Sorting

Sorting

Sorting is a process of ordering or placing a list of elements from a collection in some kind of order. It is nothing but storage of data in sorted order. Sorting can be done in ascending and descending order. It arranges the data in a sequence which makes searching easier.

For example: The below list of characters is sorted in increasing order of their ASCII values. That is, the character with lesser ASCII value will be placed first than the character with higher ASCII value.

<table>
<tr><td>d a t a s t r u c t u r e</td><td>a a c d e r r s t t t u u</td></tr>
<tr><td>Input</td><td>Output</td></tr>
</table>

Categories of Sorting

The techniques of sorting can be divided into two categories. These are:

- Internal Sorting
- External Sorting

Internal Sorting: If all the data that is to be sorted can be adjusted at a time in the main memory, the internal sorting method is being performed.

External Sorting: When the data that is to be sorted cannot be accommodated in the memory at the same time and some has to be kept in auxiliary memory such as hard disk, floppy disk, magnetic tapes etc, then external sorting methods are performed.

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**.
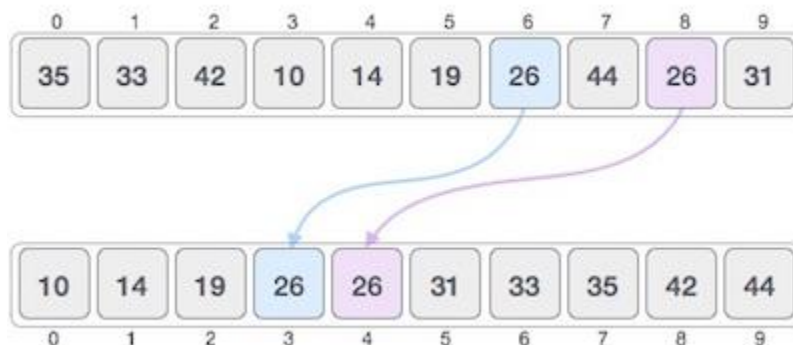
If a sorting algorithm, after sorting the contents, changes the sequence of similar content in which they appear, it is called **unstable sorting**.
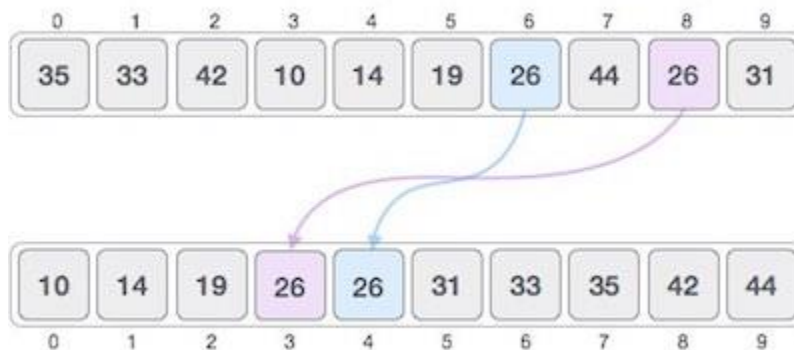
Adaptive and Non-Adaptive Sorting Algorithm

A sorting algorithm is said to be **adaptive**, if it takes advantage of already 'sorted' elements in the list that is to be sorted. That is, while sorting if the source list has some element already sorted, adaptive algorithms will take this into account and will try not to re-order them.

A **non-adaptive** algorithm is one which does not take into account the elements which are already sorted. They try to force every single element to be re-ordered to confirm their sortedness.

Important Terms

Some terms are generally coined while discussing sorting techniques, here is a brief introduction to them −

Increasing Order

A sequence of values is said to be in **increasing order**, if the successive element is greater than the previous one. For example, 1, 3, 4, 6, 8, 9 are in increasing order, as every next element is greater than the previous element.

Decreasing Order

A sequence of values is said to be in **decreasing order**, if the successive element is less than the current one. For example, 9, 8, 6, 4, 3, 1 are in decreasing order, as every next element is less than the previous element.

Non-Increasing Order

A sequence of values is said to be in **non-increasing order**, if the successive element is less than or equal to its previous element in the sequence. This order occurs when the sequence contains duplicate values. For example, 9, 8, 6, 3, 3, 1 are in non-increasing order, as every next element is less than or equal to (in case of 3) but not greater than any previous element.

Non-Decreasing Order

A sequence of values is said to be in **non-decreasing order**, if the successive element is greater than or equal to its previous element in the sequence. This order occurs when the sequence contains duplicate values. For example, 1, 3, 3, 6, 8, 9 are in non-decreasing order, as every next element is greater than or equal to (in case of 3) but not less than the previous one.

