Three $O(n^2)$ Sorting Algorithms

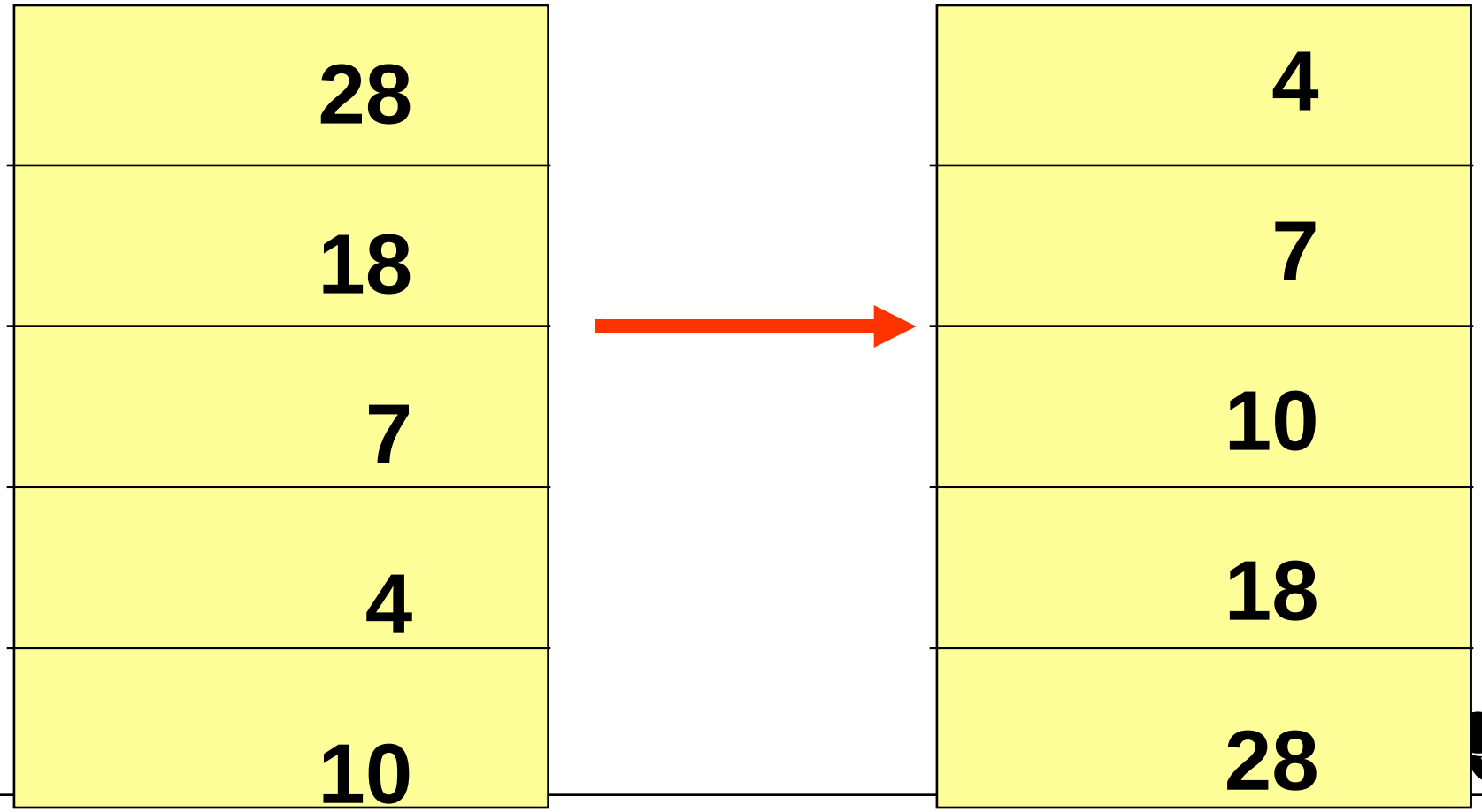
October 26, 2009

Why do we need to sort things?

- Internal Telephone Directory
  - sorted by department then by name
- My local video store holds more than 4,000 movies
  - how can I find "The Incredibles"

Sorting Integers

How to sort the integers in this array?



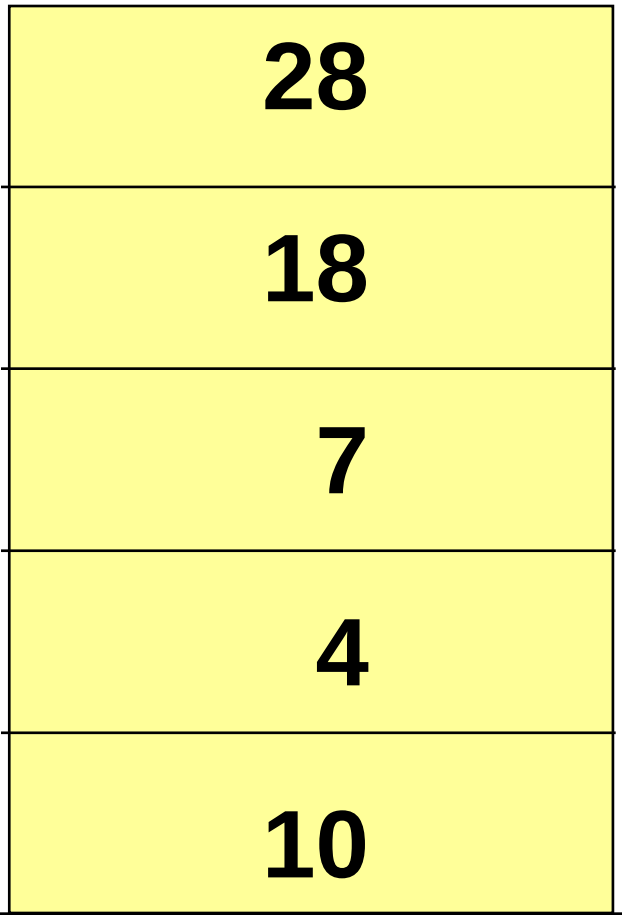
Elementary Sorting Algorithms

- Selection Sort
- Insertion Sort
- Bubble Sort

Selection Sort

- Main idea:
  - find the smallest element
  - put it in the first position
  - find the next smallest element
  - put it in the second position
  - ...

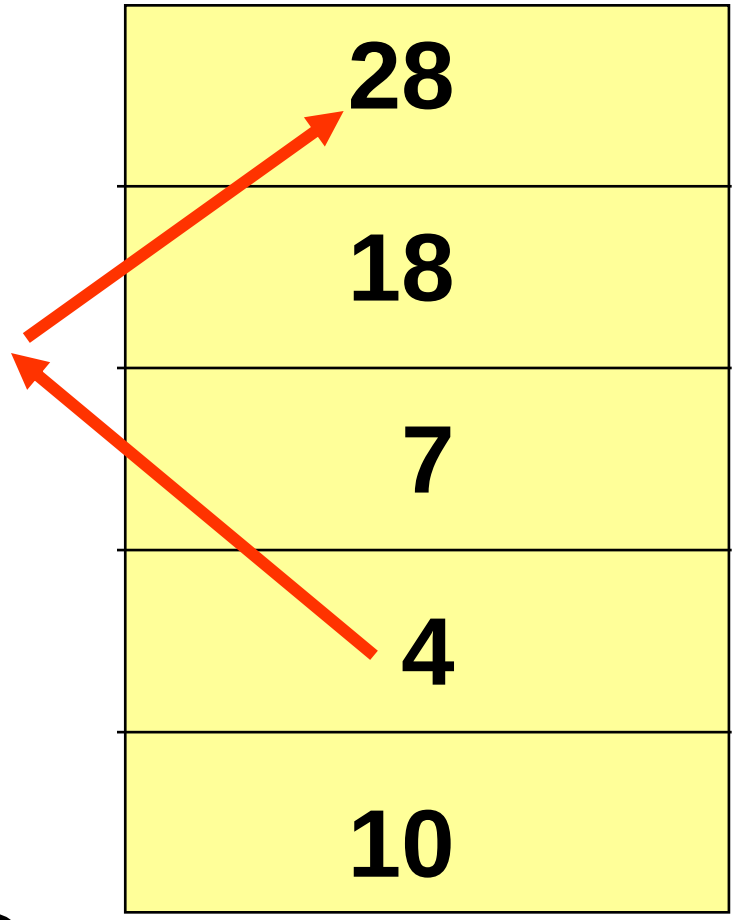
Straight Selection Sort

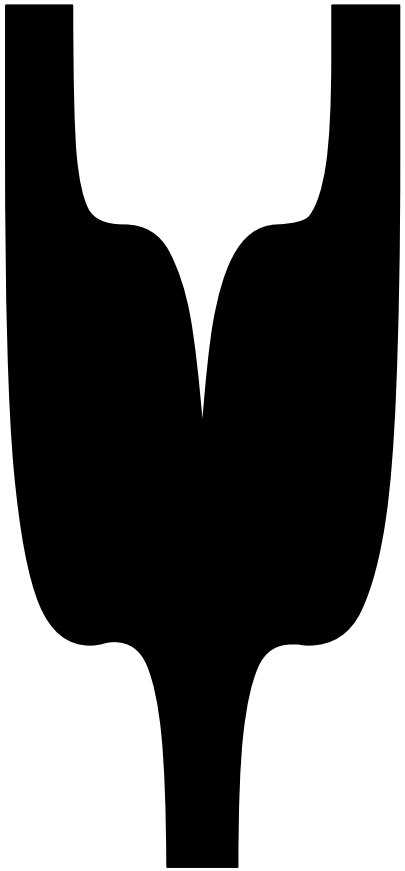


The algorithm splits the array into two parts: already sorted, and not yet sorted.

