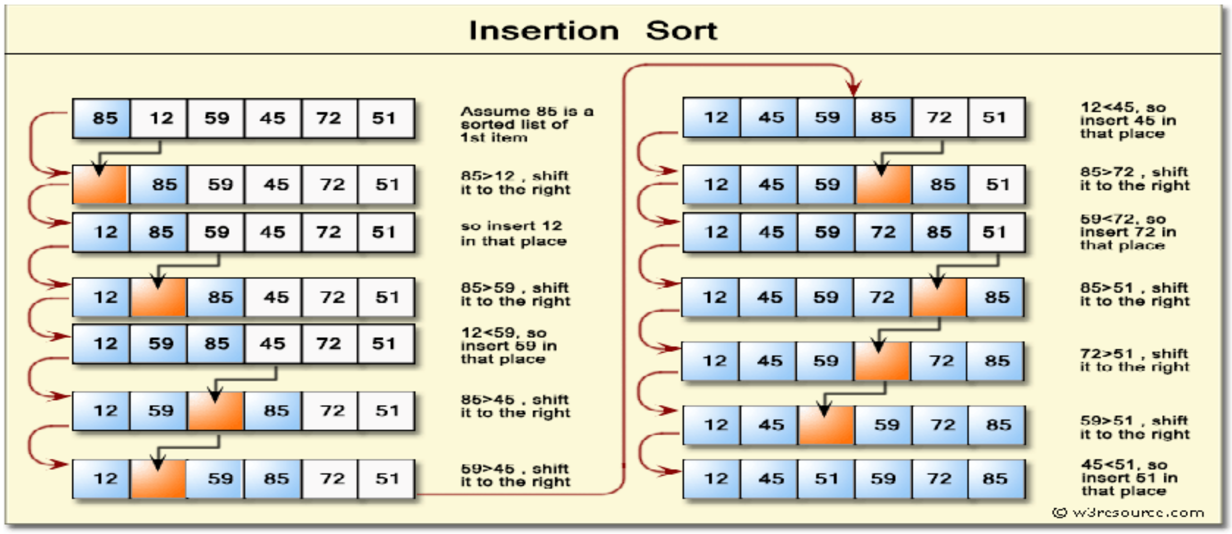
1. Insertion sort

Insertion sort is an elementary sorting algorithm that sorts one element at a time. Most humans, when sorting a deck of cards, will use a strategy similar to insertion sort. The algorithm takes an element from the list and places it in the correct location in the list. This process is repeated until there are no more unsorted items in the list.

Example:


Insertion Sort
Assume 85 is a sorted list of 1st item
85>12 , shift it to the right
so insert 12 in that place
85>59 , shift it to the right
12<59, so insert 59 in that place
85>45 , shift it to the right
59>45 , shift it to the right
12<45, so insert 45 in that place
85>72 , shift it to the right
59<72, so insert 72 in that place
85>51 , shift it to the right
72>51 , shift it to the right
59>51 , shift it to the right
45<51, so insert 51 in that place
© w3resource.com

Program:

```
a=list()
n=int(input("Enter size of list"))
for i in range(n):
    a.append(int(input("Enter list elements")))
 print("Before sorting",a)
for i in range(1,n):
  key=a[i]
  j=i-1
  while j>=0 and key<a[j]:
     a[j+1]=a[j]
     j-=1
     a[j+1]=key
print("After sorting(using insertion sort)",a)
```

Output

```
Enter size of list6
Enter listelements4
Enter listelements33
Enter list elements6
Enter listelements22
Enter list elements6
Enter list elements-9
Before sorting [4, 33, 6, 22, 6, -9]
After sorting(using insertion sort) [-9, 4, 6, 6, 22, 33]
```

2. Selection Sort

The selection sort algorithm starts by finding the minimum value in the array and moving it to the first position. This step is then repeated for the second lowest value, then the third, and so on until the array is sorted.

