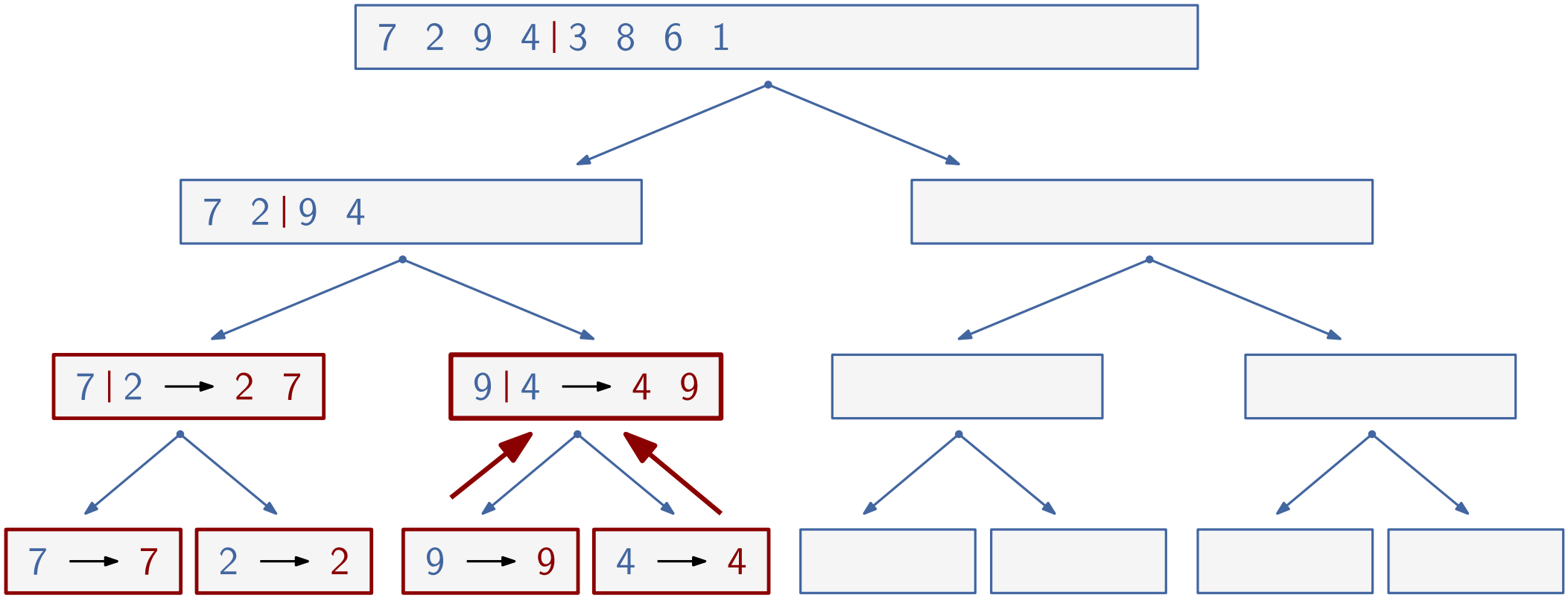
Ταξινόμηση μέσω συγχώνευσης: Παράδειγμα εκτέλεσης



