ΗΥ-150
Προγραμματισμός

Ταξινόμηση και Αναζήτηση

Το πρόβλημα της Αναζήτησης

- Δοθέντος δεδομένων, λ.χ. σε Πίνακα (P)
- Ψάχνω να βρω κάποιο συγκεκριμένο στοιχείο (key)
- Αν ο πίνακας δεν είναι ταξινομημένος
  - Γραμμική Αναζήτηση (Linear search)
    - Απλούστερη δυνατή
    - Σύγκρινε σειριακά κάθε στοιχείο του πίνακα με την τιμή-κλειδί
    - Χρήσιμο για μικρούς και ΜΗ ταξινομημένους πίνακες

```
int linearSearch(int P[],int apo,int eos,int key)
{
   int i;

   for (i = apo; i <= eos; ++i)
   {
        if (P[i] == key)
            return i;
   }
   return -1;
}
```

Το πρόβλημα της Αναζήτησης

- Αν ο πίνακας είναι ταξινομημένος
  - λ.χ. τηλεφωνικός κατάλογος
  - Μπορώ να κάνω πολύ πιο γρήγορα την αναζήτηση
- Δυαδική Αναζήτηση (Binary Search)
  - Συγκρίνει το P[**`middle`**] στοιχείο με το ζητούμενο **`key`**
    - Αν είναι ίσα βρέθηκε
    - Αν **`key < P[middle]`**, ψάχνει στο 1ο μισό του πίνακα
    - Αν **`key > P[middle]`**, ψάχνει στο 2ο μισό του πίνακα
    - Επανάληψη
  - Πολύ γρήγορη – χειρότερη περίπτωση $\log_2(N)$, N # στοιχείων πίνακα
    - Πίνακας 100 στοιχείων χρειάζεται το πολύ 7 βήματα
    - Πίνακας 100.000 στοιχείων χρειάζεται το πολύ 20 βήματα
    - Πίνακας 100.000.000 στοιχείων χρειάζεται το πολύ 27 βήματα

Binary Search – Υλοποίηση με επανάληψη

```
int binaryLoopSearch(int p[], int searchkey, int low, int high)
{
   int middle;
   while ( low <= high )
   {
        middle = (low + high ) / 2;

        if (searchkey == p[middle])
               return middle;
        else if (searchkey < p[middle] )
               high = middle - 1;
        else
               low = middle + 1;
   }
   return -1;
}
```

Binary Search – Υλοποίηση με αναδρομή

```
int binarySearch(int p[], int searchkey, int low, int high)
{
   int middle;

   middle = (low + high ) / 2;
    if (high < low)
        return -1;
    if (searchkey == p[middle])
        return middle;
   else if (searchkey < p[middle])
        return binarySearch(p, searchkey, low, middle-1);
   else
        return binarySearch(p, searchkey, middle+1,high);

   return -1;
}
```

Το πρόβλημα της Ταξινόμησης

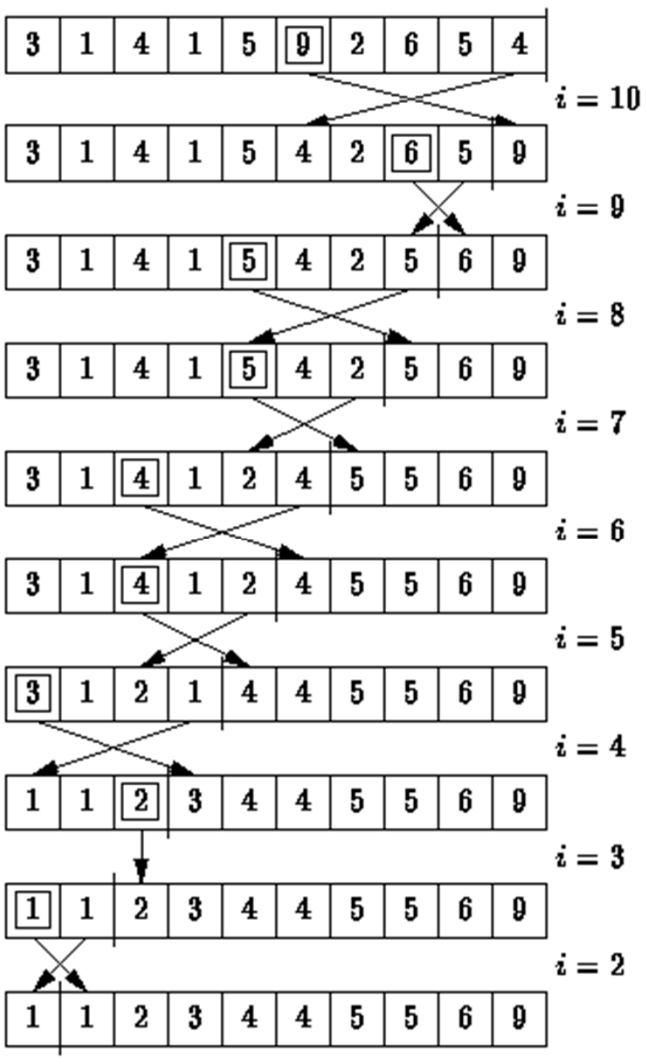
- Δοθέντος δεδομένων, λ.χ. σε Πίνακα (P)

- Να γίνει αναδιάταξη των στοιχείων του ώστε να βρεθούν σε αύξουσα (ή φθίνουσα) σειρά, δηλαδή ταξινομημένα

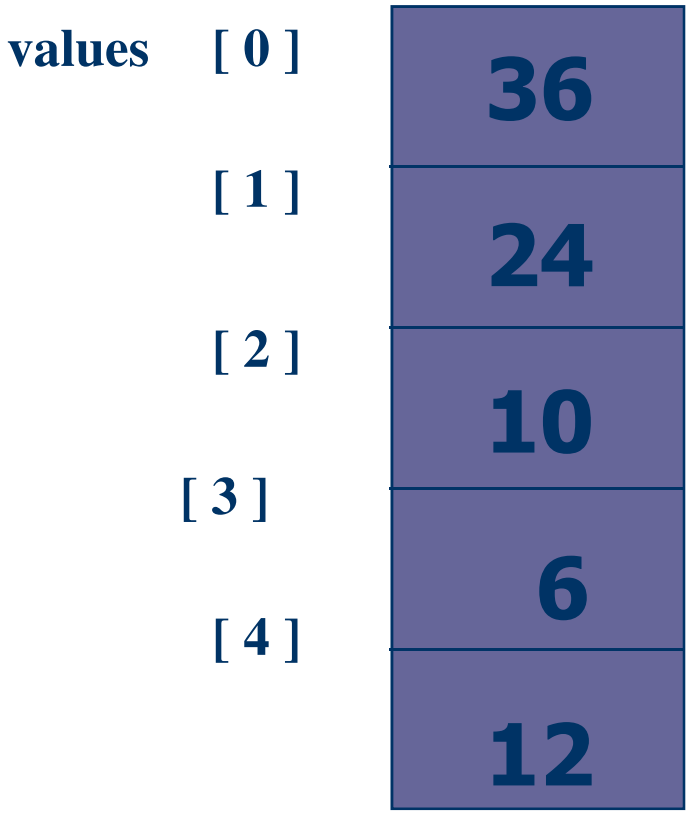
12	31	4	32	134	13	42	1	43	2
1	2	4	12	13	31	32	42	43	134

- Λύση : Υπάρχουν διάφοροι αλγόριθμοι που το πετυχαίνουν και διαφοροποιούνται στο υπολογιστικό
  - Straight Selection Sort - Bubble Sort : $O(N^2)$
  - Quick Sort : O(N log N)