Example

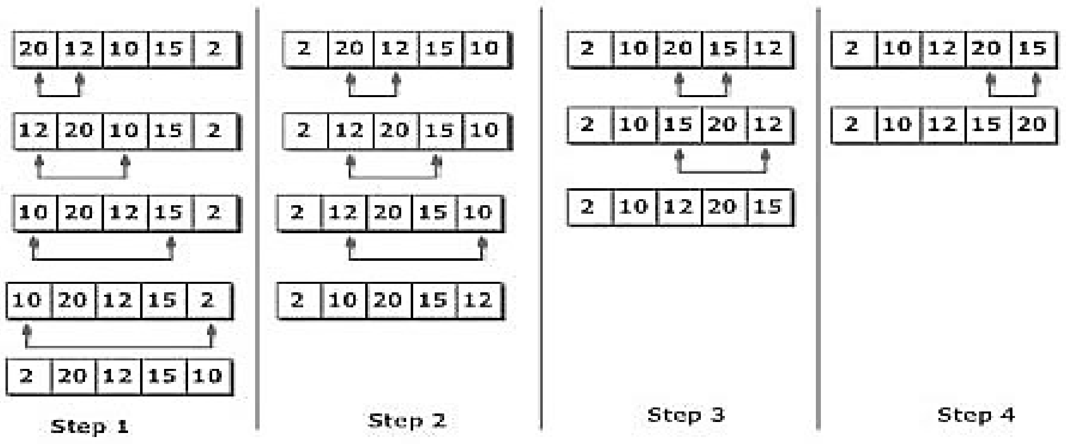


Figure: Selection Sort

```
Program
a=list()
n=int(input("Enter size of list"))
for i in range(n):
    a.append(int(input("Enter list elements")))
print("List before sorting",a)
for i in range(0,n):
   j=i+1
  for j in range(j, n):
     if a[i]> a[j]:
         temp=a[i]
         a[i]=a[j]
         a[j]=temp
print("Sorted list(using Selection Sort)=",a)
```

Output:

```
Enter size of list5
Enter list elements12
Enter list elements-5
Enter list elements4
Enter listelements48
Enter listelements98
List before sorting [12, -5, 4, 48, 98]
Sorted list(using Selection Sort)= [-5, 4, 12, 48, 98]
```

3. Quadratic Equation:

Formula :

$$ax^2+bx+c \qquad = \frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

Program

```
import cmath
 a = int(input("Enter the coefficients a:"))
b=int(input("Enter the coefficients b: "))
c = int(input("Enter the coefficients c: "))
  d = b**2-4*a*c # discriminant
  x1 = (-b+cmath.sqrt((b**2)-(4*(a*c))))/(2*a)
  x2 = (-b-cmath.sqrt((b**2)-(4*(a*c))))/(2*a)
  print ("This equation has two solutions: ", x1, " or", x2)
```

Output

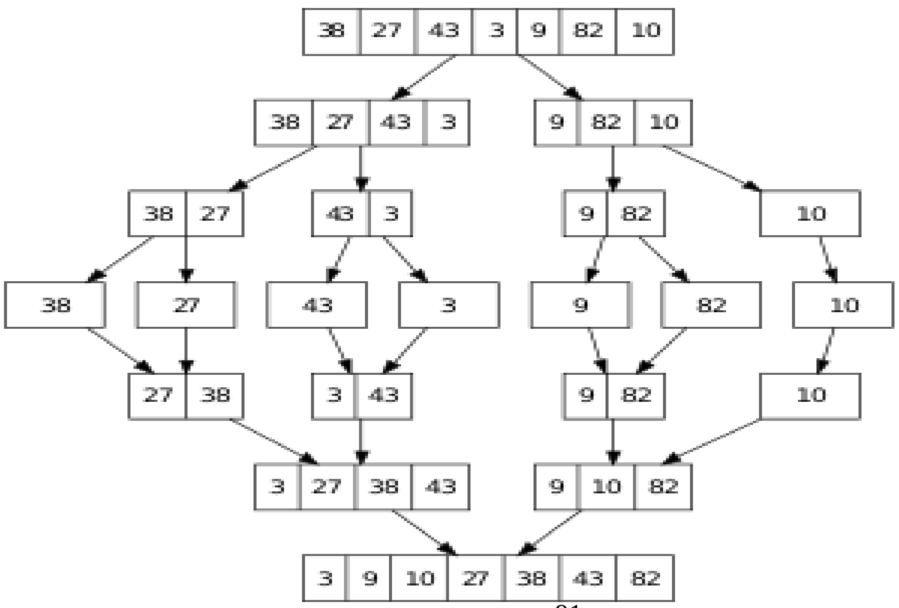
```
Enter the coefficients a: 5
Enter the coefficients b: 1
Enter the coefficients c: 2
This equation has two solutions: (-0.1+0.6244997998398398j) or (-0.1-0.6244997998398398j)

Enter the coefficients  a: 1
Enter the coefficients b: -5
Enter the coefficients  c: 6
This equation has two solutions: (3+0j) or (2+0j)
```

