Sorting:

The arrangement of data in a preferred order is called sorting in the data structure. By sorting data, it is easier to search through it quickly and easily. The simplest example of sorting is a dictionary. Before the era of the Internet, when you wanted to look up a word in a dictionary, you would do so in alphabetical order. This made it easy.

Imagine the panic if you had to go through a big book with all the English words from the world in a jumbled order! It is the same panic an engineer will go through if their data is not sorted and structured.

Sorting Categories

There are two different categories in sorting:

- **Internal sorting**: If the input data is such that it can be adjusted in the main memory at once, it is called internal sorting.
- **External sorting**: If the input data is such that it cannot be adjusted in the memory entirely at once, it needs to be stored in a hard disk, floppy disk, or any other storage device. This is called external sorting.

Internal sorting algorithms are sorting algorithms that sort data that can fit into main memory. Examples of internal sorting algorithms include:

- Bubble sort
- insertion sort
- selection sort
- merge sort
- quick sort

External sorting algorithms are sorting algorithms that sort data that is too large to fit into main memory at once. Examples of external sorting algorithms include:

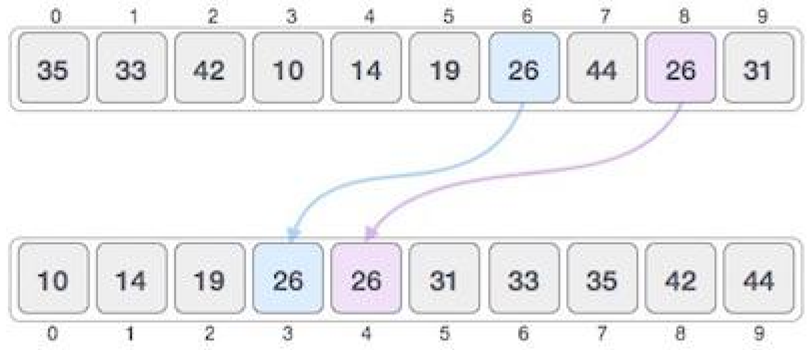
- external merge sort
- replacement selection sort
- Polyphase sort

In internal sorting the data that has to be sorted will be in the main memory always, implying faster access. Complete sorting will happen in the main memory. Insertion sort, quick sort, heap sort, radix sort can be used for internal sorting.

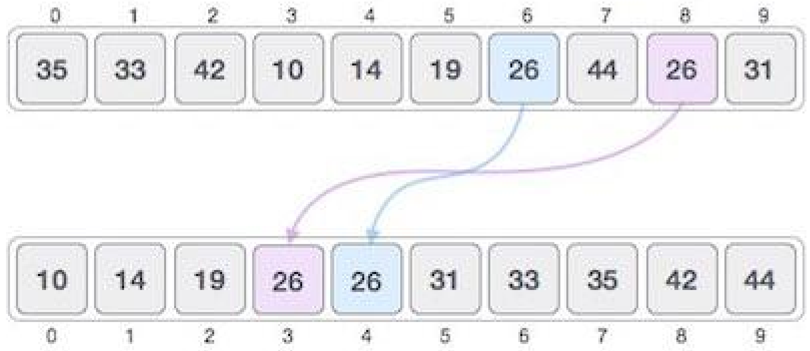
In external sorting it will be on disks, outside main memory. It can be because the data is huge and cannot be stored in main memory. While sorting the data will be pulled over in chunks from disk to main memory. Later all the sorted data will be merged and stored back to disk, where it can fit. External merge sort can be used here.

Stable and Not Stable Sorting

If a sorting algorithm, after sorting the contents, does not change the sequence of similar content in which they appear, it is called **stable sorting**.



If a sorting algorithm, after sorting the contents, changes the sequence of similar content in which they appear, it is called **unstable sorting**.

