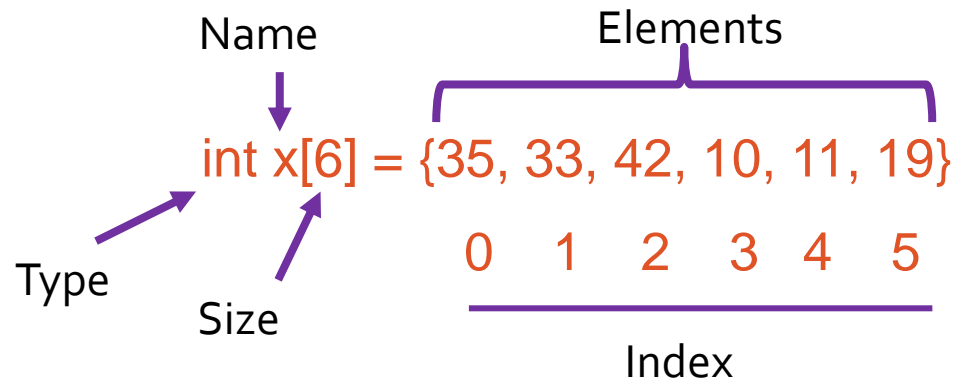# DATA STRUCTURES & ALGORITHMS

Arrays

Instructor: Engr. Laraib Siddiqui

Arrays

- Collection of cells of the same type.

- Cells are numbered with consecutive integers

- Declaration: int x[6];

- Access: use the array name and an index:
  x[0], x[1], x[2], x[3], x[4], x[5]

Name

Elements

int x[6] = {35, 33, 42, 10, 11, 19}

0   1   2   3   4   5

Type

Size

Index

Memory Layout

x[0]

x[1]

x[2]

x[3]

x[4]

x[5]

Declaring array

Syntax: data_type array_name [array_size];

- If we are declaring 'x' is an array name, 'x' is not a value.

- For example,

  int a, b;

  we can write

  b = 2;
  a = b;

  But we cannot write

  2 = a; // not allowed

```
int x[6];
int n;

x[0] = 5;
x[1] = 2;

x = 3;        // not allowed
x = a + b;    // not allowed
x = &n;       // not allowed
```

What's special about arrays?

Constant time access

3

Traversing an array

LA is a linear array with lower bound LB and upper bound UB.

1. [Initialize counter] Set K := LB.

2. Repeat steps 3 and 4 while K <= UB.

3. [Visit element] Apply PROCESS to LA[K].

4. [Increment counter] Set K := K + 1.
   [End of Step 2 loop.]

5. Exit.

Complexity

Traversing an array

e.g Array {8,9,5,12}

for (i=0; i<4; i++)

{

      printf ("%d\t", a[i]);

}

             8   9   5   12

Insertion in arrays

INSERT (LA, N, K, ITEM)

linear array with N elements and K is a positive integer such that K <= N.

Following algorithm inserts an element ITEM into the Kth position in LA.

1.[Initialize counter.] Set J := N.

2.Repeat Steps 3 and 4 while J >= K.

3.   [Move Jth element downward.] Set LA[J+1] := LA[J].

4.   [Decrease counter.] Set J := J – 1.
   [End of Step 2 loop.]

5.[Insert element.] Set LA[K] := ITEM.

6.[Reset N.] Set N := N + 1.

7.Exit.

Complexity

Insertion in arrays

25	14	56	16	38

Value to be inserted **28** in
Index position **2**

Values	25	14	28	56	16	38
Index	0	1	2	3	4	5

Deletion

DELETE (LA, N, K, ITEM)

array with N elements and K is a positive integer such that K <= N.

Following algorithm deletes the Kth element from LA.

1. Set ITEM := LA[K].

2. Repeat for J = K to N - 1.
	[Move J + 1st element upward.] Set LA[J] := LA[J + 1].
   [End of loop.]

3. [Reset the number N of elements in LA.] Set N := N - 1.

