

# Sorting

A series of horizontal lines in teal and light blue colors, located at the bottom of the slide, extending from the left edge and ending on the right side.

# Sorting

- It refers to operations of arranging data in some given sequence, i.e., ascending, descending, or lexicographic.

- **Input:**

A sequence of  $n$  numbers  $a_1, a_2, \dots, a_n$

- **Output:**

A permutation (reordering)  $a_1', a_2', \dots, a_n'$  of the input sequence

such that  $a_1' \leq a_2' \leq \dots \leq a_n'$

# Sorting - Example

- Original list

85, 76, 46, 92, 30, 41, 12, 19, 93, 3, 50, 11

- Sorted in non-decreasing order

- 3, 11, 12, 19, 30, 41, 46, 50, 76, 85, 92, 93

- Sorted in non-increasing order

93, 92, 85, 76, 50, 46, 41, 30, 19, 12, 11, 3



# Sorting Algorithms

# Sorting by comparison

- Basic operation involved in this type of sorting techniques is **comparison**. A data item is **compared with other items** in the list of item in order to find **its place in the sorted list**.

Insertion

Selection

Exchange

Enumeration

# Sorting by Comparison

## Sorting by comparison- Insertion:

From given list of items, one item is considered at a time. The item chosen is then inserted into appropriate position relative to the previously sorted items. The item can be inserted into the same list or to a different list.

e.g.: Insertion sort

## Sorting by comparison – Selection:

- First the smallest (or largest) element is located and it is separated from the rest. Then the next smallest (or next largest) is selected and so on until all items are separated.

e.g.: Selection sort, Heap sort

# Sorting by Comparison

## Sorting by comparison- Exchange:

If two items are found to be out of order, they are interchanged. The process is repeated until no more exchange is required.

e.g., Quick sort

## Sorting by comparison- Enumeration:

Two or more input lists are merged into an output list and while merging the items, an input list is chosen following the required sorting order.

e.g., Merge sort

# Sorting by Distribution

## Sorting by Distribution

No key comparison takes place.

All items under sorting are distributed over an auxiliary storage space based on the elements and then grouped them together to get the sorted list.

E.g., Radix

Note: Sorting algorithm can also be divided into different groups based on other concepts. Such as internal, external, in-place, stable. (For description refer to your class notes.)

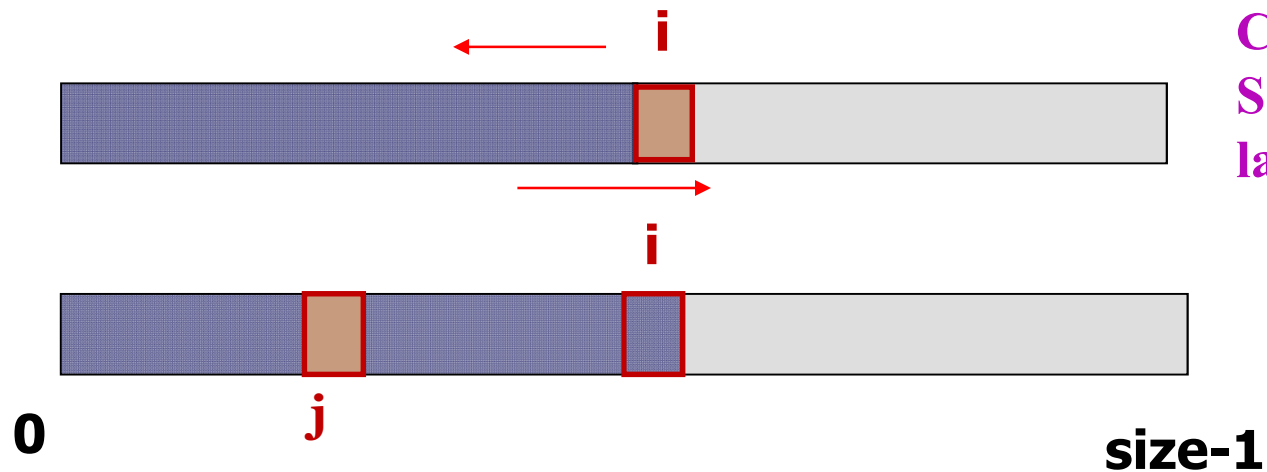




# Insertion Sort

# Insertion Sort

General situation :



Compare and  
Shift till  $x[i]$  is  
larger.

