Recursive Function

Session 15

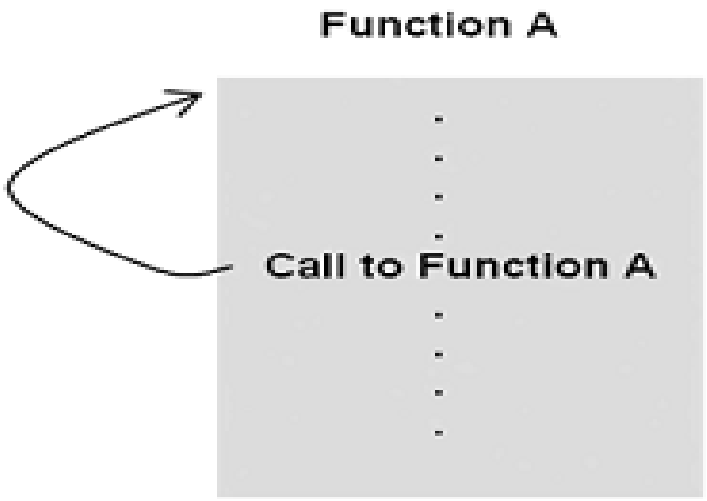
Recursive algorithms

- In *recursive problem solving*, a **problem is repeatedly broken down into similar sub problems**, until the sub problems can be directly solved without further breakdown.

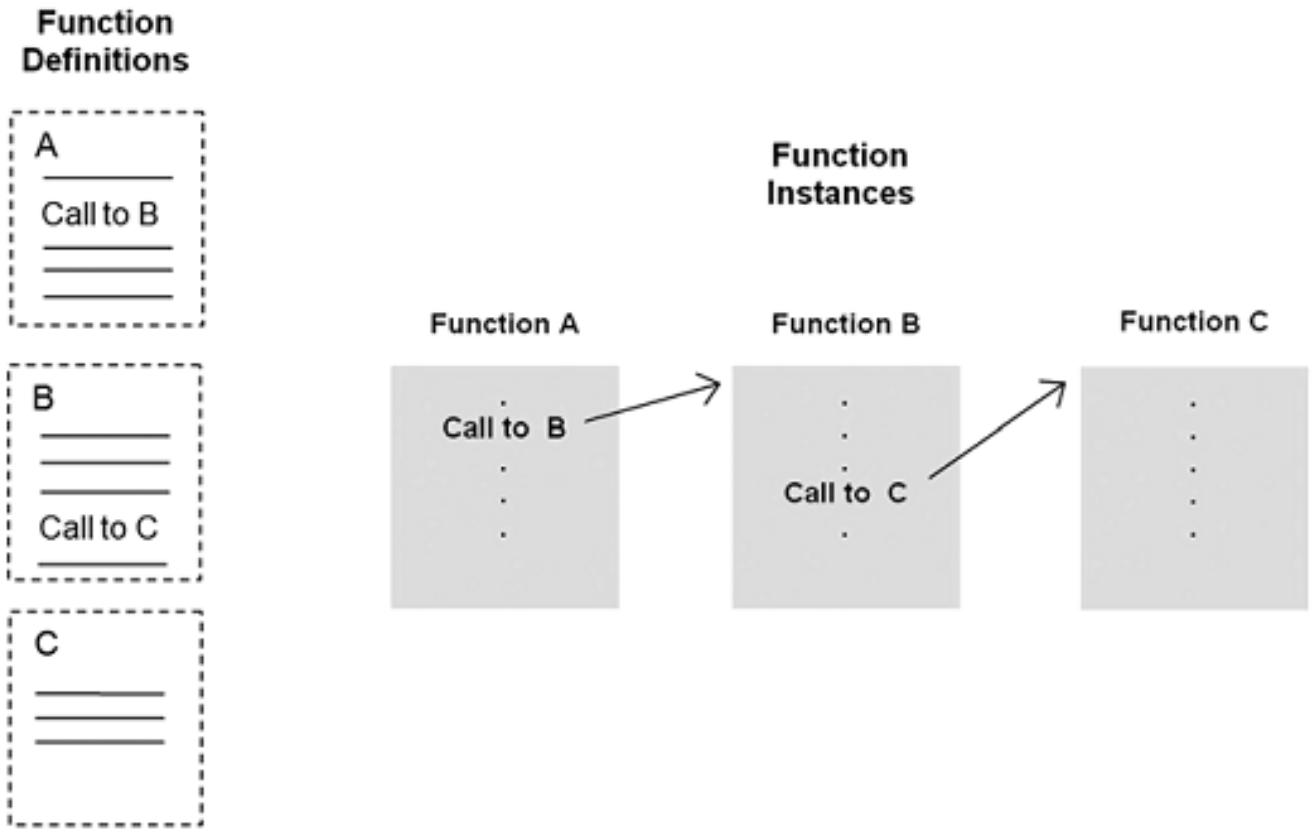
What Is a Recursive Function?

- A **recursive function** is often defined as "**a function that calls itself.**"

```
def myFun(x)
        if (x>0)
                print(x)
                myFun(x-1)


myFun(10)
10
9
8
```

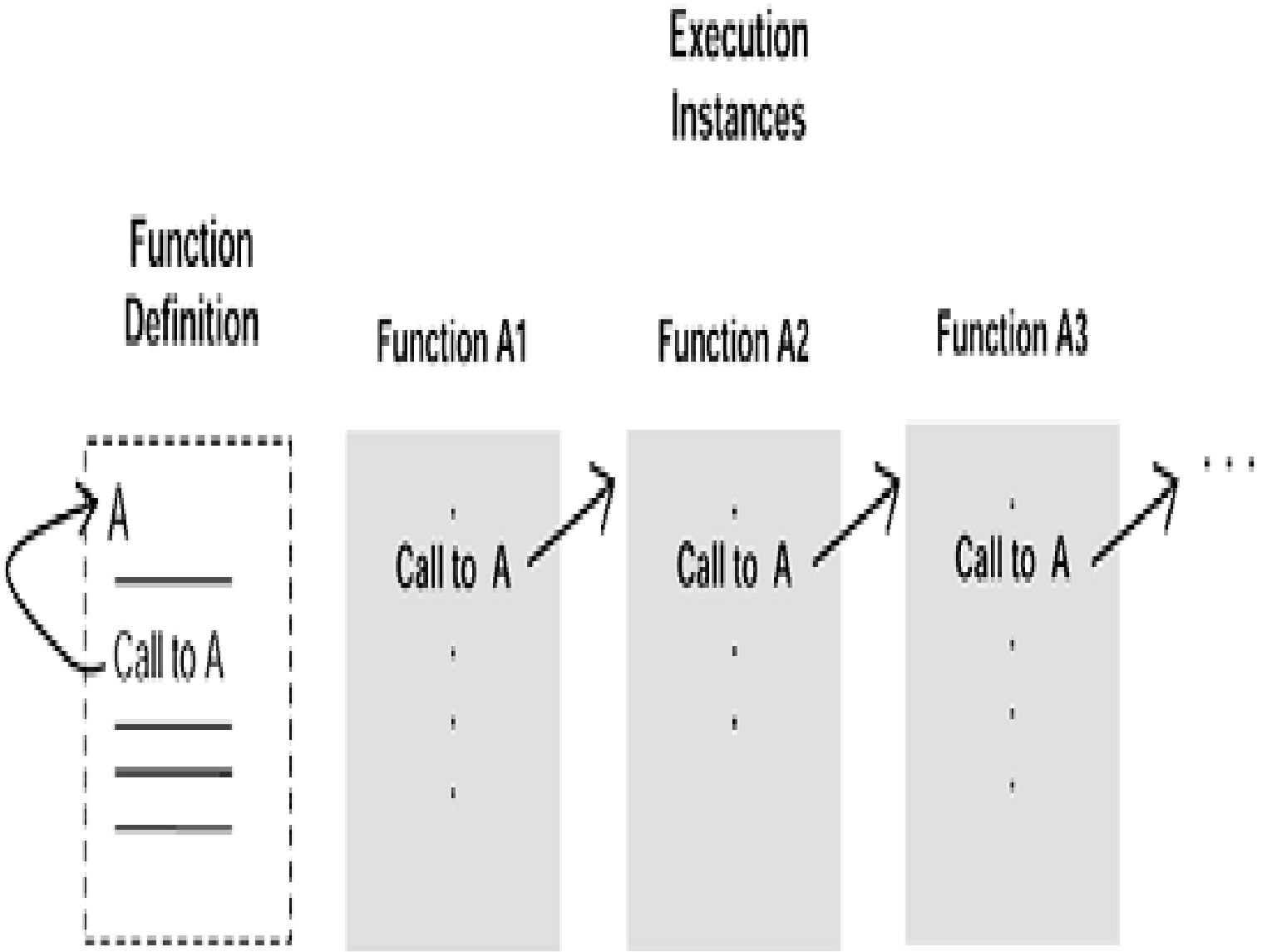


General mechanism of non-recursive function



Recursive function execution instances

- Execution of a series of recursive function instances is similar to the execution of series of non-recursive instances, except that the execution instances are “clones” of each other (that is, of the same function definition).


Execution
Instances
Function
Definition
Function A1
Function A2
Function A3
A
Call to A
Call to A
Call to A
Call to A
. . .

LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> def rfunc(n):
        print(n)
        if n > 0:
            rfunc(n - 1)

>>> rfunc(4)
???

>>> rfunc(0)
???

>>> rfunc(100)
???
```

```
>>> def rfunc(n):
        if n == 1:
            return 1
        else:
            return n + rfunc(n - 1)

>>> rfunc(1)
???

>>> rfunc(3)
???