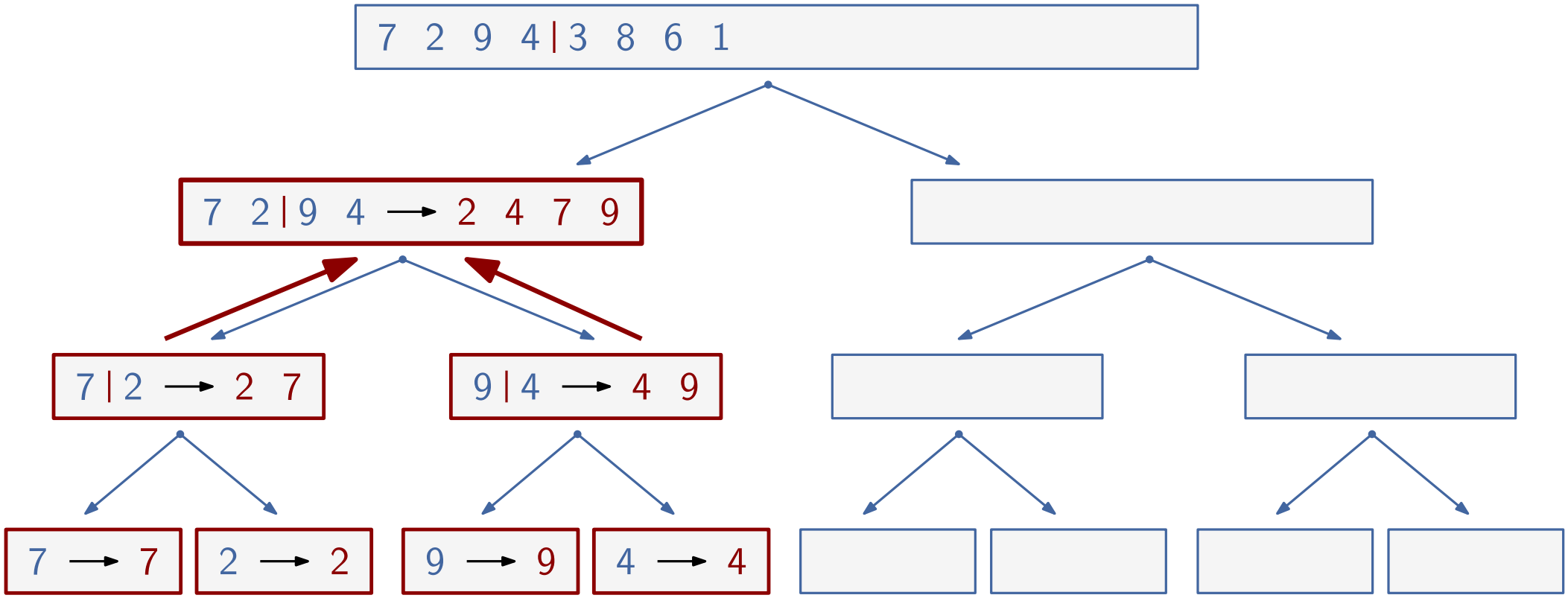
Ταξινόμηση μέσω συγχώνευσης: Παράδειγμα εκτέλεσης



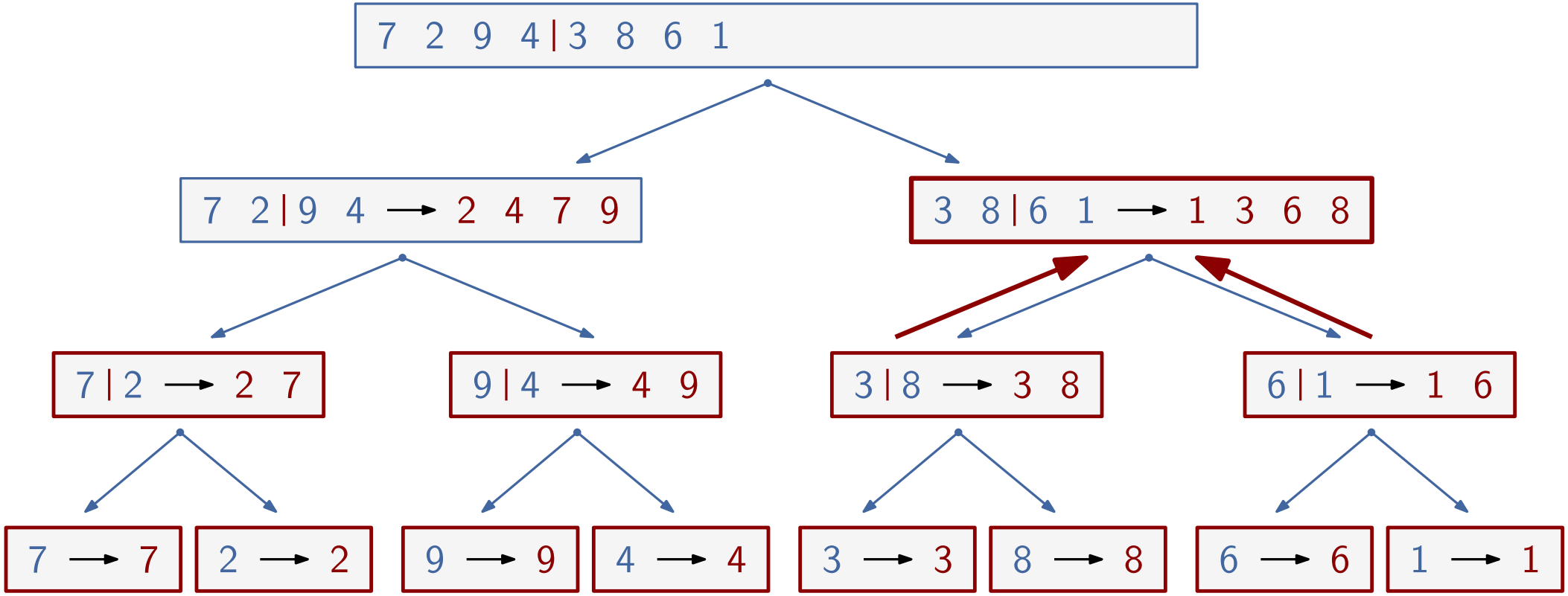
Ταξινόμηση μέσω συγχώνευσης: Παράδειγμα εκτέλεσης