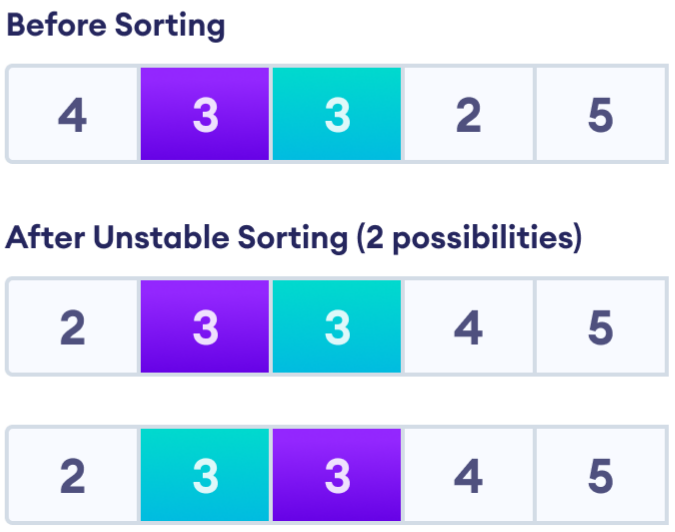


Stability of an algorithm matters when we wish to maintain the sequence of original elements, like in a tuple for example.

Stability of Sorting Algorithm

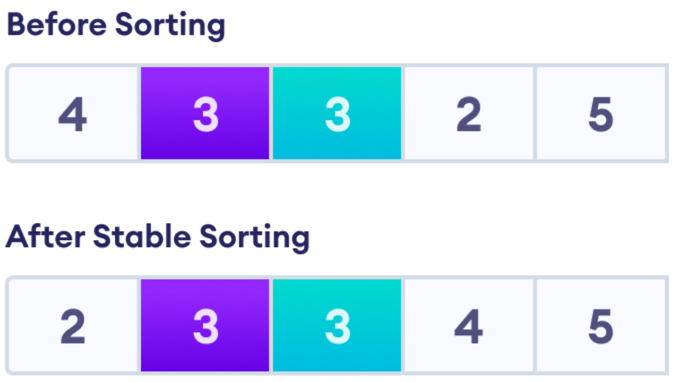
A sorting algorithm is considered stable if the two or more items with the same value maintain the same relative positions even after sorting.

For example, in the image below, there are two items with the same value 3. An unstable sorting algorithm allows two possibilities where the two positions of 3 may or may not be maintained.



Unstable sorting with two possible outcomes

However, after a stable sorting algorithm, there is always one possibility where the positions are maintained as in the original array.



Stable sorting with the positions preserved
Here's a table showing the stability of different sorting algorithms.

There are various sorting algorithms that can be used to complete this operation. And, we can use any algorithm based on the requirement.

Different Sorting Algorithms

- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge Sort
- Quicksort
- Counting Sort
- Radix Sort
- Bucket Sort
- Heap Sort
- Shell Sort

Divide and conquer

- Merge
- Quick
- heap

Sorting Algorithm	**Stability**
Bubble Sort	Yes
Selection Sort	No
Insertion Sort	Yes
Merge Sort	Yes
Quicksort	No
Counting Sort	Yes
Radix Sort	Yes
Bucket Sort	Yes
Heap Sort	No
Shell Sort	No

Complexity of Sorting Algorithms

The efficiency of any sorting algorithm is determined by the time complexity and space complexity of the algorithm.

1. Time Complexity: Time complexity refers to the time taken by an algorithm to complete its execution with respect to the size of the input. It can be represented in different forms:

- Big-O notation (O)
- Omega notation (Ω)
- Theta notation (Θ)

2. Space Complexity: Space complexity refers to the total amount of memory used by the algorithm for a complete execution. It includes both the auxiliary memory and the input.

The auxiliary memory is the additional space occupied by the algorithm apart from the input data. Usually, auxiliary memory is considered for calculating the space complexity of an algorithm.

Let's see a complexity analysis of different sorting algorithms.