On each pass, the algorithm locates the smallest of the unsorted elements, and moves into the correct position

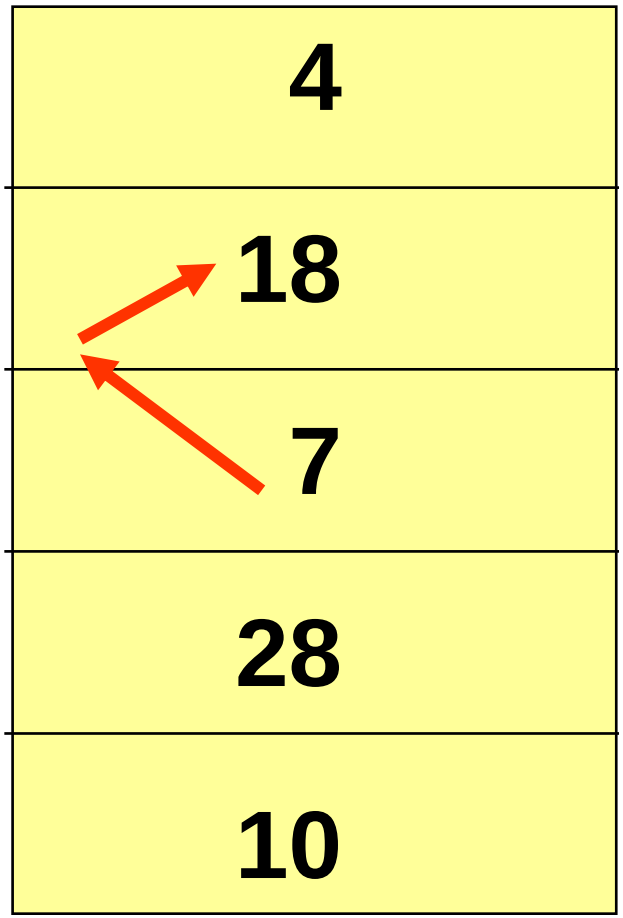
Selection Sort: Pass One

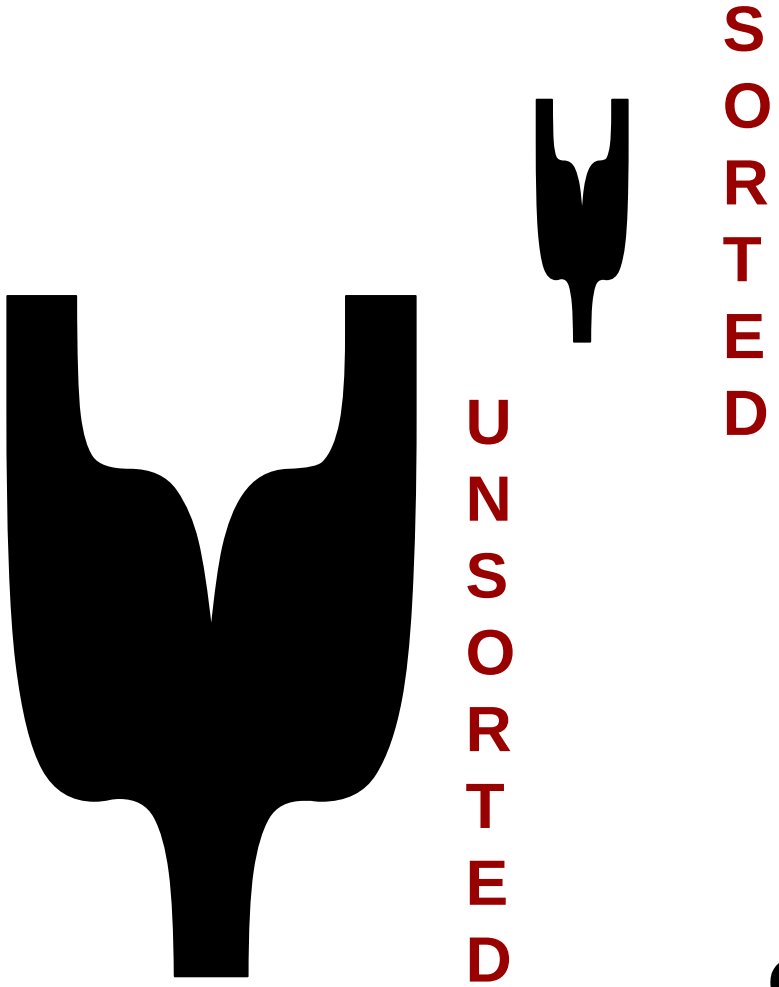




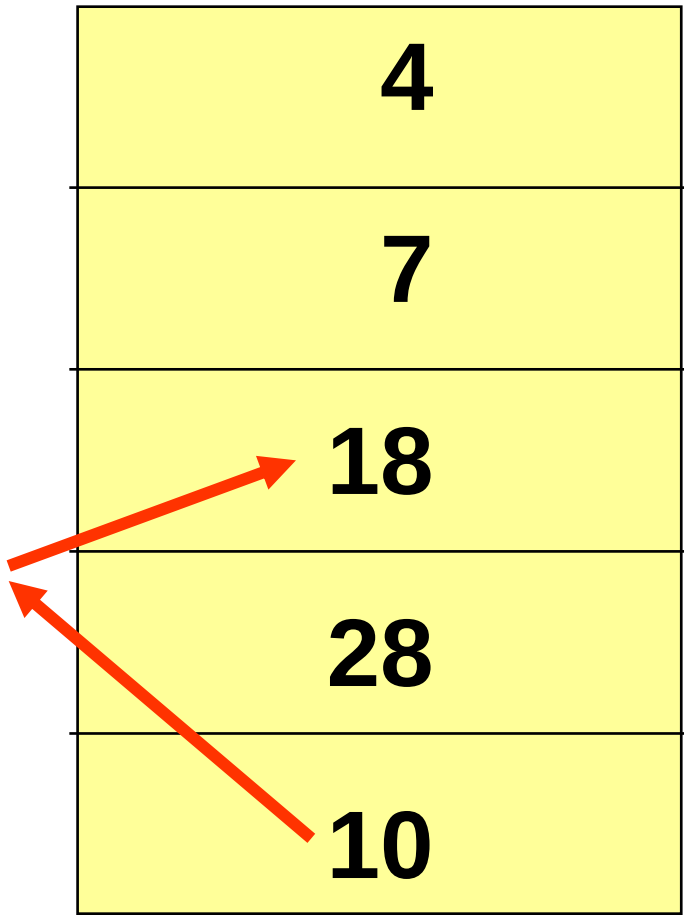
UNSORTED

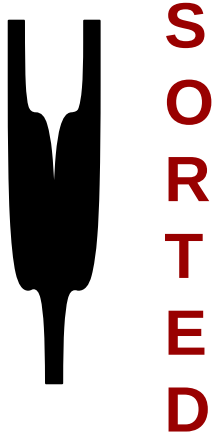
Selection Sort: End Pass One



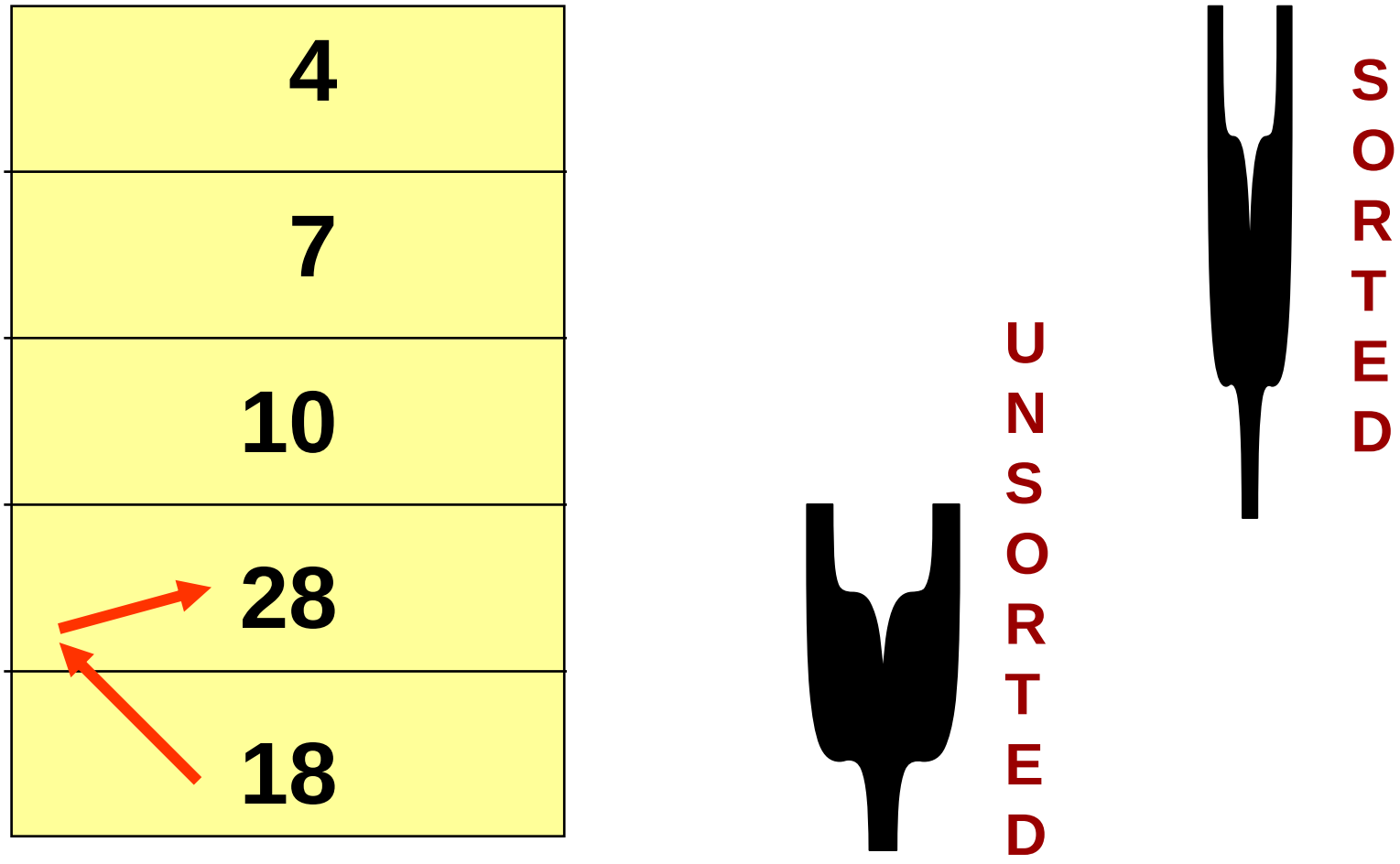


Selection Sort: End Pass Two



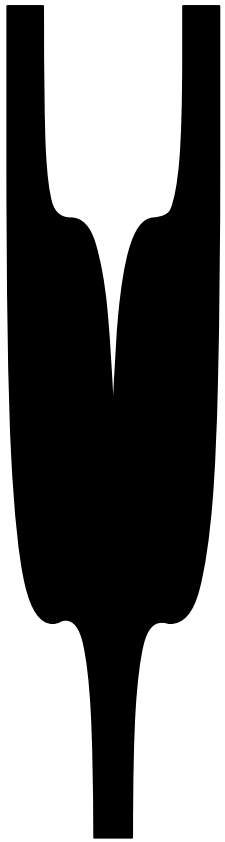


Selection Sort: Pass Three



Selection Sort: End Pass Three

4
7
10
18
28



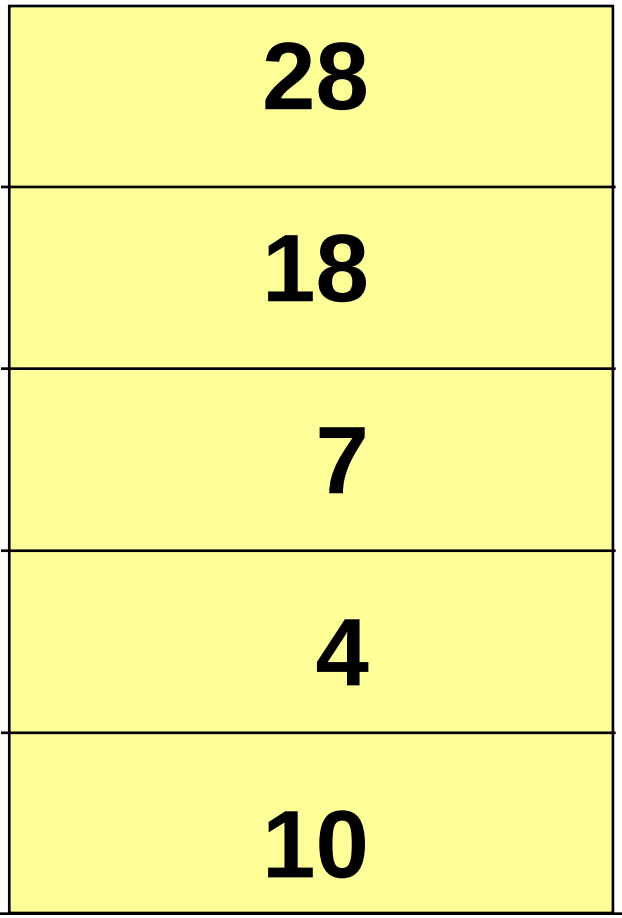
SORTED

Insertion Sort

Main Idea:

- Starts by considering the first two elements of the array data, if out of order, swap them
- Consider the third element, insert it into the proper position among the first three elements.
- Consider the forth element, insert it into the proper position among the first four elements.
- … …

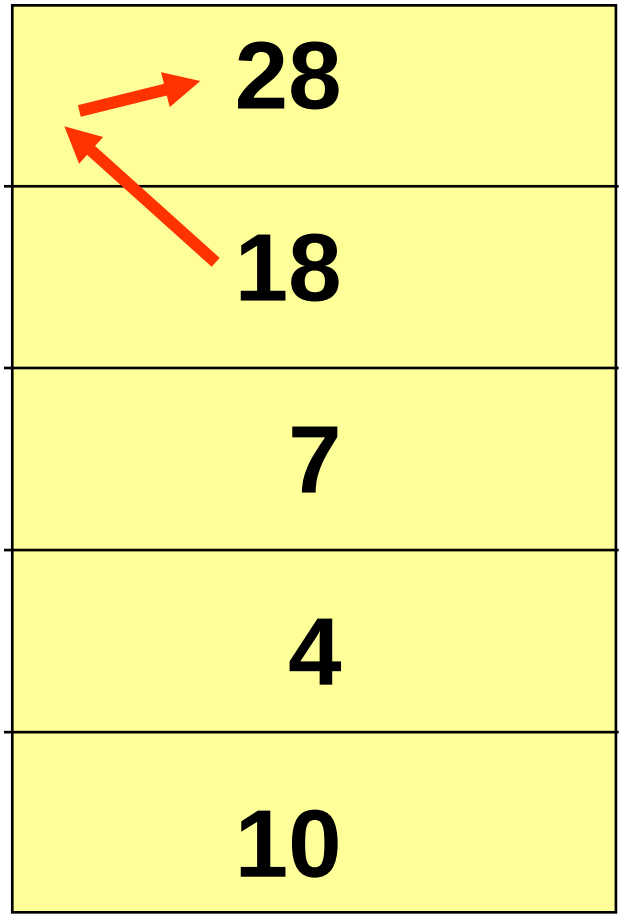
Insertion Sort

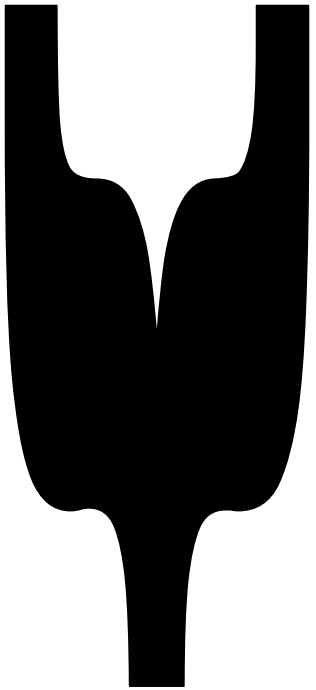


One by one, each as yet unsorted array element is inserted into its proper place with respect to the already sorted elements.

On each pass, this causes the number of already sorted elements to increase by one.

