MODULE 4
SORTING

CONTENT:

Sorting Algorithms

- A **sorting algorithm** is an algorithm that puts elements of a list in a certain order.
- Efficient sorting is important for optimizing the use of other algorithms (such as search and merge algorithms) which require input data to be in sorted lists

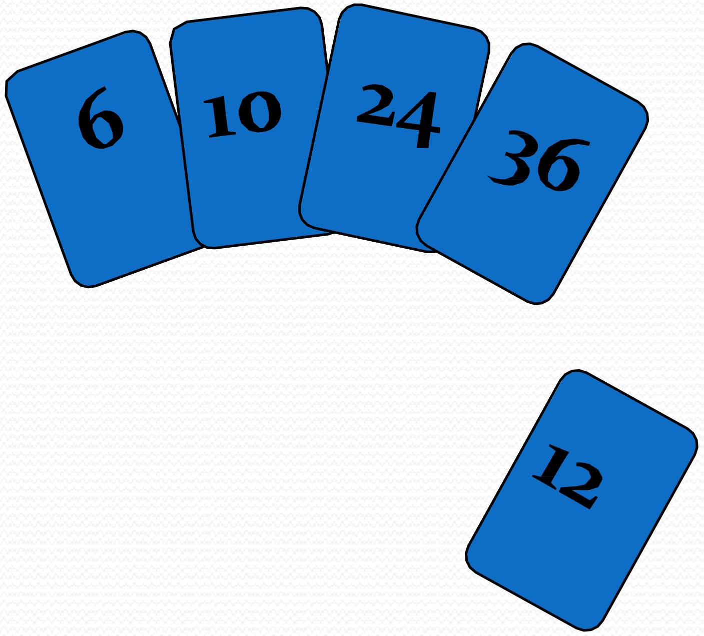
Review of Algorithms

- Selection Sort
  - An algorithm which orders items by repeatedly looking through remaining items to find the least one and moving it to a final location
- Bubble Sort
  - Sort by comparing each adjacent pair of items in a list in turn, swapping the items if necessary, and repeating the pass through the list until no swaps are done
- Insertion Sort
  - Sort by repeatedly taking the next item and inserting it into the final data structure in its proper order with respect to items already inserted.
- Merge Sort
  - An algorithm which splits the items to be sorted into two groups, recursively sorts each group, and merges them into a final, sorted sequence
- Quick Sort
  - An in-place sort algorithm that uses the divide and conquer paradigm. It picks an element from the array (the pivot), partitions the remaining elements into those greater than and less than this pivot, and recursively sorts the partitions.

Sorting Algorithms so far

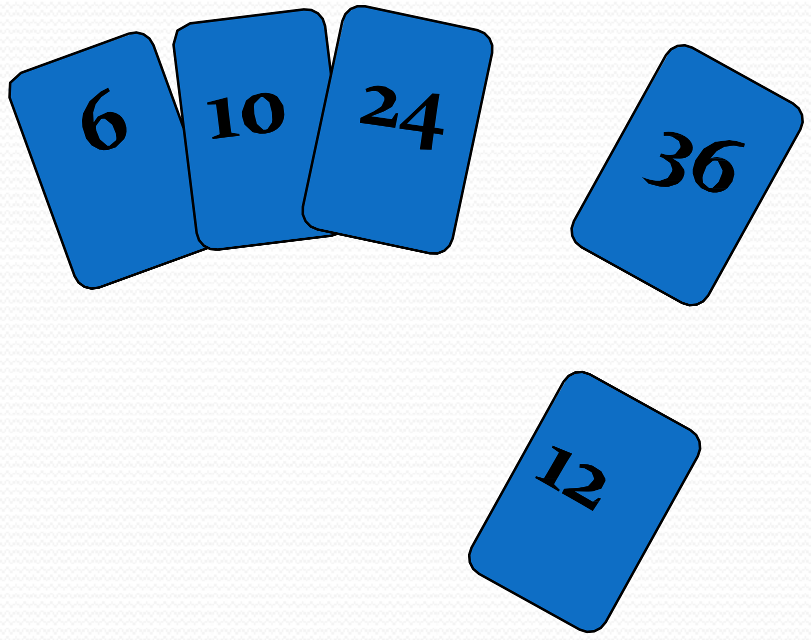
- Insertion sort, selection sort, bubblesort
  - Worst-case running time $\Theta(n^2)$; in-place
- Merge sort
  - Worst-case running time $\Theta(n \log n)$; but requires additional memory