# Insertion Sort

- **Example**

Start with an empty left hand and the cards facing down on the table.

Remove one card at a time from the table, and insert it into the correct position in the left hand

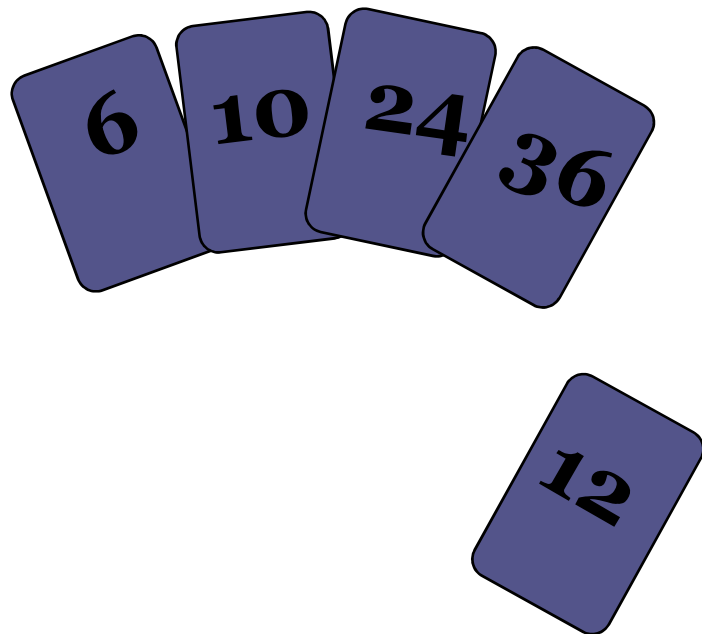
- compare it with each of the cards already in the hand, from right to left

