Data Structure -1 Linear List

Learning Objectives

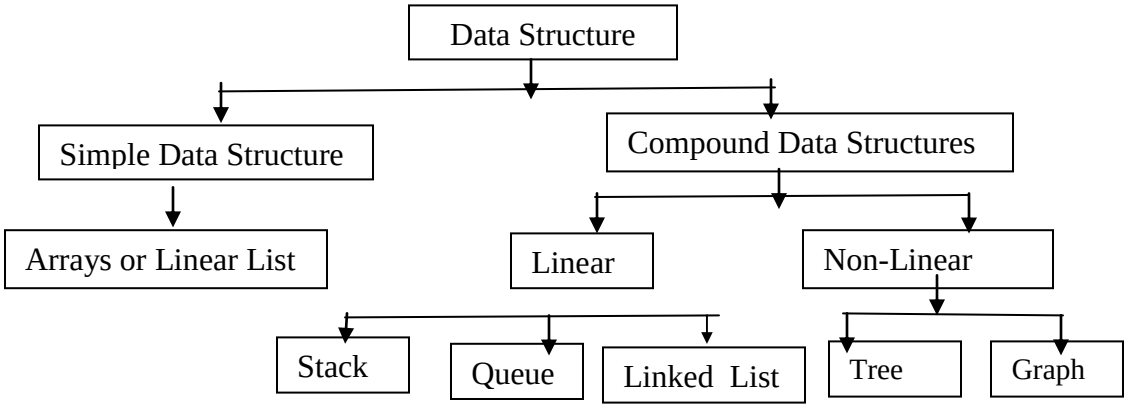
At the end of this chapter, the student will be able to:

- *Understand data structures*
- *Understand sequential memory allocation*
- *Learn basic list operations -*
- *Traversal in a list*
- *Insertion in a sorted list*
- *Deletion of an element from a list*
- *Learn Searching Techniques in a list-*
- *Learn Sorting a list*

Data Structures

A data structure is a organizing group of similar or dissimilar data which can be processed as a single unit. Data Structures are very important in a computer system, as these not only allow the user to combine various data types in a group but also allow processing of the group as a single unit thereby making things much simpler and easier.

The data structures are classified into two types:



1. Simple Data Structures. These data structures are normally built from primitive data types like integers, reals, characters, Boolean. Following data structures can be termed as simple data structures

- Array or Linear Lists

2. Compound Data Structures. Simple data structures is combined in various ways to form more complex structures called compound data structures. Compound data structures are classified into following two types:

- **Linear data structures**. These data structures are single level data structures. A data structure is said to be linear if its elements form a sequence.
  - examples: (a) Stack (b) Queue (c) Linked List
- **Non-Linear data structures**. These are multilevel data structures.
  - Example: Tree, Graph etc.

Implementation of List in memory

- **Array:** Array: It is a compact way of collecting data types where all entries must be of the same data type.

Syntax of writing an array in python:

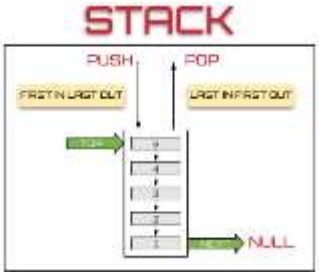
```
import array as arr
a = arr.array("I",[3,6,9])
type(a)
```

- **list:** List in Python is used to store collection of heterogeneous items. It is described using the square brackets [] and hold elements separated by comma.
  Example: [101,"Prajwal", 12, 'Computer']

In Python, arrays are implemented through List data types as **linear List** or through **NumPy arrays**.