Insertion Sort

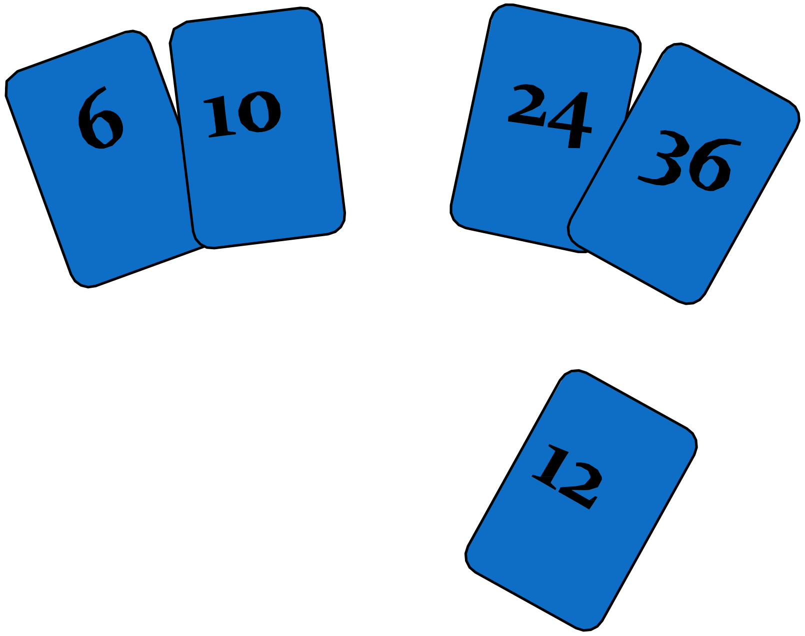


To insert 12, we need to make room for it by moving first 36 and then 24.

Insertion Sort



Insertion Sort



Insertion Sort

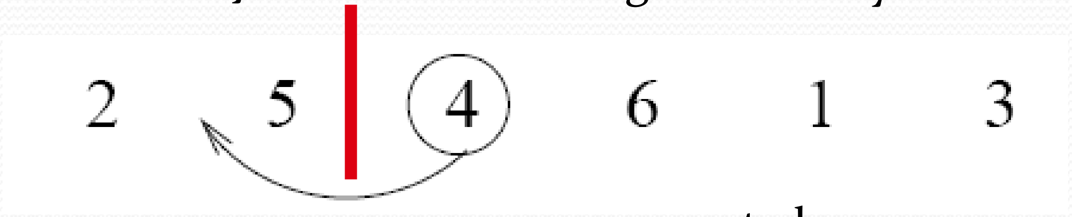
input array

5 2 4 6 1 3

at each iteration, the array is divided in two sub-arrays:

left sub-array

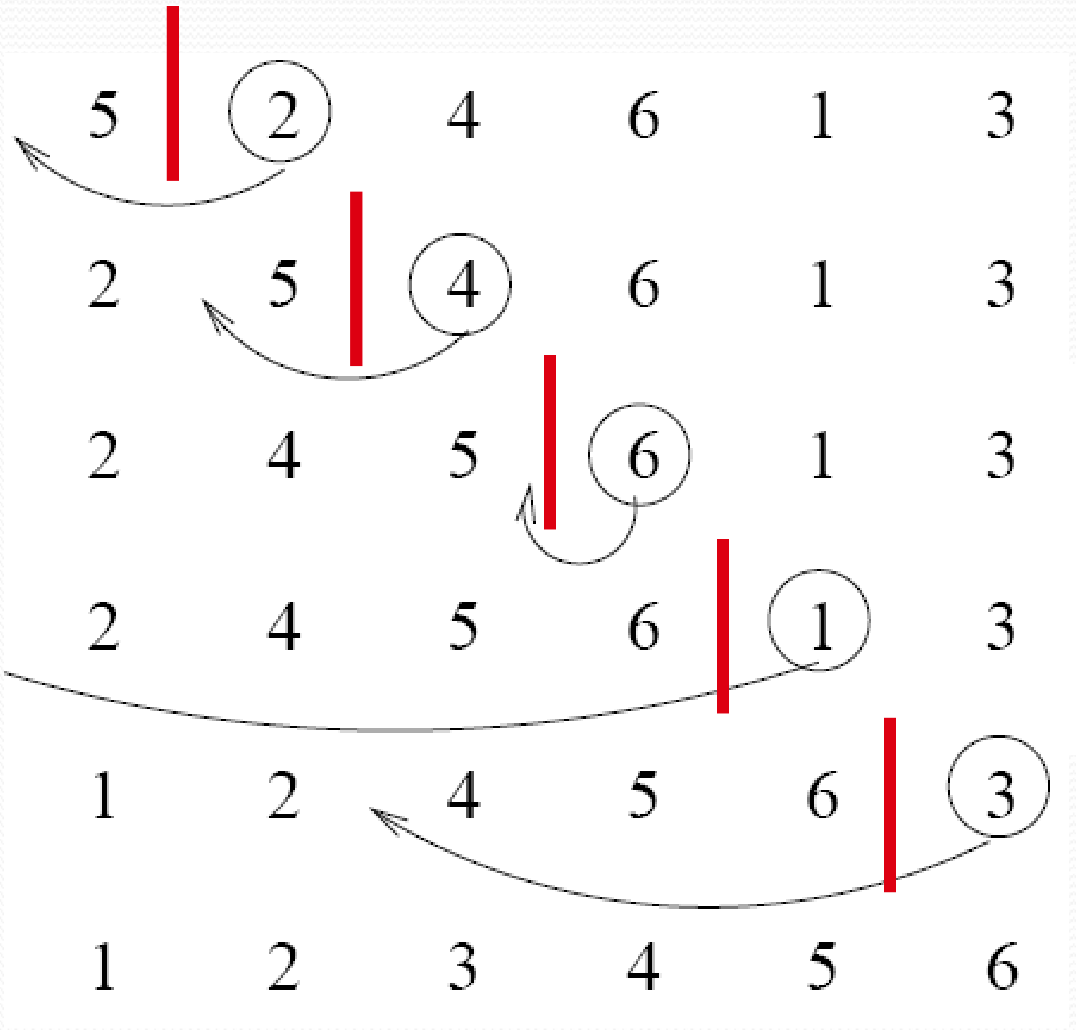
right sub-array

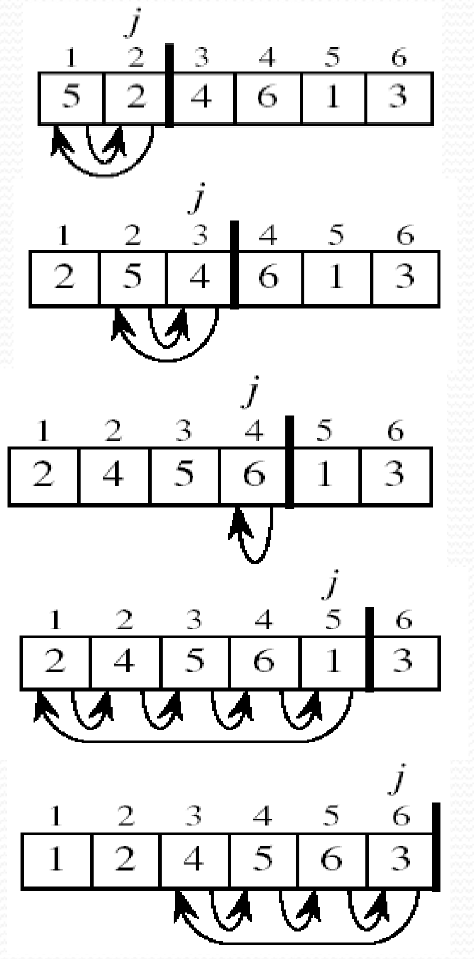


sorted

unsorted

Insertion Sort





Insertion Sort - Summary

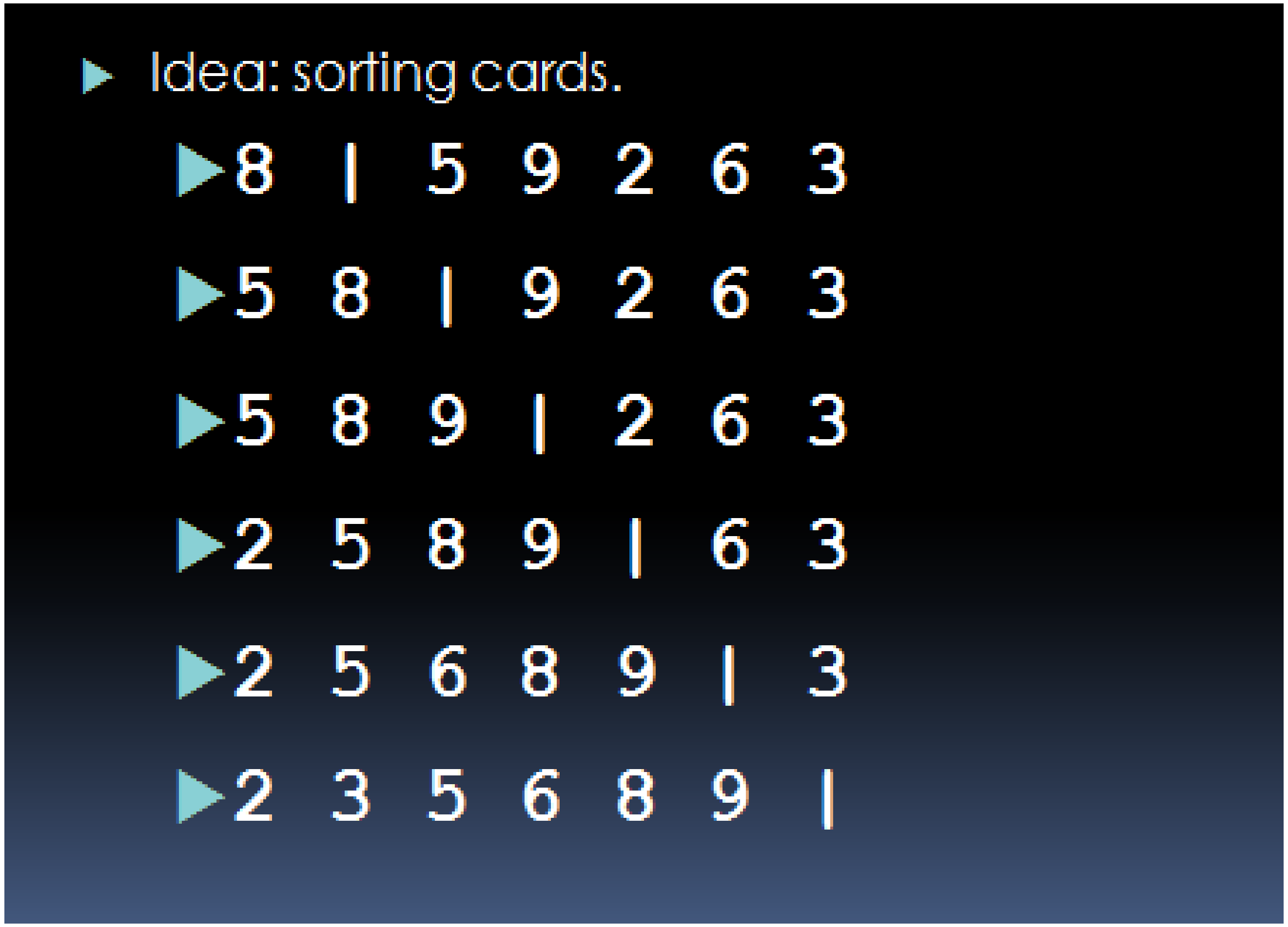
- Advantages
  - Good running time for "almost sorted" arrays $\Theta(n)$