The cards held in the left hand are sorted

- these cards were originally the top cards of the pile on the table

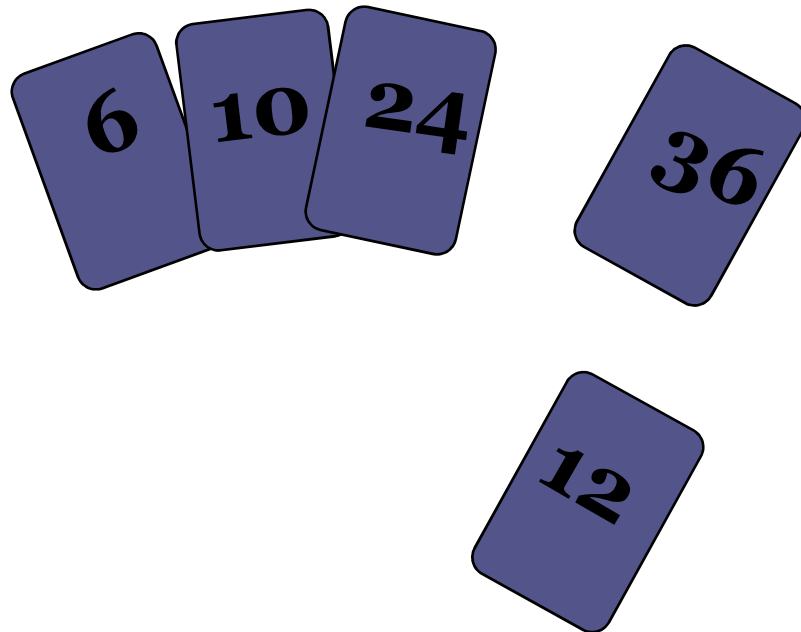
# Insertion Sort

12



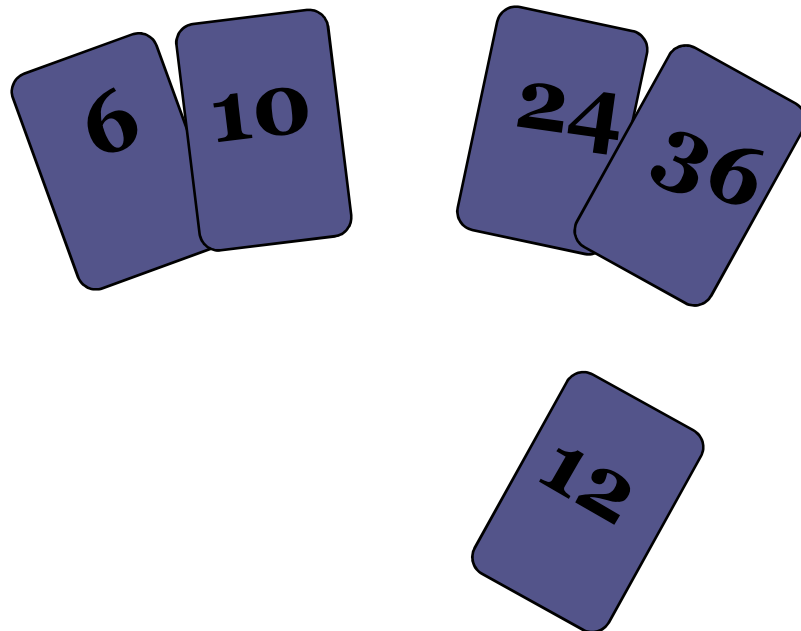
**To insert 12, we need to make room for it by moving first 36 and then 24.**

# Insertion Sort



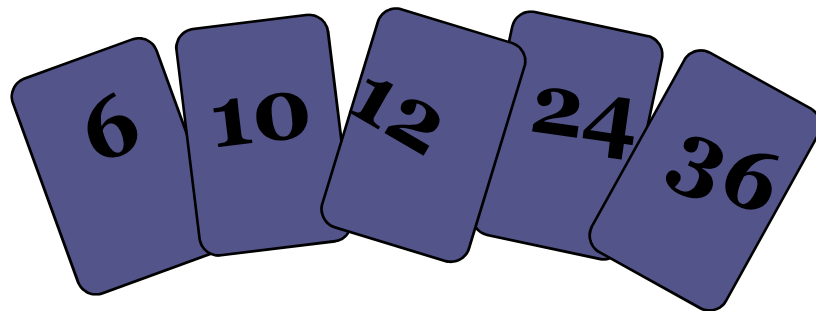
# Insertion Sort

14



# Insertion Sort

15



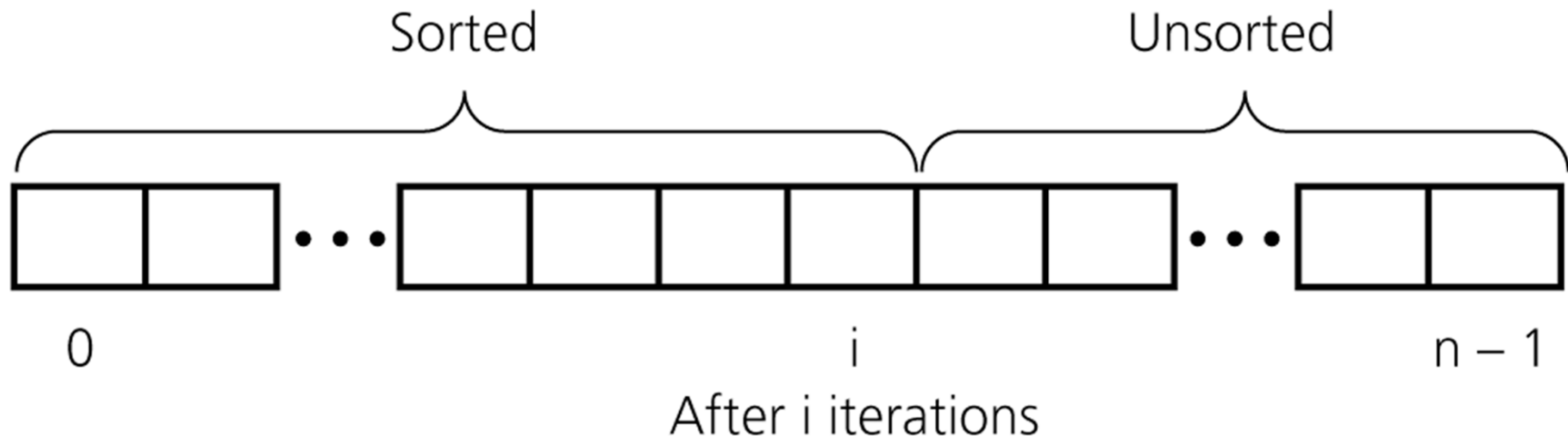
# Insertion Sort

- Steps

Assume that first element in the list is in sorted portion of the list and remaining all elements are in unsorted portion.