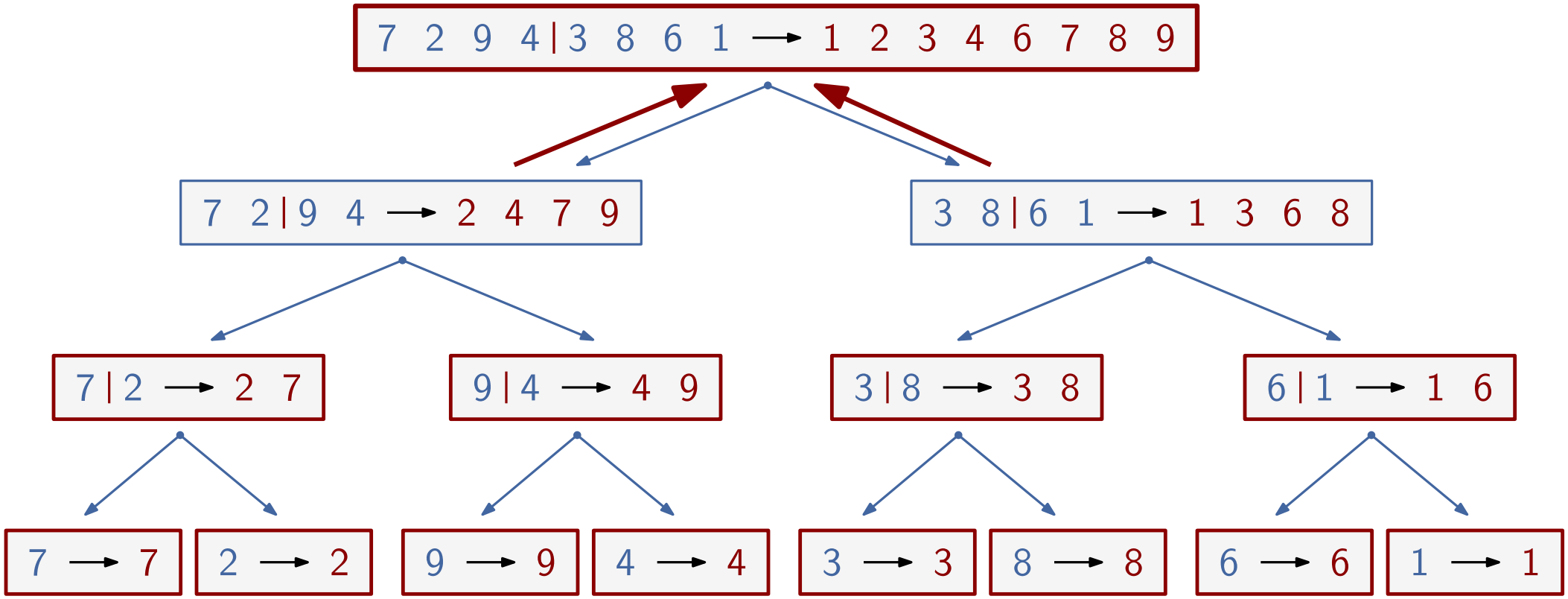
Ταξινόμηση μέσω συγχώνευσης: Παράδειγμα εκτέλεσης