4. Exit.

Deletion

25	14	28	56	16	38

Value to be deleted **28** from
Index position **2**

Values	25	14	56	16	38
Index	0	1	2	3	4

Dynamic Arrays

Problem:
- Fixed size
- Process of different data sets with same array is not possible
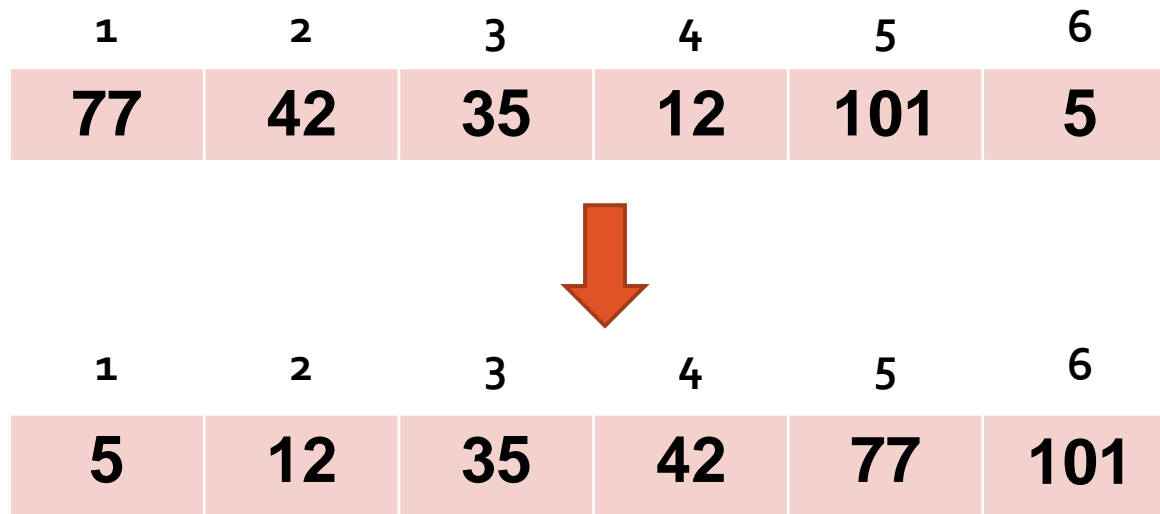
Solution:
- Dynamic Arrays

Applications

- Accessing and storing sequential data.

- Creating string objects temporary.

- I/O routines as buffers

- Dynamic programming

- Lookup tables

Sorting

Sorting can be defined as an operation of rearranging the elements such that:

$$A[1] < A[2] < \ldots\ldots < A[N]$$

1	2	3	4	5	6
77	42	35	12	101	5

1	2	3	4	5	6
5	12	35	42	77	101

Bubble Sort

- Bubble Sort is a simple yet effective sorting algorithm.

- Data is placed adjacent to each other.

- The sort starts from one end (the beginning), compares 2 adjacent data, and swaps them if they are in the wrong order.

- It moves on down the list and continues doing so.

- When it reaches the end of the data, it starts over until all the data is in the right order.

- Its not efficient when dealing with a large set of data.

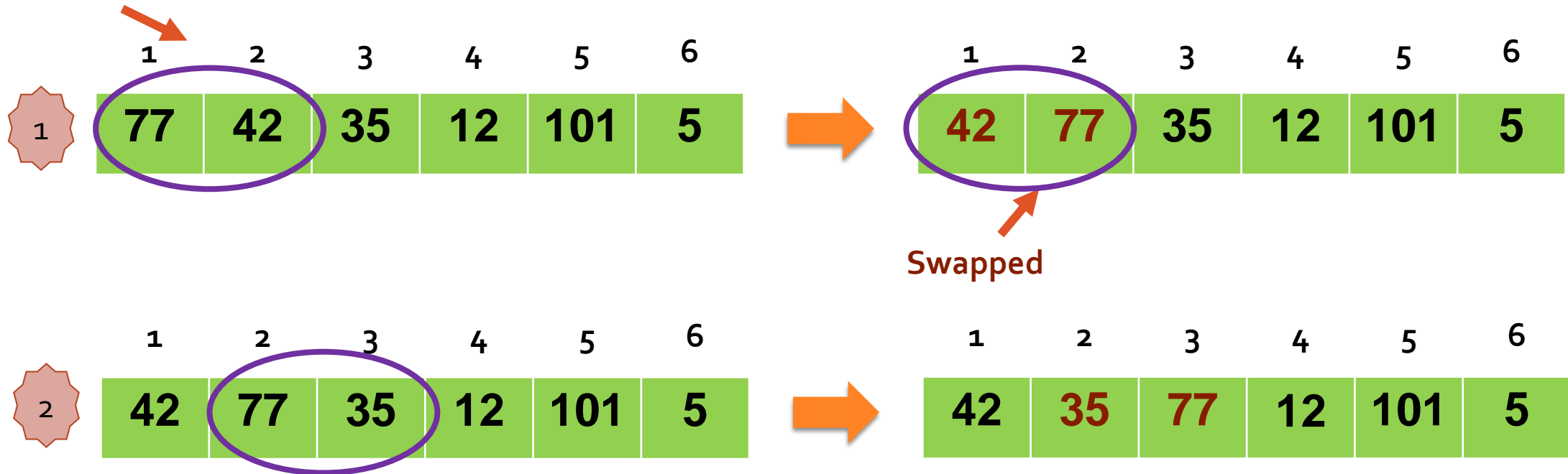- Compared to other sorting algorithm, bubble sort is slow.

Algorithm

1. Repeat steps 2 and 3 for K = 1 to N – 1.

2.   Set PTR:=1. [Initializes pass pointer PTR.]

3.   Repeat while PTR <=N-K:[Executes pass.]
       (a) If DATA[PTR] > DATA[PTR + 1], then:
             Interchange DATA[PTR] and DATA[PTR + 1].
             [End of If structure.]

       (b) Set PTR: = PTR + 1.

             [End of inner loop.]
       [End of step 1 outer loop.]