Insertion Sort: Pass One

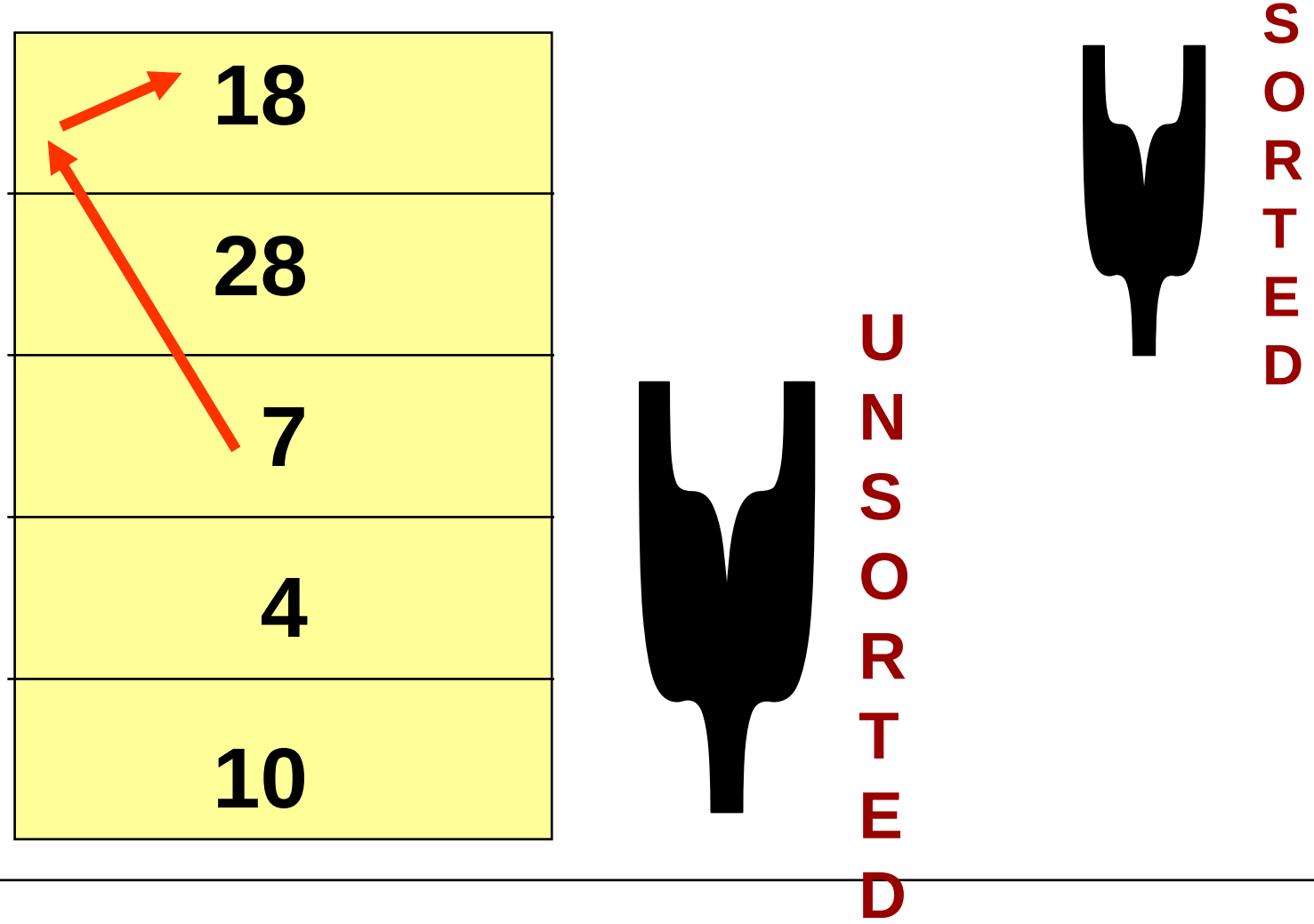




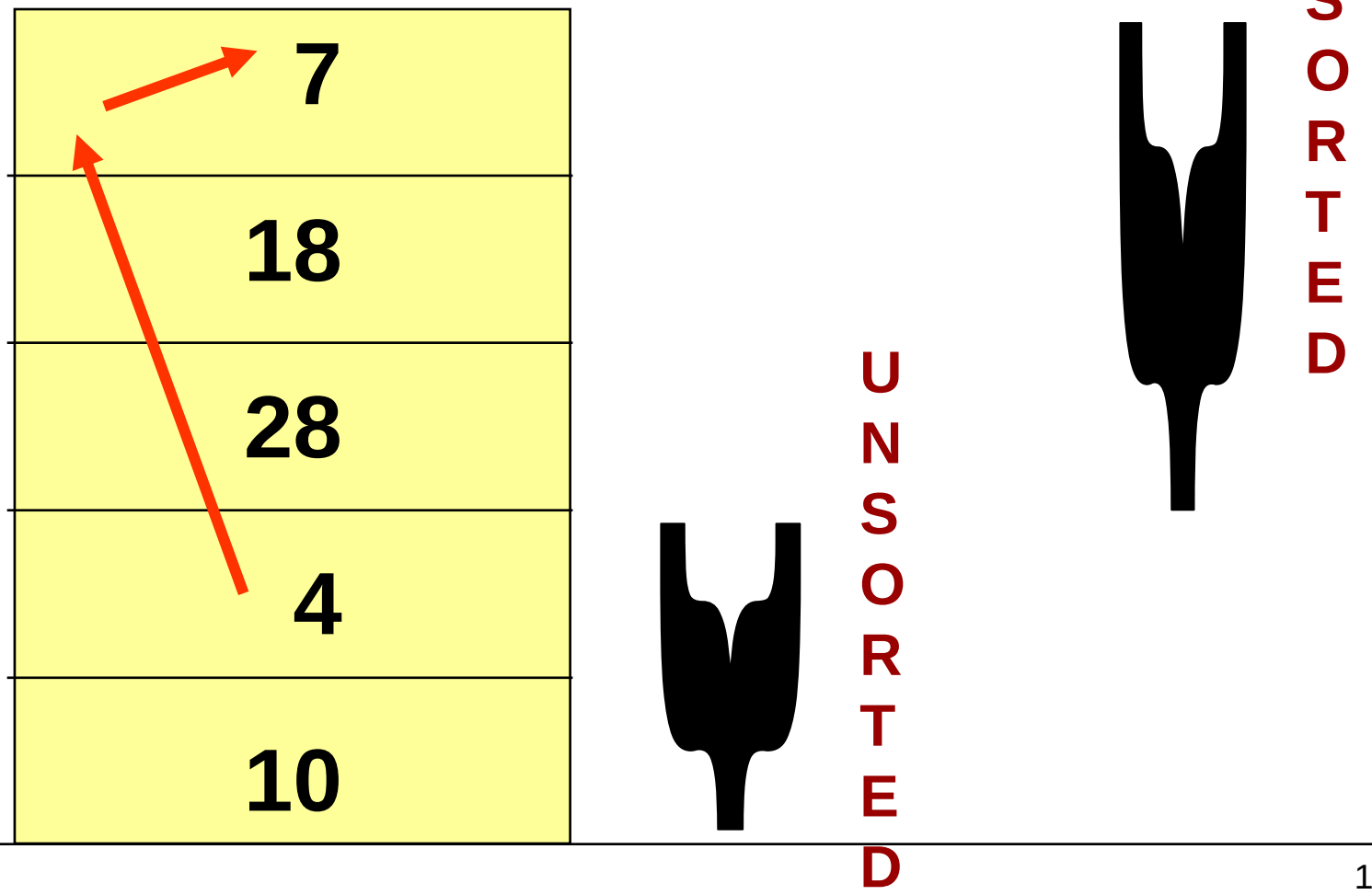
UNSORTED

SORTED

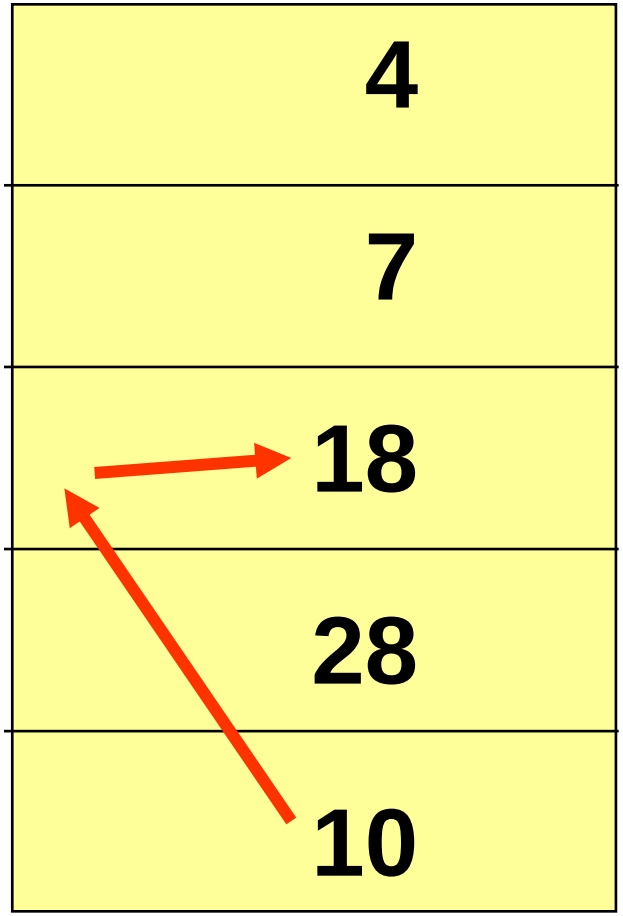
Insertion Sort: Pass Two



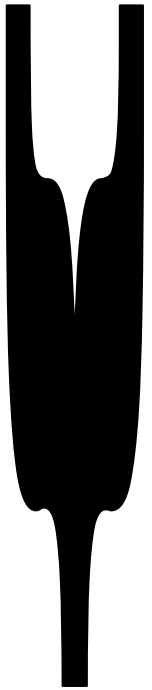
Insertion Sort: Pass Three



Insertion Sort: Pass Four

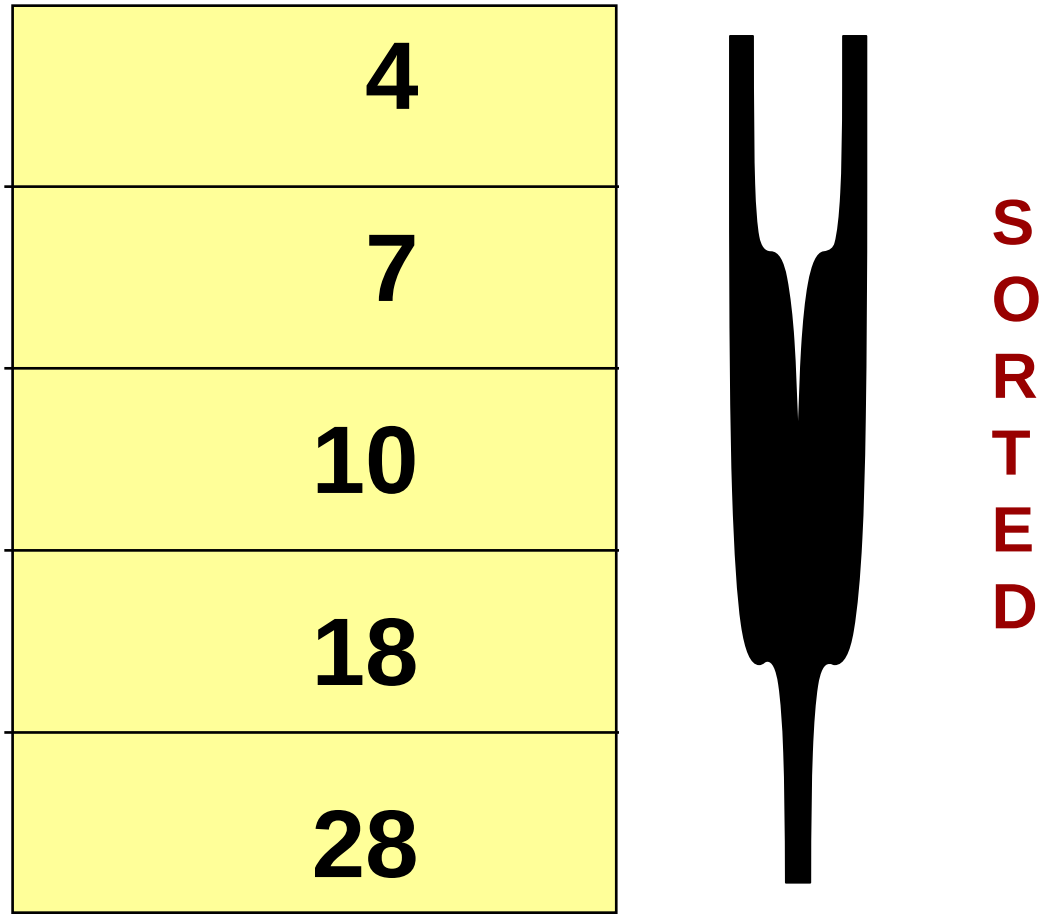


UNSORTED



SORTED

Insertion Sort: Pass Five



Asymptotic Complexity of Insertion Sort

- **$O(n^2)$**
- What does this mean?

Complexity of Insertion Sort

- Time or number of operations does not exceed $\mathbf{c.n^2}$ on any input of size **n** (**n** suitably large).
- Actually, the worst-case time is Theta($\mathbf{n^2}$) and the best-case is Theta(**n**)
- So, the worst-case time is expected to quadruple each time **n** is doubled

Complexity of Insertion Sort

- Is $\mathbf{O(n^2)}$ too much time?
- Is the algorithm practical?

Practical Complexities

10^9 instructions/second

n	*n*	*nlogn*	n^2	n^3
1000	1mic	10mic	1milli	1sec
10000	10mic	130mic	100milli	17min
10^6	1milli	20milli	17min	32years

Impractical Complexities

10^9 instructions/second

n	n^4	n^{10}	2^n
1000	17min	3.2×10^{13} years	3.2×10^{283} years
10000	116 days	*???*	*???*
10^6	3×10^7 years	*??????*	*??????*

Faster Computer Vs Better Algorithm

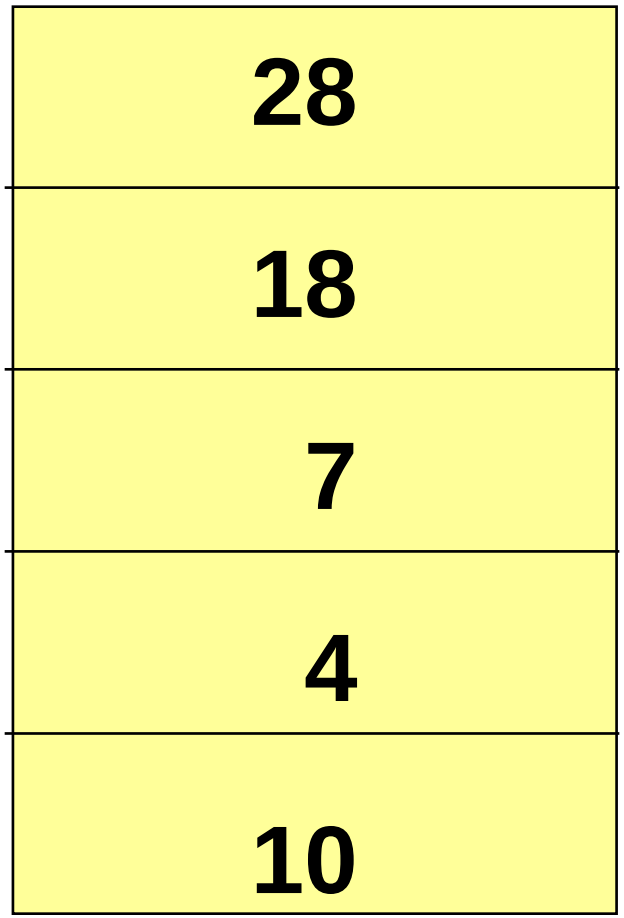
Algorithmic improvement more useful than hardware improvement.

E.g. 2^n to n^3

Bubble Sort

- The bubble sort works by comparing each item in the list with the item next to it, and swapping them if required.
- The algorithm repeats this process until it makes a pass all the way through the list without swapping any items (in other words, all items are in the correct order).
- This causes larger values to "bubble" to the end of the list while smaller values "sink" towards the beginning of the list.