- Disadvantages
  - $\Theta(n^2)$ running time in worst and average case
  - $\approx n^2/2$ comparisons and exchanges

ALGORITHM:- (Insertion sort)

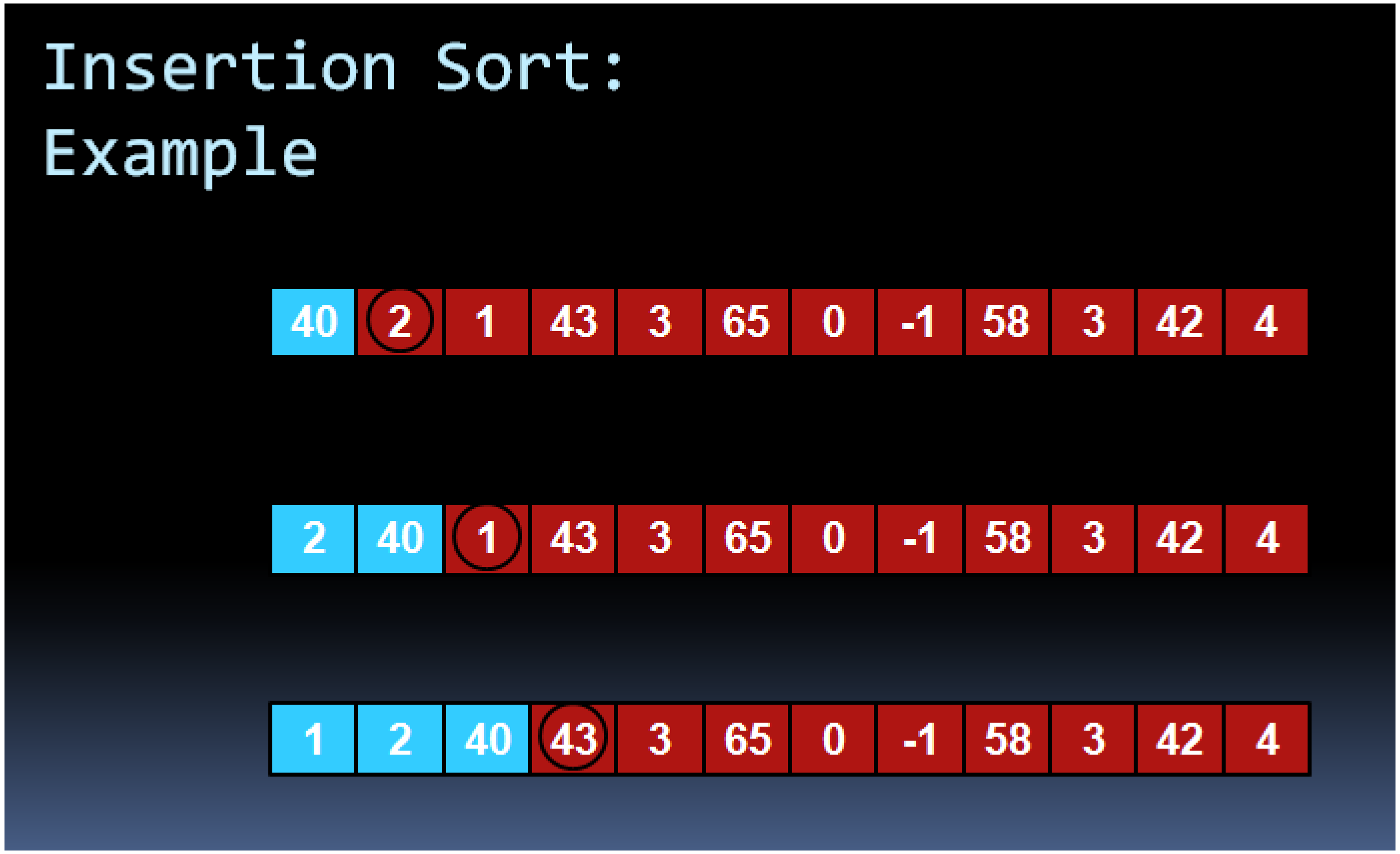
- **INSERTION(A,N)**
- **This algorithm sorts the array A with N elements.**

```
1.Set A[0]:= - INFINITE [initializes sentinel elements ]
2.Repeat steps 3 to 5 for K:=2,3,…..N
3.Set TEMP:=A[K] & PTR:=K-1
4.Repeat while TEMP<A[PTR]
A)Set A[PTR+1]:=A[PTR] [moves elements forward]
B)Set PTR:PTR-1
[End of loop ]
5.Set A[PTR+1]:=TEMP [inserts elements in proper place]
[end of step 2 loop ]
6.Exit
```