4. Exit.

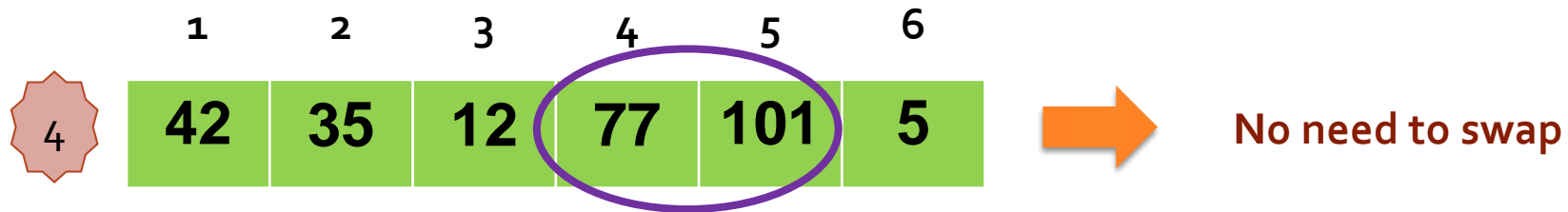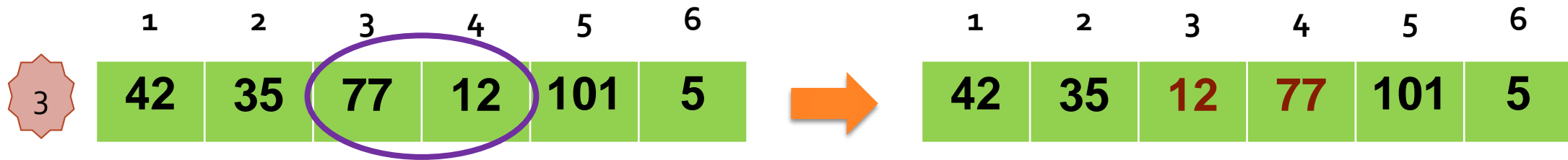Here DATA is an array with N elements.   This   algorithm sorts the elements in DATA.
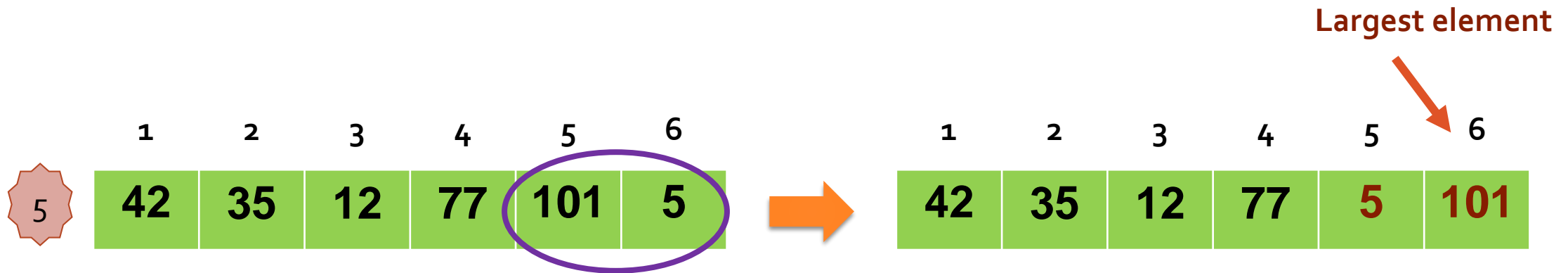
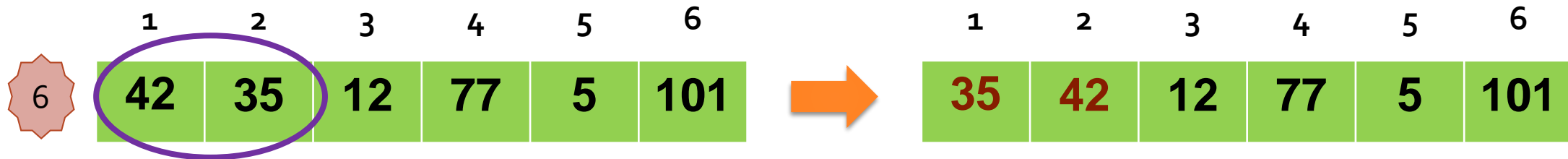# Bubble Sort - Example