There are three types of arrays
i. One-dimensional Array
ii. Two-dimensional Array
iii. Multi-dimensional Array

STACKS: ***A stack is an ordered collection of items in which an Element may be inserted or deleted only at same end.***

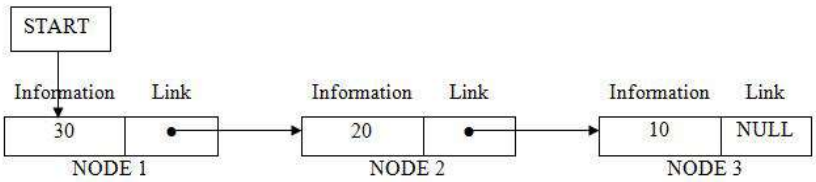


- This end is called as the **top** of the stack.
- The end opposite to top is known as the **base**.
- Stacks are sometimes known as **LIFO** (Last In First Out).

QUEUES:*A queue is an ordered collection of items where an item is inserted at one end called the "rear" and an existing item is removed at the other end, called the "front".*

- Queue is also called as **FIFO** list i.e. **First-In First-Out**.

LINKED LIST:
A linked list is a linear collection of data elements called nodes.



GRAPH: It is a data structure that consists of a finite set of vertices called nodes, and a finite set of ordered pair (u,v) called edges. It can be classified as direction and weight

TREE: Trees are multilevel data structure having a hierarchical relationship among its elements called nodes. Here each node has at most two children.

Sequential allocation of memory:
Elements in list are stored in memory in sequence of their declaration. This makes indexing of a list independent of the size of the list or the value of the index. When items are appended or inserted, the array of references is resized.

List Operations: Main operations of any list in Python

- **Traversal:** Traversal of a data structure means processing all the data elements of it, sequentlly.
- **Insertion:** Insertion means addition of a new data element in a data structure.
- **Deletion:** Deletion means removal of a data element from a data structure. The data element is searched for before its removal.
- **Searching:** Searching involves searching for the specified data element in a data structure.
- **Sorting:** Arranging data elements of a data structure in a specified order is called sorting.
- **Merging:** Combining elements of two similar data structures to form a new data structure of same type, is called merging.

Traversal in a List
Traversal means to move in a list in sequential manner starting from first element to the last element. The algorithm and code for traversal in the list is as follows:

Algorithm	**Program Code**

1. count=0
2. Repeat steps 3 through 4 until count>U
3. print DATA_LIST[count]
4. count=count+1
#End of repeat
5. END
	def traversal(DATA_LIST):
	for x in DATA_LIST:
	print (x)
	print ( )

In the above code, the for loop runs till the time all elements of the list are displayed.

Insertion of an element in a sorted list
Insertion of a new element in a sorted list has to be inserted at an appropriate position so that it falls in order with the other elements of the list.

- If insertion is done in an unsorted list, the new element is inserted at the end of the list, using append().
- If the list is sorted, then the new element has to be inserted at an appropriate position so that it falls in order with the other elements of the list.
- To insert the new element at the proper position, the rest of the elements have to be shifted as shown in

Element to be inserted: **40**

Original List	10	20	30	50	60	70	
Space created	10	20	30		50	60	70
after insertion	10	20	30	40	50	60	80

Algorithm and Python code to Insertion in Linear List

First the appropriate position for ITEM is to be determined 1.e., if the appropriate position is I+ 1 then AR[I] <= ITEM <= AR[I+1], LST specifies maximum possible index in array, u specifies index of last element in Array

```
1.      ctr=L                                   # Initialize the counter
2.      If LST = U then
                Print ("Overflow:")
                Exit from program
3.      if AR[ctr] > ITEM then
                pos 1
        else
        {
4.      while ctr < U perform steps 5 and 6
5.              if AR[ctr] <= ITEM and ITEM <= AR[ctr + 1] then
                        pos = ctr+ 1
                        break
6.              ctr= ctr +1
7.      if ctr = U then
                pos =U + 1
}               #end of if step 3
# shift the elements to create space
8.      ctr= U                                  # Initialize the counter
9.      while ctr>= pos perform steps 10 through 11
10.     {       AR[ctr+1] AR [ctr]
11.             ctr=ctr-1
        }
12.     AR[pos] = ITEM                          # Insert the element
13.     END
```

Deletion of an element from the sorted array:

To delete the element from the sorted list, the element to be deleted has to be first searched for in the array using search algorithm. If the search is successful, the element is removed from the list and the rest of the elements of the list are shifted such that the order of the array is maintained. Since the elements are shifted upwards, the free space is available at the end of the array.