>>> rfunc(100)
???
```

Example: Factorial

- Problem:

The factorial function is an often-used example of the use of recursion. The computation of the factorial of 4 is given as,

factorial(4) = 4 * 3 * 2* 1= 24

In general, the computation of the factorial of any (positive, nonzero) integer n is,

factorial(n) = n . (n-1). (n-2) . . . 1

The one exception is the factorial of 0, defined to be 1.

logic

- The factorial of n can be defined as n times the factorial of n – 1

$$\text{factorial}(n) = n \cdot \underbrace{(n-1) \cdot (n-2) \cdot \cdots \cdot 1}_{\text{factorial}(n-1)}$$

Thus, the complete definition of the factorial function is,

$$\begin{aligned} \text{factorial}(n) &= 1, && \text{if } n = 0 \\ &= n \cdot \text{factorial}(n-1), && \text{otherwise} \end{aligned}$$

A Recursive Factorial Function Implementation

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Input

Factorial(4)

Factorial Recursive Instance Calls

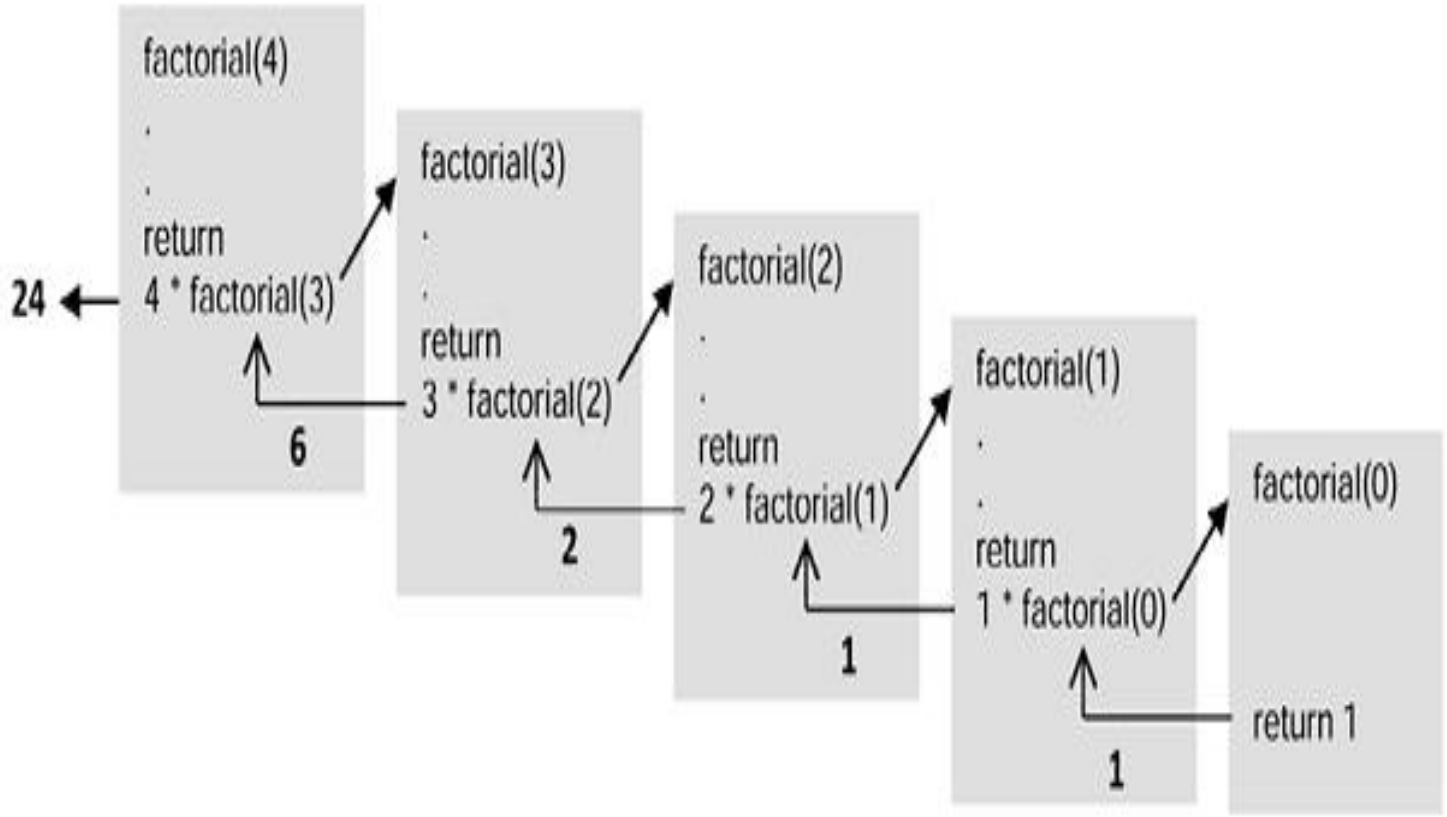


FIGURE 11-6 Factorial Recursive Instance Calls

LET'S TRY IT

From the Python Shell, enter the following and observe the results.

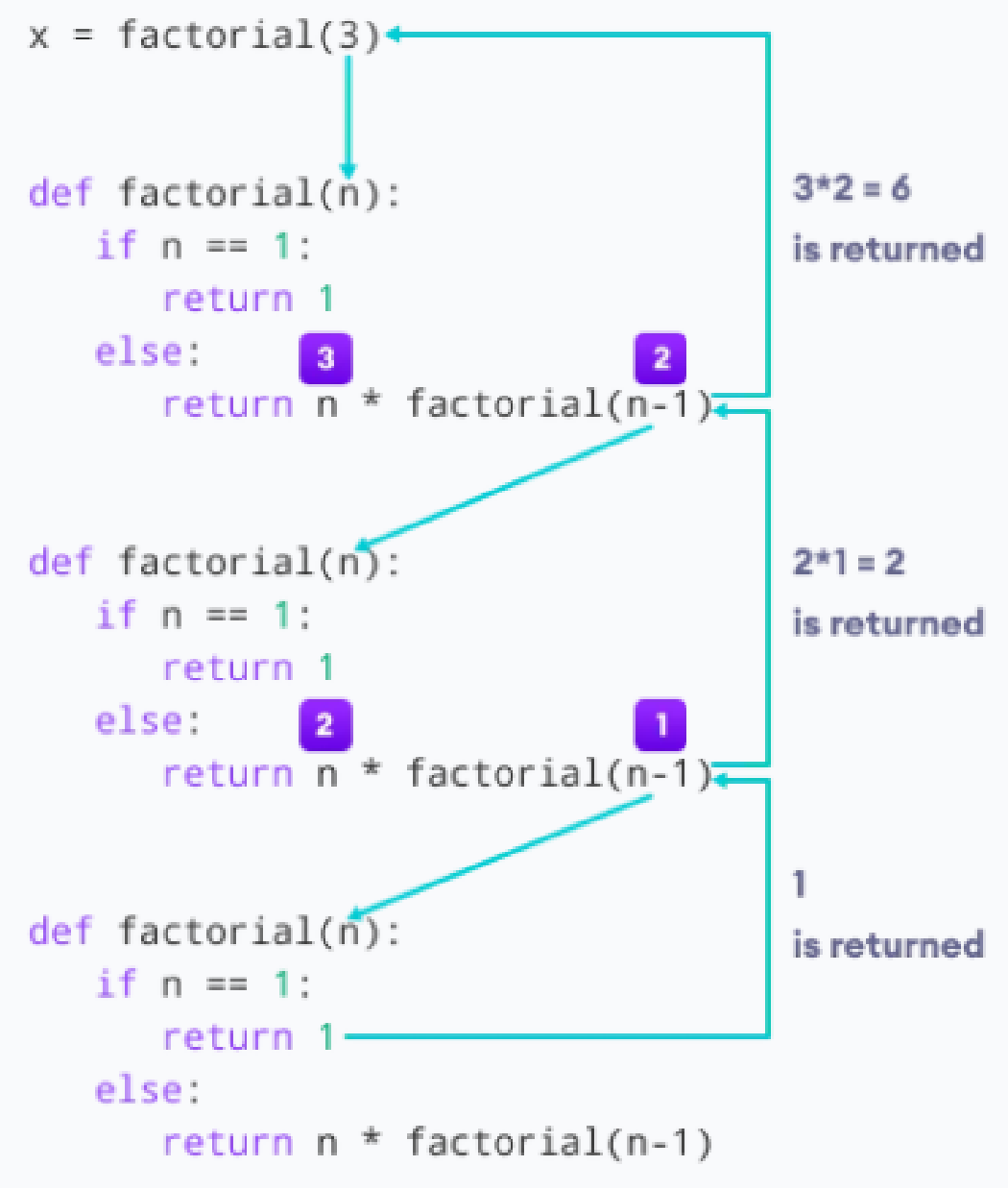
```
>>> def factorial(n):
        if n == 0:
            return 1
        return n * factorial(n - 1)


>>> factorial(4)
???
>>> factorial(0)
???
>>> factorial(100)
???
>>> factorial(10000)
???
```

```
>>> def ifactorial(n):
    result = 1
    if n == 0:
        return result
    for k in range(n, 0, -1):
        result = result * k
    return result
>>> ifactorial(0)
???
>>> ifactorial(100)
???
>>> ifactorial(10000)
???
```

Working of a recursive factorial function

```
factorial(3)            # 1st call with 3
3 * factorial(2)        # 2nd call with 2
3 * 2 * factorial(1)    # 3rd call with 1
3 * 2 * 1               # return from 3rd call as number=1
3 * 2                   # return from 2nd call
6                       # return from 1st call
```

Working of a recursive factorial function



Python Lambda Function

- A lambda function is a small anonymous function.

- A lambda function can take any number of arguments, but can only have one expression.

- Syntax

lambda arguments : expression

The expression is executed and the result is returned

Lambda Function

Add 10 to argument a, and return the result:

```
x = lambda a : a + 10
print(x(5))
```

Output:

15

Lambda Function

- Lambda functions can take any number of arguments:

Example - Multiply argument a with argument b and return the result:

```
x = lambda a, b : a * b
print(x(5, 6))
```

Output:

30

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

Output:

13

Lambda function inside another function

Use the function definition (myfunc()) to make a function that always doubles the number you send in.

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

Output:

22

Lambda function

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

Output:

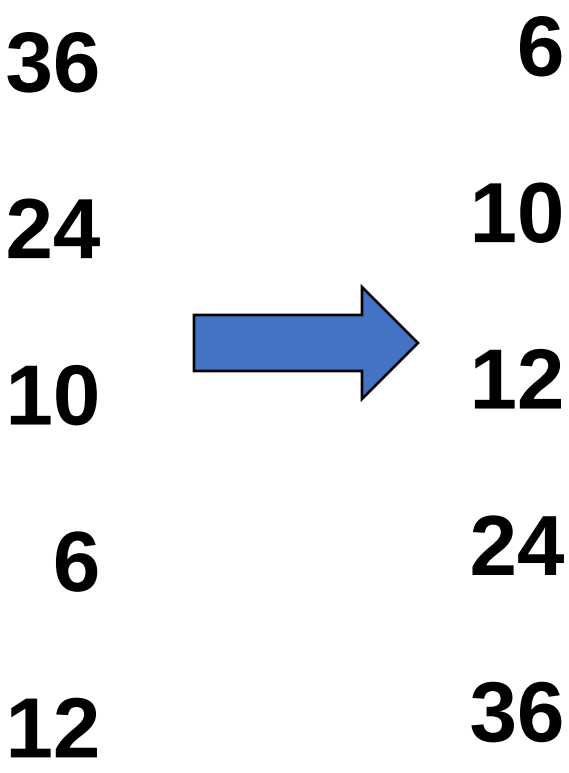
22

33

Sorting

Sorting means . . .

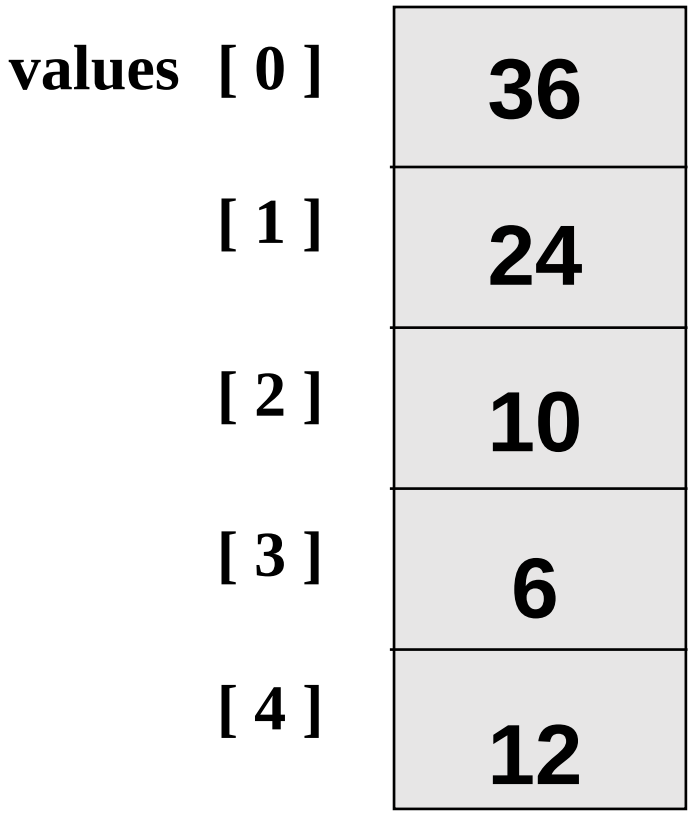
- Sorting rearranges the elements into either ascending or descending order within the array (we'll use ascending order).



Sorting also means…

- There are several sorting algorithms available like bubble sort, selection sort, insertion sort, quick sort, merge sort, radix sort etc.
- Sorting operation is performed in many applications to provide the output in desired order.
- For example listing all the product in the increasing order of their names or decreasing order of supplier names
- Searching will be easier in a sorted collection of elements
- List containing exam scores sorted from lowest to highest or vice versa
- We study Bubble Sorting, Selection Sorting and Insertion Sorting in this lab course

Bubble Sort



- **Compares neighboring pairs of array elements, starting with the last/first array element, and swaps neighbors whenever they are not in correct order.**
- **On each pass, this causes the smallest element to "bubble up" to its correct place in the array.**

Algorithm

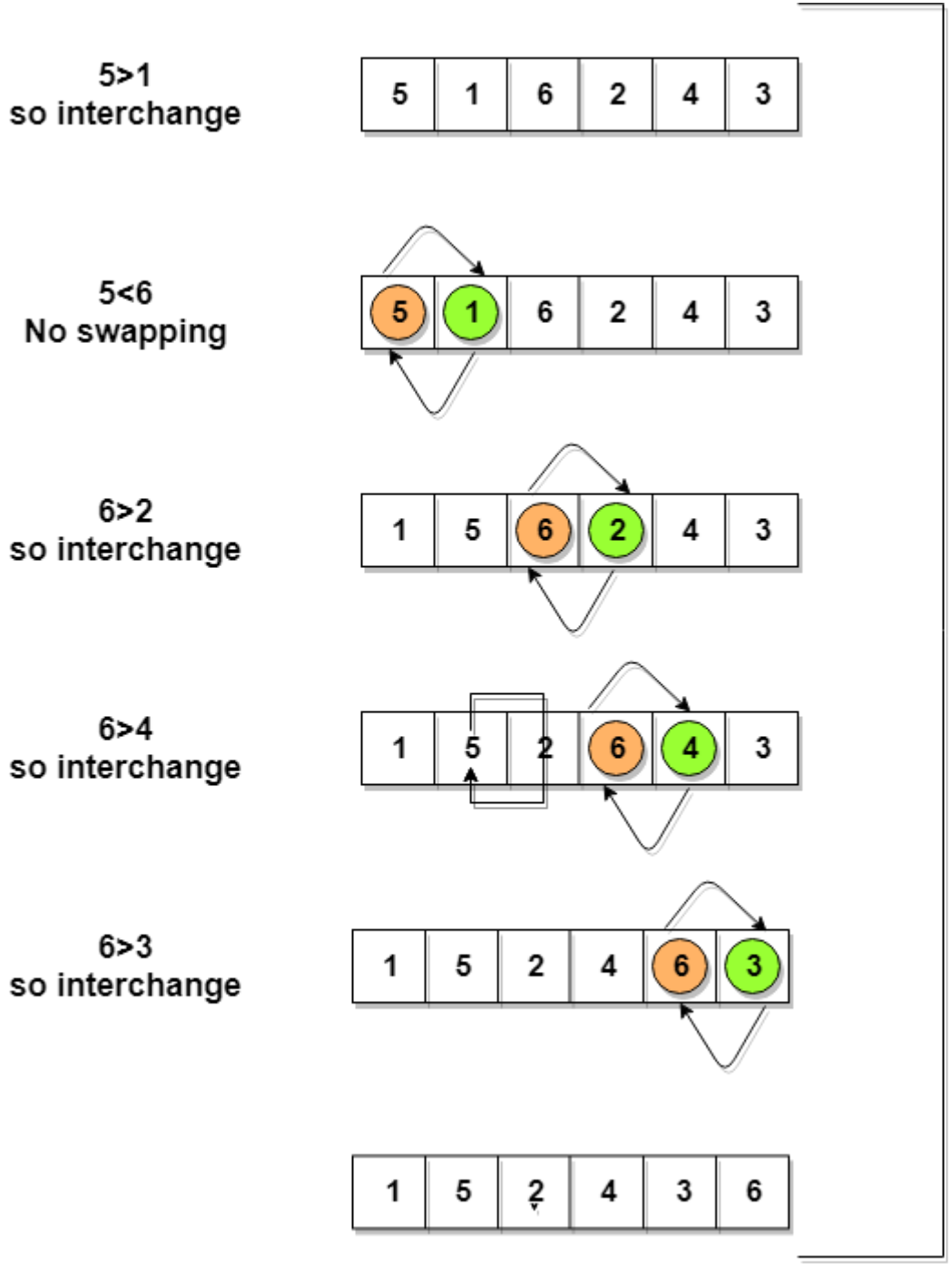
```
begin BubbleSort(list)
   for all elements of list
      if list[i] > list[i+1]
         swap(list[i], list[i+1])
      end if
   end for

