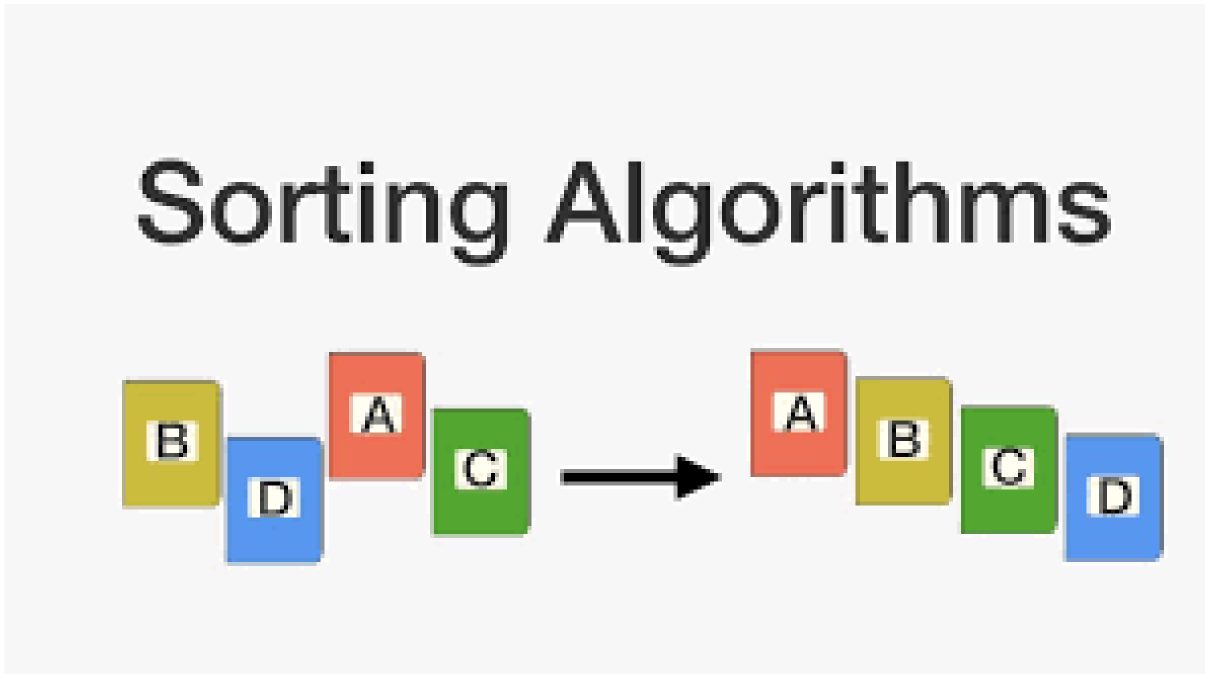
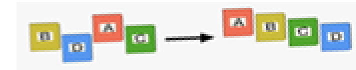


Dr. Nadia M. Mohammed

Sorting



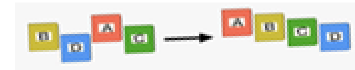
- **Sorting** : is a process that organizes a collection of data into either ascending or descending order.
- **There are two type of sorting:**

1- Internal sort requires that the collection of data fit entirely in the computer’s main memory.

2- External sort when the collection of data cannot fit in the computer’s main memory all at once but must reside in secondary storage such as on a disk.

- **There are many sorting algorithms, such as:**
  – Selection Sort
  – Insertion Sort
  – Bubble Sort
  – Merge Sort
  – Quick Sort

Selection Sort



- The list is divided into two sublists, sorted and unsorted, which are divided by an imaginary wall.
- We find the smallest element from the unsorted sublist and swap it with the element at the beginning of the unsorted data.
- After each selection and swapping, the imaginary wall between the two sublists move one element ahead, increasing the number of sorted elements and decreasing the number of unsorted ones.
- Each time we move one element from the unsorted sublist to the sorted sublist, we say that we have completed a sort pass.
- A list of n elements requires n-1 passes to completely rearrange the data.