Algorithm and Python code to delete an element at the position pos, from a sorted list

Algorithm	**Program Code**

1. Loc=position # position is the position of the element to be deleted
2. Repeat steps 3 and 4 until Loc<=U
3. DATA_LIST[Loc]=DATA_LIST[Loc+1]
4. Loc=Loc+1
#end of repeat
	def del_element(DLIST , pos):
	Loc=pos
	while(Loc<=N): #N= size of list-1
	DLIST[Loc]=DLIST[Loc+1]
	Loc=Loc+1

Searching algorithms - There are many searching algorithms.

- **Linear Search:** The search element is compared with each element of the list, starting from the beginning of the list to the end of the list.
- **Binary Search:** The array should be sorted in either ascending or descending order the search element is compared with MID element of the list recresverly.

Linear search

- In linear search, the search element is compared with each element of the list, starting from the beginning of the list. This continues till either the element has been found or you have reached to the end of the list.
- The algorithm given below searches for an element, ELEM in the list named D_LIST containing size number of elements.

Algorithm	Program Code

1. Set Loc=0 , N= size -1 # size is the size of the list L
2. Repeat steps 3 through 4 until Loc>N
3. if DATA_LIST[Loc]==ELEM then
{ print " Element found at "
print Loc+1
break
}
4. ctr= Loc+1# repeat ends
5. if Loc >N then print "Element not found"
6. END
	def linear_search(D_LIST, ELEM):
	flag=0
	for Loc in DATA_LIST:
	if DATA_LIST[Loc]==ELEM :
	print " Element found at "
	print Loc+1
	flag=1
	break
	if flag==0:
	print "Search not successful----Element not found"

Binary search

This searching technique reduces the number of comparisons and hence saves on processing time. Binary search, the condition that has to be fulfilled is that the array should be sorted in either ascending or descending order.

To search for an element, ELEM in a list that is sorted in ascending order, the ELEM is first compared with the middle element.

- If ELEM is found in the middle of list then Search successful.
- If ELEM is greater than the middle element, latter part of the list is searched.
- If ELEM is smaller than the middle element, former part of the list becomes the new segment.

Therefore, the number of comparisons to be made, in this case are reduced proportionately, thereby decreasing the time spent in searching the element.

Algorithm	Program Code

1. Set low=0 , high=size-1
2. Repeat steps 3 through 6 until low<high
3. mid=(int) (low+high)/2
4. if D-LIST[mid]==ELEM then
{
print "Element found at"
	def binary_search(D_LIST, ELEM, low, high):
	low=0
	high=len(D_LIST)
	while(low<high):
	mid=(int)(low+high/2)
	if D_LIST[mid]==ELEM:

print mid
break;
}
5. if D_LIST[mid]<ELEM then
low=mid + 1
6.if DATA_LIST[mid]>ELEM then
high=mid-1 #End of Repeat
7. if low >= high
Print "ELEMENT NOT FOUND"
8. END
	print "Element found at"
	print mid
	break
	if D_LIST[mid]<ELEM:
	low=mid+1
	if D-LIST[mid]>ELEM:
	high=mid-1
	if low >= high
	print "ELEMENT NOT FOUND"

Sorting a list

Sorting is to arrange the list in an ascending or descending order. Three sorting techniques used to sort the list in ascending or descending order.

- **Selection Sort:** To repeatedly select the smallest element in the unsorted part of the array and then swap it with the first element of the unsorted part of the list.
- **Bubble sort:** algorithm, every adjacent pair is compared and swapped if they are not in the right order
- **Insertion Sort:** Selects the next element to the right of what was already sorted, slides up each larger element until it gets to the correct location

Selection Sort

The basic logic of selection sort is to repeatedly select the smallest element in the unsorted part of the array and then swap it with the first element of the unsorted part of the list.