Consider first element from the unsorted list and insert that element into the sorted list in order specified.

Repeat the above process until all the elements from the unsorted list are moved into the sorted list.





# Example

Consider the following unsorted list of elements

1	2	3	4	5	6	7	8
15	20	10	30	50	18	5	45

Sorted	Unsorted
15	20
10	30
50	18
5	45

Sorted	Unsorted
15	20
10	30
50	18
5	45

Sorted		Unsorted					
15	20	10	30	50	18	5	45

Sorted			Unsorted				
10	15	20	30	50	18	5	45

Sorted				Unsorted			
10	15	20	30	50	18	5	45

Sorted					Unsorted		
10	15	20	30	50	18	5	45

Sorted						Unsorted	
10	15	18	20	30	50	5	45

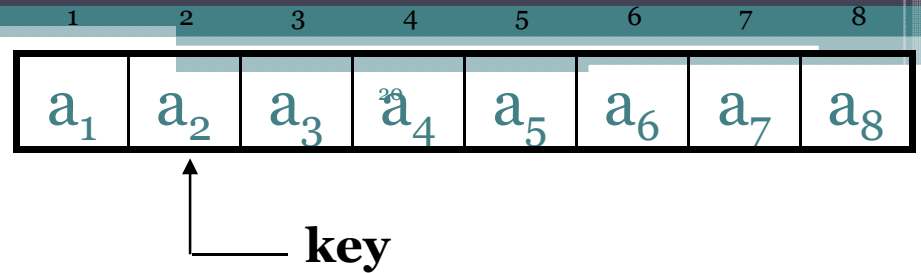
Sorted							Unsorted
5	10	15	18	20	30	50	45

Sorted							Unsorted
5	10	15	18	20	30	45	50

Final Result

5	10	15	18	20	30	45	50
---	----	----	----	----	----	----	----

# INSERTION-SORT



**Alg.:** INSERTION-SORT( $A$ )

**for**  $j \leftarrow 2$  **to**  $n$  // Start with the 2nd element because the first element is trivially sorted

$\text{key} \leftarrow A[j]$  //  $x$  is the element you want to insert in right place into the already sorted set of elements

$i \leftarrow j - 1$  // Last index of the already sorted elements because that's where you want to start comparing  $x$

**while**  $i > 0$  and  $A[i] > \text{key}$  // Whenever there is an element greater than  $x$

$A[i + 1] \leftarrow A[i]$  // shift it to the right

$i \leftarrow i - 1$

$A[i + 1] \leftarrow \text{key}$  // The correct position to insert  $x$

- Insertion sort – sorts the elements in place

# Insertion sort: Complexity analysis

**Case 1:** If the input list is already sorted order.

**Number of comparisons:** Number of comparison in each iterations is 1. Total number of comparisons to sort a list of size n.

$$c(n) = 1 + 1 + 1 \dots + 1 \text{ upto to } (n - 1)^{th} \text{ iterations}$$

**Number of movement :** No data movement takes place in any iterations. Hence, total movement is

$$m(n) = 0$$

# Insertion sort: Complexity analysis

**Case 2:** If the input list is sorted but in reverse order.

**Number of comparisons:**  $i^{th}$  iteration requires  $i$  number of comparisons. Hence, considering the total  $(n-1)$  iterations.

$$\begin{aligned} c(n) &= 1 + 2 + 3 + \dots + (n-1) \\ &= \frac{n(n-1)}{2} \end{aligned}$$

**Number of movement :** Number of movements take place in any  $i^{th}$  iteration is  $i$ .

$$\begin{aligned} m(n) &= 1 + 2 + 3 + \dots + (n-1) \\ &= \frac{n(n-1)}{2} \end{aligned}$$

# Insertion sort: Complexity analysis

**Case 3:** If the input list is in random order.

Let  $p_j$  be the probability that the key will go to the  $j$ th location ( $1 \leq j \leq i + 1$ ). Then the number of comparison will be  $j \cdot p_j$ .