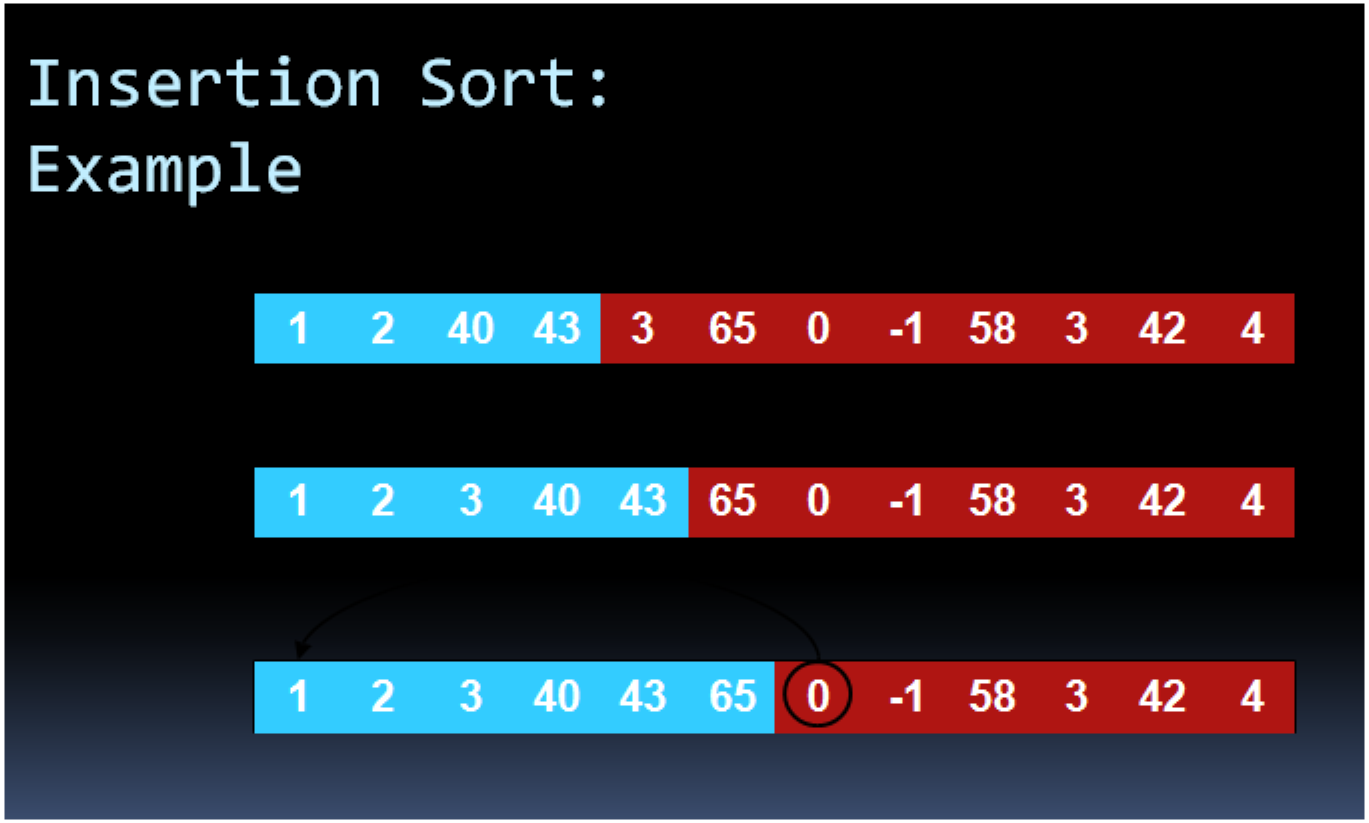
Insertion Sort


- Idea: sorting cards.
  - 8 | 5 9 2 6 3
  - 5 8 | 9 2 6 3
  - 5 8 9 | 2 6 3
  - 2 5 8 9 | 6 3
  - 2 5 6 8 9 | 3
  - 2 3 5 6 8 9 |

