**Sorting Techniques**

* **Internal Sorting**
* **Bubble Sort**
* Starting from the first index, compare the first and the second element.
* If the first element is larger than the second element, they are swapped.
* Now, compare the second and the third element . Swap them if they are not in order.
* The above process continues until the last element.
* The same process continues for the remaining iterations.
* After each iteration , the largest element among the unsorted elements is placed at the end.

* **Insertion Sort**
* The first element in the array is assumed to be sorted. Take the second element and store it separately in a temporary variable.
* Compare the temporary variable value with the first element. If the first element is greater than temporary variable value , then that value is placed in front of the first element.
* Now, the first two elements are sorted . Take the third element and compare it with the elements on the left of it . Placed it just behind the element smaller than it. If there is no element smaller than it, then place it at the beginning of the array.
* Similarly , place every unsorted element at its correct position.
* **Selection Sort**
* Set the first element as **minimum.**
* Compare **minimum** with the second element. If the second element is smaller than **minimum**, assign the second element as **minimum.** Compare **minimum** with the third element. Again, if the third element is smaller, then assign **minimum** to the third element otherwise do nothing. The process continues until the last element.
* After each iteration , **minimum** is placed in the front of the unsorted list .
* For each iteration , indexing starts from the first unsorted element. Above two steps are repeated until all the elements are placed at their correct positions .
* **Quick Sort**
* This sorting algorithm based on the **divide and conquer approach .**
* An array is divided into subarrays by selecting a **pivot element** (element selected from the array). While dividing the array , the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot.
* The left and right subarrays are also divided using the same approach. This process continues until each subarray contains a single element.
* At this point , elements are already sorted. Finally, elements are combined to form a sorted array .
* **Merge Sort**
* Using the **Divide and Conquer** technique, we divide a problem into sub-problems . When the solution to each sub-problem is ready, we 'combine' the results from the sub-problems to solve the main problem .
* We have to follow these steps while calling mergeSort recursively:

mergeSort(a,lb,ub)

if(lb<ub){

mid=(lb+ub)/2

mergeSort(a,lb,mid);

mergeSort(a,mid+1,ub);

merge(a,lb,mid,ub);

}

* After this recursive mergeSort function we have to focus on the merge function which is the backbone of merge sort because in this we have to merge the sub-arrays in respective order .
* Merge(a,p,q,r){

N1<-[q-p+1]

N2<-[r-q]

Declare the two arrays of length N1 and N2 respectively and assign the values of sub-arrays to that arrays .

L[i]=a[p+i]

M[j]=a[q+j+1]

And , then compare the elements of both the arrays and merge them according to the order .