Selection Sort Example

Sorted **Unsorted**

23	78	45	8	32	56	Original List
8	78	45	23	32	56	After pass 1
8	23	45	78	32	56	After pass 2
8	23	32	78	45	56	After pass 3
8	23	32	45	78	56	After pass 4
8	23	32	45	56	78	After pass 5

Selection Sort Function

```python
# Function to implement selection sort
def selection_sort(arr):
    # Traverse through all array elements
    for i in range(len(arr)):
        # Find the minimum element in remaining
        # unsorted array
        min_idx = i
        for j in range(i + 1, len(arr)):
            if arr[min_idx] > arr[j]:
                min_idx = j
        # Swap the found minimum element with
        # the first element
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
```

Insertion Sort

- Insertion sort is a simple sorting algorithm that is appropriate for small inputs.
- The list is divided into two parts: sorted and unsorted.
- In each pass, the first element of the unsorted part is picked up, transferred to the sorted sublist, and inserted at the appropriate place.
- A list of *n* elements will take at most *n-1* passes to sort the data.

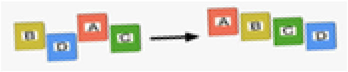
Insertion Sort Example 

Sorted			Unsorted			
23	78	45	8	32	56	Original List
23	78	45	8	32	56	After pass 1
23	45	78	8	32	56	After pass 2
8	23	45	78	32	56	After pass 3
8	23	32	45	78	56	After pass 4
8	23	32	45	56	78	After pass 5

Insertion Sort Function

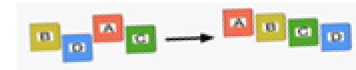
```python
# Function to implement insertion sort
def insertion_sort(arr):
    # Traverse through 1 to len(arr)
    for i in range(1, len(arr)):
        key = arr[i]
        # Move elements of arr[0..i-1], that are
        # greater than key, to one position ahead
        # of their current position
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```

Bubble Sort



- The list is divided into two sublists: sorted and unsorted.
- The smallest element is bubbled from the unsorted list and moved to the sorted sublist.
- After that, the wall moves one element ahead, increasing the number of sorted elements and decreasing the number of unsorted ones.
- Each time an element moves from the unsorted part to the sorted part one sort pass is completed.
- Given a list of n elements, bubble sort requires up to n-1 passes to sort the data.

Bubble Sort

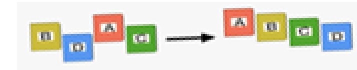


23	78	45	8	32	56	Original List
8	23	78	45	32	56	After pass 1
8	23	32	78	45	56	After pass 2
8	23	32	45	78	56	After pass 3
8	23	32	45	56	78	After pass 4

Bubble Sort Function