The Bubble Sort

The **bubble sort** makes multiple passes through a list. It compares adjacent items and exchanges those that are out of order. Each pass through the list places the next largest value in its proper place. In essence, each item "bubbles" up to the location where it belongs.

Example. Sort {5, 1, 12, -5, 16} using bubble sort.

5	1	12	-5	16	unsorted
5	1	12	-5	16	5 > 1, swap
1	5	12	-5	16	5 < 12, ok
1	5	12	-5	16	12 > -5, swap
1	5	-5	12	16	12 < 16, ok
1	5	-5	12	16	1 < 5, ok
1	5	-5	12	16	5 > -5, swap
1	-5	5	12	16	5 < 12, ok
1	-5	5	12	16	1 > -5, swap
-5	1	5	12	16	1 < 5, ok
-5	1	5	12	16	-5 < 1, ok
-5	1	5	12	16	sorted

Table 1: Comparisons for Each Pass of Bubble Sort

Pass	Comparisons
1	n−1
2	n−2
3	n−3
…	…
n−1	1

Algorithm for bubble sort
1. Input array A[1….n]
2. for (i = 0; i<= n − 1; i++)
```
  {
   for (j= 0; j<=n - i – 1; j++)
   {
    if (A[j] > A[j+1])     {
      temp = A[j];
      A[j] = A[j+1];
      A[j+1] = temp;
     }
   }
  }
```
3. Output: Sorted list

The Selection Sort
In this method, at first we select the smallest data of the list. After selecting, we place the smallest data in the first position and the data in first position is placed in the position where the smallest data was. After that we consider the list except the data in the first position. Again we select the (second) smallest data from the list and place it in the second position of the list and place the data in the in the second position, in the position where the second smallest data was. By repeating the process, we can sort the whole list.

Example. Sort {5, 1, 12, -5, 16, 2, 12, 14} using selection sort.

| 5 | 1 | 12 | -5 | 16 | 2 | 12 | 14 |

| 5 | 1 | 12 | -5 | 16 | 2 | 12 | 14 |

| -5 | 1 | 12 | 5 | 16 | 2 | 12 | 14 |

| -5 | 1 | 12 | 5 | 16 | 2 | 12 | 14 |

| -5 | 1 | 2 | 5 | 16 | 12 | 12 | 14 |

| -5 | 1 | 2 | 5 | 16 | 12 | 12 | 14 |

| -5 | 1 | 2 | 5 | 12 | 16 | 12 | 14 |

| -5 | 1 | 2 | 5 | 12 | 12 | 16 | 14 |

| -5 | 1 | 2 | 5 | 12 | 12 | 14 | 16 |
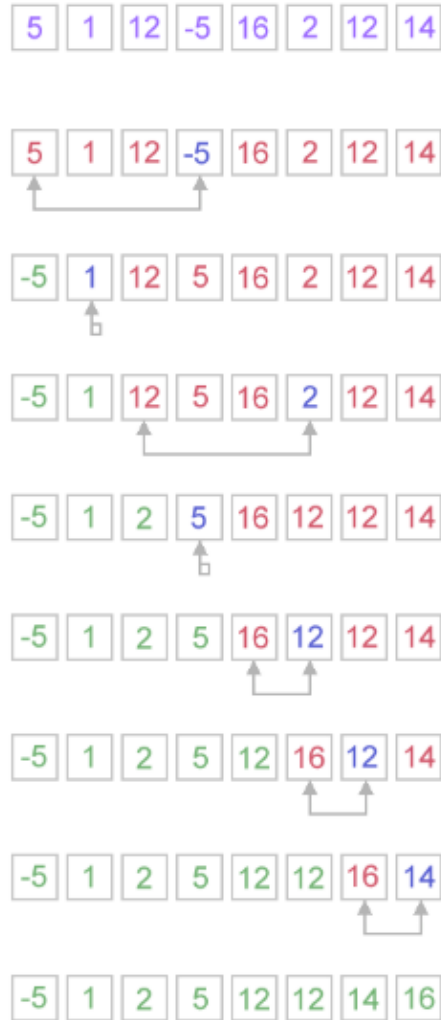
Algorithm for selection sort
1. Input array A[1…..n]
2. for(i=1; i<=n-1; i++)
       {
               small_index=i;
       for(j=i+1; j<=n; j++)
       {
               if(A[j] < A[small_index])
                       small_index=j;
       }
       temp=A[i];
       A[i]=A[small_index];
       A[small_index]=temp;
       }
3. Output: Sorted list

The Insertion Sort

It always maintains a sorted sublist in the lower positions of the list. Each new item is then "inserted" back into the previous sublist such that the sorted sublist is one item larger. Figure shows the insertion sorting process. The shaded items represent the ordered sublists as the algorithm makes each pass. We can derive simple steps by which we can achieve insertion sort.