   return list

end BubbleSort
```

Bubble Sort Pseudo Code

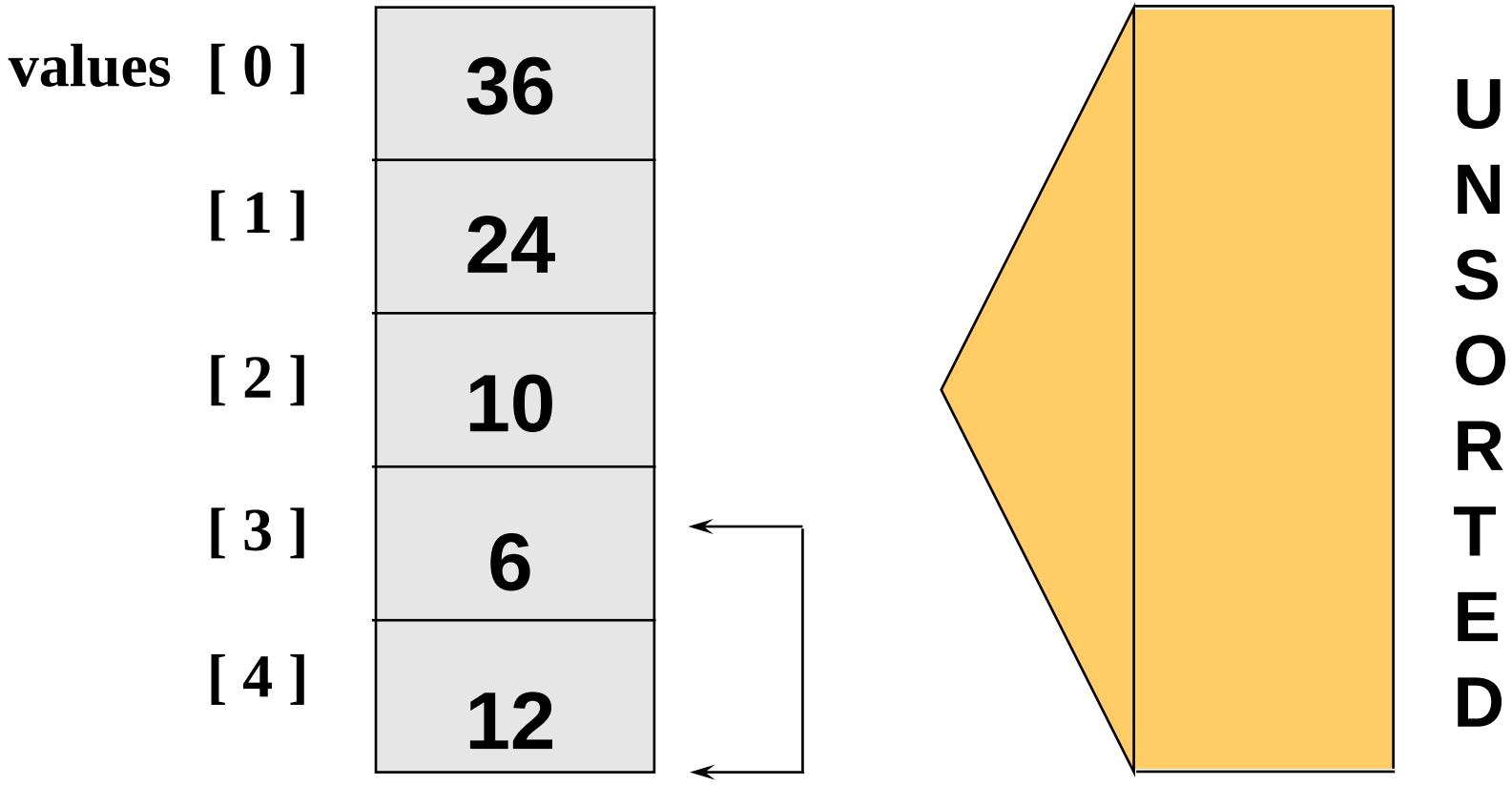
```
def bubbleSort(lyst):
    n = len(lyst)
    while n > 1:                        # Do n - 1 bubbles
        i = 1                           # Start each bubble
        while i < n:
            if lyst[i] < lyst[i - 1]:   # Exchange if needed
                swap(lyst, i, i - 1)