Algorithm	Program Code

1. small = D_LIST[0] #initialize small with the first element
2. for i=0 to U do
{
3. small= D_LIST [i]
position=i
4. for j=i to U do
{
5. if D_LIST [j]<small then
6. { small = D_LIST [j]
7. position=j
}
j=j+1
}
8. temp= D_LIST [i]
9. DATA_LIST[i]=small
10. D_LIST [position]=temp
}
11. END
	Selection Sort Program
	
	def selection_sort(D_LIST):
	for i in range(0, len (D_LIST)):
	min = i
	for j in range(i + 1, len(D_LIST)):
	if DATA_LIST[j] < D_LIST [min]:
	min = j
	temp= D_LIST [min];
	D_LIST [min] = D_LIST [i]
	D_LIST [i]=temp # swapping

Bubble Sort

In case of bubble sort algorithm, comparison starts from the beginning with every adjacent pair is compared and swapped if they are not in the order. After each iteration of the loop, one less element is needed to be compared until there are no more elements left to be compared to be sorted in order.

Algorithm	Program Code

1. for i=L to U
2. { for j=L to ((U-1)-i) #the unsorted array reduces with every iteration
3. { if(DATA_LIST[j]>DATA_LIST[j+1] then
4. { temp=DATA_LIST[j]
5. DATA_LIST[j]=DATA_LIST[j+1]
6. DATA_LIST[j+1]=temp
}
}
}
7. END
	Bubble Sort Program
	#Bubble Sort
	def bubble_Sort(DATA_LIST):
	i = 0
	j = 0
	for i in range(len(DATA_LIST)):
	for j in range(len(DATA_LIST) - i):
	if DATA_LIST[j] < DATA_LIST[j-1]:
	temp = DATA_LIST[j-1]
	DATA_LIST[j-1] = DATA_LIST[j]
	DATA_LIST[j] = temp
	print DATA_LIST
	print DATA_LIST

Insertion Sort

As far as the functioning of the outer loop is considered, insertion sort is similar to selection sort. The difference is that the insertion sort does not select the smallest element and put it into place; rather it selects the next element to the right of what was already sorted. Then it slides up each larger element until it gets to the correct location to insert into the sorted part of the array.

Algorithm	Program Code

1. ctr=0
2. repeat steps 3 through 8 for K=1 to size-1
{
3. temp=DATA_LIST[k]
4. ptr=K-1
5. repeat steps 6 to 7 while temp<DATA_LIST[ptr]
{
6. DATA_LIST[ptr+1]=DATA_LIST[ptr]
7. ptr=ptr-1
}
8. DATA_LIST[ptr+1]=temp
}
9. END
	def insertion_sort(DATA_LIST):
	for K in range (1, len(DATA_LIST)):
	temp=DATA_LIST[K]
	ptr=K-1
	while(ptr>=0)AND
	DATA_LIST[ptr]>temp:
	DATA_LIST[ptr+1]=DATA_LIST[ptr]
	ptr=ptr-1
	DATA_LIST[ptr+1]=temp

Two-dimensional list:

A two dimensional list is a list having all its elements as lists of same shapes, ie, a two dimensional list is a list of lists, e.g.,

L1- [[10, 20], [30, 40], [50, 60]]

L1 is a two dimensional list as it contains three lists, each having same shape, ie., all are singular lists (i.e., non-nested) with a length of 2 elements. Following representation will make it clearer.

L1- [[10, 20],

[30, 40],

[50, 60]]

Regular two-dimensional lists are the nested lists with these properties:

- All elements of a 2D list have same shape (i.e., same dimension and length)
- The length of a 2D list tells about Number of Rows in it (ie., len(list))
- The length of single row gives the Number of Columns in it (i.e., len(list[n]))

Ragged list. A list that has lists with different shapes as its elements is also a 2d list but it is an irregular 2d list, also known as a ragged list.