Ταξινόμηση με επιλογή



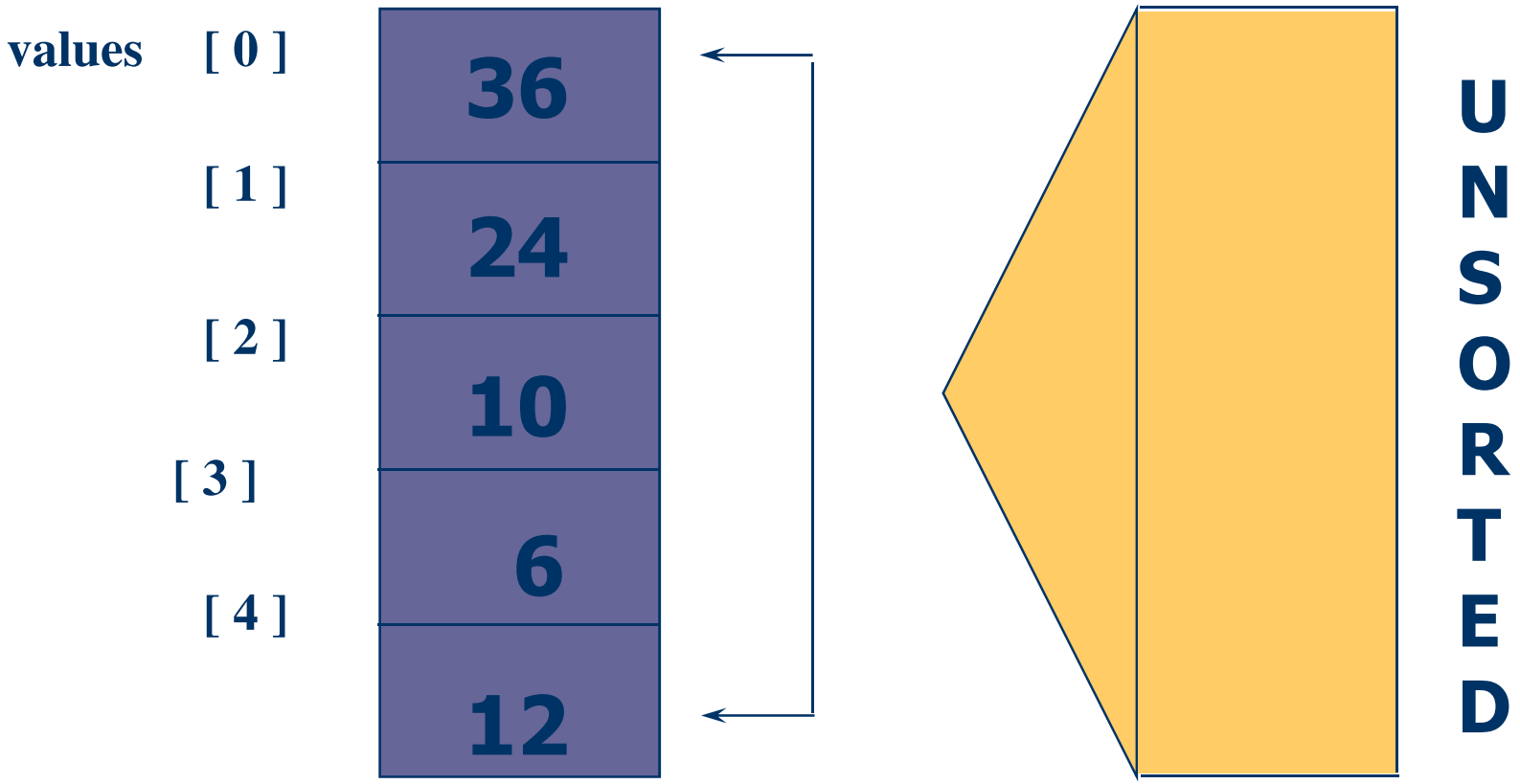
Ταξινόμηση με επιλογή



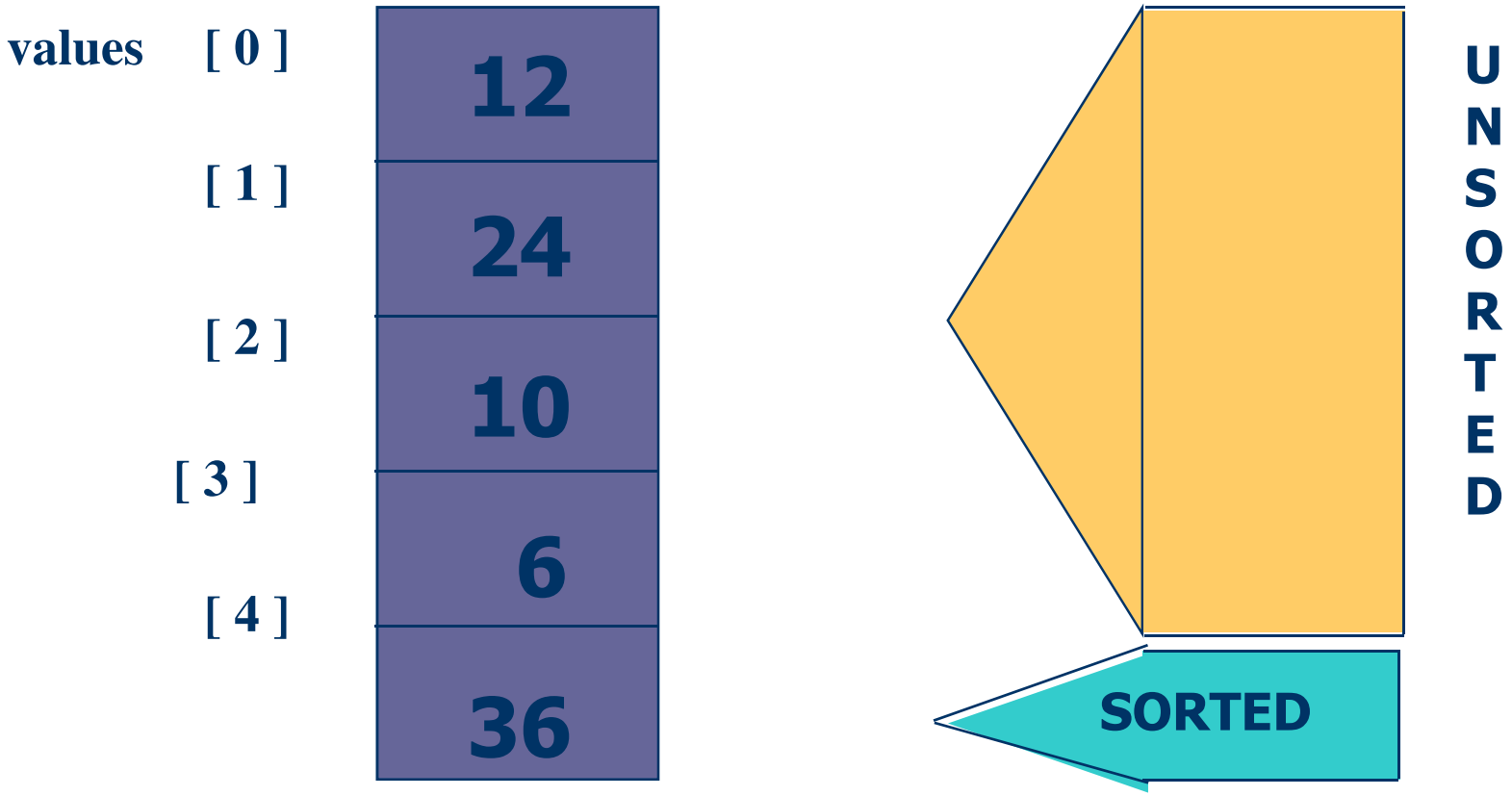
Χωρίζει νοητά τον πίνακα σε 2 μέρη:
- **Ταξινομημένο**
- **Μη ταξινομημένο**

Σε κάθε επανάληψη βρίσκει το μεγαλύτερο στοιχείο και το ανταλλάσει με το πρώτο του αταξινόμητου υποπίνακα.