            i += 1
        n -= 1
```

Let's consider an array with values

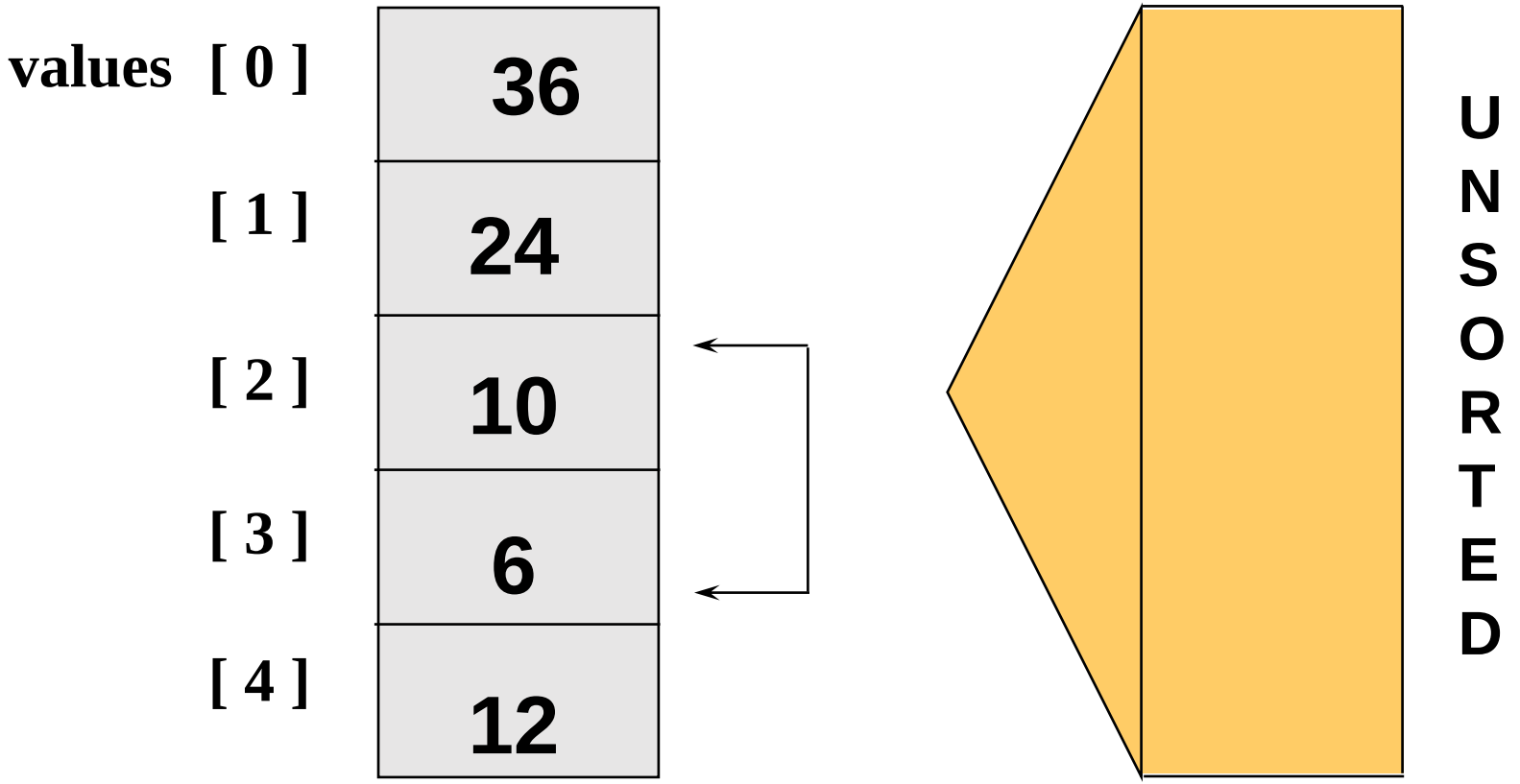
`{5, 1, 6, 2, 4, 3}`



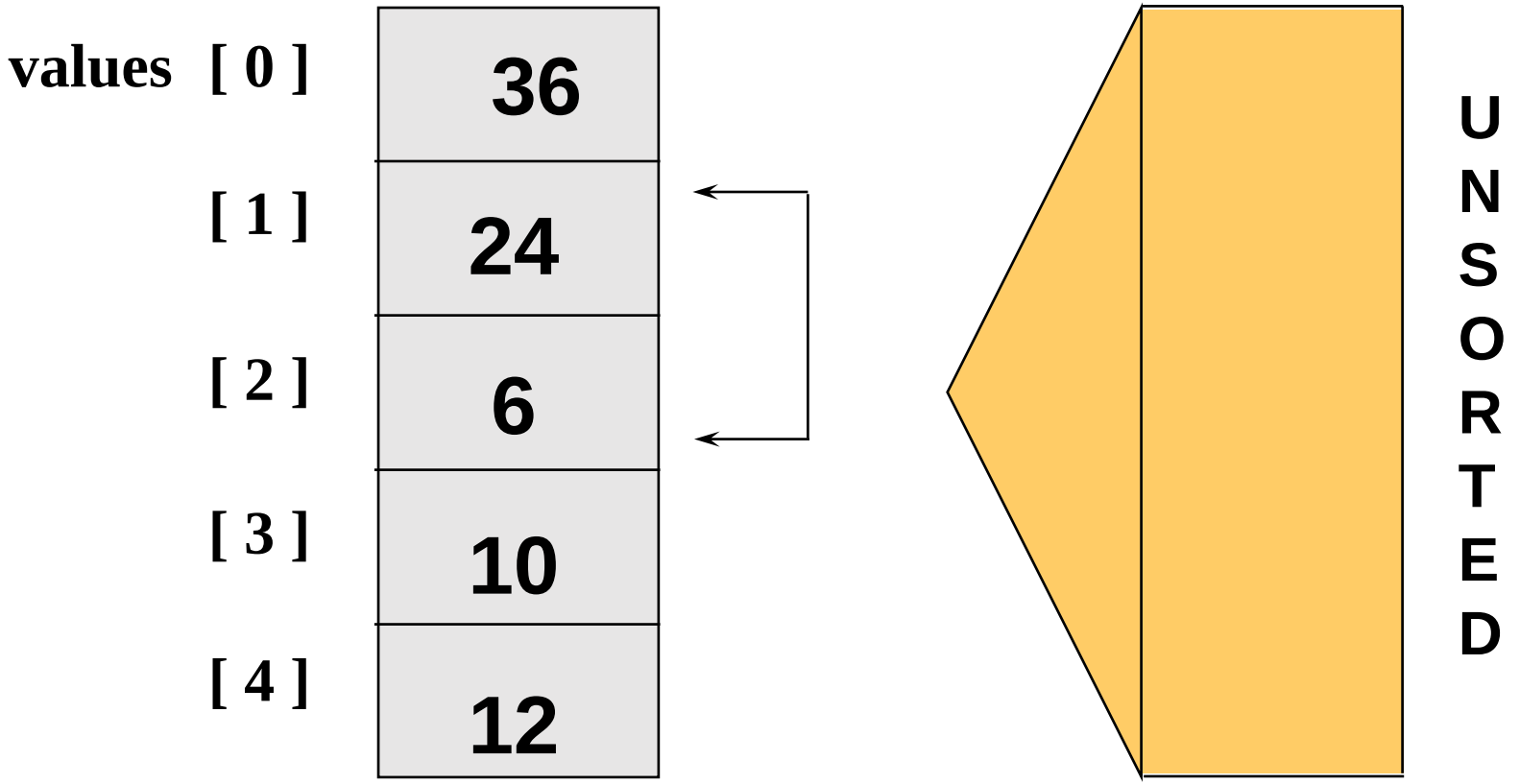
Bubble Sort: Pass One



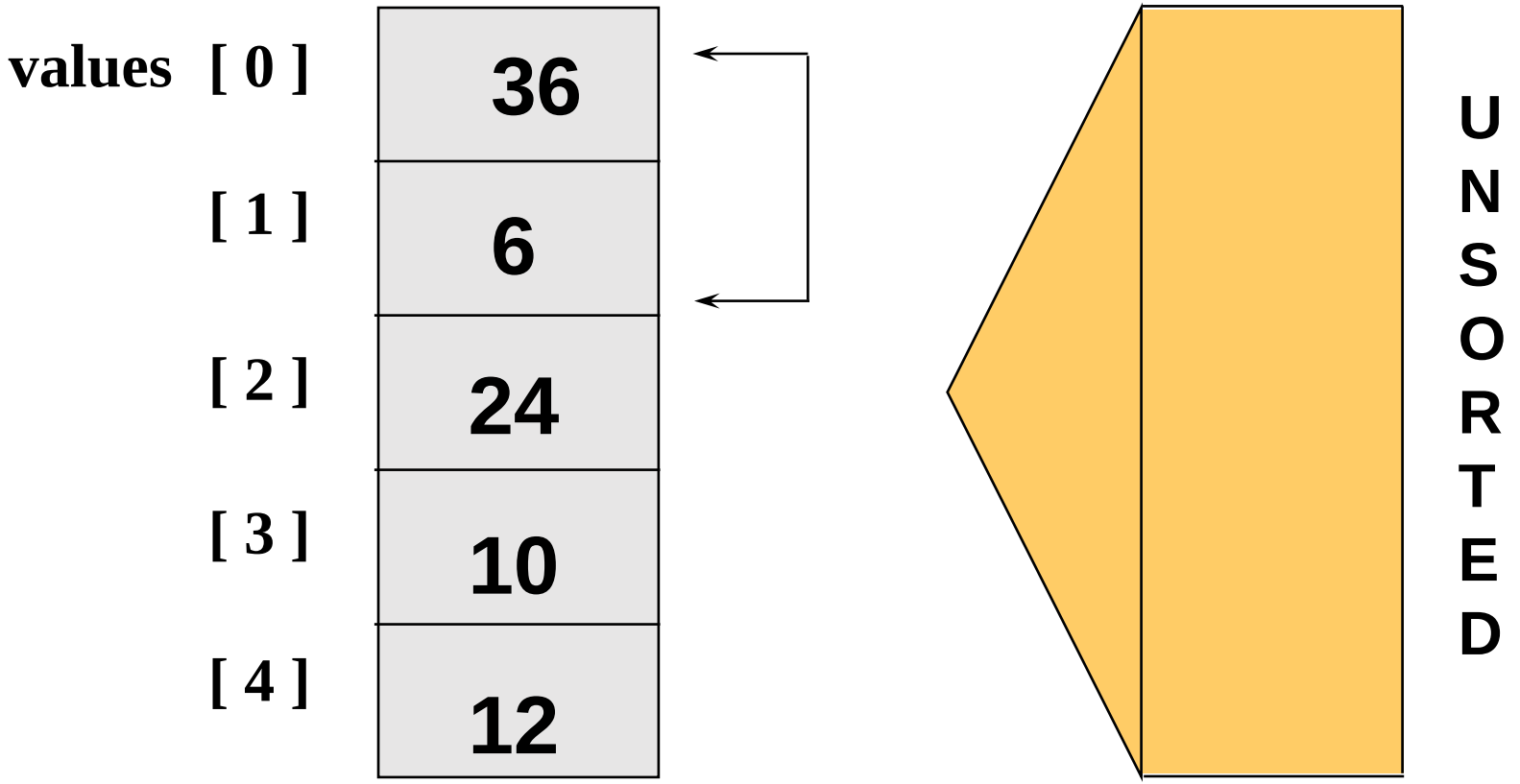
Bubble Sort: Pass One



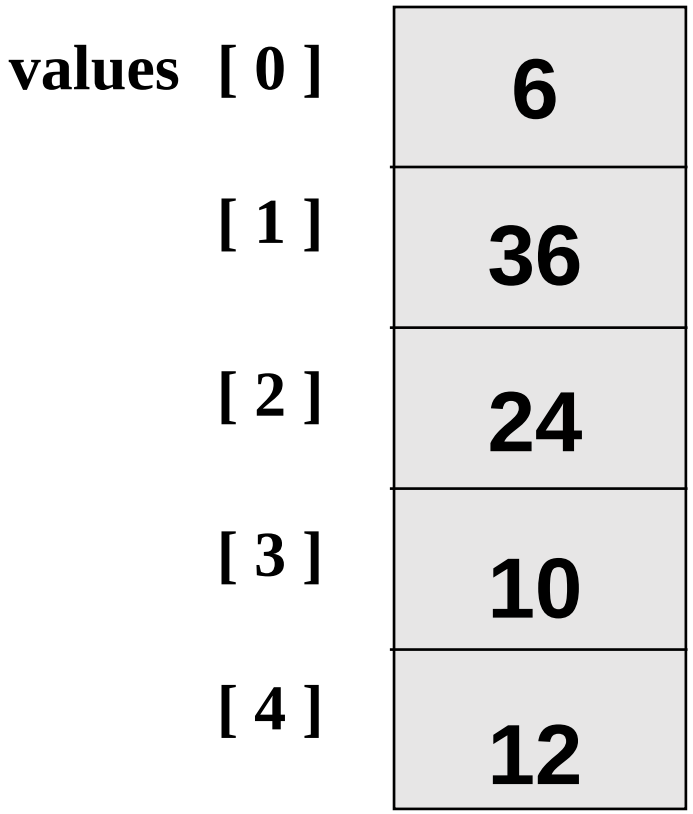
Bubble Sort: Pass One