For instance, following list () is a ragged list:

L2= [[1, 2, 3], [5, 6]]

Here the elements is list L2 has two sub lists where the first-sub list has 3 elements whereas second sub-list has 2 elements which is called Ragged list.

A Creating a 2D List

To create a 2D list by inputting element by element, you can employ a nested loop as shown below: one loop for rows and the other loop for reading individual elements of each row.

```
List1 = []
row = int (input ("How many rows?"))
col =  int (input ( "How many columns?") )
for i in range(row) :
        Lst = [ ]
        for j in range (col):
                ele = int (input("Element"+ str(i) +", "+str(j) + ": "))
                Lst.append (ele)
        List1.append (Lst)
print("List created is : ", List1)
```

Traversing a 2D List: To traverse a 2D list element by element, you can employ a nested loop as earlier one loop for rows and another for traversing individual elements of each row.
Example:

```
#We have added following nested loop to previous code of 2d list creation
print ("List created is : ")
print("Lst = [ ")
for i in range(row):
        print ("\t[", end = " ")
```

```
            for j in range (col):
                    print( Lst[i] [j], end "
            print("]")
            print("\t]")
```

The output produced is like :

```
    List created is
    Lst= [
            [ 10, 20 30 ]
            [ 11, 22 33 ]
            [ 50 60 70 ]
        ]
```

Solved Questions

Short Answer Type Questions.

1. What do you mean by the term?

(i) Raw data (ii) Data item

Ans. (i) Raw data are raw facts.

(ii) Data item represent single unit of values of certain type.

2. Define a data structure.

Ans. A data structure is a group of data, which can be processed as a single unit. This group of data may be of similar or dissimilar data types. Data Structures are very useful while programming because they allow processing of the entire group of data as a single unit.

3. Name the two types of data structures.

Ans. 1. Linear data structure examples: Arrays, Linked List, Stack and Queue

2. Non-Linear Data Structure Examples: Tree and graph.

4. What are linear data structures?

Ans. These are data structures whose elements form a sequence e. g. Stack, queue and linked lists.

5. What do you understand by simple data structures?

Ans. Data structure built from primitive data type are called simple data structure e.g. arrays and Lists.

6. What are Compound data structures?

Ans. These are formed by combining sample data structures. These are of two types Linear and non linear

7. What are nonlinear data structures?

Ans. These are multilevel data structures. e. g. tree.

8. What are the overhead sizes for sequence type strings, tuples and lists?

Ans. String-21+ (1*length)-32 bit implementation
37+ (1*length)-64 bit implementation
Tuples-28+ (4*length)-32 bit implementation
64+ (8*length)-64 bit implementation
Lists 36(4*length)-32 bit implementation
72+ (8*length)-64 bit implementation

9. When would you go for linear search in an array and when for binary search?

Ans. When the array is unsorted liner search is used Binary search is performed in a sorted array.

10. What is getsizeof ( )?

Ans. **getsizeof ( )** function is used to check memory size of any object. It is present in sys module

11. What is bisect ( )?

Ans. In a list sorted in ascending order, bisect0 function of bisect module returns the appropriate index where the new element should be inserted.

Syntax: **bisect.bisect(list, <element>)**

12. What is insort ( )?

ans. The insort ( ) function of bisect module inserts an item in the sorted sequence.

Syntax: **bisect.insort (list, <newelement >)**

13. What is a list comprehension? How is it useful?

Ans: A List Comprehension is a concise description of a list that shorthand the list creating for loop in the form of a single statement.

List comprehensions make the code compact and are faster to execute.

Q. 14. Give difference between an array and a list in Python.

Ans. An array is defined as a set of contiguous data of similar data type. Python lists are actually arrays of variable length. The elements of a list are of heterogeneous types which means they are of different data types.