```python
# Function to implement Bubble Sort
def bubble_sort(arr):
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):
        # 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]
```

Merge Sort



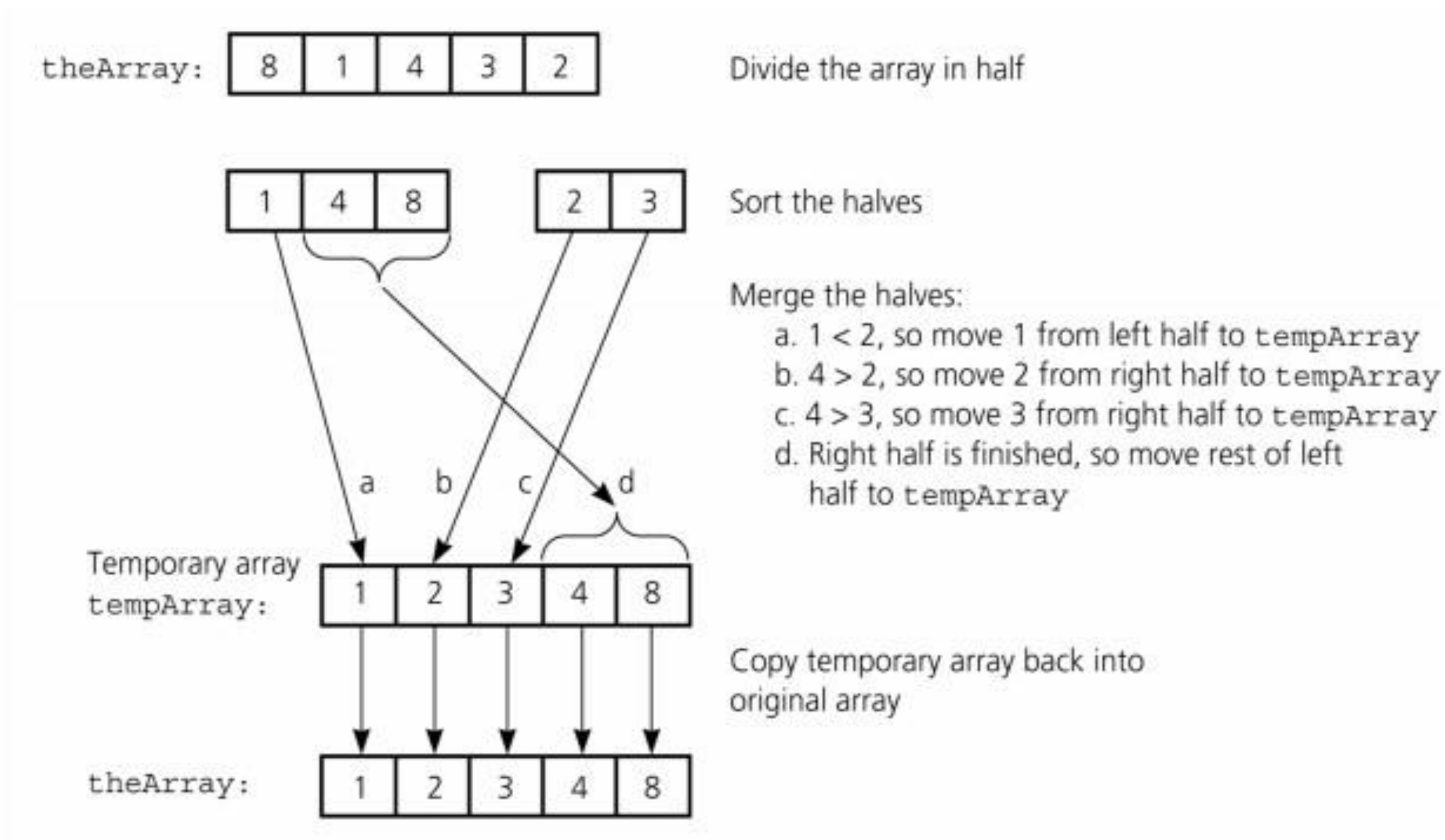
- Merge sort algorithm is one of two important divide-and-conquer sorting algorithms (the other one is quicksort).
- It is a recursive algorithm.

– Divides the list into halves,

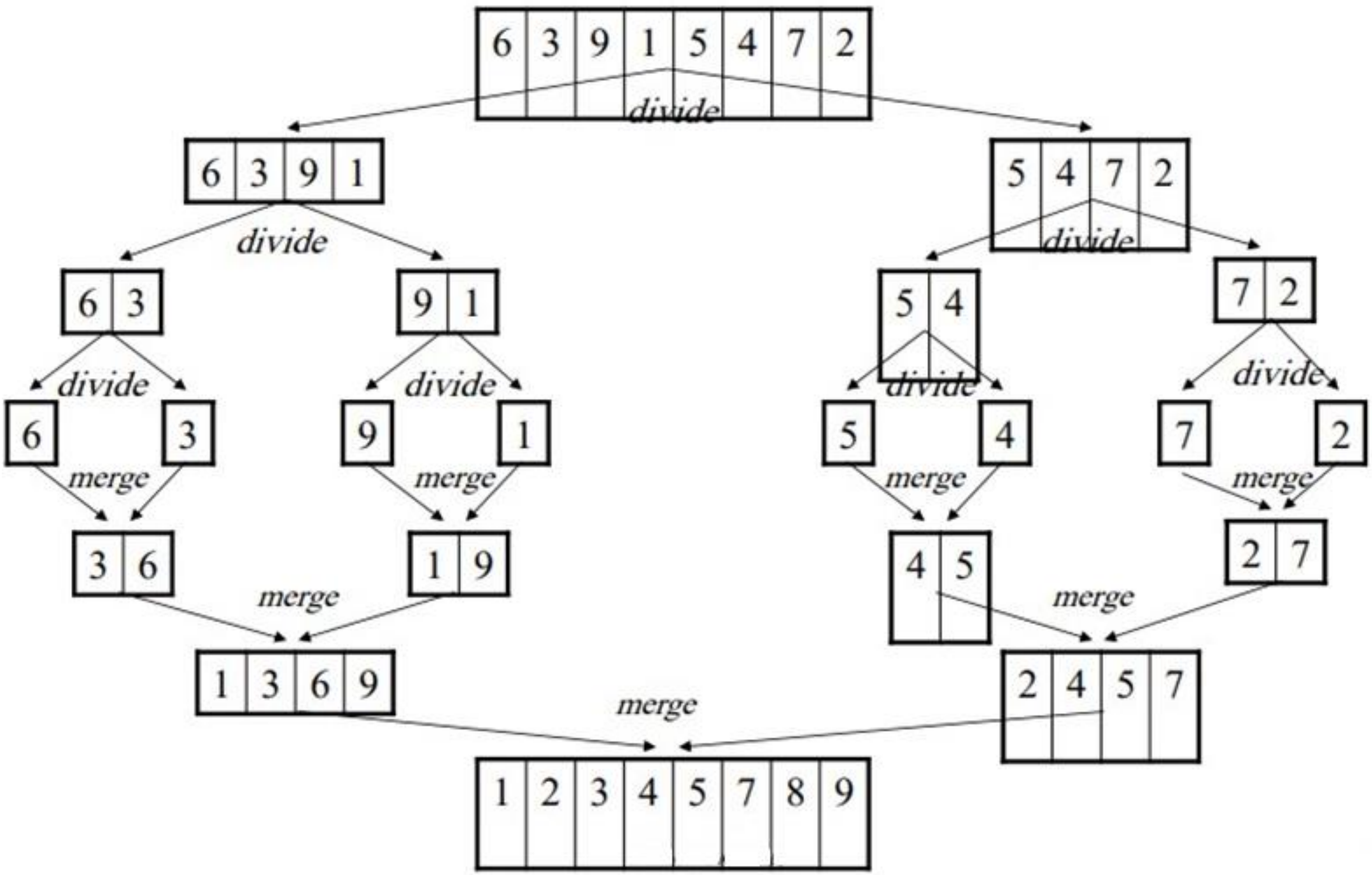
– Sort each halve separately, and

– Then merge the sorted halves into one sorted array.

Mergesort - Example



Mergesort - Example



Merge Sort Function

```
# Function to implement Merge Sort
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2  # Finding the mid of the
array
        L = arr[:mid]  # Dividing the array elements
        R = arr[mid:]  # into 2 halves

        merge_sort(L)  # Sorting the first half
        merge_sort(R)  # Sorting the second half

        i = j = k = 0

        # Copy data to temp arrays L[] and R[]
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
```

Merge Sort Function cont.

```
            arr[k] = R[j]
            j += 1
        k += 1

# Checking if any element was left
while i < len(L):
    arr[k] = L[i]
    i += 1
    k += 1

while j < len(R):
    arr[k] = R[j]
    j += 1
    k += 1
```