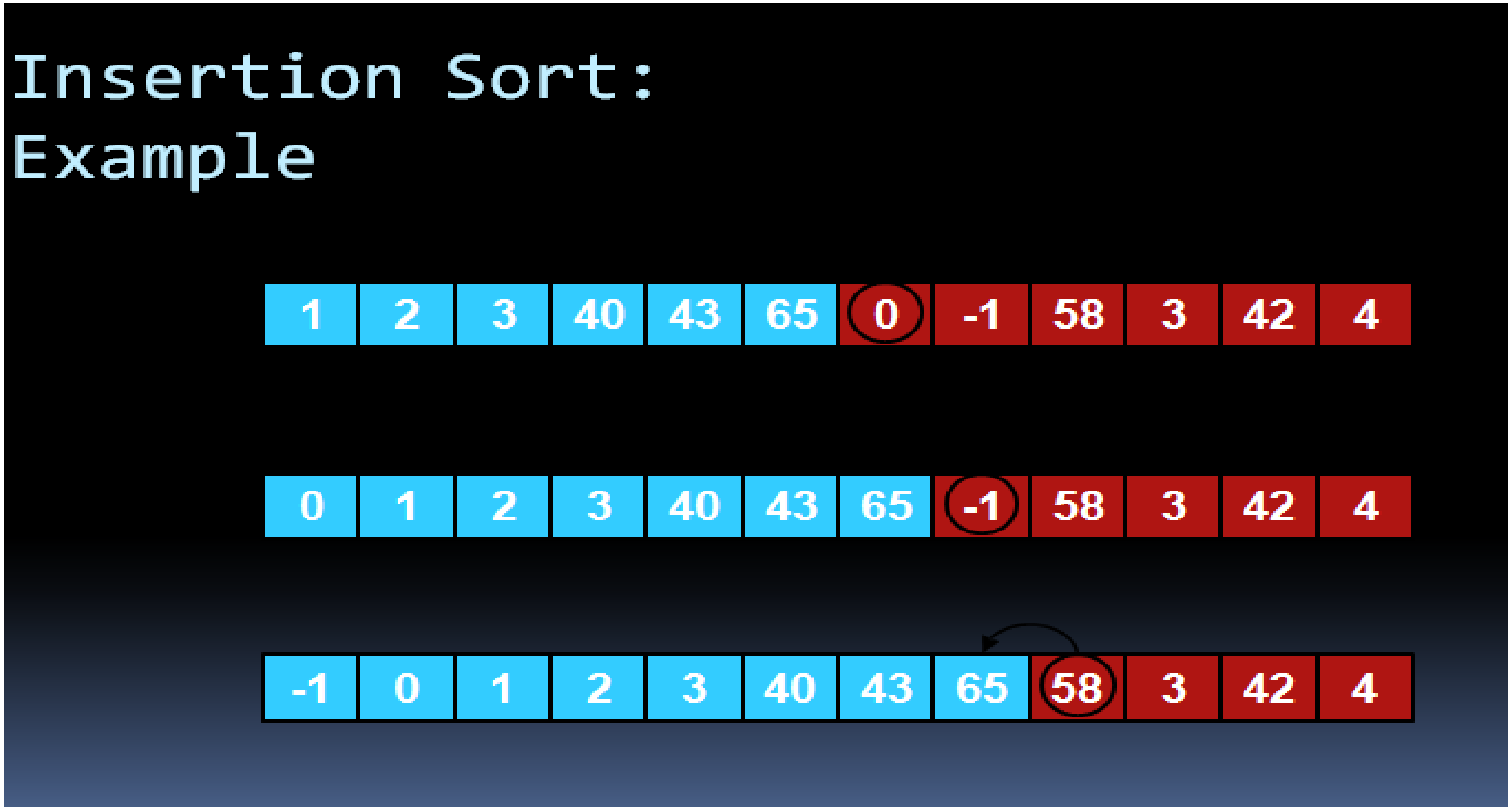
Insertion Sort



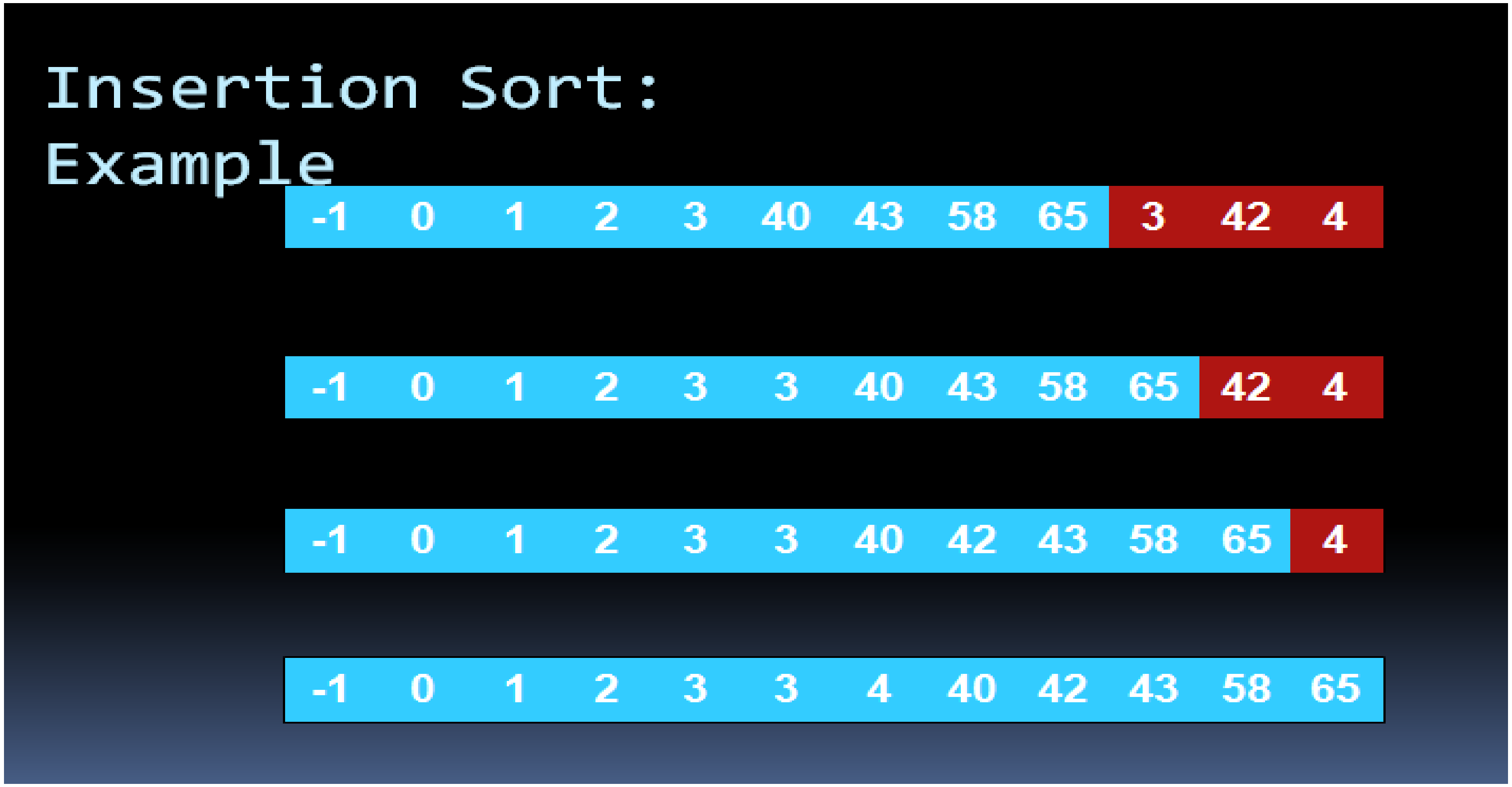
Insertion Sort



Insertion Sort



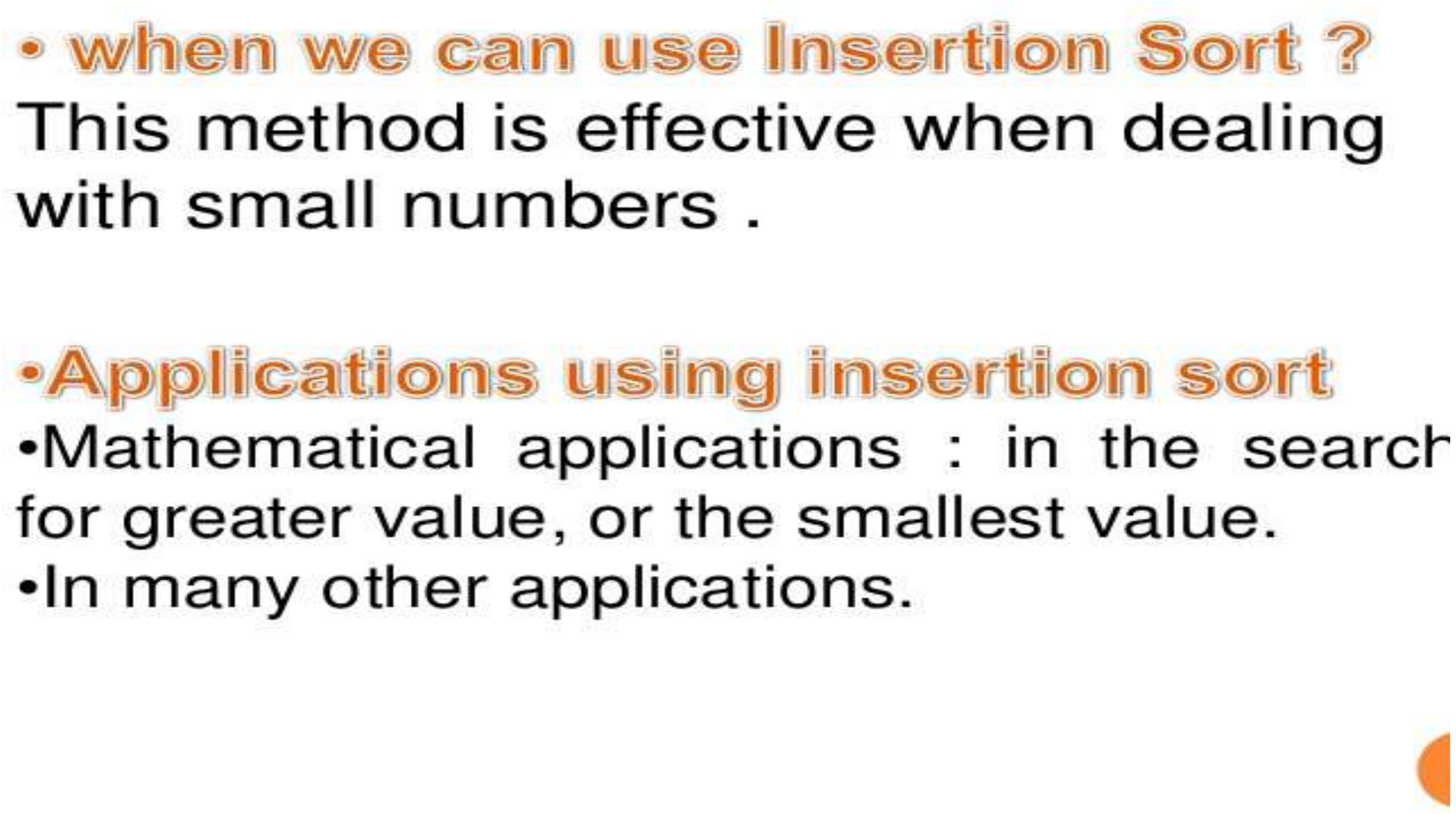
Insertion Sort



Insertion Sort:
Example

-1	0	1	2	3	40	43	58	65	3	42	4
-1	0	1	2	3	3	40	43	58	65	42	4
-1	0	1	2	3	3	40	42	43	58	65	4
-1	0	1	2	3	3	4	40	42	43	58	65

Insertion Sort



• when we can use Insertion Sort ?

This method is effective when dealing with small numbers .

•Applications using insertion sort

•Mathematical applications : in the search for greater value, or the smallest value.

•In many other applications.

PRACTICE QUESTION

- SORT THE FOLLOWING LIST OF ELEMENTS:
- 1. 30,20,50,10,60,70
- 2. A,P,P,L,E

1. 30,20,50,10,60,70

30	20	50	10	60	70
1	2	3	4	5	6

30	20	50	10	60	70
1	2	3	4	5	6

K=2
A[K]=20
TEMP:=A[K]
TEMP= 20
PTR:=K-1 = 1
A[PTR]=30

```
TEMP<A[PTR]
20<30
A[PTR+1]:=A[PTR]
A[2]:=A[1]
A[2]= 30
PTR:=PTR-1
PTR:=0
A[PTR+1]= TEMP
A[1]=20
```

20	30	50	10	60	70
1	2	3	4	5	6

K=3

A[K]=50

TEMP:=50

PTR:=K-1= 2

A[PTR]=30

TEMP<A[PTR]

PTR:=3

A[PTR+1]=TEMP

A[3]=50

20	30	50	10	60	70

10	20	30	50	60	70

10	20	30	50	60	🟨 70

FINAL SORTED ARRAY:

10	20	30	50	60	70

ANY QUERY??