Q.15. Compare a data type with a Data structure.

Ans. A Data type defines a set of values along with well-defined operations stating its input-output behavior e.g., you cannot put decimal point in an integer or two strings cannot not be multiplied etc.

On the other hand, a Data structure is a physical implementation that clearly defines way of storing, accessing, manipulating data stored in a data structure. The data stored in a data structure has a specific data type e.g., in a stack, all insertions and deletions take place at one end only.

Q. 16. Why Python does stores overheads to store every type of data?

Ans. Python keeps a reference count for every value it stores. Because it stores every value only once in memory and makes every variable having this value refer to the same memory location. Python remembers how many variables are referring to this value through reference count. Thus, for every value, it allocates additional memory for overheads like reference count, indexing etc,.

Q.17. How are lists implemented in memory?

(or)

How is memory allocated to a list in Python?

Ans: A list in Python is an array that contains elements (pointers to objects) of a specific size only and this is a common feature of all dynamically typed languages. For implementation of a list, a contiguous array of references to other objects is used. Python keeps a pointer to this array and the array's length is stored in a list head structure. This makes indexing of a list independent of the size

of the list or the value of the index when items are appended or inserted, the array of references is resized.

Q.18. What is sequential allocation of memory? Why do we say that lists are stored sequentially?

Ans: A list is a allocated memory in sequential manner. This means that the elements of the list are stored in memory in sequence of their declaration. Therefore, if you want to view the fifth element of the list, you have to first traverse through first four elements of the list. This is called sequential allocation of memory.

Q.19. What is the difference between a regular 2D list and a ragged List?

Ans: A regular two dimensional list is a list having lists as its elements and each element-list has the same shape i.e., same number of elements (length).

On the other hand, a list that contains lists with different shapes as its elements is also a 2D list but it is an irregular 2D list, also known as a ragged list.

For instance, in the example code below List1 is a regular 2D list while List2 is a ragged list:

Listl = [[1, 2, 3], [7, 9, 8]]

List2 [[4, 5, 6], [1, 7]]

Q.20. What common operations are performed on different Data Structures 7

Ans. Some commonly performed operations on data structures are

- Insertion. To add a new data item in the given collection of data items.
- Deletion. To delete an existing data item from the given collection of data items
- Traversal. To access each data item exactly once so that it can be processed.
- Searching. To find out the location of the data item if it exists in the given collection of data items.
- Sorting. To arrange the data items in some order either in ascending or descending order in case of numerical data and in dictionary order in case of alphanumeric data.

Q.21. What purpose Linear lists data structures are mostly used for?

Ans: Linear lists data structures are used to store and process elements that are similar in type and are to be processed in the same way. For example, to maintain a shopping list, a linear list may be used where items to be shopped are inserted to it one by one and as soon as an item is shopped, it is removed from the list.

Q.22. How is linear search different from binary search.

Ans. (i) Binary search requires the input data to be sorted but linear search does not.

(ii) Binary search requires an ordering comparison; linear search only requires equality comparison.

(iii) Binary search has complexity O(log n); linear search has complexity O(n).

(iv) Binary search requires random access to the data; linear search only requires sequential access (this can be very important-it means a linear search can stream data of arbitrary size).

Q.23. What is bubble sort?

Ans: Bubble sort is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted. Comparing each pair of adjacent items and swapping them if they are in the wrong order the pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm gets its name from the way smaller elements bubble to the top of the list because it only uses comparisons to operate on elements; it is also called a comparison sort.

Q.24. Write an algorithm for selection sort?

Ans: Selection sort performs the following steps:
1. Starting at index 0, search the entire array to find the next smallest or largest value
2. Swap the smallest or largest value found with the value at index 0
3. Repeat steps 1 & 2 starting from the next index**.**

Q. 25: Explain insertion sort.

Ans: Every repetition of insertion sort removes an element from the input data, inserting it into the correct position in the already-sorted list, until no input elements remain. The choice of which element to remove from the input is arbitrary and can be made using almost any choice algorithm.