4. Merge sort

Merge sort works as follows:

a. Divide the unsorted list into n sub lists, each containing 1 element (a list of 1 element is considered sorted).
b. Repeatedly merge sub lists to produce new sorted sub lists until there is only 1 sub list remaining. This will be the sorted list.

Example



Program:

```
def merge(left, right):
   result = []
   i, j = 0, 0
   while (i < len(left) and j<len(right)):
      if left[i] < right[j]:
          result.append(left[i])
          i+= 1
      else:
          result.append(right[j])
          j+= 1
   result=result+left[i:]
   result=result+right[j:]
   return result
def mergesort(list):
    if len(list) < 2:
        return list
    middle = len(list)//2
    left = mergesort(list[:middle])
    right = mergesort(list[middle:])
    return merge(left, right)
a=list()
n=int(input("Enter size of list"))
for i in range(n):
     a.append(int(input("Enter list elements")))
print("Unsorted list is",a)
print("Sorted list using merge sort is",a)
```

Output

```
Enter size of list5
Enter list elements21
Enter list elements1
Enter list elements-8
Enter list elements14
Enter list elements18
Unsorted list is [21, 1, -8, 14, 18]
Sorted list using merge sort is [-8, 1, 14, 18, 21]
```

5. LIST

- List is a sequence of values, which can be of different types. The values in list are called "elements" or "items"
- Each elements in list is assigned a number called "position" or "index"
- A list that contains no elements is called an empty list. They are created with empty brackets[]
- A list within another list is nested list

Creating a list :

The simplest way to create a new list is to enclose the elements in square brackets ([])

```
[10,20,30,40]
[100, "python" , 8.02]
```

1.LIST OPERATIONS:

1. Concatenation of list
2. Repetition of list

Concatenation: the '+' operator concatenate list

```
>>> a = [1,2,3]
>>> b = [4,5,6]
>>> c = a+b
 >>> Print (a*2) => [1,2,3,1,2,3]
```

Repetition: the '*' operator repeats a list a given number of times

```
>>> a = [1,2,3]
>>> b = [4,5,6]
>>> print (a*2)= [1,2,3,1,2,3]
```

2. List looping: (traversing a list)

1. Looping in a list is used to access every element in list
2. "for loop" is used to traverse the elements in list

```
eg: mylist = ["python","problem",100,6.28]
        for i in range (len (mylist)):
            print (mylist [i])
```

3.List Slices:

A subset of elements of list is called a slice of list.

```
Eq: n = [1,2,3,4,5,6,7,8,9,10]
        print (n[2:5])
        print (n[-5])
         print (n[5: ])
         print (n[ : ])
```

4. Aliasing and cloning:

- when more than one variables refers to the same objects or list, then it is called aliasing.

```
a= [5,10,50,100]
 b=a
 b[0] = 80
 print ("original list", a) = [5,10,50,100]
 print ("Aliasing list", b) = [80,5,10,50,100]
```

- Here both a & b refers to the same list. Thus, any change made with one object will affect other, since they are mutable objects.
- in general, it is safer to avoid aliasing when we are working with mutable objects

5. Cloning:

- Cloning creates a new list with same values under another name. Taking any slice of list create new list.
- Any change made with one object will not affect others. the easiest way to clone a new list is to use "slice operators"

```
 a = [5,10,50,100]
b= a[ : ]
b[0] = 80
Print (" original list", a) = [5,10,50,100]
 Print (" cloning list", b) = [5,10,50,100]
```

List parameter:

- List can be passed as arguments to functions the list arguments are always passed by reference only.
- Hence, if the functions modifies the list the caller also changes.

```
Eq:          def head ():
                 del t[ 0 ]
           >>> letters = ['a','b','c']
          >>> head (letters)
          >>> letters
                 ['b','c']
```

In above,

The parameters 't' and the variable 'letters' or aliases for the same objects

An alternative way to write a function that creates and return a new list

```
Eq:          def tail (t):
                 return t [1:]
           >>> letters = ['a','b','c']
           >>> result = tail (letters)
           >>> result
                 ['b','c']
```

In above,

The function leaves the original list unmodified and return all element in list except first element

6. TUPLES:

A tuple is a sequence of value which can be of any type and they are indexed by integers. Values in tuple are enclosed in parentheses and separated by comma. The elements in the tuple cannot be modified as in list (i.e) tuple are immutable objects

Creating tuple:

Tuple can be created by enclosing the element in parentheses separated by comma

```
t =  ('a','b','c','d')
```

To create a tuple with a single element we have to include a final comma

```
>>> t = 'a',
>>> type (t)
        < class 'tuple'>
```

Alternative way to create a tuple is the built-in function tuple which mean, it creates an empty tuple

```
>>> t = tuple ()
>>> t
>>> ( )
```

Accessing element in tuple:

If the argument in sequence, the result is a tuple with the elements of sequence.

```
>>>t= tuple('python')
>>> t
        ('p','y','t','h','o','n')
      t = ('a','b',100,8.02)
      print (t[0]) = 'a'
      print (t[1:3]) = ('b', 100 , 8.02)
```

Deleting and updating tuple:

Tuple are immutable, hence the elements in tuple cannot be updated / modified

But we can delete the entire tuple by using keyword 'del'

```
Eg 1: a = (' programming', 200, 16.54, 'c', 'd')
#Try changing an element.
          a[ 0 ] = 'python'       <-------- Error,modifying not possible
          print (a [0])
 Eg: # Deletion of tuple
          a = ('a','b','c','d')
         del (a) :-------- delete entire tuple
         del a [1] <--------- error,deleting one element in tuple not possible
 Eg: # replacing one tuple with another
          a = ('a','b','c','d')
          t = ('A',) + a[1: ]
          print (t) <------ ('a','b','c','d')
```

Tuple Assignment:

- Tuple assignment is often useful to swap any number of values
- the number of variables in left and right of assignment operators must be equal
- A single assignment to paralleling assign value to all elements of tuple is the major benefit of tuple assignment

Eg: Tuple swapping in python

```
A= 100
B= 345
C= 450
print (" A & B:", A,"&",B)
     # Tuple assignments for two
variables A,B = B,A
print (" A&B after tuple assignment : ",A,"&",B)
     # Tuple assignment can be done for no of
variables A,B,C = C,A,B
print (" Tuple assignment for more variables:",
A,"&",B,"&",C) Output
A & B: 100 & 345
 A&B after tuple assignment : 345 & 100
 Tuple assignment for more variables: 450 & 345 & 100
```

Tuple as return value:

- Generally, function can only return one value but if the value is tuple the same as returning the multiple value
- Function can return tuple as return value

Eg: # the value of quotient & remainder are returned as tuple

```
def mod_div
    (x,y): quotient
    = x/y remainder
    = x%y
    return quotient, remainder
# Input the seconds & get the hours minutes &
second sec = 4234
minutes,seconds= mod_div
(sec,60)
hours,minutes=mod_div(minutes,
60)
print("%d seconds=%d hrs:: %d min:: %d
sec"%(sec,hours,minutes,seconds)) Output:
4234onds=1 hrs:: 10 min:: 34 sec
```

7. Histogram

```
def histogram( items ):                    Output
    for n in items:                            **
        output = ''                            ***
        times = n                              ******
        while( times > 0 ):                    *****
          output += '*'
          times = times - 1
        print(output)

histogram([2, 3, 6, 5])
```

Two marks:

1. **Write a program to create list with n values**

```
a=list()
n=int(input("Enter the  size of list”))
for i in range (n):
  a.append(int(input("Enter the list element")))
print("Created List=",a)
```

Output

```
Enter the size of list 5
Enter the list of element20
Enter the list of element30
Enter the list of element78
Enter the list of element12
Enter the list of element65
Created List= [20, 30, 78, 12, 65]
```

2. **What is dictionary?**

A dictionary is an unordered set of key: value pair. In a list, the indices have to be integers; in a dictionary they can be any type. A dictionary contains a collection of indices, which are called keys, and a collection of values. Each key is associated with a single value. The association of a key and a value is called a key-value pair. Dictionary is created by enclosing with curly braces {}.