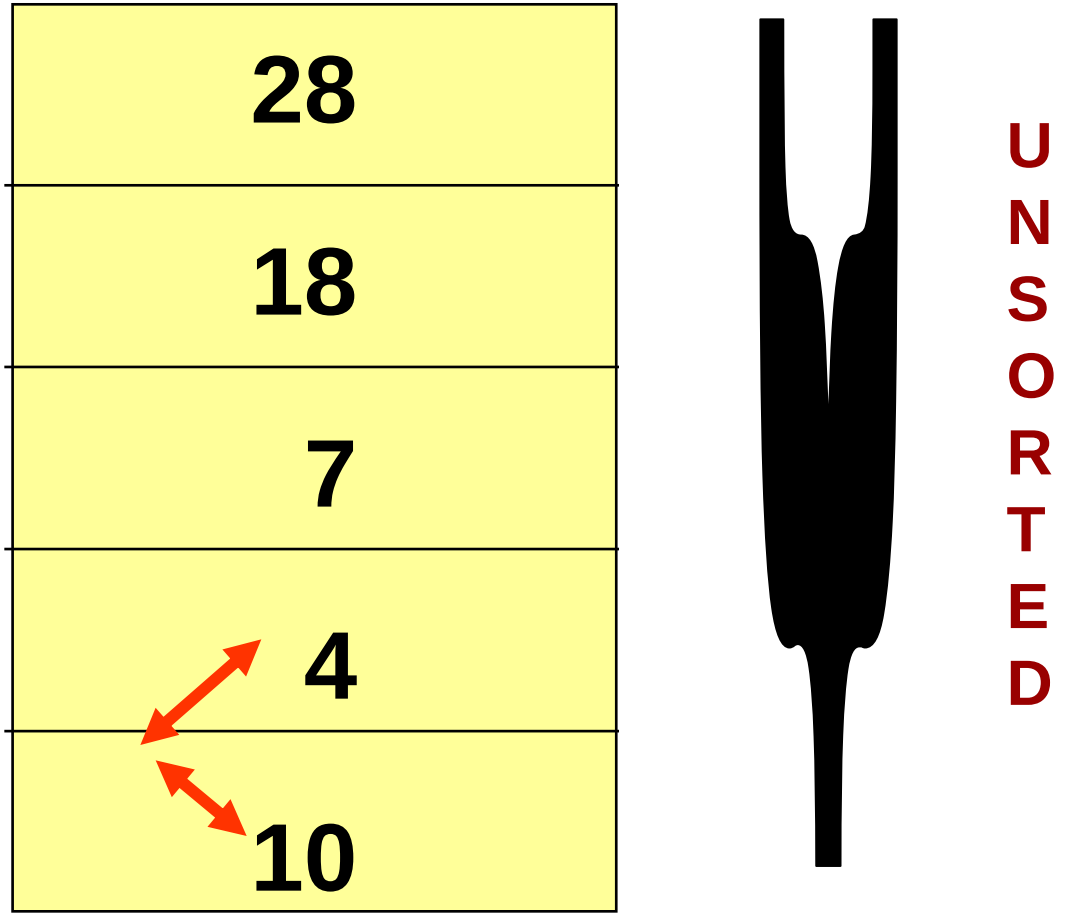
Bubble Sort



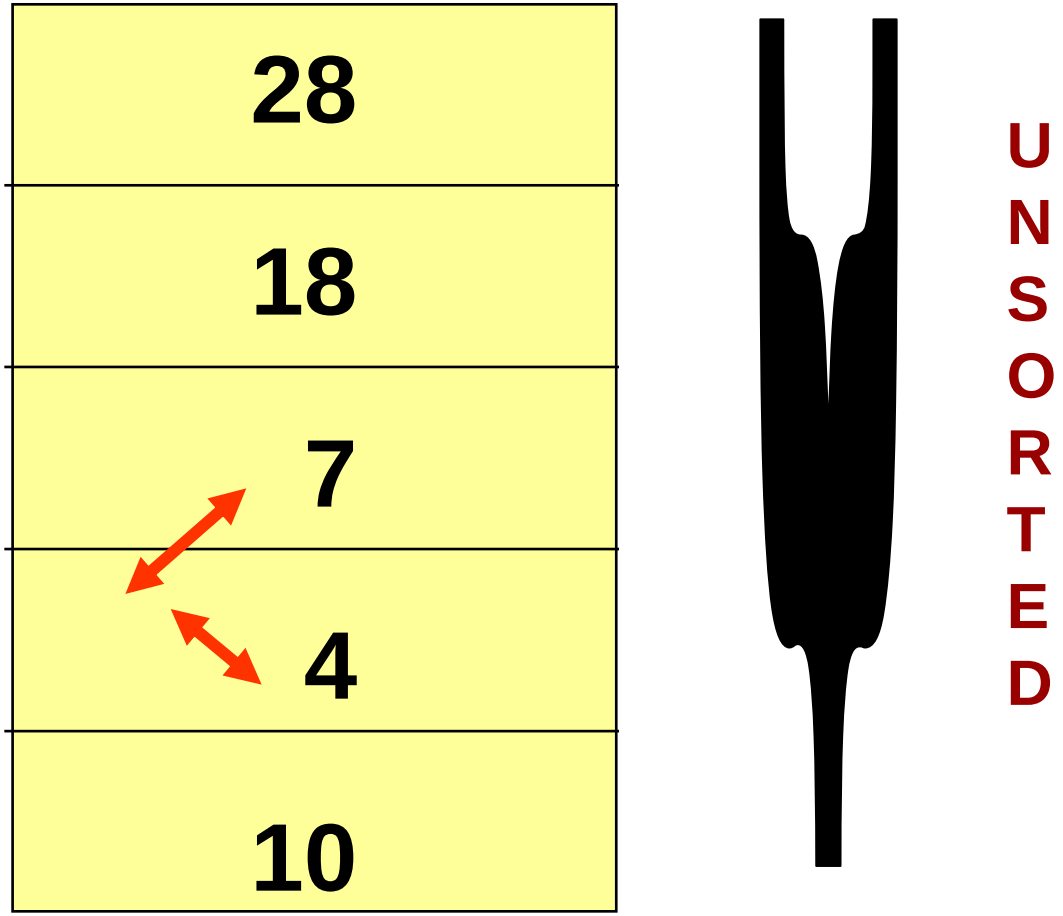
Compares neighboring pairs of array elements, starting with the last array element, and swaps neighbors whenever they are not in correct order.

On each pass, this causes the smallest element to "bubble up" to its correct place in the array.