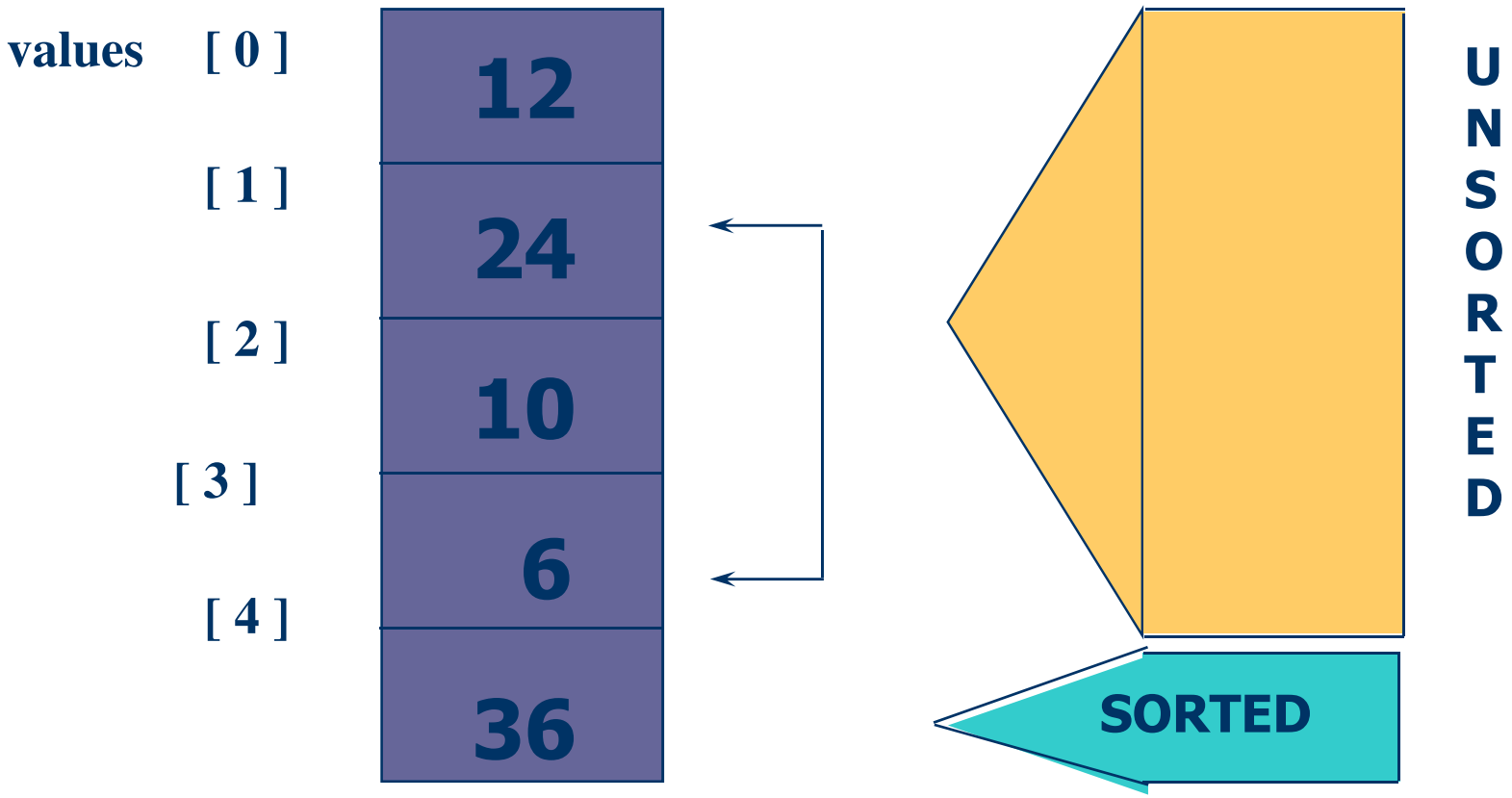
Ταξινόμηση με επιλογή



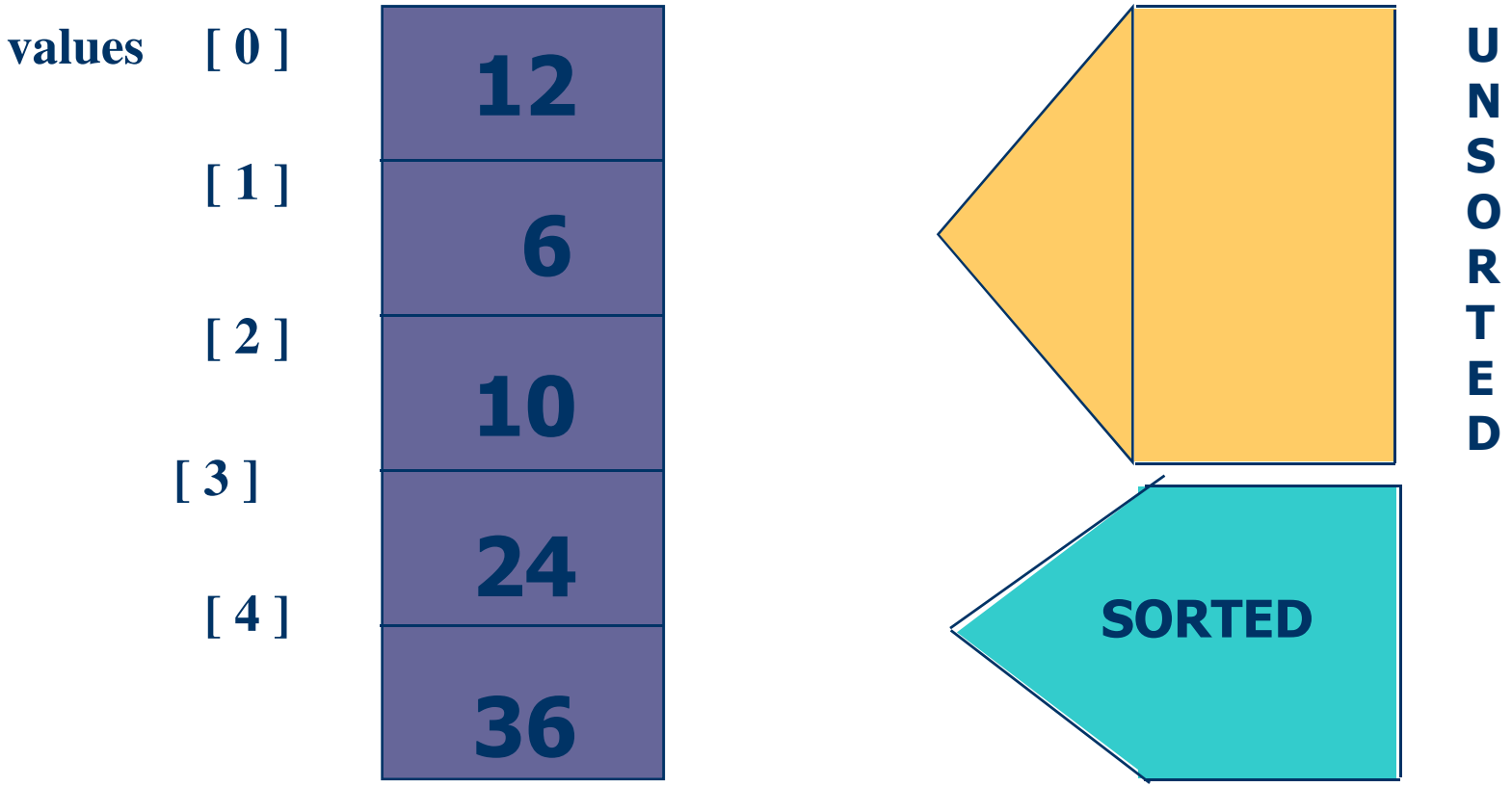
Ταξινόμηση με επιλογή



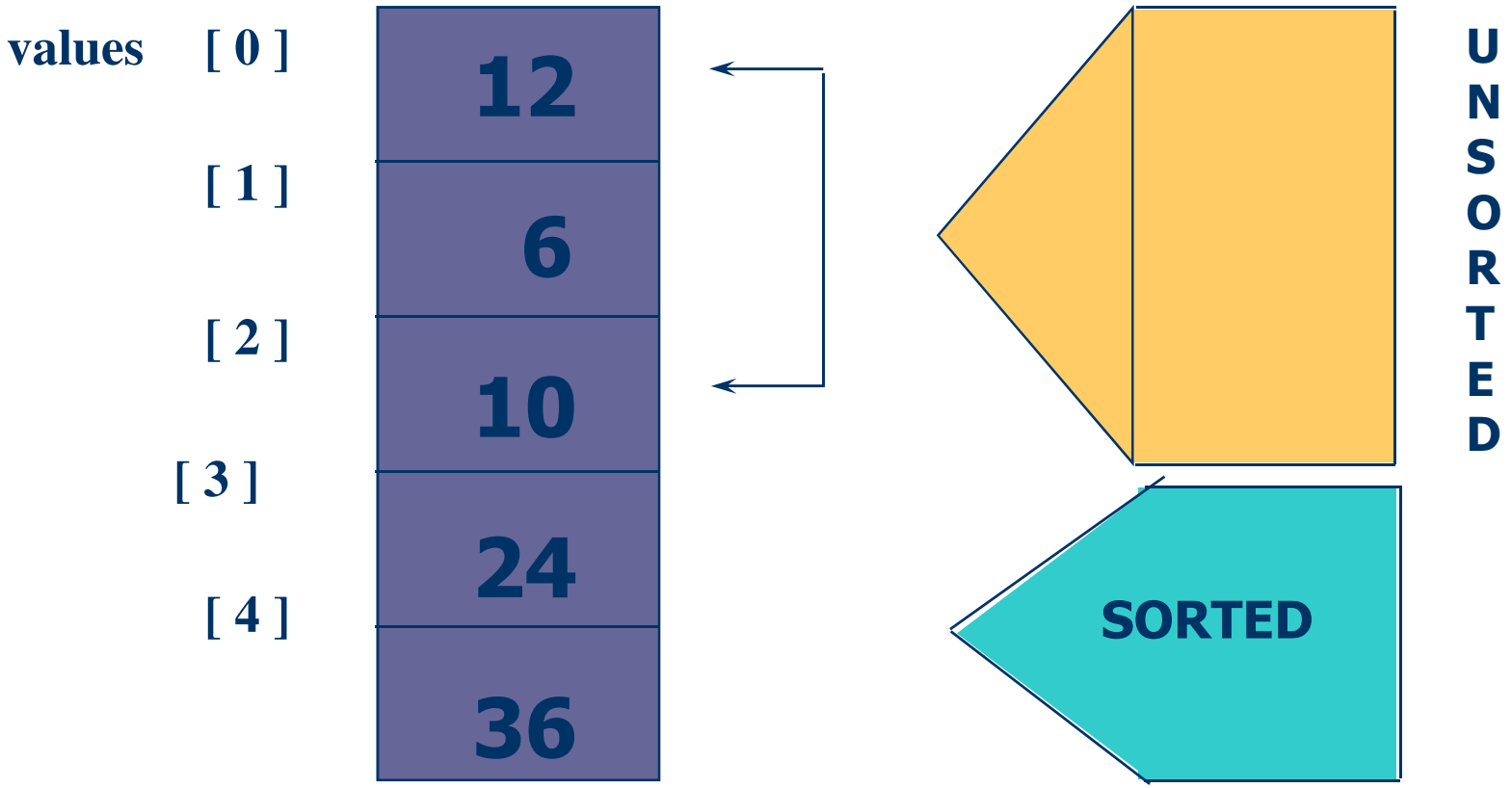
Ταξινόμηση με επιλογή



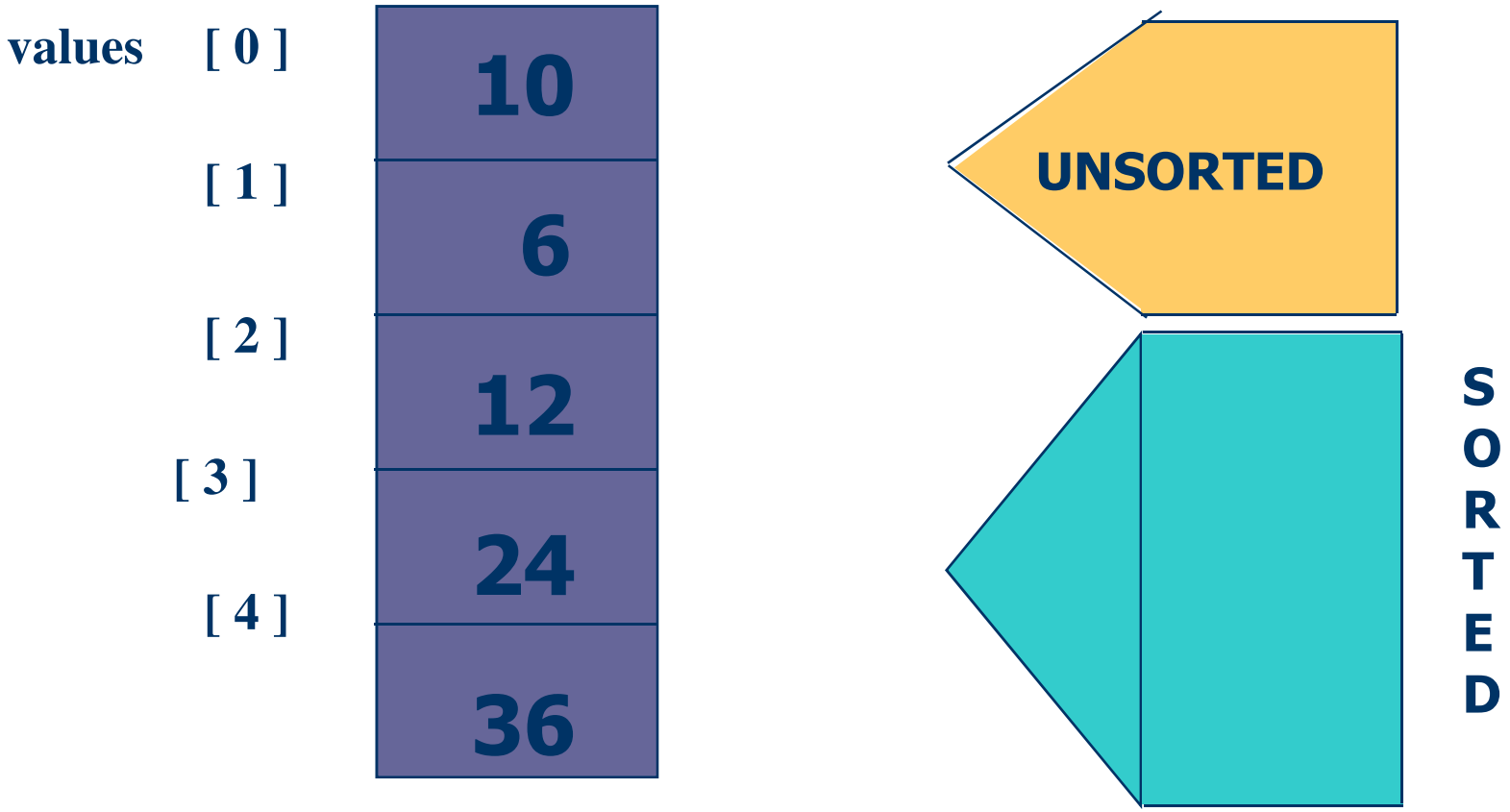
Ταξινόμηση με επιλογή



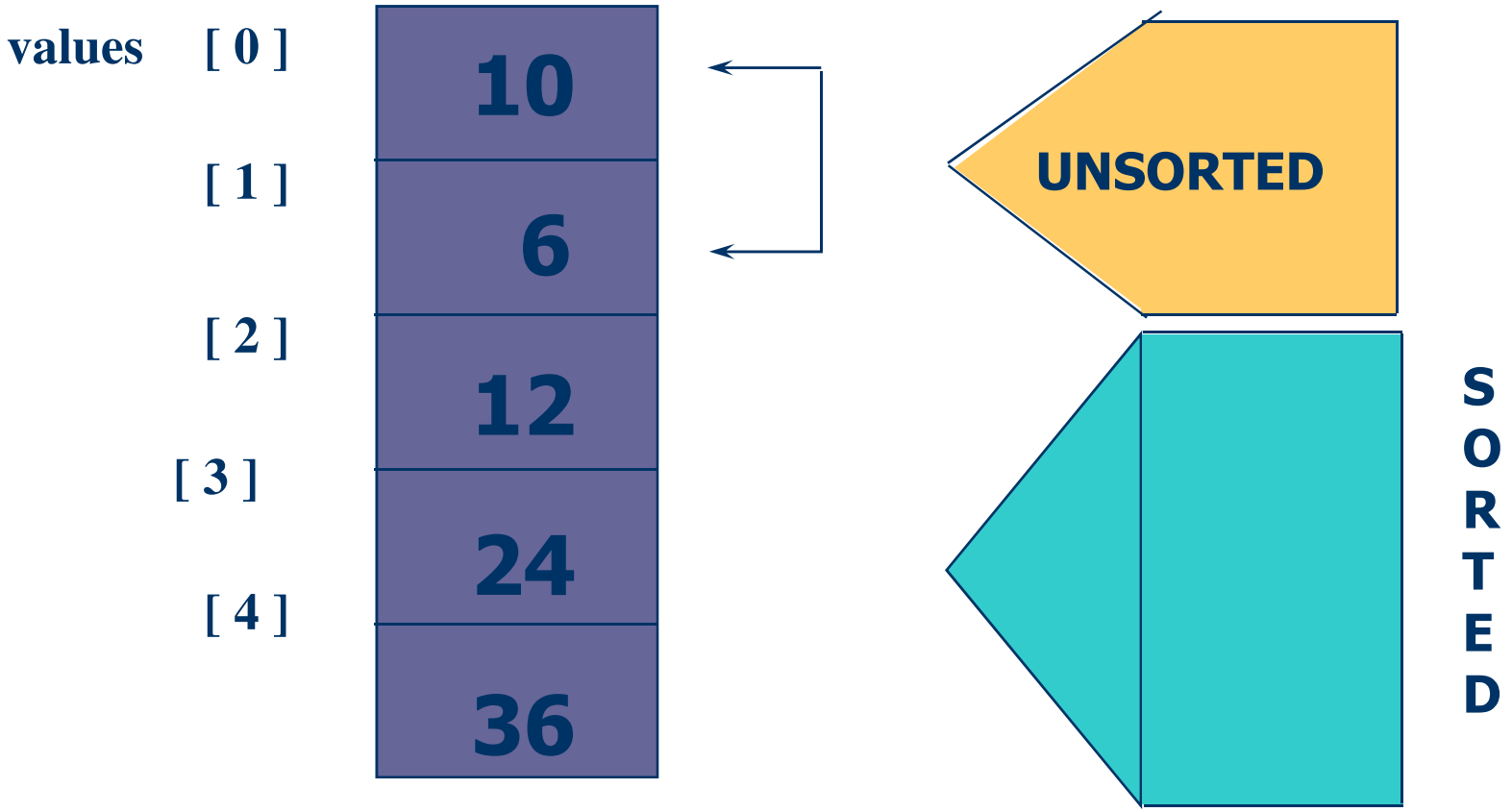
Ταξινόμηση με επιλογή



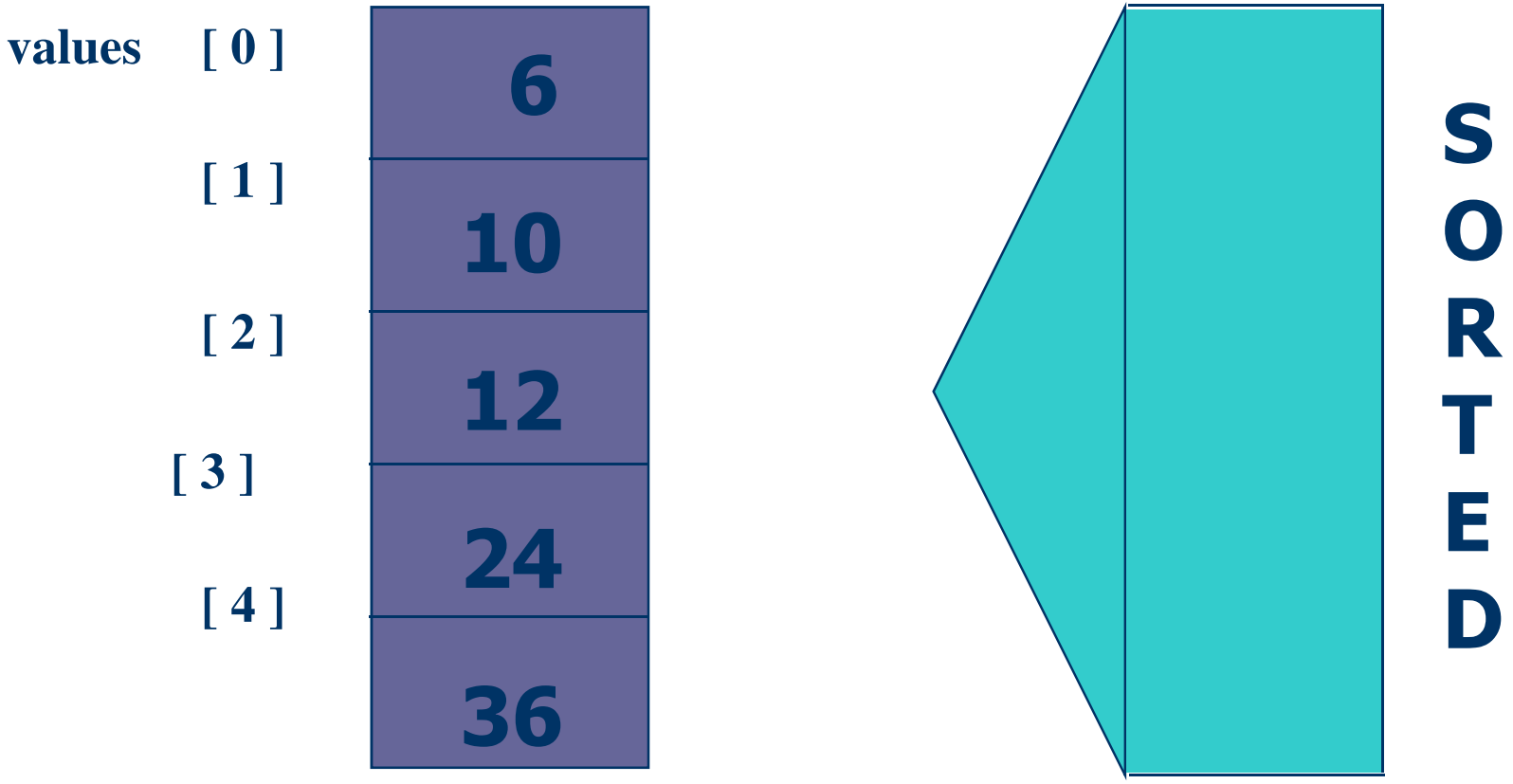
Ταξινόμηση με επιλογή



Ταξινόμηση με επιλογή

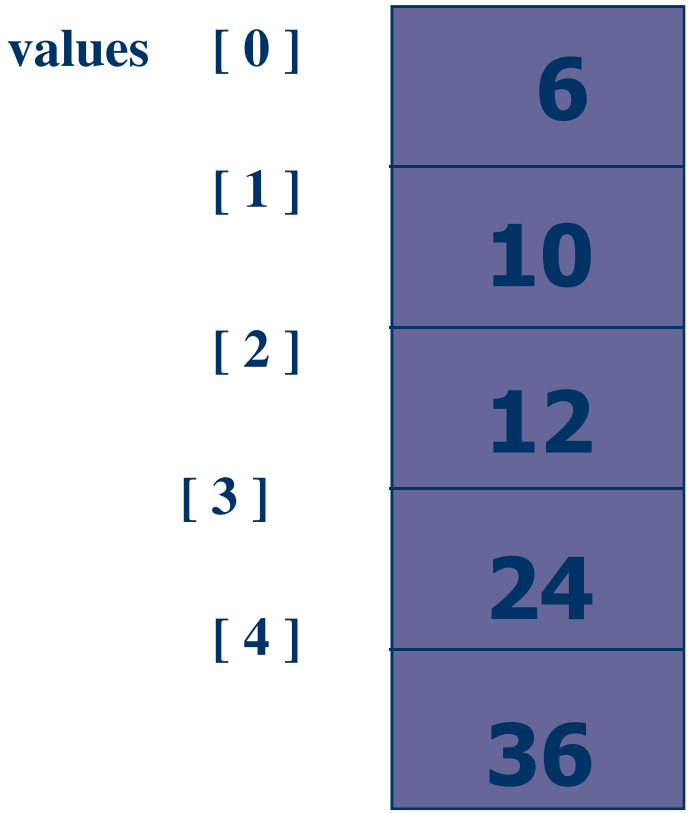