Eg:

```
>>>
dictionary={"RollNo":101,2:(1,2,3),"Name":"Ramesh",20:20.50,Loc":['Chennai']}
>>> dictionary
{'Name':'Ramesh', 'Loc':['Chennai'], 2:(1,2.3), 20: 20.0, 'RollNo': 101}
```

3. **Write program to rotate values in the list.(counter-clock wise)**

```
a=list()
n=int(input("Enter the number of list elements"))
for i in range (n):
  a.append(int(input("Enter list element")))
rotate=int(input("Enter the rotation value(Give negative value for
counter cock-wise)"))
print("Created List=",a)
print("List rotated is",a[rotate:]+a[:rotate] )
```

Output

```
Enter the number of list elements 5
Enter list element 30
Enter list element 98
Enter list element 45
Enter list element 49
Created List= [30, 98, 45, 49]
Enter the rotation value(Give negative value for counter cock-wise)-2
List rotated in counter clockwise [45, 49, 30, 98]
```

4. **What is data structure? List out the data structures used in Python**

A data structure is a particular way of organizing and storing data in a computer so that it can be accessed and modified efficiently.

Python data structures:-

1. List
2. Tuples
3. Dictionary

5. **Compare all the three data structures in Python**

	List	**Tuples**	**Dictionary**
Mutable	List is mutable	Tuples are immutable	Keys must be immutable. Values may mutable
Indexing	A positive integer is used for indexing and always starts with zero. Reverse index is supported.	A positive integer is used for indexing and always starts with zero. Reverse index is supported.	Indexing is done with ‘key’. Index may be of any type. Values can be accessed only through key
Declaration	List=[05,’Ashok’,450]	Tuple=(‘Sun’,’Mon’)	Dictionary={“Key”:”value”}

6. **Difference between list append and list extend**

1. append() is a function adds a new element to the end of a list.
2. extend() is a function takes a list as an argument and appends all of the elements.

append()	extend()
>>>a=[10,20,30] >>>b=[40,50] >>>a.append(b) >>>print(a) [10,20,30,[40,50]]	>>>a=[10,20,30] >>>b=[40,50] >>>a.extend(b) >>>print(a) [10,20,30,40,50]

7. **What is mutability? Is tuple is mutable**

In object-oriented and functional programming, an immutable object (unchangeable object) is an object whose state cannot be modified after it is created. This is in contrast to a mutable object (changeable object), which can be modified after it is created.

Tuple is immutable.

8. **Write a program to add or change elements in a dictionary.**

```
>>> dictionary={"Roll No":101,2:(20.00,30),"Name":"Ramesh",20:200.00, "Loc":['Chennai']}
>>> dictionary
{'Name': 'Ramesh', 'Loc': ['Chennai'], 2: (20.0, 30), 20: 200.0, 'Roll No': 101}
>>> dictionary['Roll No']=105
>>> dictionary
{'Name': 'Ramesh', 'Loc': ['Chennai'], 2: (20.0, 30), 20: 200.0, 'Roll No': 105}
```

9. **How to convert a string to list of characters and words**.

```
>>> str1=”Hello”
>>> list1=list(str1)
>>> list1
['H', 'e', 'l', 'l', 'o']
```

10. **What is zip operation in tuples. Give an example**.

Zip is a built-in function that takes two or more sequences and returns a list of tuples where each tuple contains one element from each sequence. This example zips a string and a list:

```
>>> s = 'abc'
>>> t = [0, 1, 2]
>>> zip(s, t)
<zip object at 0x7f7d0a9e7c48>
```

The result is a zip object that knows how to iterate through the pairs. The most common use of zip is in a for loop:

```
>>> for pair in zip(s, t):
        print(pair)
('a', 0)
('b', 1)
('c', 2)
```