54	26	93	17	77	31	44	55	20	Assume 54 is a sorted list of 1 item
26	54	93	17	77	31	44	55	20	inserted 26
26	54	93	17	77	31	44	55	20	inserted 93
17	26	54	93	77	31	44	55	20	inserted 17
17	26	54	77	93	31	44	55	20	inserted 77
17	26	31	54	77	93	44	55	20	inserted 31
17	26	31	44	54	77	93	55	20	inserted 44
17	26	31	44	54	55	77	93	20	inserted 55
17	20	26	31	44	54	55	77	93	inserted 20

Algorithm for insertion sort
1. Input array A[1….n]
2. for (i = 1 ; i <= n - 1; i++) {
   j = i;
   while (j > 0 && a[j-1] > a[j]) {
    temp   = a[j];
    a[j]   = a[j-1];
    a[j-1] = temp;
    j--;
   }
  }
3. Output: Sorted list.

The Merge Sort

Merge sort is a recursive algorithm that continually splits a list in half. If the list is empty or has one item, it is sorted by definition (the base case). If the list has more than one item, we split the list and recursively invoke a merge sort on both halves. Once the two halves are sorted, the fundamental operation, called a **merge**, is performed. Merging is the process of taking two smaller sorted lists and combining them together into a single, sorted, new list.
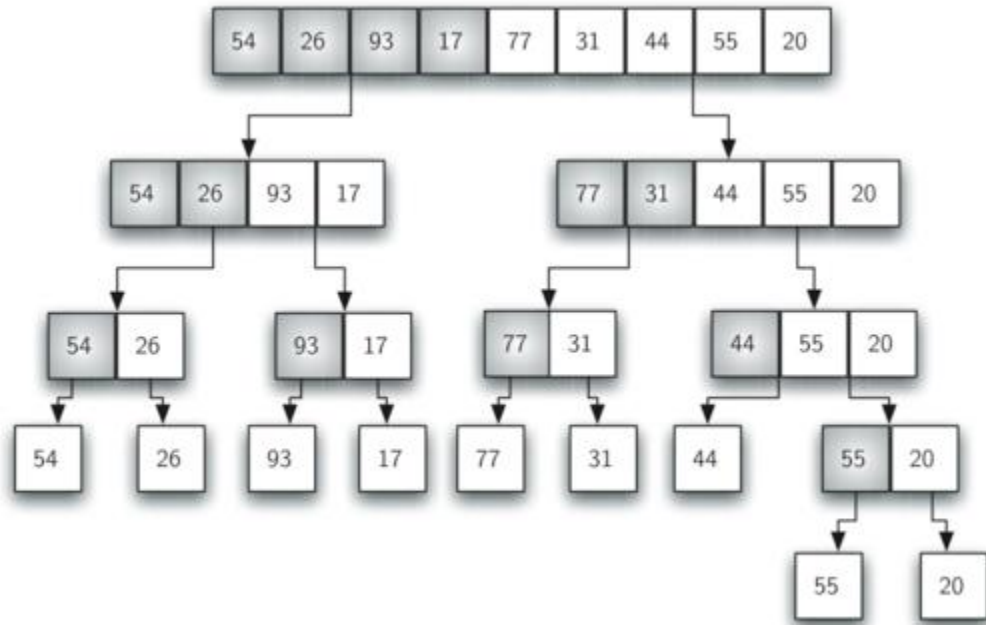
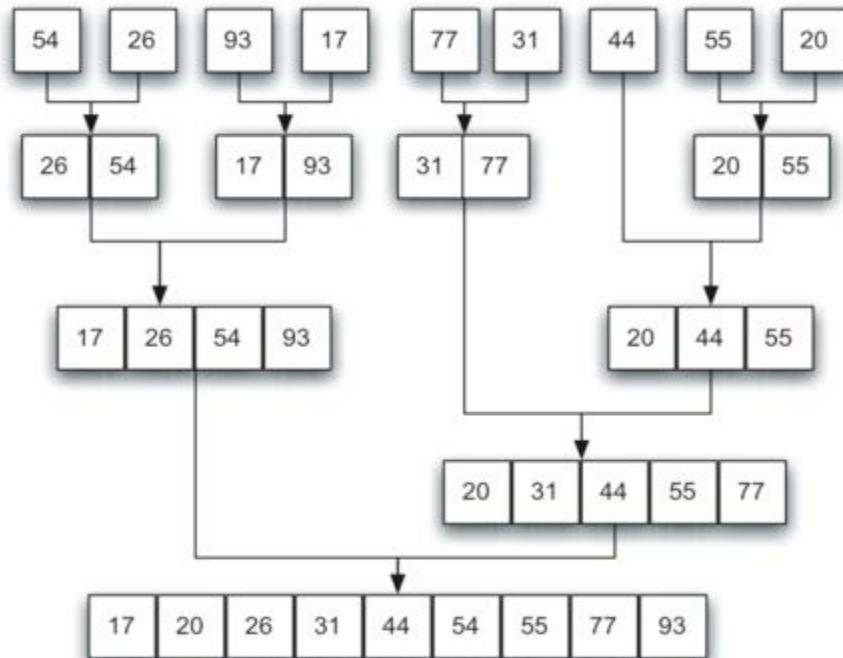Figure 1: Splitting the List in a Merge Sort