Ταξινόμηση με επιλογή



Ταξινόμηση με επιλογή:

Απαρίθμηση συγκρίσεων (υπολογιστικό κόστος)



4 compares for values[4]

3 compares for values[3]

2 compares for values[2]

1 compare for values[1]

= 4 + 3 + 2 + 1

QSort: Ψευδοκώδικας

- QSort(Πίνακας Π, δείκτης Α, δείκτης Τ)
  Ταξινομεί τον υποπίνακα Π[Α] έως Π[Τ]
- Αν Α ≥ Τ, επέστρεψε
- Διαφορετικά,
  - Διάλεξε ένα στοιχείο του πίνακα (pivot), έστω το μεσαίο Λ = Π[(Α+Τ)/2]
  - Βρες την θέση Θ του Λ που θα έχει στην τελική ταξινόμηση
  - Μετέφερε τα στοιχεία του Π[Α] έως και Π[Τ] έτσι ώστε:
    - Π[Χ] ≤ Λ, αν Χ < Θ
    - Π[Χ] > Λ, αν Χ > Θ
- QSort(Π, Α, Θ-1)
- QSort(Π, Θ+1, Τ)

Qsort

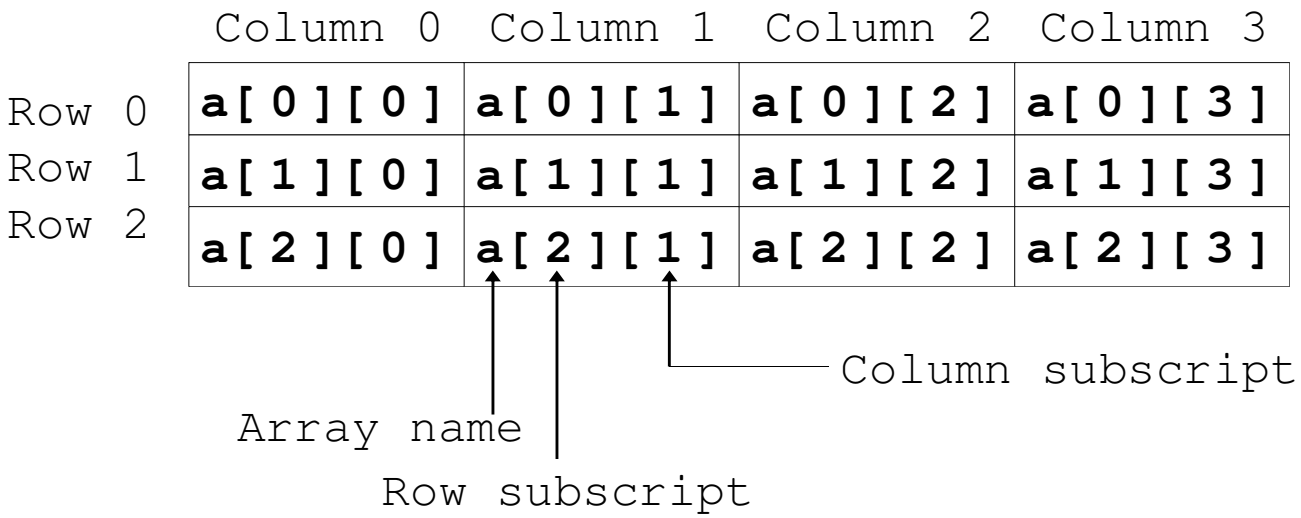
```
void qsort(int v[], int left, int right)
{
   int i, last;
   if(left >= right)
       return;
   swap(v, left, (left + right)/2);
   last = left;
   for (i = left+1; i <= right; i++)
       if (v[i] < v[left])
              swap(v, ++last, i);
   swap(v, left, last);
   qsort(v, left, last-1);
   qsort(v, last+1, right);
}
void swap(int v[], int i, int j)
{
   int temp;
   temp = v[i];
   v[i] = v[j];
   v[j] = temp;
}
```

ΗΥ-150
Προγραμματισμός

Πολυδιάστατοι Πίνακες

Πίνακες πολλών διαστάσεων

- Πίνακες πολλών διαστάσεων
  - Πίνακες με γραμμές και στήλες (**m** X **n** πίνακας)
  - Όπως και στα μαθηματικά: πρώτα γραμμή και μετά στήλη