First Iteration/pass

Wrong order

1	2	3	4	5	6
77	42	35	12	101	5

1	2	3	4	5	6
42	77	35	12	101	5

Swapped

1	2	3	4	5	6
42	77	35	12	101	5

1	2	3	4	5	6
42	35	77	12	101	5

Bubble Sort - Example

		1	2	3	4	5	6
3		42	35	77	12	101	5

➡️

	1	2	3	4	5	6
	42	35	12	77	101	5

		1	2	3	4	5	6
4		42	35	12	77	101	5

➡️ **No need to swap**

Bubble Sort - Example

Largest element

	1	2	3	4	5	6
5	42	35	12	77	101	5

➡️

	1	2	3	4	5	6
	42	35	12	77	5	101

Second Iteration/pass

	1	2	3	4	5	6
6	42	35	12	77	5	101

➡️

	1	2	3	4	5	6
	35	42	12	77	5	101

Bubble Sort - Example

Bubble Sort - Example

Bubble Sort - Example

11

1	2	3	4	5	6
12	35	42	5	77	101

➡️ **No need to swap**

12

1	2	3	4	5	6
12	35	42	5	77	101

➡️

1	2	3	4	5	6
12	35	5	42	77	101

3rd Largest element
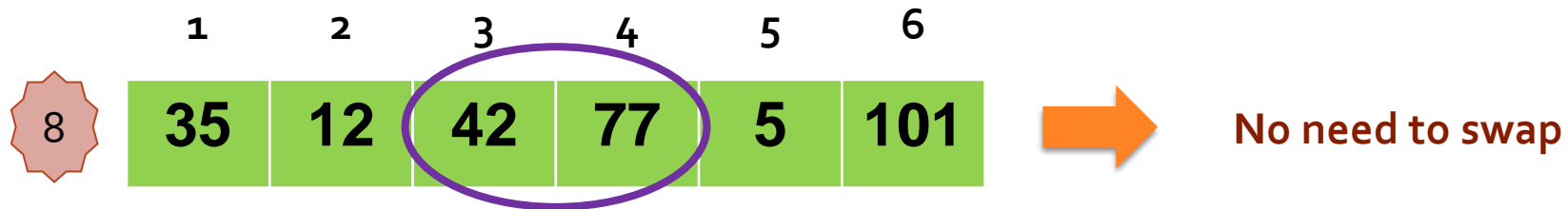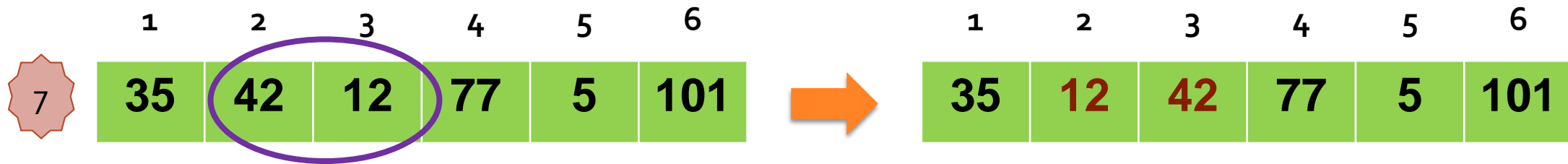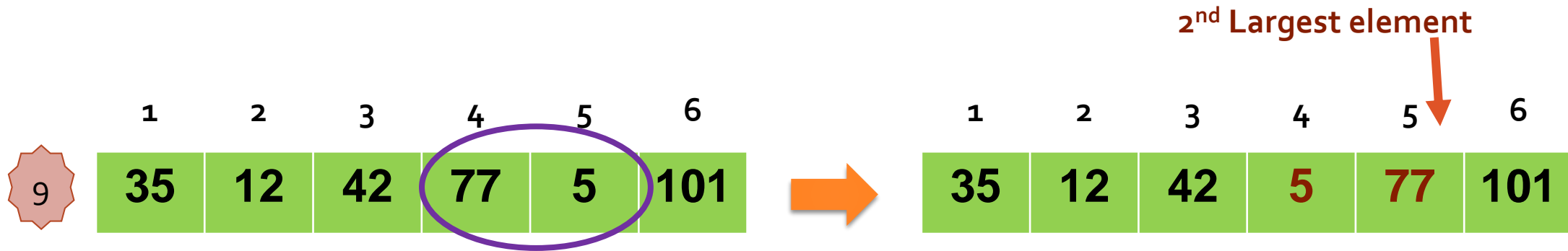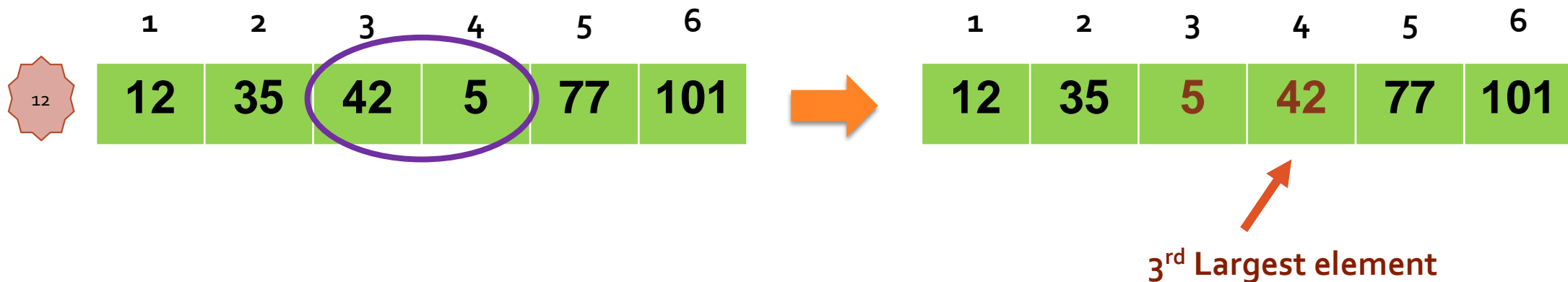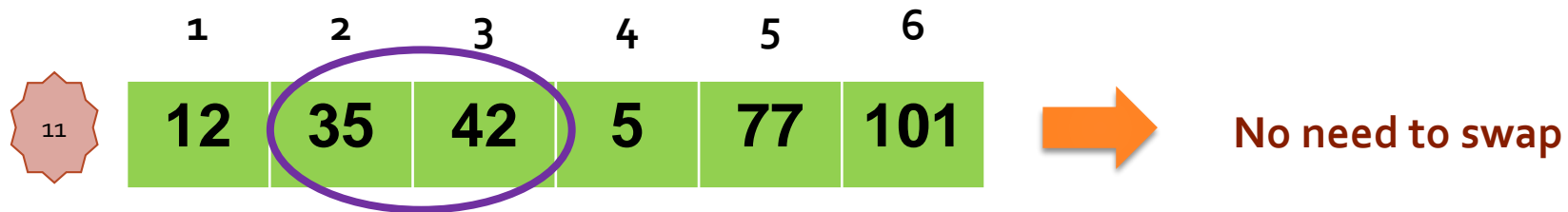
Bubble Sort - Example

