Types of Sorting

Sorting algorithms are divided into two categories: internal and external sorts.

If all the records to be sorted are kept internally in the main memory, they can be

sorted using an internal sort. However, if there are a large number of records to be

sorted, they must be kept in external files on auxiliary storage. They have to be sorted

using external sort.

1. Bubble sort

2. Insertion sort

3. Selection sort

4. Quick sort

5. Heap sort

6. Shell sort

7. Bucket sort

8. Radix sort

9. File sort

10. Merge sort

General Sort Concepts

The order in which the data is organized, either ascending or descending, is called *sort order*.

*Sort Stability*

A sorting method is said to be stable if at the end of the method, identical elements occur

in the same relative order as in the original unsorted set.

Consider the following unsorted sequence of marks to be sorted in descending order. Sort this sequence using the stable and unstable sort methods.

**Name** Uma Sera Sakira Kesha Ayse Harsha Lelo

**Marks** 80 90 93 95 83 90 83

***Solution*** The stable sort method will sort the sequence as

**Name** Kesha Sakira Sera **Harsha Ashish Lelo** Uma

**Marks** 95 93 **90 90 83 83** 80

whereas, the unstable sort method may sort the same sequence as

**Name**

**Marks**

Kasturi Sanika **Harsha Saurabh Lelo Ashish** Uma

95 93 **90 90 83 83** 80

Quick Sort

Quick sort is based on the divide-and-conquer strategy. This sort technique initially

selects an element called as *pivot* that is near the middle of the list to be sorted, and then

the items on either side are moved so that the elements on one side of pivot are smaller

and on the other side are larger.

Now, the pivot is at the right position with respect to the

sorted sequence. These two steps, selecting the pivot and arranging the elements on either

side of pivot, are now applied recursively to both the halves of the list till the list size

reduces to one.

To choose the pivot, there are several strategies. The popular way is considering the

first element as the pivot.

Thus, the recursive algorithm consists of four steps:

1. If the array size is 1, return immediately.

2. Pick an element in the array to serve as a ‘pivot’ (usually the left-most element in the

list).

3. Partition the array into two parts—one with elements smaller than the pivot and

the other with elements larger than the pivot by traversing from both the ends and

performing swaps if needed.

4. Recursively repeat the algorithm for both partitions.

Let us consider an example. Let the list of numbers to be sorted be {13, 11, 14, 11, 15,

19, 12, 16, 15, 13, 15, 18, 19}. Now, the first element 13 becomes pivot. We need to place

13 at a proper location so that all elements to its left are smaller and the right are greater.

Initially, the array is pivoted about its first element *A*[pivot] 13.

* Starting from the left end, find the first element that is greater than or equal to the pivot
* Searching backward from the right end, find the first element that is less than the pivot
* Interchange (swap) these two elements
* Repeat, searching from where we left off, until done