The average number of comparisons in the  $(i + 1)^{th}$  iteration is

$$A_{i+1} = \sum_{j=1}^{i+1} j \cdot p_j$$



Locations to place a key at an iterations with five keys

# Insertion sort: Complexity analysis

Assume that all keys are distinct and all permutations of keys are equally likely.

$$p_1 = p_2 = p_3 = \dots = p_{i+1} = \frac{1}{i+1}$$

## Case 3: Number of comparison

The average number of comparison is in the  $(i+1)^{th}$  iteration as

$$A_{i+1} = \frac{1}{i+1} \sum_{j=1}^{i+1} j = \frac{1}{i+1} \cdot \frac{(i+1) \cdot (i+2)}{2} = \frac{i+2}{2}$$

The total number of comparison for all  $(n-1)$  iterations is

$$C(n) = \sum_{i=0}^{n-1} A_{i+1} = \sum_{i=0}^{n-1} \left( \frac{i}{2} + 1 \right) = \frac{1}{2} \cdot \sum_{i=0}^{n-1} i + (n-1) = \frac{1}{2} \cdot \frac{n(n-1)}{2} + (n-1)$$



# Insertion sort: Complexity analysis

## Case 3: Number of movement

On average, the number of movement in the  $i^{th}$  iteration

$$M_i = \frac{i + (i - 1) + (i - 2) + \dots + 2 + 1}{i} = \frac{i + 1}{2}$$

The total number of movements

$$M(n) = \sum_{i=1}^{n-1} M_i = \frac{1}{2} \cdot \sum_{i=1}^{n-1} i + \frac{(n - 1)}{2} = \frac{1}{2} \cdot \frac{n(n - 1)}{2} + \frac{n - 1}{2}$$

# Insertion sort: Summary of Complexity Analysis

Case	Comparisons	Movement	Remarks
Case 1	$C(n) = (n - 1)$	$M(n) = 0$	Input list is in sorted order
Case 2	$C(n) = \frac{n(n - 1)}{2}$	$M(n) = \frac{n(n - 1)}{2}$	Input list is sorted in reverse order
Case 3	$C(n) = \frac{(n - 1)(n + 4)}{4}$	$M(n) = \frac{(n - 1)(n + 2)}{4}$	Input list is in random order

$$T(n) = t_1 \cdot C(n) + t_2 \cdot M(n)$$

Case	Run time, $T(n)$	Complexity	Remarks
Case 1	$T(n) = c(n - 1)$	$T(n) = O(n)$	Best case
Case 2	$T(n) = c n(n - 1)$	$T(n) = O(n^2)$	Worst case
Case 3	$T(n) = c \frac{(n - 1)(n + 3)}{2}$	$T(n) = O(n^2)$	Average case

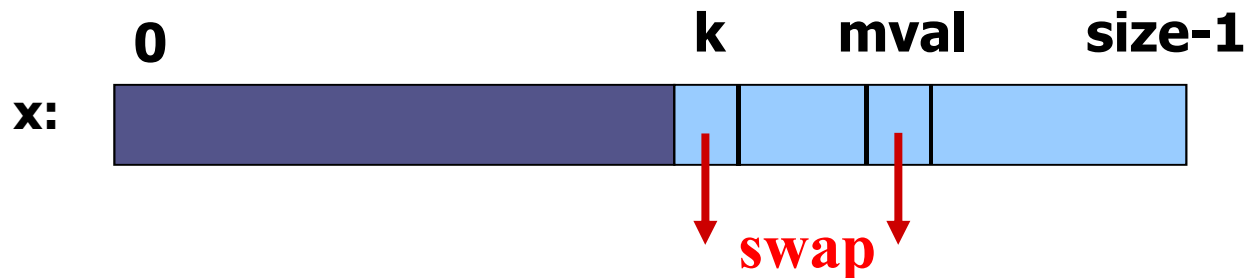
# Selection sort

## General situation :



## Steps :

- Find smallest element, **mval**, in **x[k...size-1]**
- Swap smallest element with **x[k]**, then **increase k**.