Forth Iteration/pass

	1	2	3	4	5	6
13	12	35	5	42	77	101

➡ **No need to swap**

	1	2	3	4	5	6
14	12	35	5	42	77	101

➡

1	2	3	4	5	6
12	5	35	42	77	101

4th Largest element

21
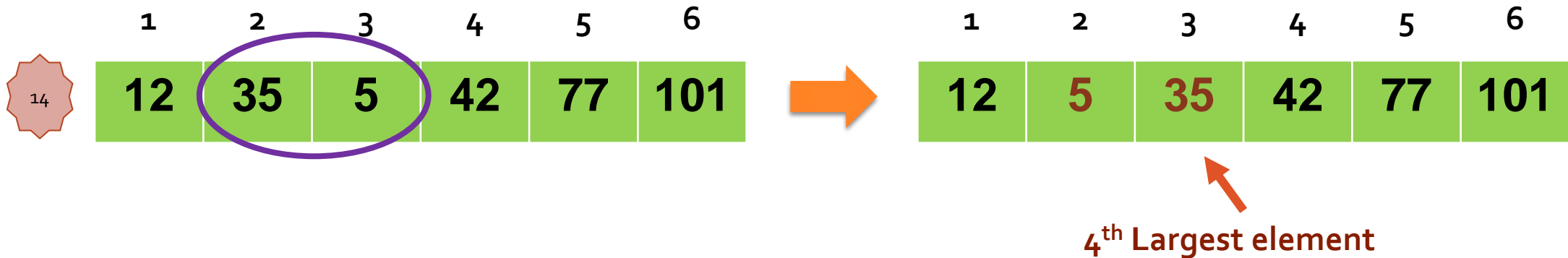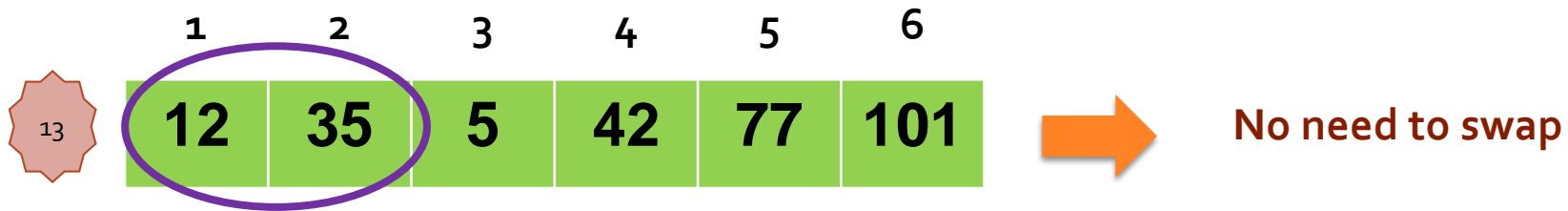
Bubble Sort - Example

Fifth Iteration/pass

	1	2	3	4	5	6
15	12	5	35	42	77	101

→

1	2	3	4	5	6
5	12	35	42	77	101
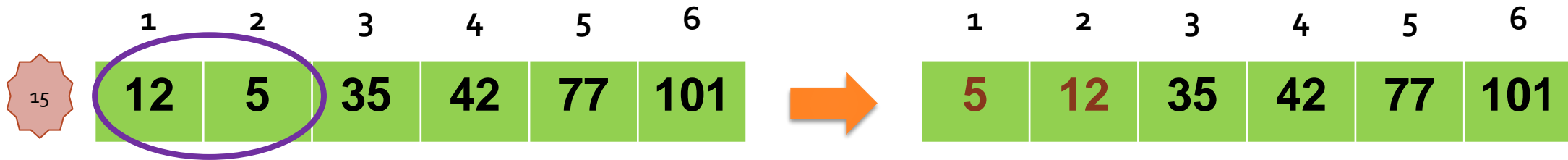
Sorted list

Selection Sort

- The Selection Sort searches (linear search) all the elements in a list until it finds the smallest element. It "swaps" this with the first element in the list.

- Next it finds the smallest of the remaining elements, and "swaps" it with the second element and so on.

- Its not very efficient when dealing with a huge list of items.