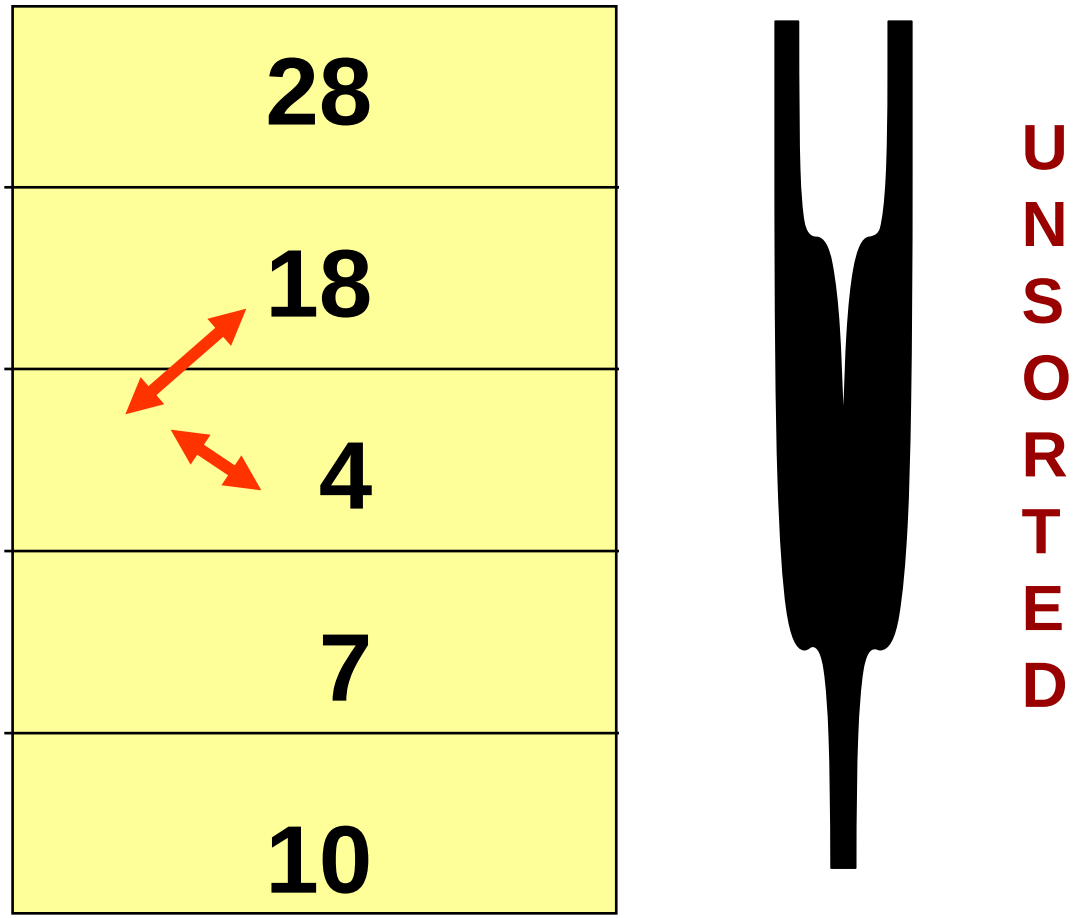
Bubble Sort: Pass One



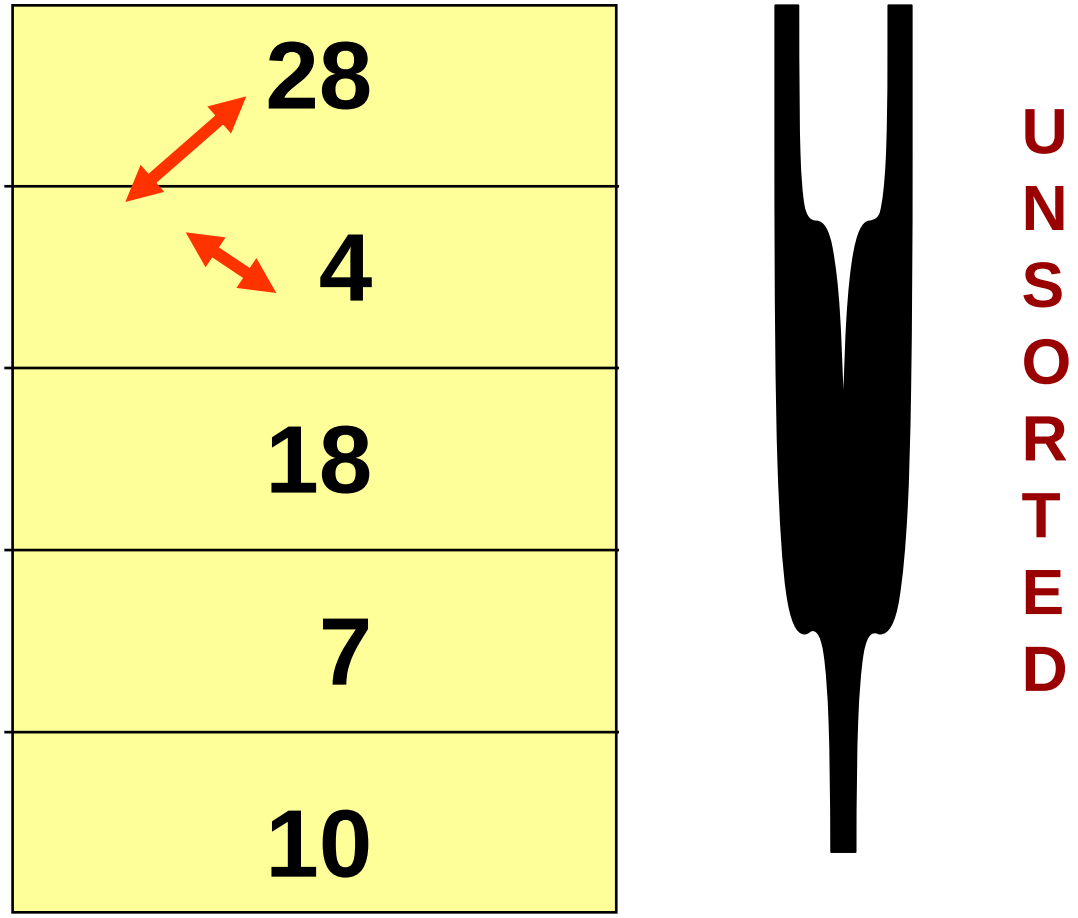
Bubble Sort: Pass One



Bubble Sort: Pass One

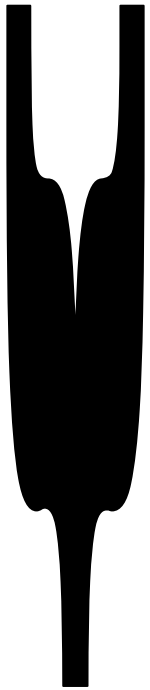


Bubble Sort: Pass One



Bubble Sort: End Pass One

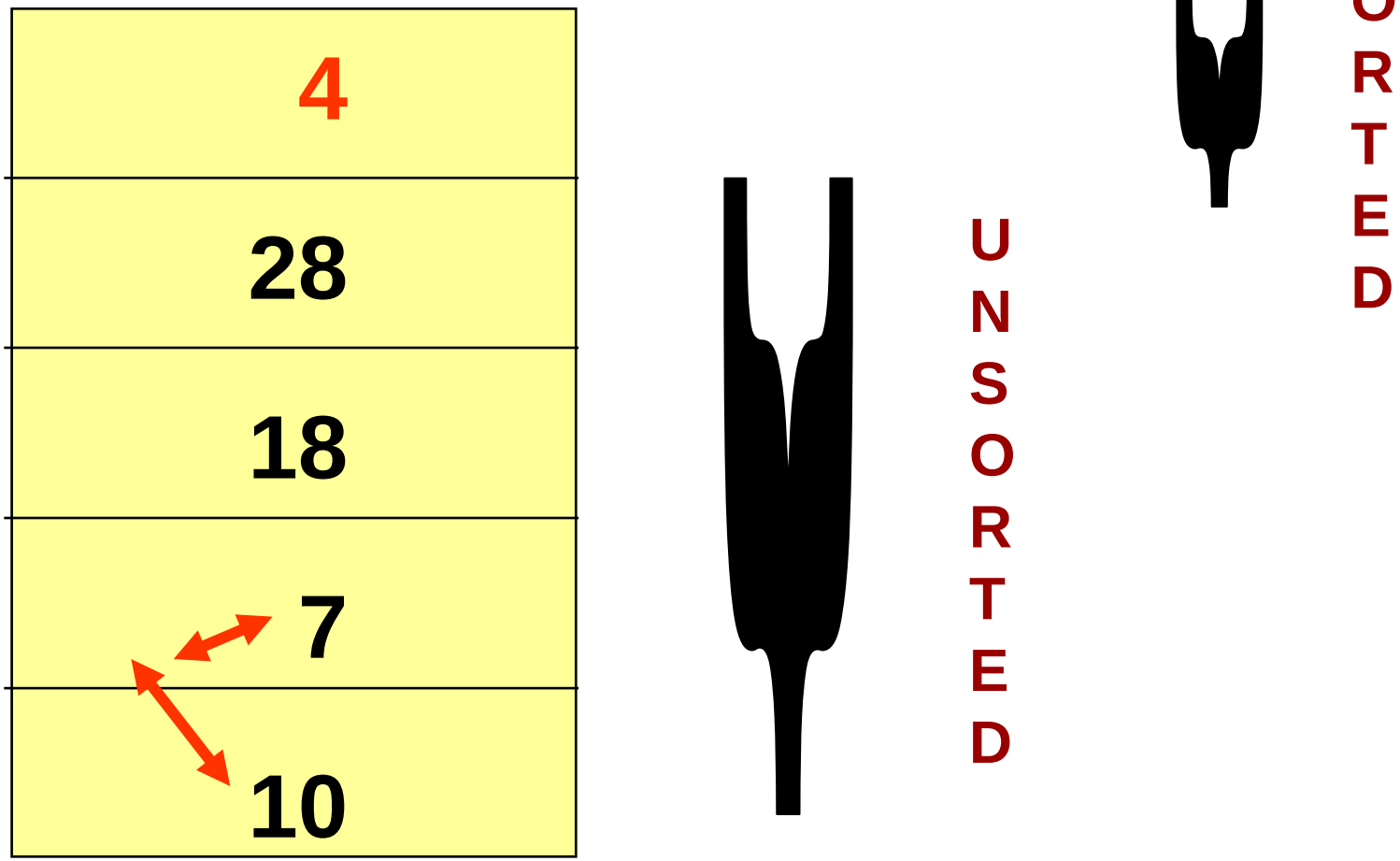
4
28
18
7
10



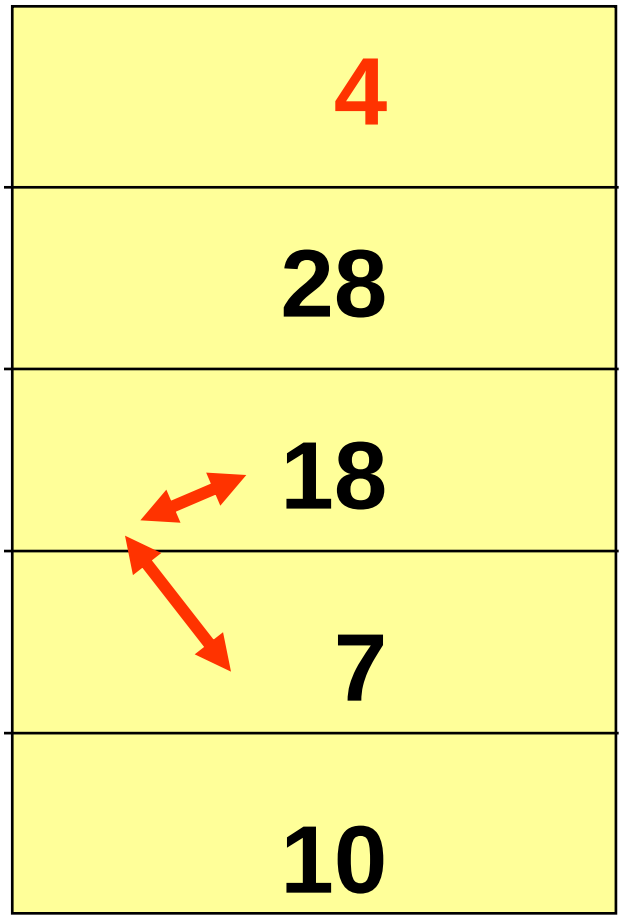
UNSORTED

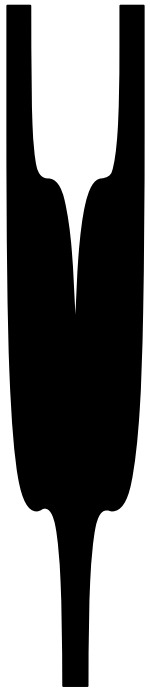
SORTED

Bubble Sort: Pass Two



Bubble Sort: Pass Two

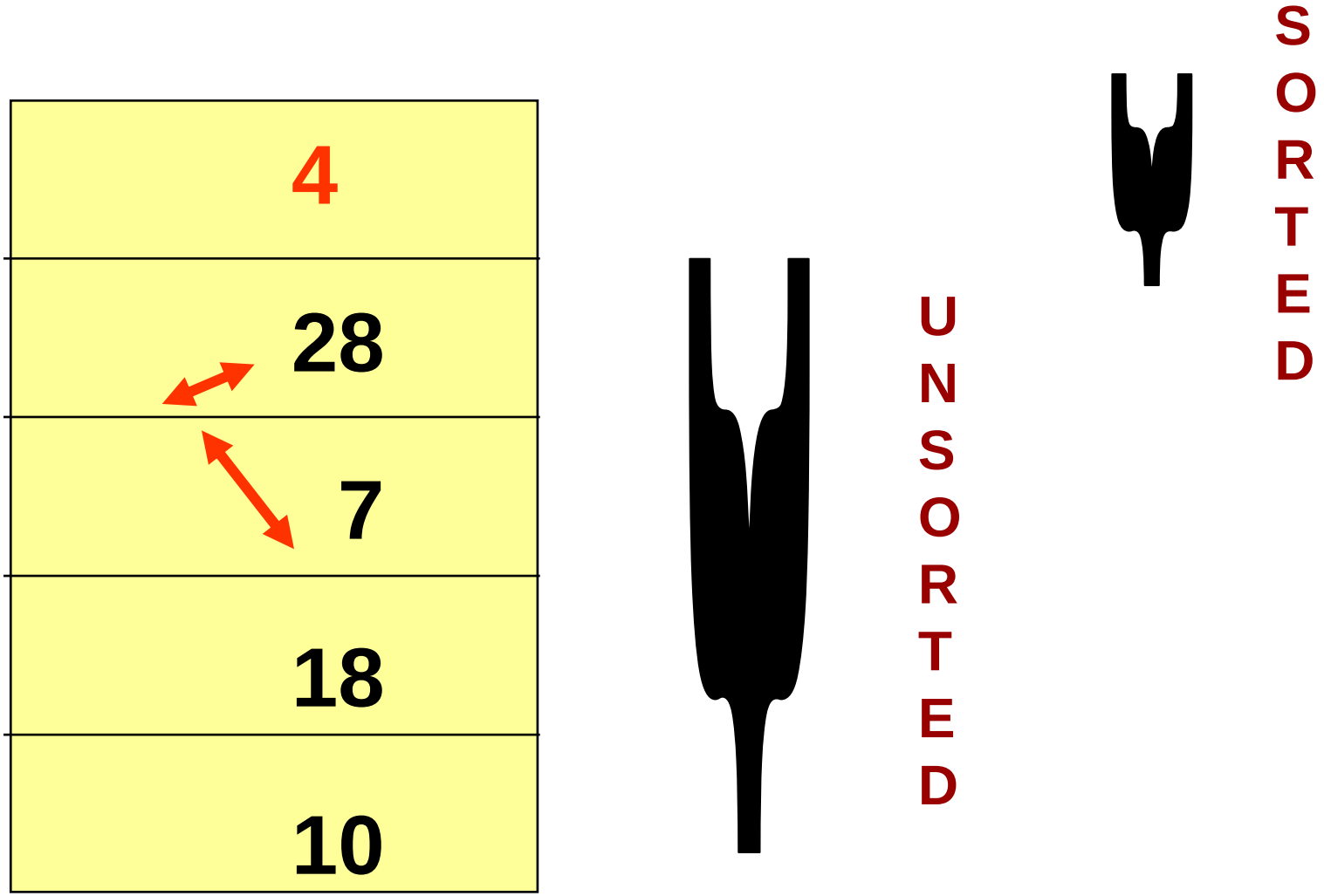




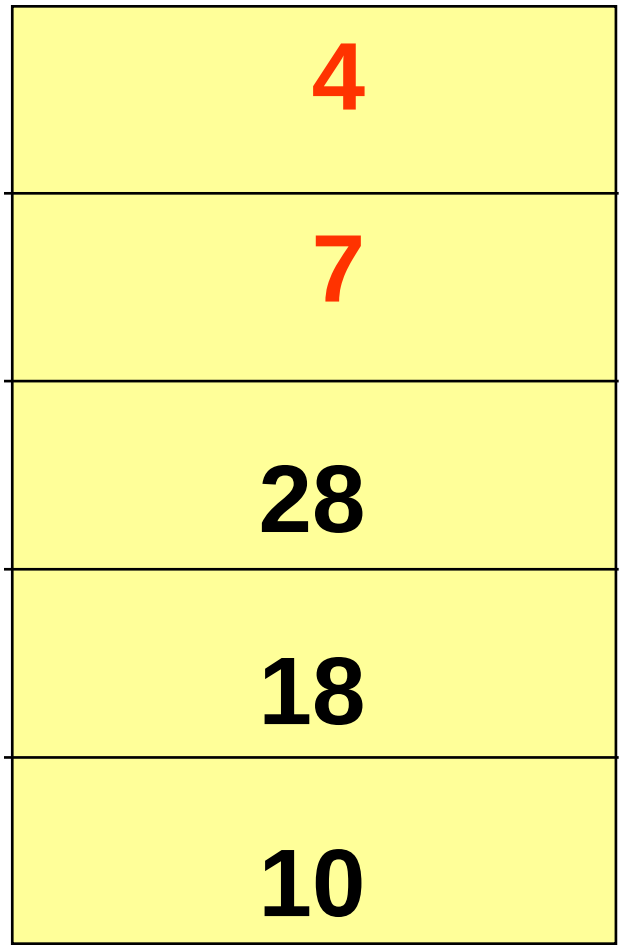
UNSORTED

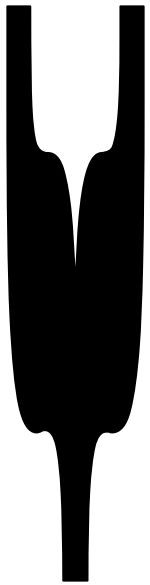
SORTED

Bubble Sort: Pass Two

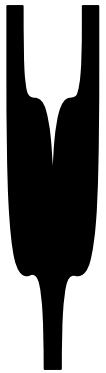


Bubble Sort: End Pass Two



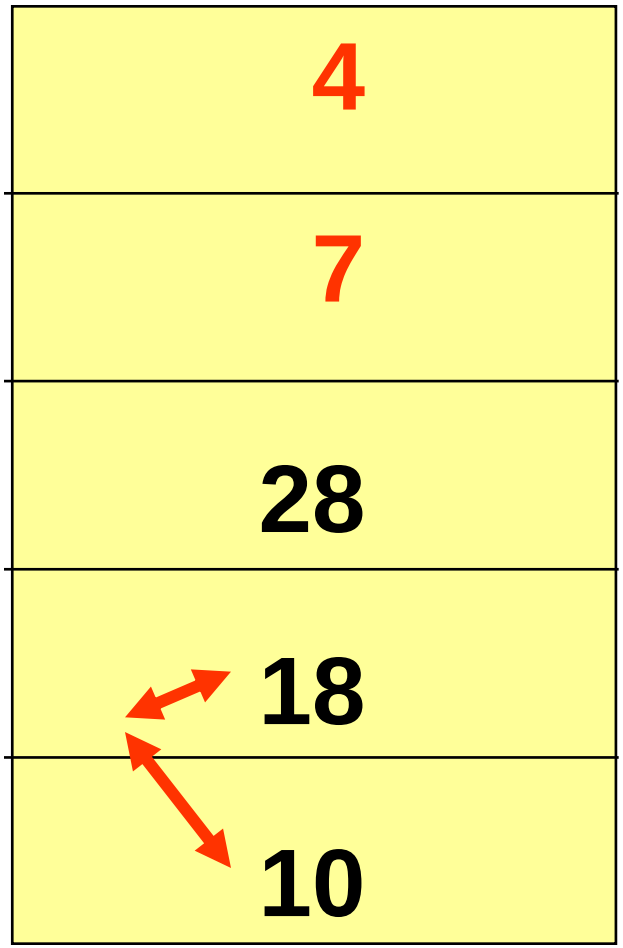


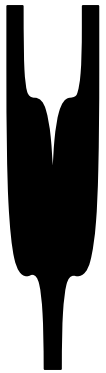
UNSORTED



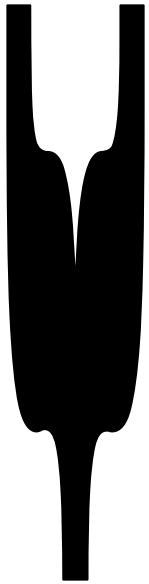
SORTED

Bubble Sort: Pass Three