Sorting Algorithm	**Time Complexity - Best**	**Time Complexity - Worst**	**Time Complexity - Average**	**Space Complexity**
Bubble Sort	n	n^2	n^2	1
Selection Sort	n^2	n^2	n^2	1
Insertion Sort	n	n^2	n^2	1
Merge Sort	nlog n	nlog n	nlog n	n
Quicksort	nlog n	n^2	nlog n	log n
Counting Sort	n+k	n+k	n+k	max

Radix Sort	n+k	n+k	n+k	max
Bucket Sort	n+k	n^2	n	n+k
Heap Sort	nlog n	nlog n	nlog n	1
Shell Sort	nlog n	n^2	nlog n	1

Bubble Sort Problems:

- 7, 11, 9, 2, 17, 4
- 14, 33, 27, 35, 10
- 6, 3, 7, 1, 2, 4
- -2, 45, 0, 11, -9

Bubble Sort Algorithm(one)

We assume a list is an array of n elements. We further assume that the swap function swaps the values of the given array elements.

```
begin BubbleSort(list)
  for all elements of list
    if list[i] > list[i+1]
      swap(list[i], list[i+1])
    end if
  end for
  return list
end BubbleSort
```

Bubble Sort Algorithm(two)

```
bubbleSort(array)

  for i <- 1 to indexOfLastUnsortedElement-1

    if leftElement > rightElement

      swap leftElement and rightElement

end bubbleSort
```

Bubble Sort Code in C

```c
#include <stdio.h>

// perform the bubble sort

void bubbleSort(int array[], int size) {

  // loop to access each array element

  for (int step = 0; step < size - 1; ++step) {

    // loop to compare array elements

    for (int i = 0; i < size - step - 1; ++i) {

      // compare two adjacent elements

      // change > to < to sort in descending order

      if (array[i] > array[i + 1]) {

        // swapping occurs if elements

        // are not in the intended order

        int temp = array[i];

        array[i] = array[i + 1];

        array[i + 1] = temp;

      }

    }

  }

}

// print array

void printArray(int array[], int size) {

  for (int i = 0; i < size; ++i) {

    printf("%d  ", array[i]);

  }
  printf("\n");

}

int main() {

  int data[] = {-2, 45, 0, 11, -9};

  // find the array's length

  int size = sizeof(data) / sizeof(data[0]);

  bubbleSort(data, size);

  printf("Sorted Array in Ascending Order:\n");

  printArray(data, size);

}
```

1. What are Heap Sort and Quick Sort?

Different sorting techniques are utilized for performing the sorting procedures as per the requirements. Usually, Quick Sort is used as it is faster, but one would use Heap Sort when memory usage is the concern.
Heap Sort is a comparison-based sorting algorithm completely based on the binary heap data structure. This is why heap sort can take advantage of the heap's properties. In the quick sort algorithm, the Divide-and-Conquer approach is utilized. Here, the entire algorithm is divided into 3 steps. The first one is to pick an element that acts as the pivot element. Next, the elements to the left of the pivot element are smaller ones and to the right are the bigger ones in value. On every partition, the previous step is repeated to sort the entire array of elements.

2. Which is the easiest sorting algorithm?

If you are dealing with sorting algorithms, then you will have noticed that Bubble Sort is the simplest one among all the others. The basic idea behind this algorithm is to scan the entire array of elements and compare every adjacent element. Now, the swapping action occurs only when the elements are not sorted.
With Bubble Sort, you just have to compare the adjacent elements, and the array gets sorted. This is why it is considered to be the simplest sorting algorithm.

3. Which is the fastest sorting algorithm in data structures?

Quicksort is considered to be the fastest one among all the other sorting algorithms. The time complexity of Quicksort is O(n log n) in its best case, O(n log n) in its average case, and O(n^2) in its worst case. Quicksort is known to be the fastest sorting algorithm because of its best performance in all the average case inputs. The speed will depend a lot on the amount of data too. As per the comparison between all the sorting algorithms, Quicksort is the fastest because of its average case inputs.