Algorithm

1. Repeat steps 2 and 3 for K = 1 to N – 1.

2. Call MIN (A, K, N, LOC).

3. [Interchange A[K] and A[LOC].]
   Set TEMP: A[K], A[K]:=A[LOC] and A[LOC]: = TEMP.
   [End of step 1 loop.]

4. Exit.

Procedure

1. Set MIN:= A[K] and LOC:=K [Initializes pointers]

2. Repeat for J = K + 1, to N:

   If MIN > A[J], then: Set MIN: = A[J] and LOC:= A[J] and LOC:=J.

   [End of loop.]

3. Exit.

An array A is in memory. This procedure finds the location LOC of the smallest element among A[k], A[K+1] ...., A[N].

Selection Sort - Example

1st iteration/Pass:

- ✓ Smallest = 77
- ✓ 42 < 77, smallest = 42
- ✓ 35 < 42, smallest = 35
- ✓ 12 < 35, smallest = 12
- ✓ 101 > 12, smallest = 12
- ✓ 5 < 12 smallest = 5

Swap 77 and 5

Smallest ??

77
42
35
12
101
5

Selection Sort - Example

Next Smallest ??

5
42
35
12
101
77

5
12
35
42
101
77
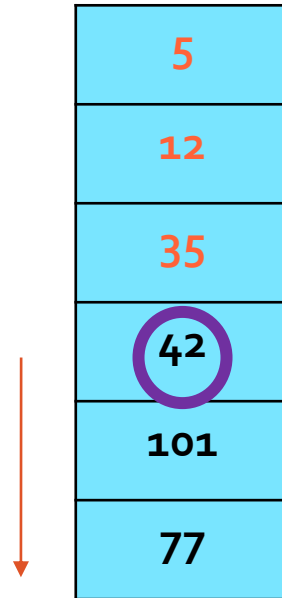
Selection Sort - Example
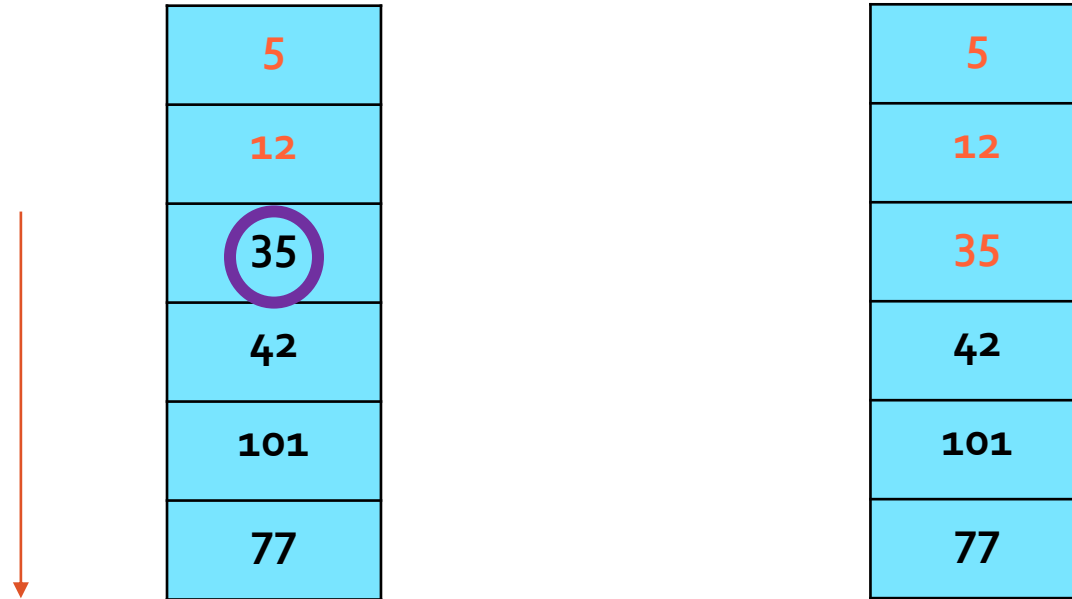
Next Smallest ??

Selection Sort - Example
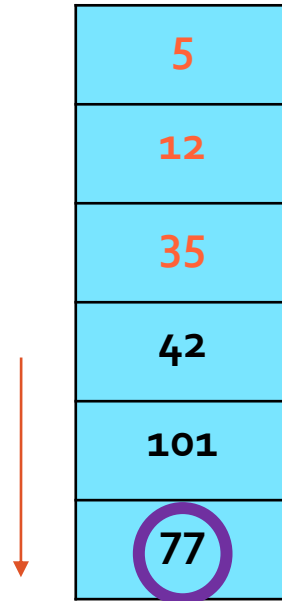
Next Smallest ??

5
12
35
42
101
77

5
12
35
42
77
101

Selection Sort - Example

Next Smallest ??

5
12
35
42
101
(77)

5
12
35
42
77
101

Insertion Sort

- Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands.

- The array is virtually split into a sorted and an unsorted part.