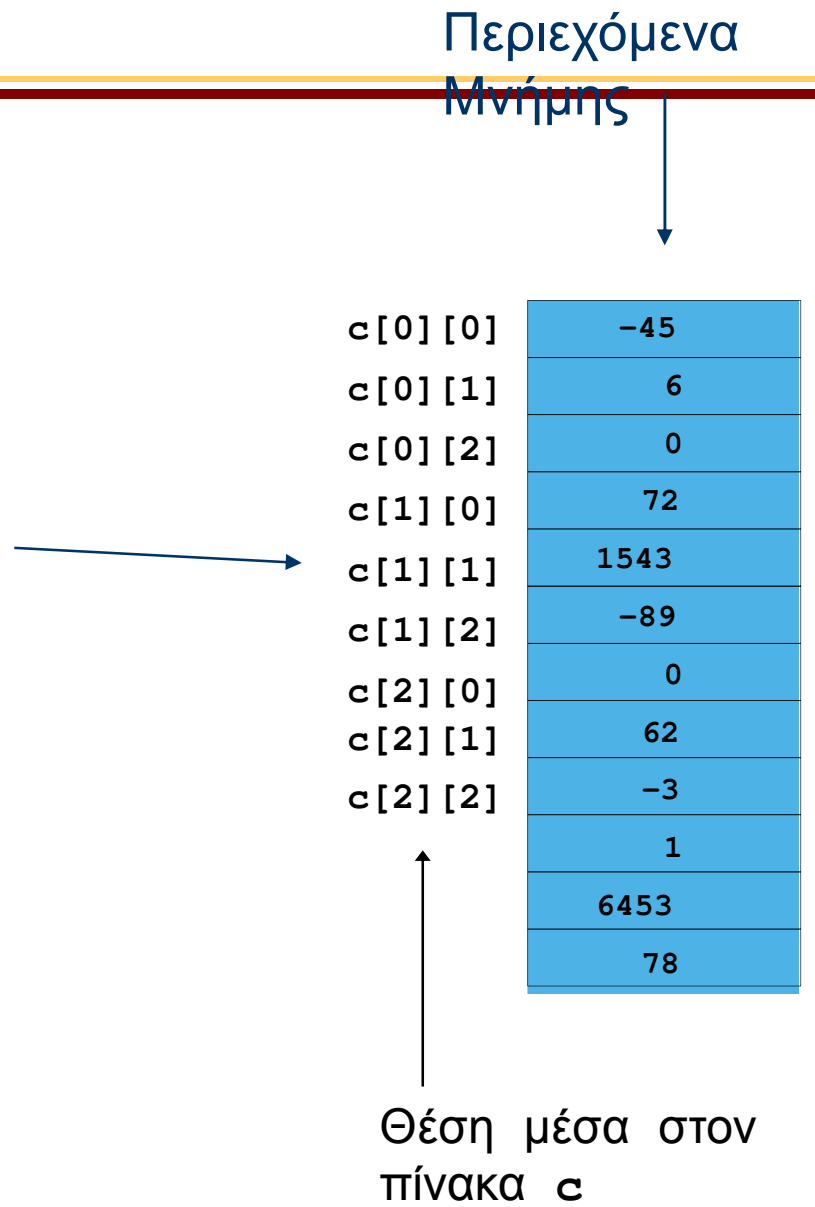
- Ένας διδιάστατος πίνακας ορίζεται ως εξής:

  τύπος όνομα[πλήθος_στοιχείων_1][πλήθος_στοιχείων_2];
- Πίνακες περισσότερων διαστάσεων ορίζονται ανάλογα. Ένας ακέραιος 6 x 8 διδιάστατος πίνακας είναι ο **int** a[6][8];
- Το στοιχείο π.χ. (3,2) του πίνακα αυτού είναι προσπελάσιμο ως a[3][2] .
- Απόδοση αρχικών τιμών γίνεται ανά γραμμή, ως εξής:

int b[2][3] = **{ {**0, 1, 2**}**, **{**3, 4, 5**} }**;

Δισδιάστατοι Πίνακες

- Δήλωση: int c[3][3];
- Χρήση: c[k][j] = m;
- Αποθήκευση στη μνήμη



Πίνακες πολλών διαστάσεων

- Αρχικοποίηση
  - **int b[2][2] = { { 1, 2 }, { 3, 4 } };**
  - {} διαχωρίζουν γραμμές του πίνακα
  - Αν δεν είναι αρκετά τα στοιχεία, τα υπόλοιπα γίνονται 0

    int b[2][2] = { { 1 }, { 3, 4 } };

1	2
3	4

1	0
3	4

Παράδειγμα 2Δ πίνακα

```
#include <stdio.h>

int main()
{
	int magic[3][3] =
	{	{8, 1, 6},
		{3, 5, 7},
		{4, 9, 2}
	};

	int i, j, sum;
	int sumrows[3];
	int sumcolumns[3];
	int sumdiagonal1;
	int sumdiagonal2;

	// Print the magic square
	for (i=0; i < 3; i++)
	{
		for (j = 0; j < 3; j++)
		{
			printf("%3d", magic[i][j]);
		}
		printf("\n");
	}
```

```
	// Now check that it is a magic square:
		// all rows and columns need to add to 15
		sumdiagonal1 = 0;
		sumdiagonal2 = 0;
		for (i = 0 ; i < 3; i++)
		{









		}

		for (i=0; i < 3 ; i++)
		{
			printf("Sum Row %d = %d\n", i, sumrows[i]);
			printf("Sum Column %d = %d\n", i,
		sumcolumns[i]);
		}
		printf("Sum diagonal 1 is %d\n", sumdiagonal1);
		printf("Sum diagonal 2 is %d\n", sumdiagonal2);

		return 0;
}
```

Παράδειγμα 2Δ πίνακα

```
#include <stdio.h>

int main()
{
	int magic[3][3] =
	{ {8, 1, 6},
		{3, 5, 7},
		{4, 9, 2}
	};

	int i, j, sum;
	int sumrows[3];
	int sumcolumns[3];
	int sumdiagonal1;
	int sumdiagonal2;

	// Print the magic square
	for (i=0; i < 3; i++)
	{
		for (j = 0; j < 3; j++)
		{
			printf("%3d", magic[i][j]);
		}
		printf("\n");
	}
```

```
// Now check that it is a magic square:
	// all rows and columns need to add to 15
	sumdiagonal1 = 0;
	sumdiagonal2 = 0;
	for (i = 0 ; i < 3; i++)
	{
		sumrows[i] = 0;
		sumcolumns[i] = 0;
		for (j = 0 ; j < 3; j++)
		{
			sumrows[i] += magic[i][j];
			sumcolumns[i]+= magic[j][i];
		}

		sumdiagonal1 += magic[i][i];
		sumdiagonal2 += magic[i][2-i];
	}

	for (i=0; i < 3 ; i++)
	{
		printf("Sum Row %d = %d\n", i, sumrows[i]);
		printf("Sum Column %d = %d\n", i,
	sumcolumns[i]);
	}
	printf("Sum diagonal 1 is %d\n", sumdiagonal1);
	printf("Sum diagonal 2 is %d\n", sumdiagonal2);

	return 0;
}
```

Πολυδιάστατους Πίνακες ως Ορίσματα

- Πάντα πρέπει να δηλώνεται το μέγεθος κάθε διάστασης, εκτός από την πρώτη διάσταση (αριθμός γραμμών)

```
int f(int a[][10][20])
{
        …
}
int main(void)
{
   int b[40][10][20];
   f(b);
}
```

- Γιατί μας υποχρεώνει η γλώσσα σε αυτόν τον περιορισμό; Σκεφτείτε.

```c
/*
   Double-subscripted array example */
#include <stdio.h>
#define STUDENTS 3
#define EXAMS 4

int minimum( const int [][ EXAMS ], int, int );
int maximum( const int [][ EXAMS ], int, int );
double average( const int [], int );
void printArray( const int [][ EXAMS ], int, int );

int main()
{
   int student;
   const int studentGrades[ STUDENTS ][ EXAMS ] =
      { { 77, 68, 86, 73 },
        { 96, 87, 89, 78 },
        { 70, 90, 86, 81 } };

   printf( "The array is:\n" );
   printArray( studentGrades, STUDENTS, EXAMS );
   printf( "\n\nLowest grade: %d\nHighest grade: %d\n",
           minimum( studentGrades, STUDENTS, EXAMS ),
           maximum( studentGrades, STUDENTS, EXAMS ) );

   for ( student = 0; student <= STUDENTS - 1; student++ )
      printf( "The average grade for student %d is %.2f\n",
              student,
              average( studentGrades[ student ], EXAMS ) );

   return 0;
}
```

Each row is a particular student, each column is the grades on the exam.