Q.26. Accept a list containing integers randomly. Accept any number and display the position at which the number is found in the list.

Ans:
```
maxrange input("Enter Count of numbers: ")
Marks=[ ]
Flag=False
for i in range(0, maxrange):
        marks.append(input (“?”))
number=input(“Enter number to be searched”)
for i in range(0, maxrange):
        if marks[i]= =number:
                print (“number, “found at position”, i)
                flag= =True:
if fag= =False:
        print( number, "not found in list”)
```

Q.27. What will be the output produced by following code?

```
Text = [ 'h', 'e', 'l', 'l', 'o']
print(text)
vowels = "aeiou"
newText=[x.upper ( ) for x in text if x not in vowels ]
print(newText)
```

Ans: ['h', e', '1', '1', 'o']

['H','L','L']

Q.28. Predict the output:

I. LA = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
LB = [num/3 for num in LA if num % 3= = 0]
print (LB)

Ans: [3.0, 12.0, 27.0]

II. [x + y for x in 'ball' for y in 'boy']

Ans: ['bb', 'bo', 'by', 'ab', 'ao', 'ay', 1b', lo', ly', 1b', 'lo', 'ly']

III. L1 [1, 2, 3, 4, 5, 6, 7, 8, 9]
k [elem1*elem2 for elem1 in L1 if (elem1 - 4) >1 for elem2 in L1[:4]]
print (k)

Ans: [6, 12, **18, 24, 7, 14, 21, 28, 8, 16, 24, 32, 9, 18, 27, 36]**

Q.29. What will be the status of the following list after the First, Second and Third pass of the Slection sort method used for arranging the following elements in descending order?

LIST=[12, 14, -54, 64, 90, 24]

Ans: 12, 14, -54, 64, 90, 24
Pass 1: 90, 14, -54, 64, 12, 24
Pass 2: 90, 64, -54, 14, 12, 24
Pass 3: 90, 64, 24, 14, 12, -54

Q.30. For a given list of values in descending order, write a method in Python to search for a value with the help of Binary search method. The method should return position of the value and should return 1 if the value not present in the list.

Ans.
```
def binarysrch (nums, x):
    high=len (nums)
    while low<high:
    mid-(low+high)/2
    midval=nums[mid]
    if midval >x:
```

```
            low = mid +1
    elif midval < x:
            high = mid
    else:
            return mid
    return-1
```

Q.31. What will be the status of the following after the First, Second and Third pass of the insertion sort method used for arranging the following elements in descending order?

Ele= [22, 24, -64, 34, 80, 43]

Ans: 22, 24, -64, 34, 80, 43
Pass 1: 24, 22, -64, 34, 80, 43
Pass 2: 24, 22, -64, 34, 80, 43
Pass 3: 34, 22, 24, -64, 80, 43

Long Answer Type Questions

Q.1.Consider the following unsorted list: [90, 78, 20, 46, 54, 1] Write the list after:

1. **3rd iteration of selection sort**
2. **4th iteration of bubble sort**
3. **5th iteration of insertion sort Ans. Working:**

Ans: Working:

Pass	Bubble Sort	Selection Sort	Insertion Sort
P1	**[1, 90, 78, 46, 54, 20]**	**[1, 78, 20, 46, 54, 90]**	**[78, 90, 20, 46, 54, 1]**
P2	**[1, 20, 90, 78, 54, 46]**	**[1, 20, 78, 46, 54, 90]**	**[20, 78, 90, 46, 54, 1]**
P3	**[1, 20, 46, 90, 78, 54]**	**[1, 20, 46, 78, 54, 90)**	**[20, 46, 78, 90, 54, 1]**
P4	**[1, 20, 46, 54, 90, 78]**	**[1, 20, 46, 54, 78, 90]**	**[20, 46, 54, 78, 90, 1]**
P5	**[1, 20, 46, 54, 78, 90]**	**[1, 20, 46, 54, 78, 90]**	**[1, 20, 46, 54, 78, 90]**
P6	**[1, 20, 46, 54, 78, 90]**	**[1, 20, 46, 54, 78, 90]**	