# Example

15	20	10	30	50	18	5	45
1	2	3	4	5	6	7	8

15	20	10	30	50	18	5	45
----	----	----	----	----	----	---	----

5	20	10	30	50	18	15	45
---	----	----	----	----	----	----	----

5	10	20	30	50	18	15	45
---	----	----	----	----	----	----	----

5	10	15	30	50	18	20	45
---	----	----	----	----	----	----	----

5	10	15	18	50	30	20	45
---	----	----	----	----	----	----	----

5	20	10	30	50	18	15	45
---	----	----	----	----	----	----	----

5	10	20	30	50	18	15	45
---	----	----	----	----	----	----	----

5	10	15	30	50	18	20	45
---	----	----	----	----	----	----	----

5	10	15	18	50	30	20	45
---	----	----	----	----	----	----	----

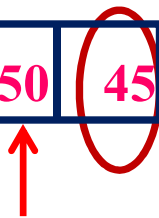
5	10	15	18	20	30	50	45
---	----	----	----	----	----	----	----



5	10	15	18	20	30	50	45
---	----	----	----	----	----	----	----

5	10	15	18	20	30	50	45
---	----	----	----	----	----	----	----

5	10	15	18	20	30	50	45
---	----	----	----	----	----	----	----



5	10	15	18	20	30	45	50
---	----	----	----	----	----	----	----

# Selection Sort

**Alg.:** SELECTION-SORT( $A$ )

$n \leftarrow \text{length}[A]$

**for**  $j \leftarrow 1$  **to**  $n - 1$

$\text{smallest} \leftarrow j$

**for**  $i \leftarrow j + 1$  **to**  $n$

**if**  $A[i] < A[\text{smallest}]$

**then**  $\text{smallest} \leftarrow i$

    exchange  $A[j] \leftrightarrow A[\text{smallest}]$

# Selection sort: Complexity Analysis

**Case 1:** If the input list is already in sorted order

**Number of comparison:**

$$C(n) = \sum_{i=1}^{n-1} n \quad i = \frac{n(n-1)}{2}$$

**Number of movement :** No data movement takes place in any iterations. Hence, total movement is

$$M(n) = 0$$

# Selection sort: Complexity Analysis

**Case 2: The input is sorted but in reverse order.**

**Number of comparison**

$$C(n) = \sum_{i=0}^{n-1} (n - i) = \frac{n(n-1)}{2}$$

**Number of movements**

$$M(n) = \frac{3}{2} (n - 1)$$



# Selection sort: Complexity Analysis

**Case 3:** The elements in the input list are in random order.

**Number of comparison:**

$$C(n) = \frac{n(n-1)}{2}$$

**Number of movement**

Let  $p_i$  be the probability that the  $i^{th}$  smallest element is in  $i^{th}$  position. Number of swap operation in  $(n-1)$  total iteration  
 $= (1 - p_i) \times (n - 1)$

where,  $p_1 = p_2 = p_3 \dots = p_n = \frac{1}{n}$

**Total number of movement**

$$M(n) = \left(1 - \frac{1}{n}\right) \times (n - 1) \times 3 = \frac{3(n-1)(n-1)}{n}$$

# Selection sort: Summary of complexity analysis

Case	Comparisons	Movement	Remarks
Case 1	$c(n) = \frac{n(n-1)}{2}$	$M(n) = 0$	Input list is in sorted order
Case 2	$c(n) = \frac{n(n-1)}{2}$	$M(n) = \frac{3(n-1)}{2}$	Input list is sorted in reverse order
Case 3	$c(n) = \frac{n(n-1)}{2}$	$M(n) = \frac{3(n-1)^2}{n}$	Input list is in random order

Case	Run time, $T(n)$	Complexity	Remarks
Case 1	$T(n) = \frac{n(n-1)}{2}$	$T(n) = O(n^2)$	Best case
Case 2	$T(n) = \frac{(n-1)(n+3)}{2}$	$T(n) = O(n^2)$	Worst case
Case 3	$T(n) \approx \frac{(n-1)(2n+3)}{2}$ (Taking $n-1 \approx n$ )	$T(n) = O(n^2)$	Average case

# Merge Sort

- Merger sort is an example of divide and conquer strategy.
- Divide and Conquer
  - Divide:** Partition the list midway, i.e., at  $\frac{l+r}{2}$  into 2 sub list with  $\frac{n}{2}$  elements in each.
  - Conquer:** Sort the two lists recursively using the merge sort.
  - Merge:** Combine the sorted sublist to obtain the sorted output.