Notice! Each studentGrades[i] is a one-dimensional array with the grades of student i.

```
33
34 /* Find the minimum grade */
35 int minimum( const int grades[][ EXAMS ],
36              int pupils, int tests )
37 {
38    int i, j, lowGrade = 100;
39
40    for ( i = 0; i <= pupils - 1; i++ )
41       for ( j = 0; j <= tests - 1; j++ )
42          if ( grades[ i ][ j ] < lowGrade )
43             lowGrade = grades[ i ][ j ];
44
45    return lowGrade;
46 }
47
48 /* Find the maximum grade */
49 int maximum( const int grades[][ EXAMS ],
50              int pupils, int tests )
51 {
52    int i, j, highGrade = 0;
53
54    for ( i = 0; i <= pupils - 1; i++ )
55       for ( j = 0; j <= tests - 1; j++ )
56          if ( grades[ i ][ j ] > highGrade )
57             highGrade = grades[ i ][ j ];
58
59    return highGrade;
60 }
61
62 /* Determine the average grade for a particular exam */
63 double average( const int setOfGrades[], int tests )
64 {
```

```
65    int i, total = 0;
66
67    for ( i = 0; i <= tests - 1; i++ )
68       total += setOfGrades[ i ];
69
70    return ( double ) total / tests;
71 }
72
73 /* Print the array */
74 void printArray( const int grades[][ EXAMS ],
75                  int pupils, int tests )
76 {
77    int i, j;
78
79    printf( "                [0]  [1]  [2]  [3]" );
80
81    for ( i = 0; i <= pupils - 1; i++ ) {
82       printf( "\nstudentGrades[%d] ", i );
83
84       for ( j = 0; j <= tests - 1; j++ )
85          printf( "%-5d", grades[ i ][ j ] );
86    }
87 }
```

Program Output

```
The array is:
                  [0]   [1]   [2]   [3]
studentGrades[0] 77    68    86    73
studentGrades[1] 96    87    89    78
studentGrades[2] 70    90    86    81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
```

Υπολογισμός Mean, Median and Mode

- Mean – average
  - 1, 1, 2, 2, 4
  - Mean = (1+1+2+2+4) / 5 = 2.0
- Median – number in middle of sorted list
  - 1, 2, 3, 4, 5
  - 3 is the median
- Mode – number that occurs most often
  - 1, 1, 1, 2, 3, 3, 4, 5
  - 1 is the mode

```c
/* Fig. 6.16: fig06_16.c
   This program introduces the topic of survey data analysis.
   It computes the mean, median, and  mode of the data */
#include <stdio.h>
#define SIZE 99

void mean( const int [] );
void median( int [] );
void mode( int [], const int [] ) ;
void bubbleSort( int [] );
void printArray( const int [] );

int main()
{
   int frequency[ 10 ] = { 0 };
   int response[ SIZE ] =
      { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
        7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
        6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
        7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
        6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
        7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
        5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
        7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
        7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
        4, 5, 6, 1, 6, 5, 7, 8, 7 };

   mean( response );
   median( response );
   mode( frequency, response );
   return 0;
}
```

```
33
34 void mean( const int answer[] )
35 {
36    int j, total = 0;
37
38    printf( "%s\n%s\n%s\n", "********", "  Mean", "********" );
39
40    for ( j = 0; j <= SIZE - 1; j++ )
41       total += answer[  j  ];
42
43    printf( "The mean is the average value of the data\n"
44            "items. The mean is equal to the total of\n"
45            "all the data items divided by the number\n"
46            "of data items ( %d ). The mean value for\n"
47            "this run is: %d / %d = %.4f\n\n",
48            SIZE, total, SIZE, (  double  ) total / SIZE );
49 }
50
51 void median( int answer[] )
52 {
53    printf( "\n%s\n%s\n%s\n%s",
54            "********", " Median", "********",
55            "The unsorted array of responses is" );
56
57    printArray( answer );
58    bubbleSort( answer );
59    printf( "\n\nThe sorted array is" );
60    printArray( answer );
61    printf( "\n\nThe median is element %d of\n"
62            "the sorted %d element array.\n"
63            "For this run the median is %d\n\n",
64            SIZE / 2, SIZE, answer[ SIZE / 2 ] );
```

```
65 }
66
67 void mode( int freq[], const int answer[] )
68 {
69    int rating, j, h, largest = 0, modeValue = 0;
70
71    printf( "\n%s\n%s\n%s\n",
72            "********", "  Mode", "********" );
73
74    for ( rating = 1; rating <= 9; rating++ )
75       freq[ rating ] = 0;
76
77    for ( j = 0; j <= SIZE - 1; j++ )
78       ++freq[ answer[ j ] ];
79
80    printf( "%s%11s%19s\n",
81            "Response", "Frequency", "Histogram");
82`
83
84    for ( rating = 1; rating <= 9; rating++ ) {
85       printf( "%8d%11d           ", rating, freq[ rating ] );
86
87       if ( freq[ rating ] > largest ) {
88          largest = freq[ rating ];
89          modeValue = rating;
90       }
91
92       for ( h = 1; h <= freq[ rating ]; h++ )
93          printf( "*" );
94
```

Notice how the subscript in **`frequency[]`** is the value of an element in **`response[]`** (**`answer[]`**)

Print stars depending on value of **`frequency[]`**

```
95       printf( "\n" );
96    }
97
98    printf( "The mode is the most frequent value.\n"
99            "For this run the mode is %d which occurred"
100           " %d times.\n", modeValue, largest );
101}
102
103 void bubbleSort( int a[] )
104 {
105   int pass, j, hold;
106
107   for ( pass = 1; pass <= SIZE - 1; pass++ )
108
109      for ( j = 0; j <= SIZE - 2; j++ )
110
111         if ( a[ j ] > a[ j + 1 ] ) {
112            hold = a[ j ];
113            a[ j ] = a[ j + 1 ];
114            a[ j + 1 ] = hold;
115         }
116 }
117
118 void printArray( const int a[] )
119 {
120   int j;
121
122   for ( j = 0; j <= SIZE - 1; j++ ) {
123
124      if ( j % 20 == 0 )
125         printf( "\n" );
```

Bubble sort: if elements out of order, swap them.

```
126
127       printf( "%2d", a[ j ] );
128    }
129 }
```

```
********
 Mean
********
The mean is the average value of the data
items. The mean is equal to the total of
all the data items divided by the number
of data items (99). The mean value for
this run is: 681 / 99 = 6.8788

********
 Median
********
The unsorted array of responses is
7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 8
 6 7 8 9 3 9 8 7 8 7 7 8 9 8 9 8 9 7 8 9
 6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5 3
 5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 8
 7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7

The sorted array is
 1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 5
 5 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

The median is element 49 of
the sorted 99 element array.
For this run the median is 7
```

Program Output

Program Output

```
********
  Mode
********
Response  Frequency        Histogram
       1          1          *
       2          3          ***
       3          4          ****
       4          5          *****
       5          8          ********
       6          9          *********
       7         23          ***********************
       8         27          ***************************
       9         19          *******************
The mode is the most frequent value.
For this run the mode is 8 which occurred 27 times.
```