Ταξινόμηση μέσω συγχώνευσης: Παράδειγμα εκτέλεσης

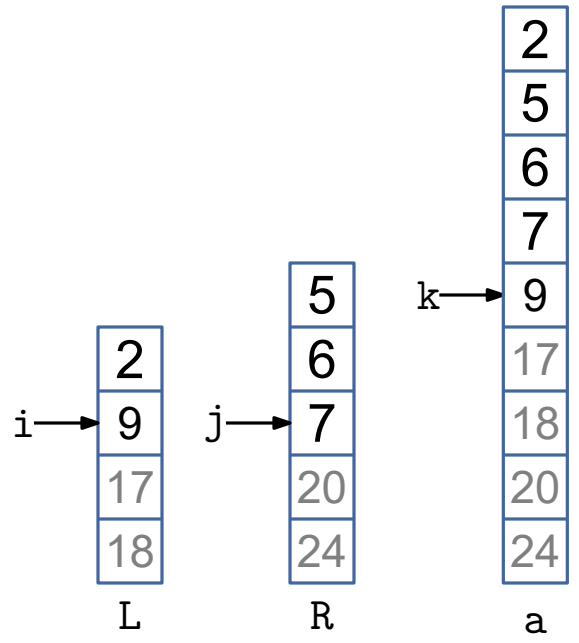


Ταξινόμηση μέσω συγχώνευσης: Παράδειγμα εκτέλεσης



Ταξινόμηση μέσω συγχώνευσης: Merge

```
void merge(int a[], int left, int middle, int right)
{
    // Create two temporary arrays
    int l = middle - left + 1, r = right - middle;
    int L[l], R[r];

    // Copy data to temporary arrays L and R
    for (int i = 0; i < l; i++) L[i] = a[left + i];
    for (int j = 0; j < r; j++) R[j] = a[middle + 1 + j];

    // Merge the L and R back into array[l...r]
    int i = 0, j = 0, k = left;
    while (i < l && j < r) {
        if (L[i] <= R[j]) { a[k] = L[i]; i++; k++;}
        else { a[k] = R[j]; j++; k++; }
    }

    // Copy the remaining elements of L and R, if any
    while (i < l) { a[k] = L[i]; i++; k++; }
    while (j < r) { a[k] = R[j]; j++; k++; }
}
```



Ταξινόμηση μέσω συγχώνευσης: Ανάλυση

- Θεώρημα: Το πλήθος συγκρίσεων της ταξινόμησης μέσω συγχώνευσης δίνεται από την αναδρομική σχέση:

$$T[n] \leq T[\lfloor \tfrac{n}{2} \rfloor] + T[\lceil \tfrac{n}{2} \rceil] + n - 1 \qquad T[1] = 0$$

η οποία ως έχει λύση:

$$T[n] = n\lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1 = \mathcal{O}(n \log n)$$

Απόδειξη:

Υποθέτουμε ότι το n είναι δύναμη του 2

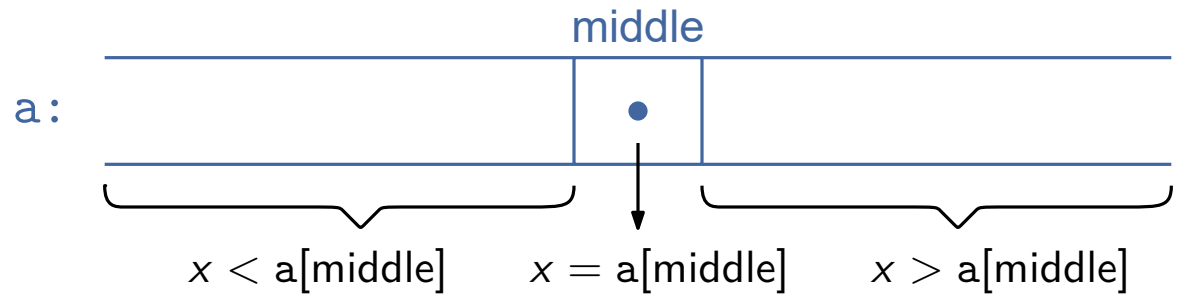
$\Rightarrow n = 2^k \iff k = \log n$

$$\begin{array}{rcl}
T[n] & \leq & 2\ T[\frac{n}{2}] + \quad n - 1 \\
2T[\frac{n}{2}] & \leq & 2^2 T[\frac{n}{4}] + 2\ \frac{n}{2} - 2 \\
\cdots & & \\
2^{k-1} T[\frac{n}{2^{k-1}}] & \leq & 2^k T[\frac{n}{2^k}] + 2^{k-1} \frac{n}{2^{k-1}} - 2^{k-1} \\
\hline
T[n] & \leq & 2^k T[\frac{n}{2^k}] + nk - \sum_{j=0}^{k-1} 2^j \\
 & = & n \log n - (2^k - 1) \\
 & = & n \log n - n + 1 = \mathcal{O}(n \log n)
\end{array}$$

Μέρος 8°:
Δυαδική αναζήτηση

Δυαδική αναζήτηση

- Αναζήτηση ενός στοιχείου `x` σε ταξινομημένο πίνακα.
- Μέθοδος: Σύγκρινε το στοιχείο x με το μεσαίο στοιχείο του πίνακα
  - $=$: το στοιχείο βρέθηκε
  - $<$: αναζήτησε το στοιχείο x στα αριστερά
  - $>$: αναζήτησε το στοιχείο x στα δεξια
- Οπτικοποίηση:



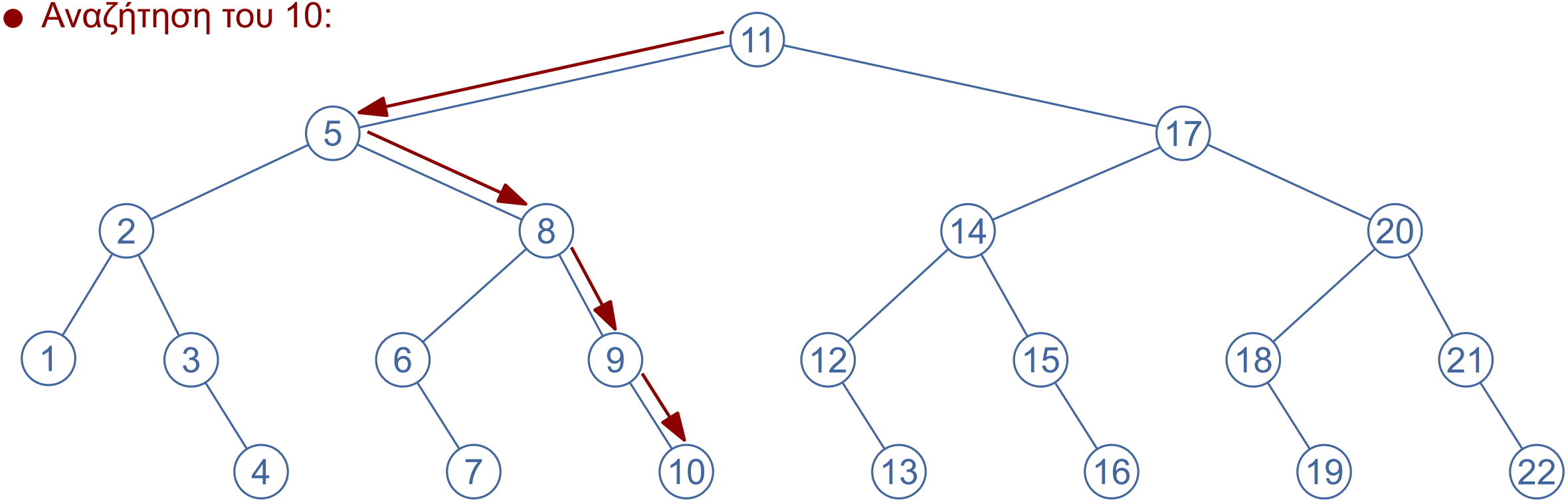

```
#include <iostream>
#include "auxiliary.cpp"
using namespace std;

int bsearch(int a[], int left,
                     int right, int x) {
  if (right >= left) {
    int middle = left+(right-left)/2;

    if (a[middle] == x)
      return middle;
    if (a[middle] > x)
      return bsearch(a,left,middle-1,x);
    else
      return bsearch(a,middle+1,right,x);
    }
    return -1;
}
int main() {
  const int SIZE = 10;
  int a[SIZE];
  initialize(a, SIZE);
  cout << "Element 10 found at position "
       << bsearch(a, 0, SIZE-1, 10);
}
```

Part03/sorting/binarysearch.cpp

Δυαδική αναζήτηση: Παράδειγμα

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

- Αναζήτηση του 10:



Δυαδική αναζήτηση: Ανάλυση

- Θεώρημα: Το πλήθος των συγκρίσεων της μεθόδου binarySearch δίνεται από την αναδρομική σχέση:

$$T[n] \leq T[\lfloor \tfrac{n}{2} \rfloor] + 1 \qquad T[1] = 1$$

η οποία ως έχει λύση:

$$T[n] \leq \lfloor \log n \rfloor + 1 = \mathcal{O}(\log n)$$

Απόδειξη:

Υποθέτουμε ότι το n είναι δύναμη του 2

$\Rightarrow n = 2^k \iff k = \log n$

$$T[n] \leq T[\tfrac{n}{2}] + 1$$
$$T[\tfrac{n}{2}] \leq T[\tfrac{n}{4}] + 1$$
$$\cdots$$
$$T[\tfrac{n}{2^{k-1}}] \leq T[\tfrac{n}{2^k}] + 1$$

$$T[n] \leq T[\tfrac{n}{2^k}] + k = 1 + \log n = \mathcal{O}(\log n)$$

Επιπλέον υλικό

- Κεφάλαιο 7 (σ.σ., 290–305):
  R. Lafore, Αντικειμενοστρεφής προγραμματισμός με τη C++,
  ISBN: 960-209-904-6, εκδόσεις Κλειδάριθμος, 2006.