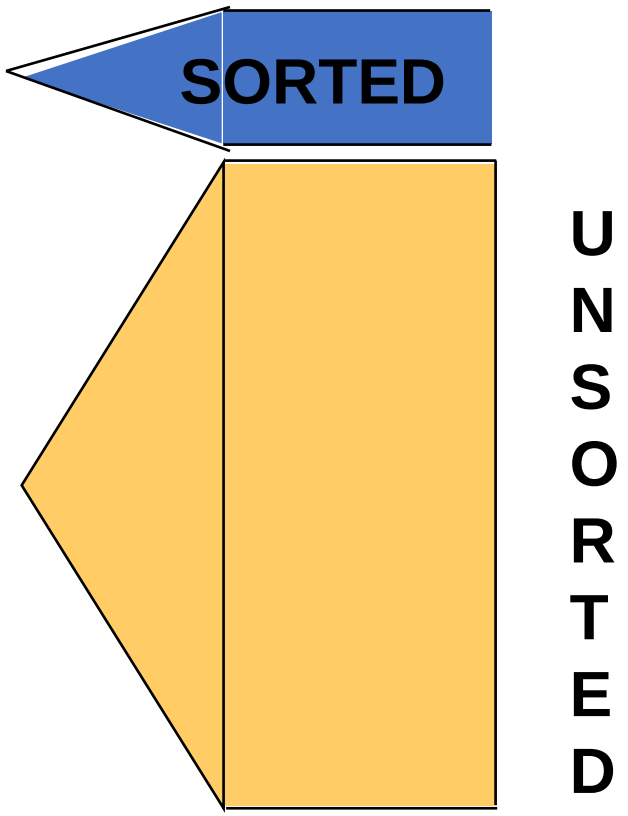


Bubble Sort: Pass One

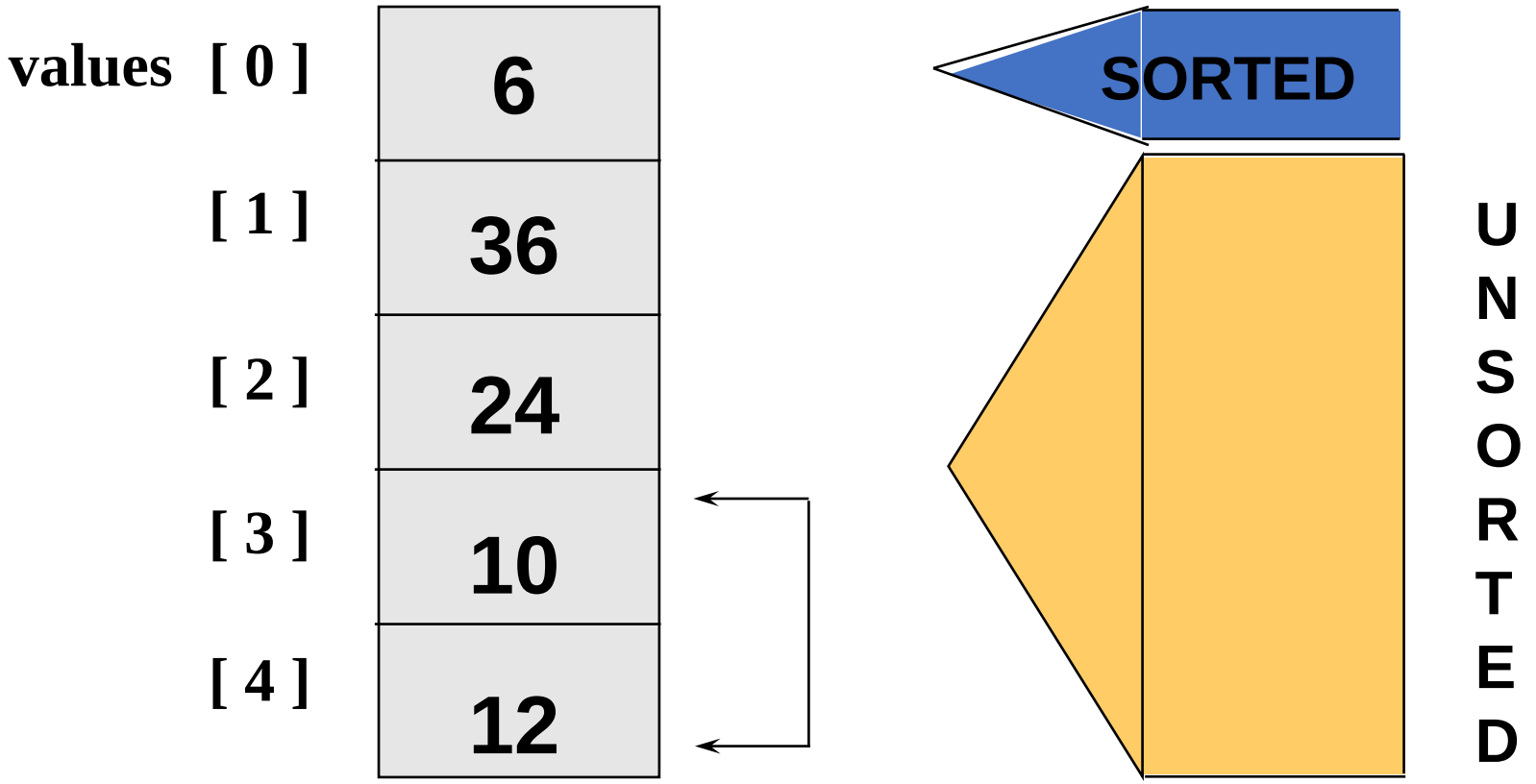


Bubble Sort: End Pass One

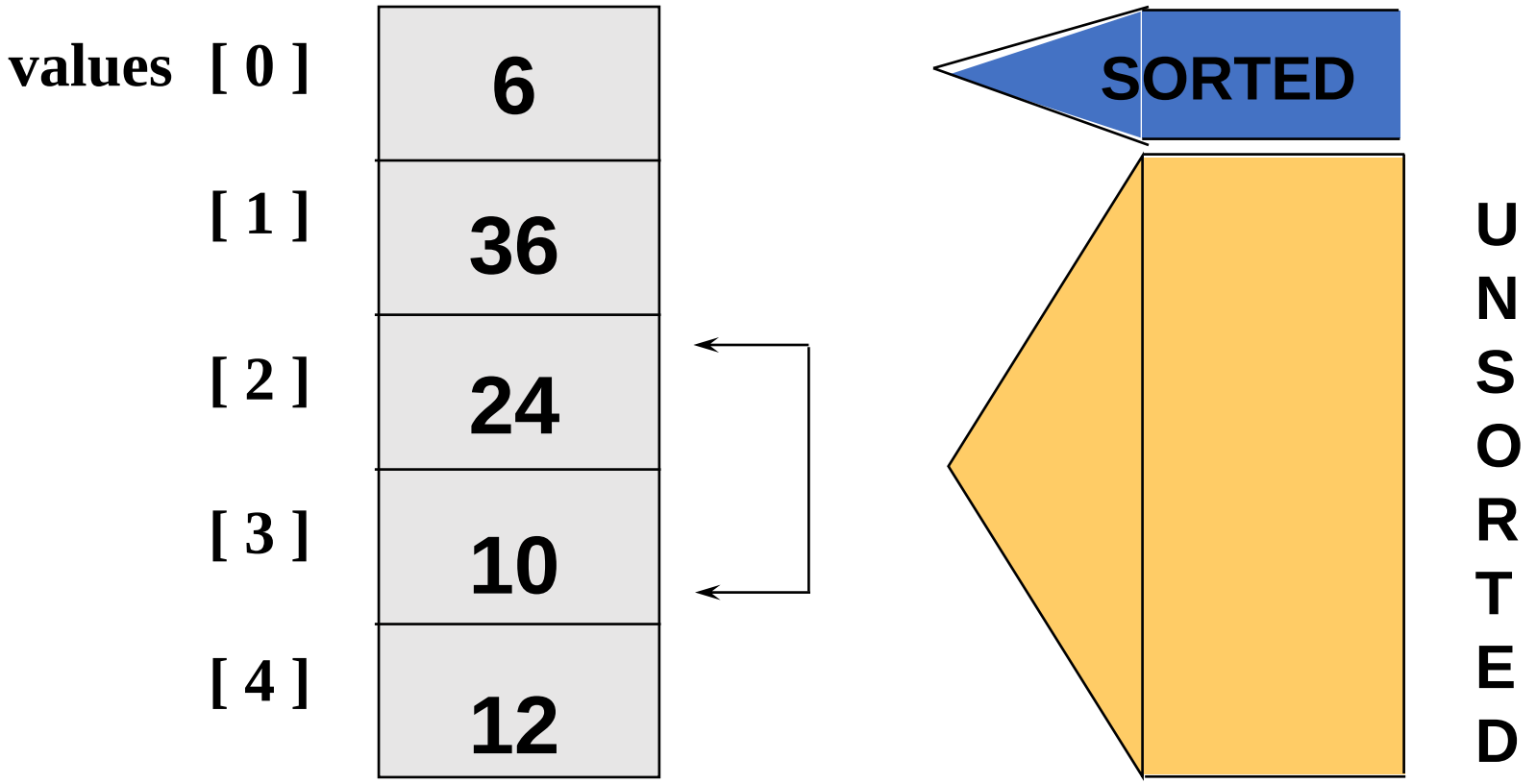




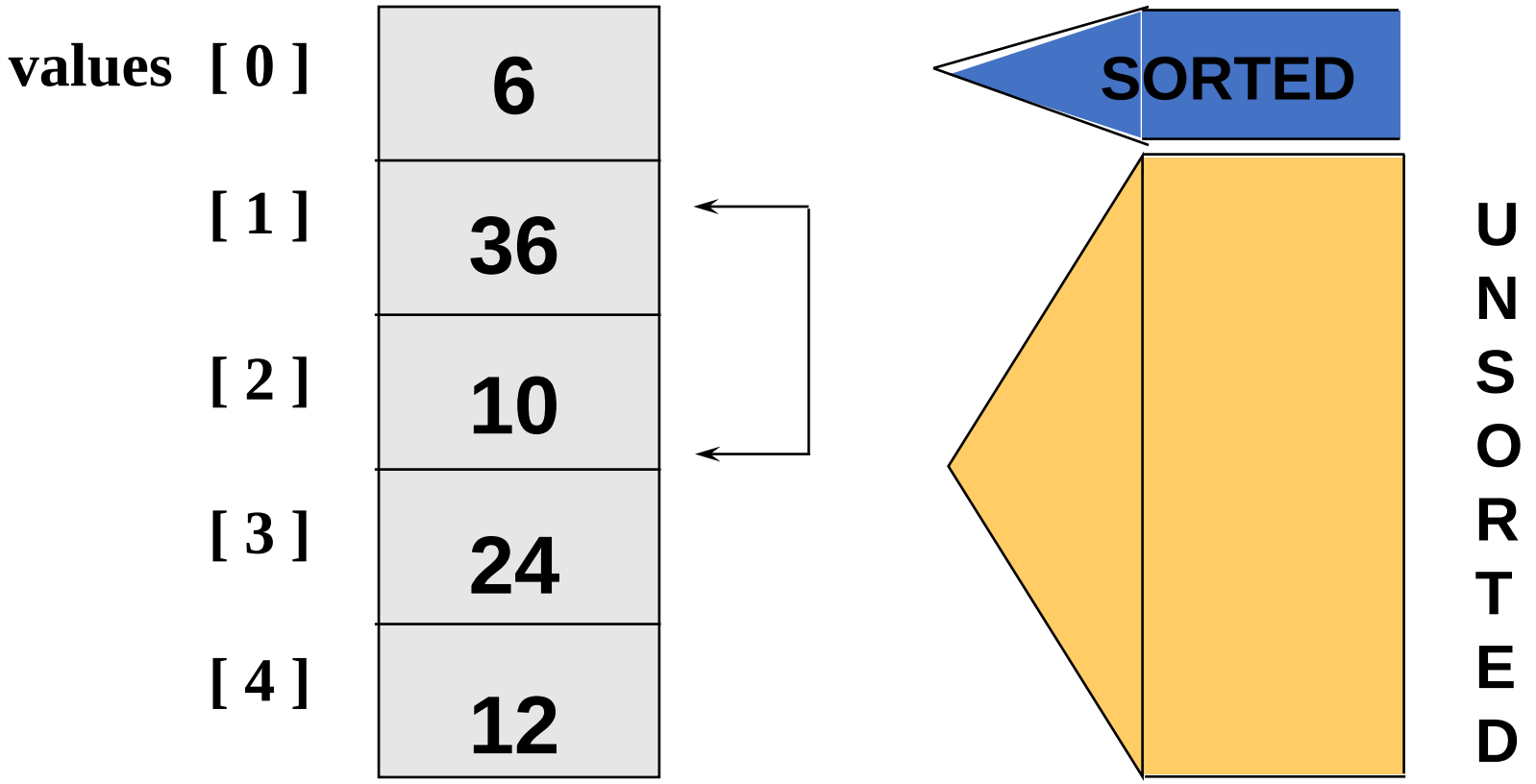
Bubble Sort: Pass Two



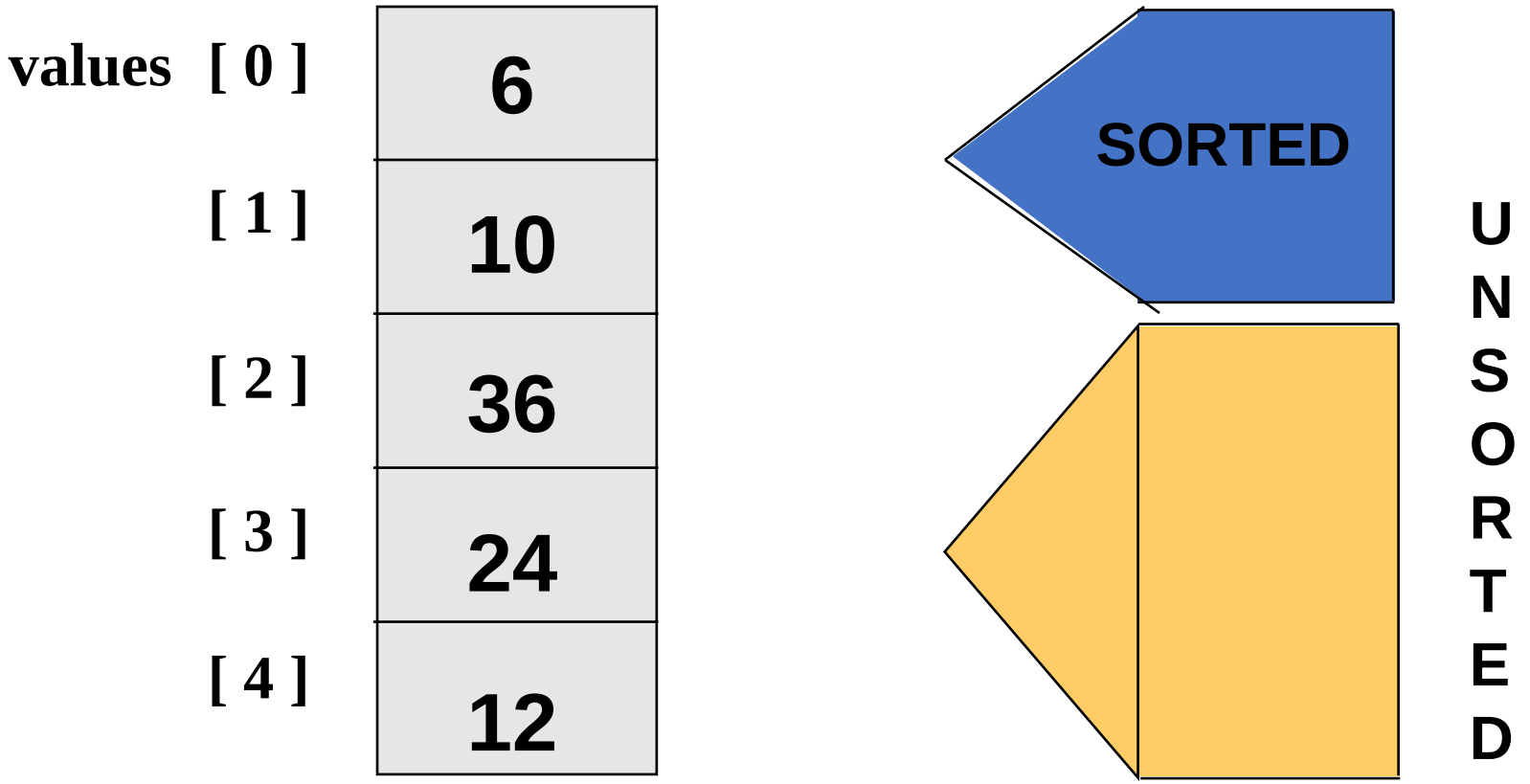
Bubble Sort: Pass Two



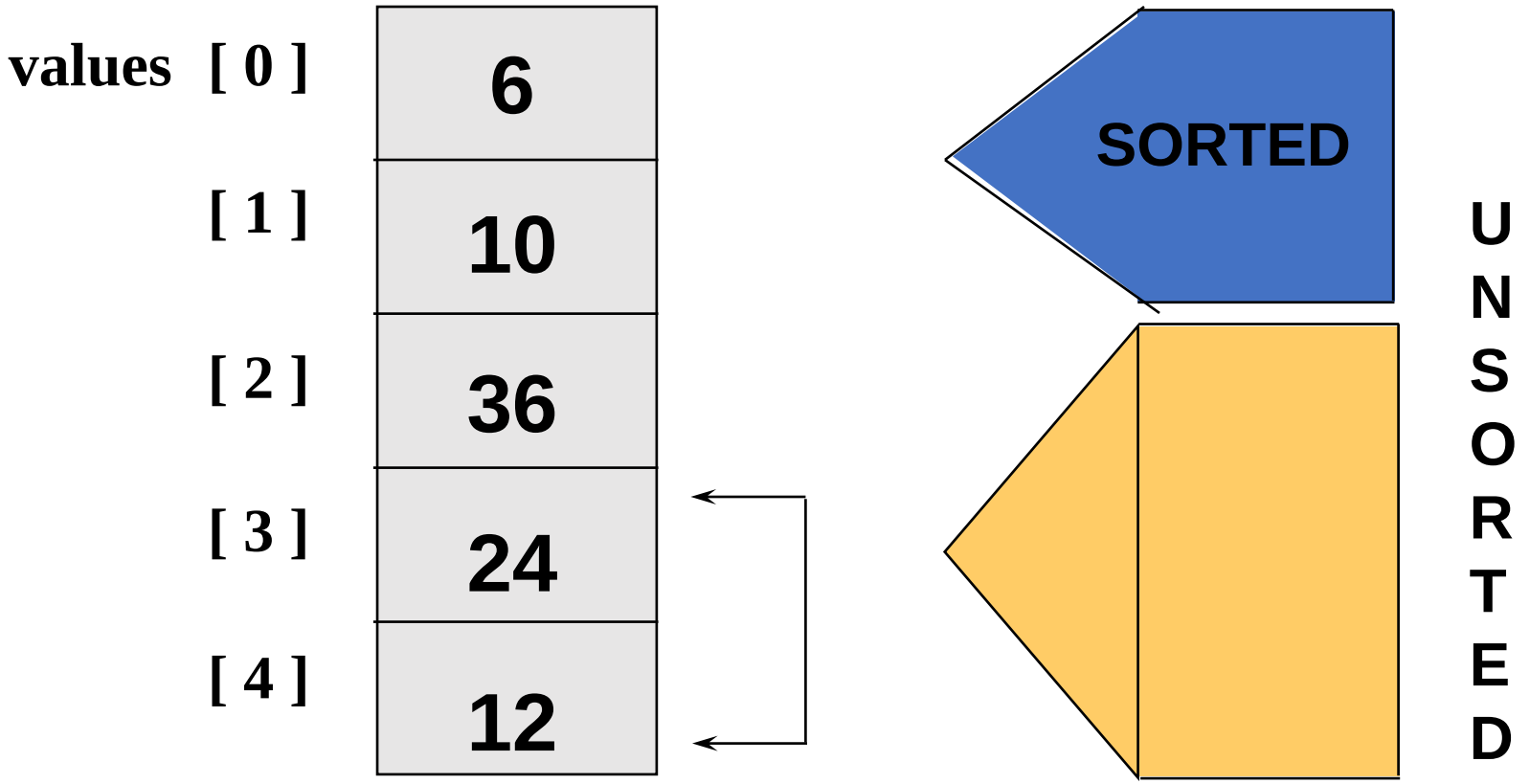
Bubble Sort: Pass Two