1. 3rd iteration of selection sort: [1, 20, 46, 78, 54, 90]
2. 4th iteration of bubble sort: [1, 20, 46, 54, 90, 78]
3. 5th iteration of insertion sort [1, 20, 46, 54, 78, 90]

Q. 2. Accept a list containing integers randomly Accept any number and display the position at which the number is found in the list.

Ans.
```
maxrange= input("Enter Count of numbers:")
Marks=[]
Flag=False
for i in range(0, maxrange):
```

```
        marks.append(input(?")
number-input("Enter number to be searched")
for i in range(0, maxrange):
        if marks[i]= =number:
        print (number,"found at position",i)
        flag=True
if flag=-False:
print number,"not found in list"
```

Q.3. Write a python program to search an element using binary search algorithm.

```
def binary_search(D_LIST, ELEM, low, high):
        low=0
        high=len(D_LIST)
        while(low<high):
        mid=(int)(low+high/2)
        if D_LIST[mid]==ELEM:
                print "Element found at"
                print mid
                break
        if D_LIST[mid]<ELEM:
                low=mid+1
                if D-LIST[mid]>ELEM:
                high=mid-1
        if low >= high
        print "ELEMENT NOT FOUND"
```

Q.4. Write a python program to sort elements using bubble sort algorithm.

```
#Bubble Sort
def bubble_Sort(DATA_LIST):
        i = 0
        j = 0
        for i in range(len(DATA_LIST)):
                for j in range(len(DATA_LIST) - i):
                        if DATA_LIST[j] < DATA_LIST[j-1]:
                                temp = DATA_LIST[j-1]
                                DATA_LIST[j-1] = DATA_LIST[j]
                                DATA_LIST[j] = temp
        print DATA_LIST
```

Q.5. Write an algorithm to sort elements using Selection sort?

Ans. Selection Sort Algorithm:

1. small = D_LIST[0] #initialize small with the first element

```
2. for i=0 to U do
{
3. small= D_LIST [i]
position=i
4. for j=i to U do
{
5. if D_LIST [j]<small then
6. { small = D_LIST [j]
7. position=j
}
j=j+1
}
8. temp= D_LIST [i]
9. DATA_LIST[i]=small
10. D_LIST [position]=temp
}
11. END
```

EXERCISE

1. Define a data structure.
2. Name the two types of data structures and give one point of difference between them.
3. Give one point of difference between an array and a list in Python.
4. How are lists implemented in memory?
5. What is sequential allocation of memory? Why do we say that lists are stored sequentially?
6. How is memory allocated to a list in Python?
7. The function should do the following:
    a. Insert the number passed as argument in a sorted list.
    b. Delete the number from the list.
8. What do you mean by the following terms?
    a. (1 raw data
    b. (ii) data item
    c. (iüi) data type
    d. (iv) data structure
9. What do you understand by the following:
    a. simple data structures
    b. compound data structures
    c. linear data structures
    d. non-linear data structures ?
10. When would you go for linear search in an array and why?
11. State condition(s) when binary search is applicable.
12. Name the efficient algorithm offered by Python to insert element in a sorted sequence.
13. State condition(s) when binary search is applicable.
14. What is a list comprehension?
15. What is a 2D list?
16. What is a nested list?
17. Is Ragged list a nested list?
18. Write a function that takes a list that is sorted in ascending order and a number as arguments.
19. How is linear search different from binary search?
20. Accept a list containing integers randomly. Accept any number and display the position at which the number is found is the list.
21. Write a function that takes a sorted list and a number as an argument. Search for the number in the sorted list using binary search.