- Values from the unsorted part are picked and placed at the correct position in the sorted part.
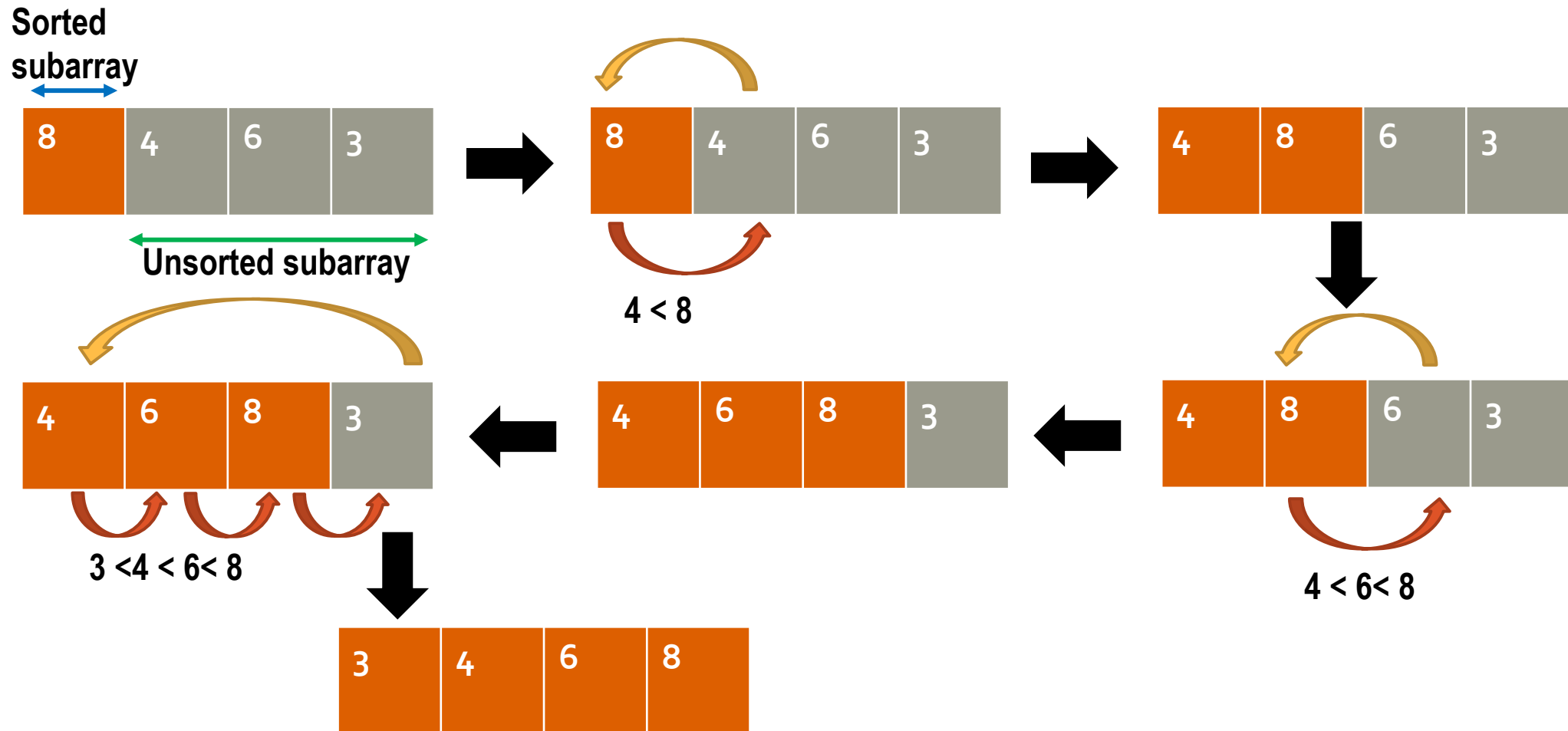
Algorithm

1. Set A[0]: = -∞. [Initialize sentinel element.]

2. Repeat Steps 3 to 5 for K = 2, 3 ,..., N:

3. Set TEMP:= A[K] and PTR:= K- 1.

4. Repeat while TEMP < A[PTR]:
   a) Set A[PTR + 1]: = A[PTR]. [Move element forward.]
   b) Set PTR:= PTR – 1.

   [End of Step 2 loop.]

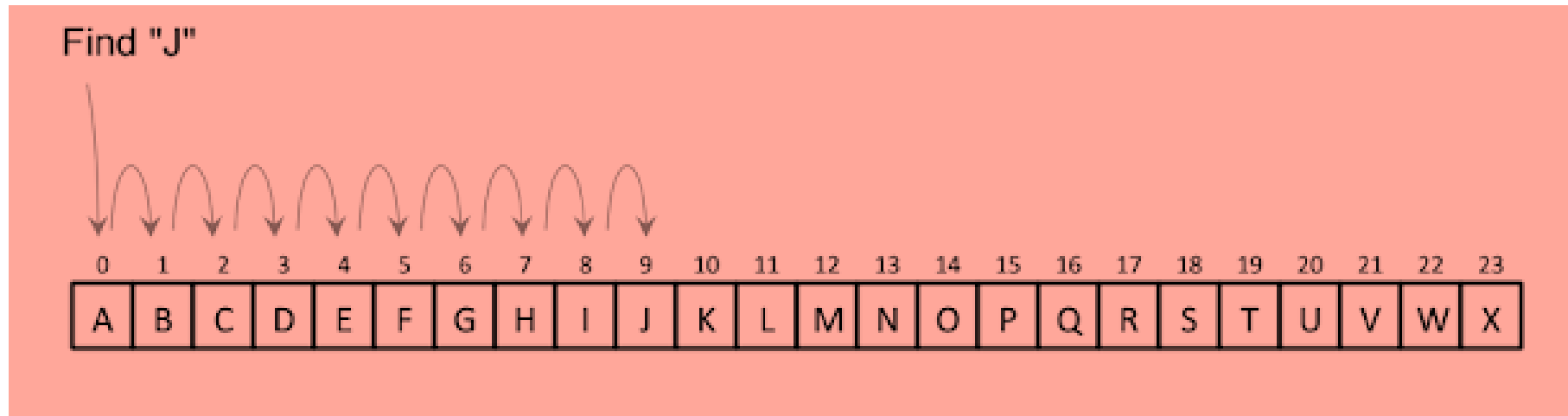5. Set A[PTR + 1]:=TEMP. [Inserts element in proper place.]