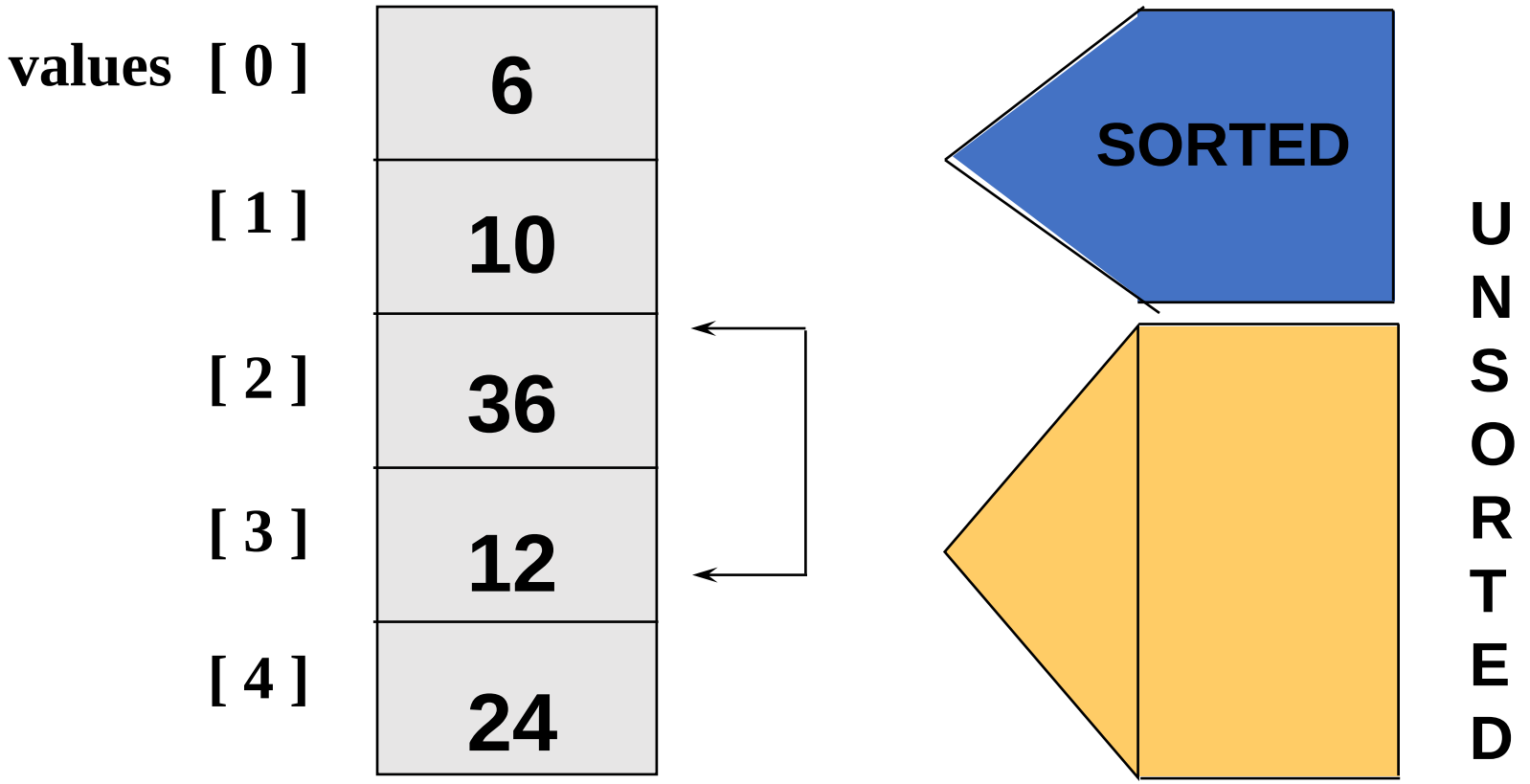
Bubble Sort: End Pass Two



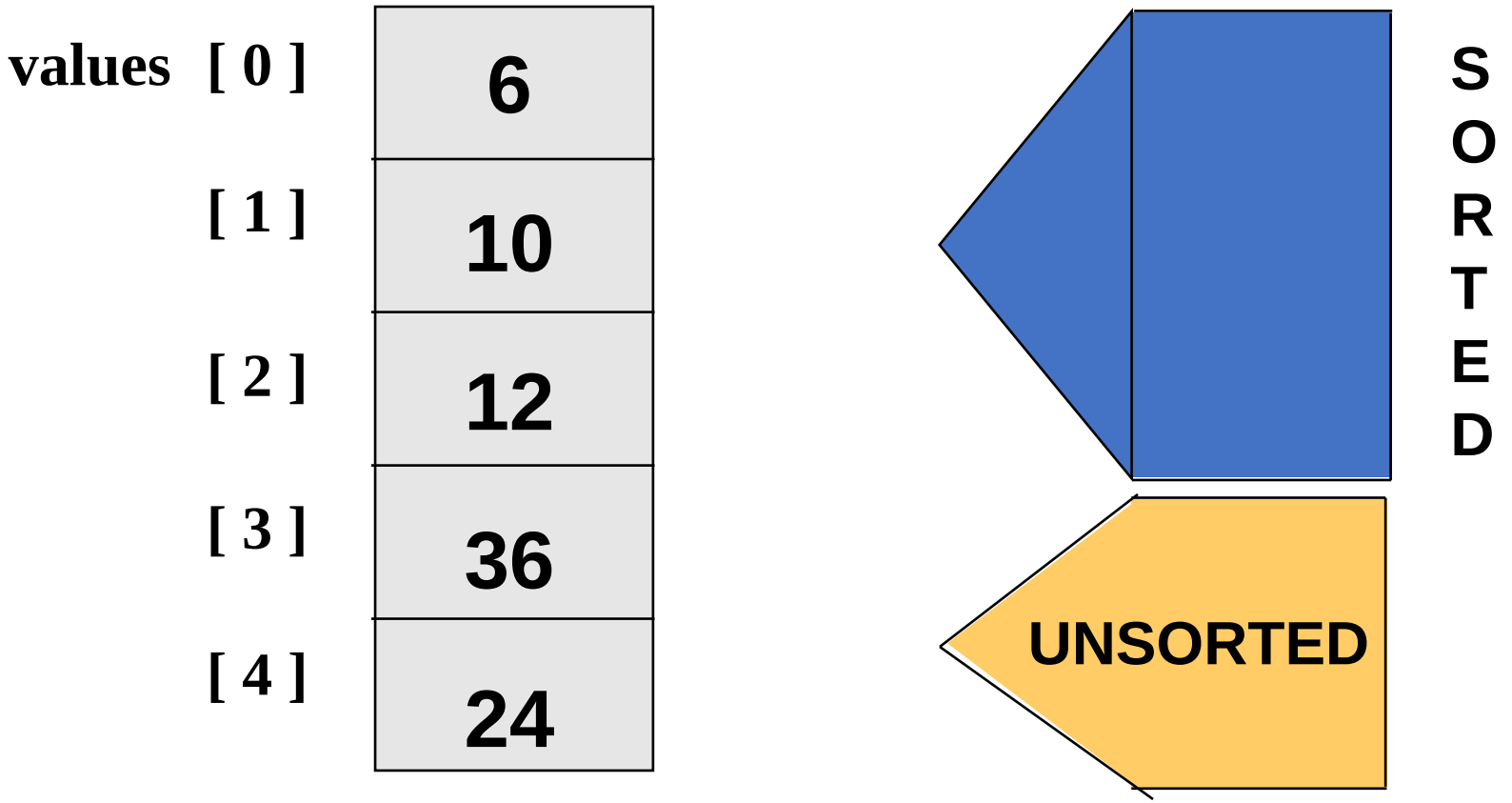
Bubble Sort: Pass Three



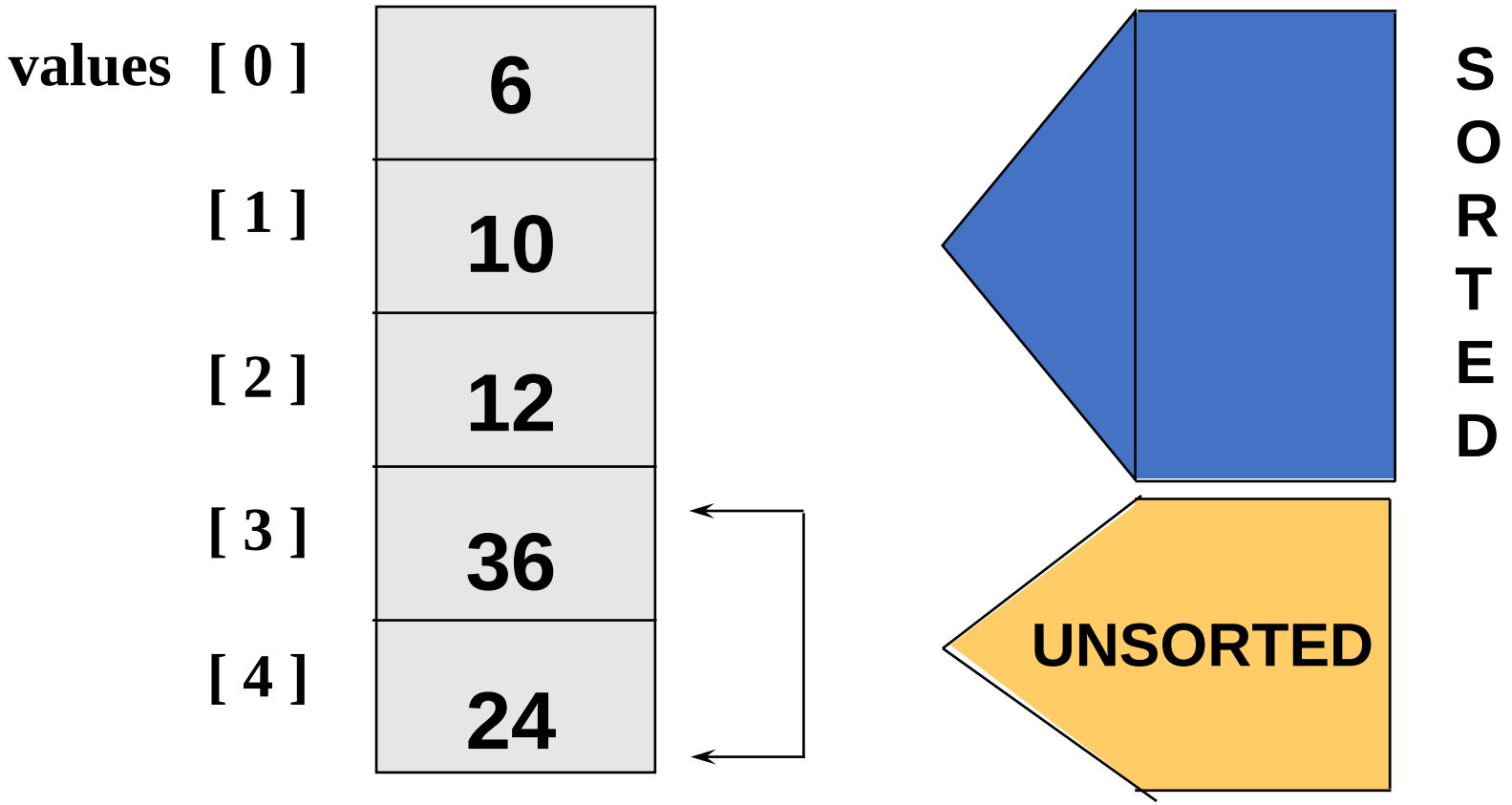
Bubble Sort: Pass Three



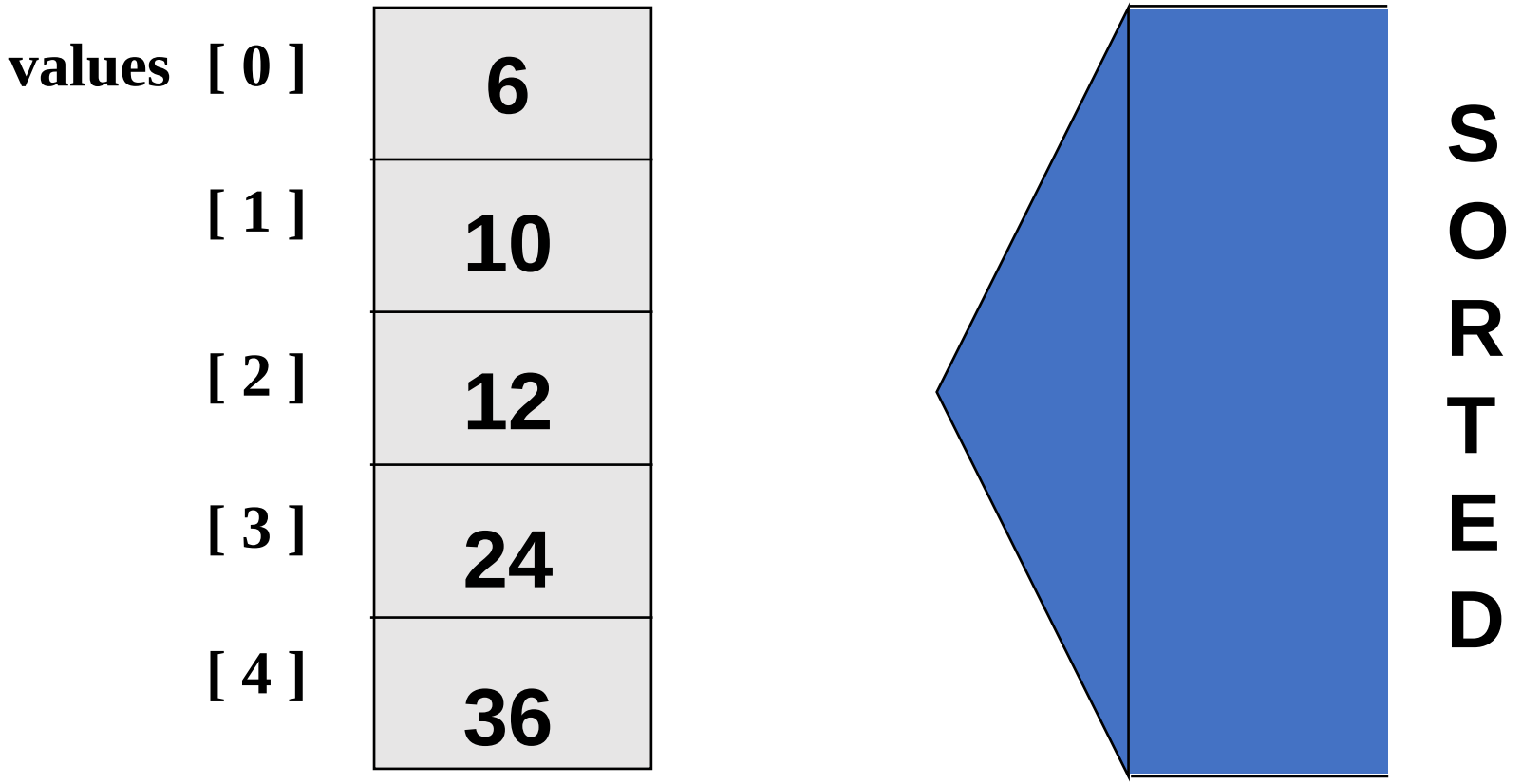
Bubble Sort: End Pass Three



Bubble Sort: Pass Four



Bubble Sort: End Pass Four



Bubble sort in Python example