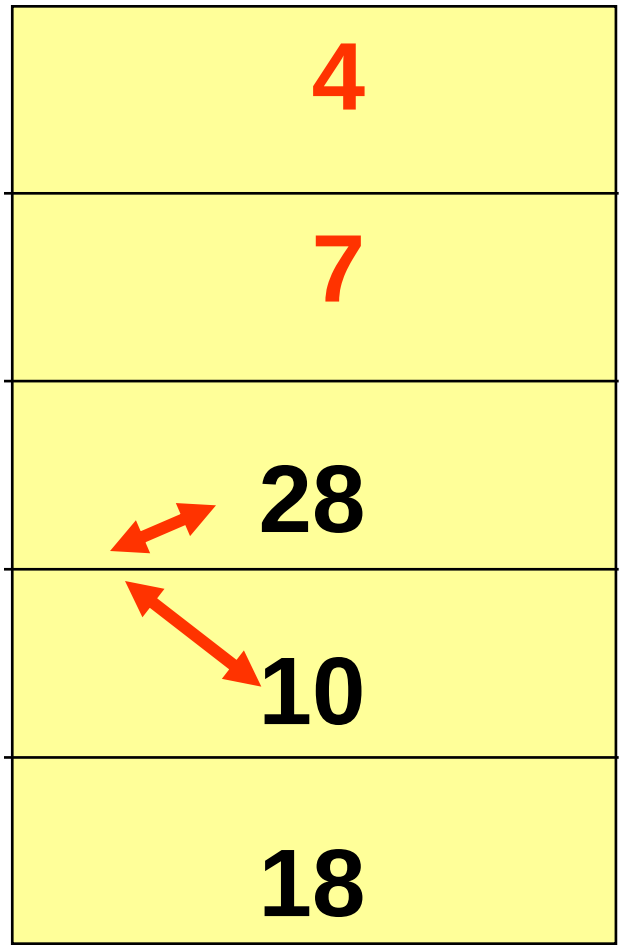


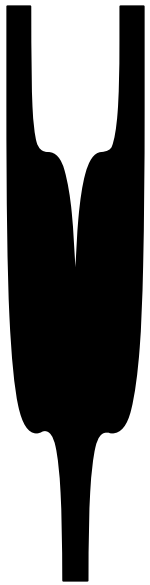
SORTED



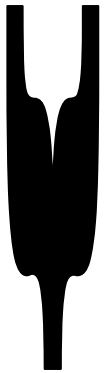
UNSORTED

Bubble Sort: Pass Three



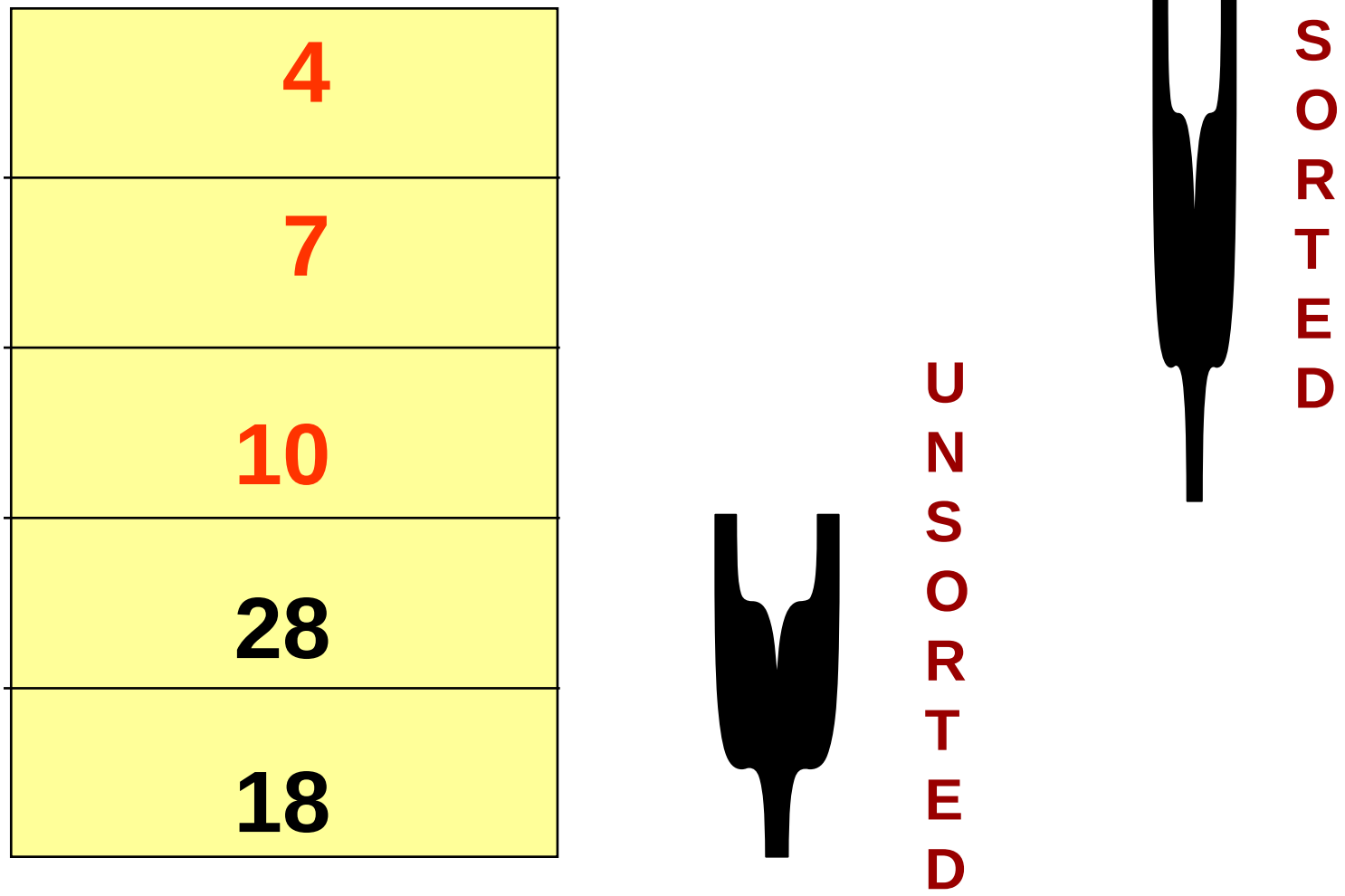


UNSORTED

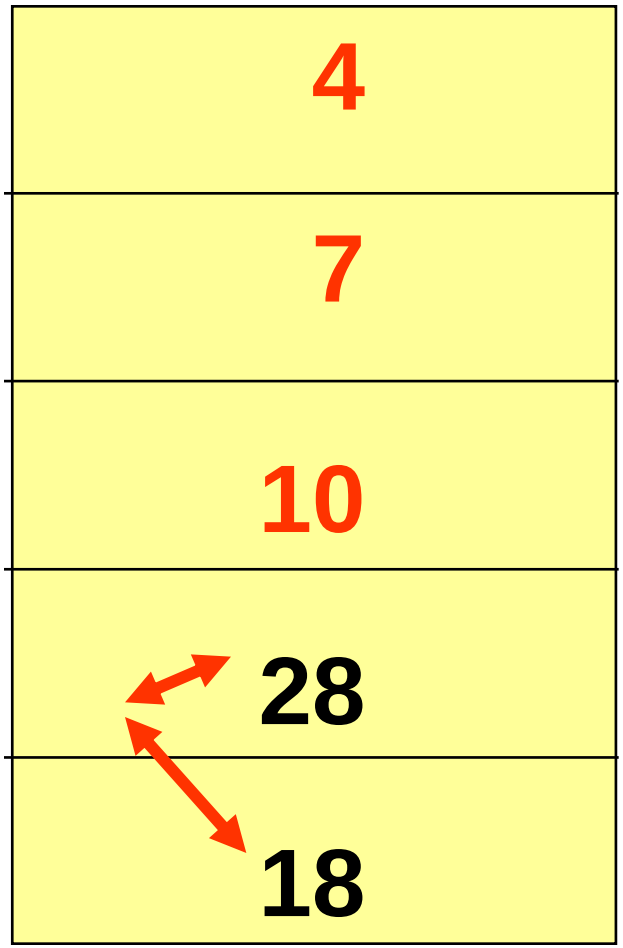


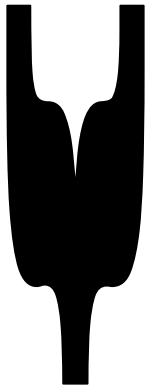
SORTED

Bubble Sort: End Pass Three

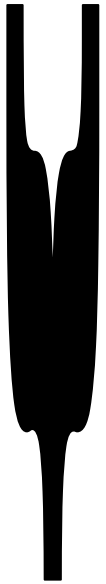


Bubble Sort: Pass Four



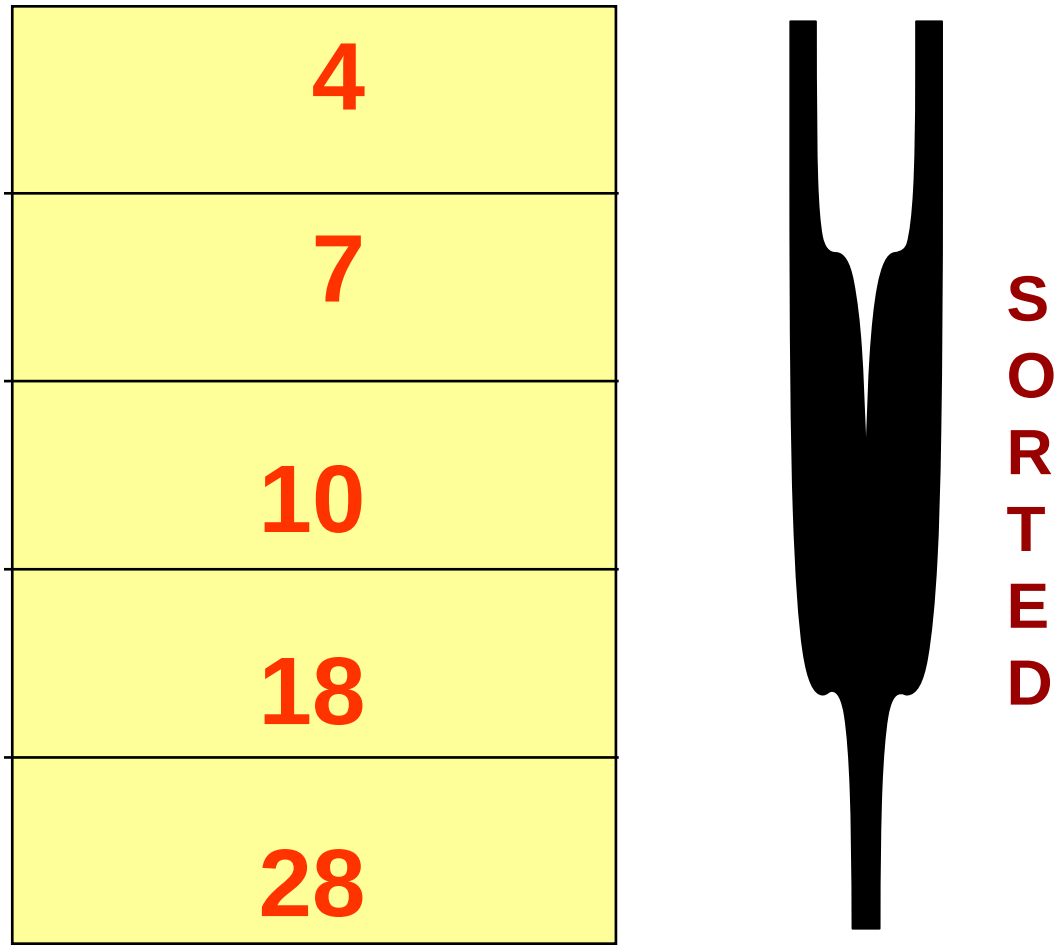


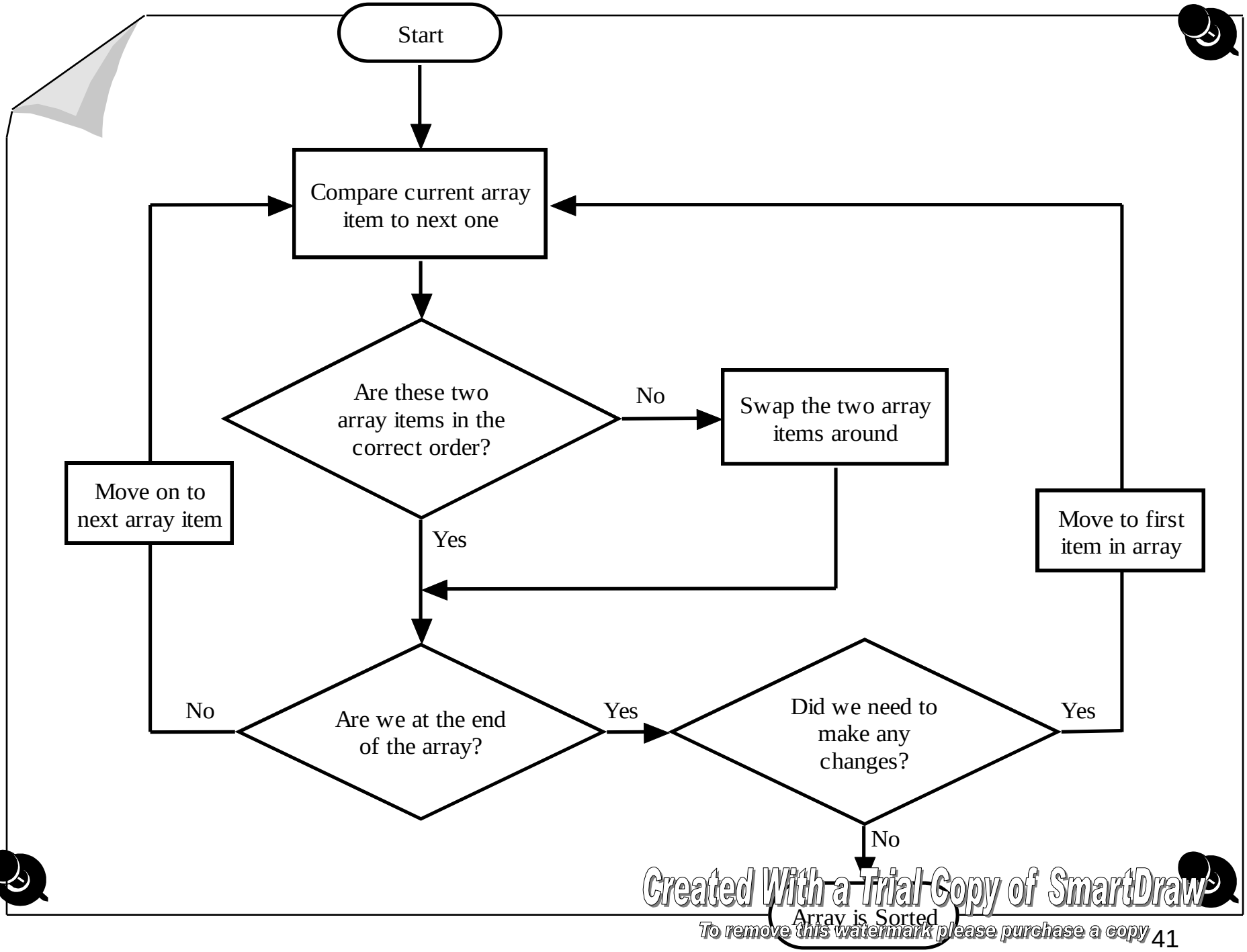
UNSORTED



SORTED

Bubble Sort: End Pass Four





A Program Incorporating a Bubble Sort

```
BEGIN
     clrscr;

     ReadMyArray;
     SortMyArray;
     DisplayMyArray
END.
```

```
PROCEDURE ReadMyArray;

     VAR count : integer;

     BEGIN

          Writeln ('How many numbers will your provide? ');
          readln (size);
          For Count := 1 to size do
               BEGIN
                         readln(MyArray[Count]);

               END;

     END;
```

```
BEGIN
     clrscr;

     ReadMyArray;
     SortMyArray;
     DisplayMyArray
END.
```

```
PROCEDURE SortMyArray;

     VAR  i, j, tmp : integer;

     BEGIN


     (* Sort using bubble sort. *)
     FOR i := size - 1 DOWNTO 1 DO
          FOR j := 1 TO i DO
               IF MyArray[j] > MyArray[j + 1] THEN BEGIN
                    tmp := MyArray[j];
                    MyArray[j] := MyArray[j + 1];
                    MyArray[j + 1] := tmp;
               END;

     END;
```

```
BEGIN
     clrscr;

     ReadMyArray;
     SortMyArray;
     DisplayMyArray
END.
```

```
PROCEDURE DisplayMyArray;
     VAR count : integer;
     BEGIN
          writeln ('The current contents of MyArray are as follows: ');
          FOR Count := 1 TO size DO
               writeln(MyArray[Count])
     END;
```