Figure 2: Lists as They Are Merged Together

Divide and Conquer Method

In divide and conquer approach, the problem in hand, is divided into smaller sub-problems and then each problem is solved independently. When we keep on dividing the subproblems into even smaller sub-problems, we may eventually reach a stage where no more division is possible. Those "atomic" smallest possible sub-problem (fractions) are solved. The solution of all sub-problems is finally merged in order to obtain the solution of an original problem.
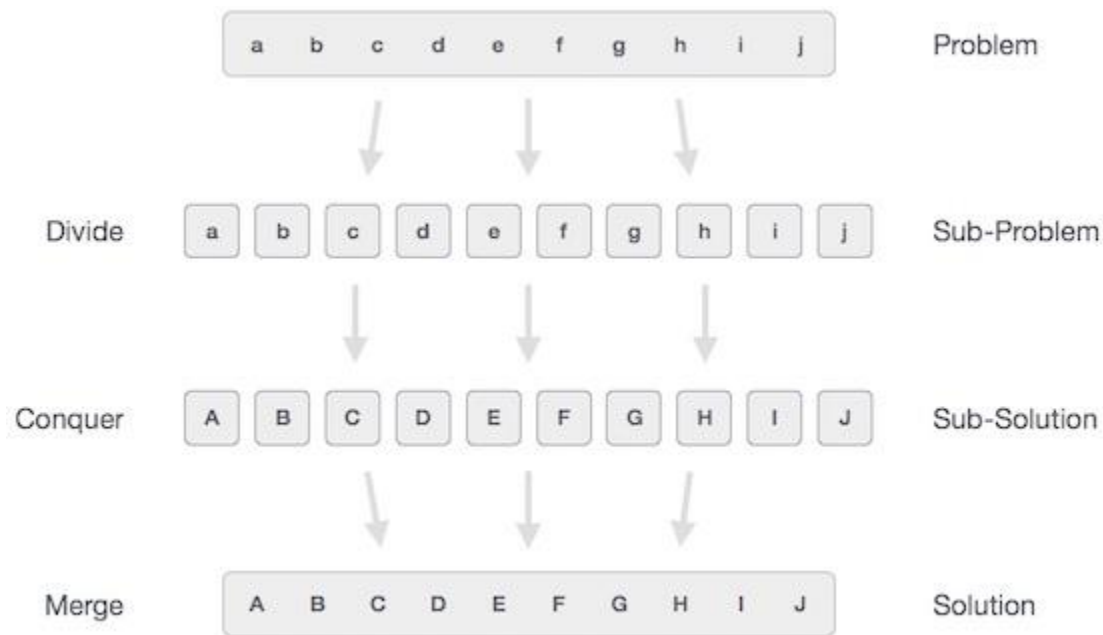
Broadly, we can understand **divide-and-conquer** approach in a three-step process.

Divide/Break

This step involves breaking the problem into smaller sub-problems. Sub-problems should represent a part of the original problem. This step generally takes a recursive approach to divide the problem until no sub-problem is further divisible. At this stage, sub-problems become atomic in nature but still represent some part of the actual problem.

Conquer/Solve

This step receives a lot of smaller sub-problems to be solved. Generally, at this level, the problems are considered 'solved' on their own.

Merge/Combine

When the smaller sub-problems are solved, this stage recursively combines them until they formulate a solution of the original problem. This algorithmic approach works recursively and conquer & merge steps works so close that they appear as one.