   [End of Step 2 loop.]

6. Return.

Insertion Sort – Example

Linear Search

- Linear search is used whether the given element is present in an array and if it present then at what location it is present.

- We keep on comparing each element with the element to search until the desired element is found or list ends.

Find "J"

| 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X

Algorithm

1. [Initialize] Set K:=1 and LOC :=0
2. Repeat Steps 3 and 4 while LOC =0 and K<=N.
3.   IF ITEM = DATA[K],then :Set LOC :=K.
4.   Set K:= K+1.[Increments counter.]
        [End of Step 2 loop.]
5. [Successful?]
        If LOC =0,then:
           Write: ITEM is not in the array DATA.
        Else:
           Write: LOC is the location of ITEM.
        [End of If structure.]
6. Exit.

A linear array DATA with N elements and a specific ITEM of information are given. This algorithm finds the location LOC of ITEM in the array DATA or sets LOC=0.

Binary Search

- Binary search is an efficient way for finding an item from an ordered list of items.

- It works by repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed down the possible locations to just one.
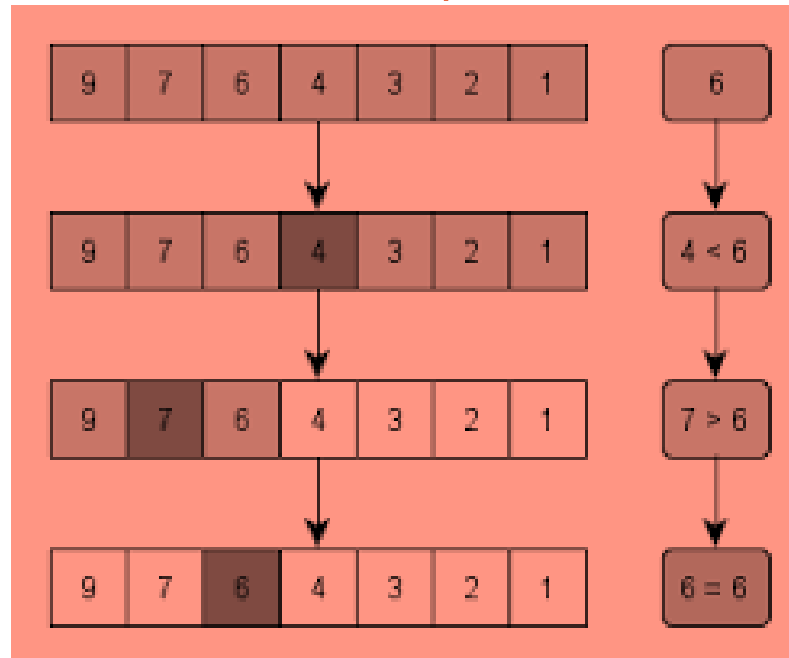
Algorithm

BINARY (DATA, LB, UB, ITEM, LOC)

1.[Initialize segment variable.] Set BEG:=LB, END:=UB, MID:=INT((BEG+END)/2).

2.Repeat step 3 and 4 while BEG<=END and DATA[MID]!=ITEM.

3.   If ITEM < DATA[MID]. then:
         Set END:=MID-1.
      Else
         Set BEG:=MID+1.
      [End of if structure.]

4.   Set MID:=INT((BEG+END)/2).
      [End of step 2 loop.]

5.If DATA[MID]=ITEM. then:
      Set LOC:=MID.
   Else
      Set LOC:=NULL.
      [End of if structure.]

6.Exit.

Here DATA is a sorted array with lower bound LB and upper bound UB and ITEM is the given item of information. The variables BEG, END, and MID denote the beginning ,the end and the middle of the segment of the element of DATA. This algorithm find the LOC of ITEM in DATA ,or sets LOC:=NULL if the search is unsuccessful.

Complexity

	Static	Dynamic
Access	O(1)	O(1)
Search	O(n)	O(n)
Delete	O(n)	O(n)
Insertion	O(1)	O(n)

Complexity

Sorting and Searching	Worst case
Bubble Sort	$O(n^2)$
Selection Sort	$O(n^2)$
Insertion Sort	$O(n^2)$
Binary Search	$O(\log n)$