INSERTION SORT

This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'inserted in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, **insertion sort**.

The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array). This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$, where **n** is the number of items.

Now let's see the visual representation of the algorithm −

Program for Insertion sort

```
def  insertionsort(a):
        for index in  range(1,len(a):
            currentvalue=a[i]
            position=i
            while position>0 and a[position-1]>currentvalue:
                a[position]=a[position-1]
                position=position-1
                a[position]=currentvalue
list=[50,60,40,30,20,70]
print( “Original list is:”,list)
insertionsort(list)
print(”List after insert:”,a)
```

Output:

Original list is=[50,60,40,30,20,70]

List afterinsert:[20,30.40,50,60,70]

2. SELECTION SORT

In the **selection sort** algorithm, an array is sorted by recursively finding the minimum element from the unsorted part and inserting it at the beginning. Two subarrays are formed during the execution of Selection sort on a given array.

- The subarray, which is already sorted
- The subarray, which is unsorted.

During every iteration of selection sort, the minimum element from the unsorted subarray is popped and inserted into the sorted subarray.

Let's see the visual representation of the algorithm −

Now let's see the implementation of the algorithm −

Program for Selection sort:

```
def selectionSort(alist):
    for i in range(len(alist)-1,0,-1):
        pos=0
        for location in range(1,i+1):
            if alist[location]>alist[pos]:
                pos= location
                temp = alist[i]
                alist[i] = alist[pos]
                alist[pos] = temp
alist = [54,26,93,17,77,31,44,55,20]
selectionSort(alist)
print(a list)
```

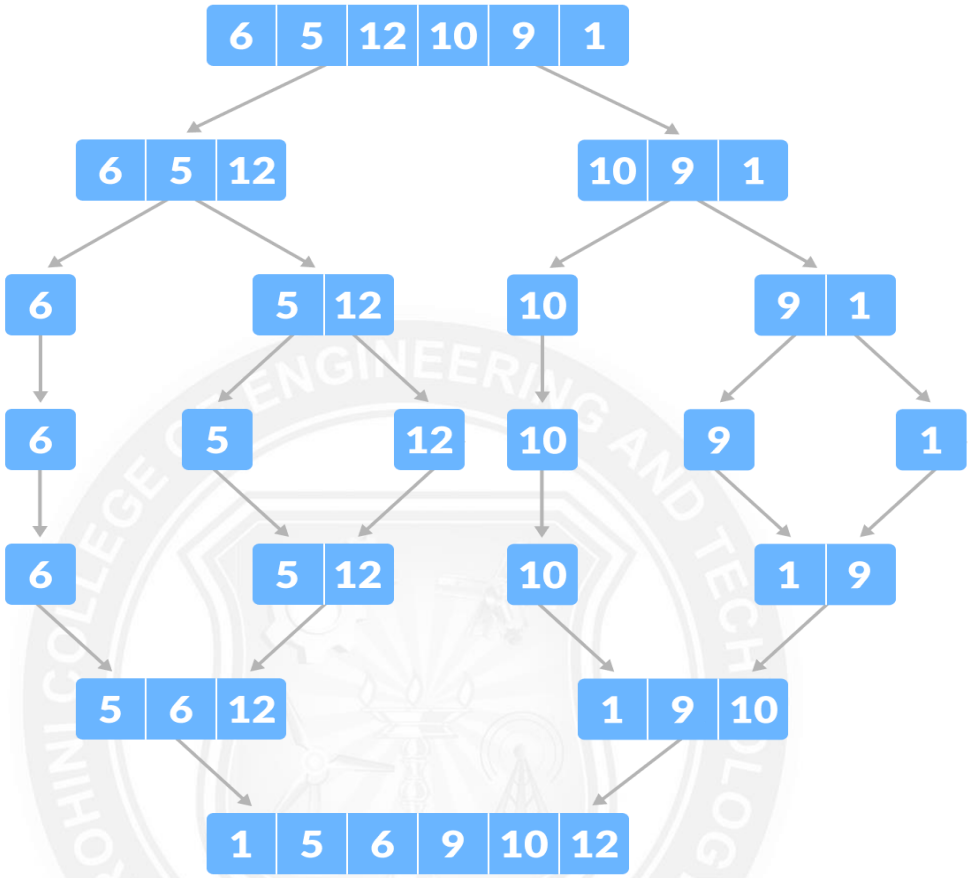
Output:

```
[17, 20, 26, 31, 44, 54, 55, 77, 93]
```

3. MERGE SORT

Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being O(n log n), it is one of the most respected algorithms.

Merge sort first divides the array into equal halves and then combines them in a sorted manner.



Program for Merge sort

```
def mergeSort(alist):
    print("Splitting ",alist)
    if len(alist)>1:
        mid = len(alist)//2
        lefthalf = alist[:mid]
        righthalf = alist[mid:]
        mergeSort(lefthalf)
        mergeSort(righthalf)
        i=0
        j=0
        k=0
```

```
    while i < len(lefthalf) and j < len(righthalf):
        if lefthalf[i] < righthalf[j]:
            alist[k]=lefthalf[i]
            i=i+1
        else:
            alist[k]=righthalf[j]
            j=j+1
        k=k+1
    while i < len(lefthalf):
        alist[k]=lefthalf[i]
        i=i+1
        k=k+1
    while j < len(righthalf):
        alist[k]=righthalf[j]
        j=j+1
        k=k+1
  print("Merging ",alist)
alist = [50, 60, 40, 20, 70, 100]]
mergeSort(alist)
print(alist)
```

Output:

Original list is: [50, 60, 40, 20, 70, 100]

Sorted list is: [20, 40, 50, 60, 70, 100]

HISTOGRAM

To create a **histogram,** the first step is to create bin of the ranges, then distribute the whole range of the values into a series of intervals, and the count the values which fall into each of the intervals. Bins are clearly identified as consecutive, non-overlapping intervals of variables.

Program:

```
def  histogram(items):
    for n in items:
        output=''
        times=n
        while (times>0):
            output+='*'
        times=times-1
        print(output)
        histogram([2,3,4,3,2])
```

Output:

```
**
***
****
***
**
```

Example :2

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import colors
from matplotlib.ticker
import PercentFormatter
```

```
# Creating dataset
np.random.seed(23685752)
N_points = 10000
n_bins = 20
# Creating distribution
x = np.random.randn(N_points)
y = .8 ** x + np.random.randn(10000) + 25
# Creating histogram
fig, axs = plt.subplots(1, 1,
 figsize =(10, 7),
tight_layout = True)
axs.hist(x, bins = n_bins)
# Show plot
plt.show()
```

Output :