```
List = [29,8,7,6,5,4,3,2,1]
for i in range(0,len(List)-1):
    for j in range(0,len(List)-1-i):
        if List[j]>List[j+1]:
            List[j],List[j+1]=List[j+1],List[j]
    print List
```

Bubble Sort (Recursive Version)

```
def bubbleSort(arr):
    n = len(arr)
    # Traverse through all list elements
    for i in range(n-1):
    # range(n) also work but outer loop will repeat one time more than needed.
            # Last i elements are already in place
            for j in range(0, n-i-1):
                    # traverse the array from 0 to n-i-1
                    # Swap if the element found is greater
                    # than the next element
                    if arr[j] > arr[j+1] :
                            arr[j], arr[j+1] = arr[j+1], arr[j]

# Now the Program to test above function
arr = [64, 34, 25, 12, 22, 11, 90]
bubbleSort(arr)
print ("Sorted list is:")
for i in range(len(arr)):
    print (arr[i]),
```

Problem

- Results of VIT entrance exam has been released. Given the details of the students such as name, address and marks scored in entrance, write a program to sort the student details so that it will be convenient to call for counselling.

Test Case

Input	Output
2 (No. of Students)	[('John', 'Chennai', 99.0), ('Mark', 'Bangalore', 80.0)]
John (Name)	
Chennai (Place)	
99 (Score)	
Mark (name)	
Bangalore (Place)	
80(score)	