1. Bubble Sort : It is simple algorithm to sort the list. It grows like a bubble by comparing adjacent values, swaps them if they are not in the correct order. whenever it reaches the end index of the list.

1.Let us assume that list is not sorted and flag is set to false
2. Continue scanning until the algorithm sorts all the element in the list.
    2.1 Switch the flag of sorted = true
    2.2 Compare the adjacent element of the list. If the adjacent element is greater than previous element, swap the element. Continue till this step till the length of the list.
    2.3 Once you are exiting, set flag to false for sorted

Time Complexity: O(n²)

2. Insertion Sort : It is another simple algorithm to sort the list. It works considerably fast on smaller input data. It states that an item is picked and compared with remaining item and is inserted into correct position by shifting remaining non-sorted data by 1 position. This process is similar to how human plays card game.ment

1. It is assumed 1st item is always sorted. Algorithm starts with 2nd item. Continue scanning element till the end of list is reached.
    1.1 Picking the unsorted element and storing in a variable.
    1.2 Let the index of current item be set to index of next item – 1
    1.3 Continue scanning the list and if the value of current item is greater than the current value, shift the remaining unsorted element by 1 position to the right. The index of that element is also reduced.
    1.4 If the current element isn't smaller than the biggest sorted element (i.e. it's bigger), it's simply inserted at the end where it belongs

Time Complexity: O(n²)

3. Selection Sort : This algorithm segments the list into two parts: sorted and unsorted. We continuously remove the smallest element of the unsorted segment of the list and append it to the sorted segment. We consider the leftmost part of the list as sorted. Search the element and swap the smallest element with the 1st element of the unsorted list.

1. Continue scanning element till the end of list is reached.
1.1 We assume that the first item of the unsorted segment is the smallest element.
1.2 Continue iterating over the unsorted item. If the value of the current index is less than first item. Set that smallest to that index.
1.3 Swap the smallest with of the lowest unsorted element with the first unsorted element.

Time Complexity:O(n²)

4 . Merge Sort : It is one of divide and conquer algorithm, which splits a list in half, and keeps splitting the list by 2 until it only has singleton element. Adjacent elements become sorted pairs, then sorted pairs are merged with other sorted pairs. This process continues until we get a sorted list with all the elements of the unsorted input list. Sorting is done by comparing the smallest elements of each half. The singleton element of each list are compared with other singleton element. If the first half begins with a smaller value, then we add that to the sorted list. We then compare the second smallest value of the first half with the first smallest value of the second half. Every time we select the smaller value at the beginning of a half, we increment the index of that item by one.

//NAME: SATYAM KUMAR
//AAMSTY AKMRU

//AAAKMMRSTUY

Time Complexity: O(n*logn)

5. Quick Sort: It is another divide and conquer sorting algorithm. It is one of the most widely used sorting algorithm. Sorting of list is carried out around the pivot element, which can be any element from starting index till length of list. Quick Sort begins by partitioning the list - picking one value of the list that will be in its sorted place. This value is called a pivot. All elements smaller than the pivot are moved to its left. All larger elements are moved to its right. Then the pivot item is swapped with the last index of smaller item. In this way,pivot gets placed in the correct index position of the list. 1st element,last element or median of the element or randomized element can be choosen as pivot.

Time Complexity: O(n*logn)

6. Heap Sort : This is one of the popular used algorithm. It is similar to the Insertion and Selection sorts, which segments the list into sorted and unsorted parts. It converts the unsorted segment of the list to a Heap data structure, so that we can efficiently determine the largest element. Convert the list into Binary tree and convert into Max Heap. The root is the largest element. We swap root with the element present in the last index of the list. Again apply Max Heap in remaining item with 1 element less from the length of list. We iterate this process of building the heap until all nodes are removed.

Time Complexity: O(n*logn)

7. Radix Sort : Radix Sort compares digit or character, starting from the last index. Radix sort works only for integer values. It compares digit by digit sorting starting from least significant digit(i.e. digit present at the right) to most significant digit(i.e. digit present at the left) for the integer part. Radix sort uses counting sort as a subroutine to sort.

Time Complexity: O(n*k), where k is constant,denoting the maximum number of digits present.

8. Bucket Sort : In this method of sorting, the elements are divided into several groups called buckets. The elements inside each bucket are sorted using any of the suitable sorting algorithms or recursively calling the same algorithm. Each bucket has some specific range. The elements ipresent inside the bucket are sorted using some other algorithm. Finally, the elements of the bucket are gathered to get the sorted array. It works for real valued data.

Worst case Time Complexity:O(n²)
When there are elements of close range in the array, they are likely to be placed in the same bucket. This may result in some buckets having more number of elements than others

Best Case Complexity: O(n+k)
It occurs when the elements are uniformly distributed in the buckets with a nearly equal number of elements in each bucket.