UNIT IV COMPOUND DATA: LISTS, TUPLES, DICTIONARIES

Lists, list operations, list slices, list methods, list loop, mutability, aliasing, cloning lists, list parameters; Tuples, tuple assignment, tuple as return value; Dictionaries: operations and methods; advanced list processing - list comprehension, Illustrative programs: selection sort, insertion sort, merge sort, quick sort.

4.1 LIST

- A list is a versatile data type contains items separated by commas and enclosed within square brackets ([ ]). An item can be of any type.
- Each and every item has its unique index.
- A list contains different data type such as string, integer, float, real and another list.

Example:

L1= [1, 2, 3] #list with integer data type

L2 = [ ] #empty list

L3= [[4,8],L1]

The above list can be expanded as

L3= [[4,8] , [1,2,3]]

4.1.1 List indexing

- Positions are numbered from left to right starting at 0 and it is called as positive indexing.Positions can also be numbered from right to left starting at -1 and it is called as negative indexing.

0	**1**	**2**	**3**	**4**	**5**
P	**Y**	**T**	**H**	**O**	**N**
-6	**-5**	**-4**	**-3**	**-2**	**-1**

4.1.2 List Operations

- List can perform operations like a string such as concatenation, indexing, slicing, the len function etc.

Concatenation Operator (+):

The concatenation operator (+) is used to add a list of elements to the existing list when both the lists are of the same type. Example.py

list1=[1,2,3,4]+[10,20,30] list2=list1+["Numbers"]

print(list1)

print(list2) Output:

[1, 2, 3, 4, 10, 20, 30]
[1, 2, 3, 4, 10, 20, 30, 'Numbers']

Repetition Operator (*):

"*" operator occurs between a sequence s and an integer n, a new sequence containing 'n' repetitions of the elements of s is obtained.

Example.py list1=[1,2,3,4]*2 list2="&"*3

print(list1) print(list2) Output:

[1, 2, 3, 4, 1, 2, 3, 4]

&&&

len(): len(s) returns the number of element in a sequence s.
Example.py

list1=[1,2,3,4]

print(len(list1)) Output:

4
max () & min(): max(s) returns the largest value in a sequence s and min(s) returns the smallest value in a sequence s Example.py

list1=[1,2,3,4]

print(max(list1))

print(min(list1)) Output:

4

1

Membership Operator:

The "in" operator is used to test the membership. The operator returns true if the element is present in a list otherwise it returns false. Example.py

```
list1=[1,2,3,4] print(2
in list1) print(6 in
list1) Output:
True
False
```

Inverse Operator:

The inverse operator “not in” returns false if the element is present in alist otherwise it returns true. Example.py

```
list1=[1,2,3,4] print(3
not in list1) print(7
not in list1) Output:
False
True
```

list():

This function is used to convert a tuple or string into list. The list() method takes sequence types and converts them into lists. Example.py tuple1=(1,2,3,4)

```
print(list(tuple1)) Output:
[1,2,3,4]
```

4.7.3 List Slices / List Index

We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.Trying to access an element other that this will raise an IndexError. The index must be an integer. We can't use float or other types, this will result into TypeError.Nested list are accessed using nested indexing.

Example.py

```
my_list = ['p','r','o','b','e']
# Output: p
print(my_list[0]) #
Output: o
print(my_list[2]) #
Output: e
print(my_list[4])
```

```
# Error! Only integer can be used for indexing
# my_list[4.0] # Nested List
n_list = ["Happy", [2,0,1,5]]
# Nested indexing #
Output: a
print(n_list[0][1])
# Output: 5
print(n_list[1][3])
```

Negative indexing

Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

Example: Script.py

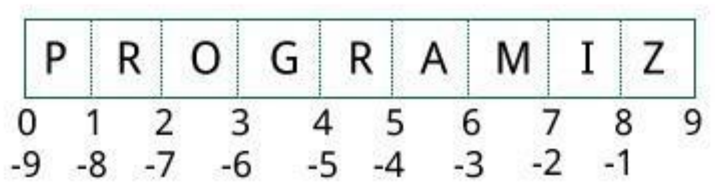
```
my_list = ['p','r','o','b','e']
# Output: e
print(my_list[-1])
# Output: p
print(my_list[-5])
```

How to slice lists in Python?

We can access a range of items in a list by using the slicing operator (colon). my_list

```
= ['p','r','o','g','r','a','m','i','z']
# elements 3rd to 5th
print(my_list[2:5]) #
elements beginning to 4th
print(my_list[:-5]) #
elements 6th to end
print(my_list[5:])
# elements beginning to end print(my_list[:])
```

Slicing can be best visualized by considering the index to be between the elements as shown below. So if we want to access a range, we need two index that will slice that portion from the list.



4.1.4 List Methods

Methods that are available with list object in Python programming are tabulated below.They are accessed as list.method(). Some of the methods have already been used above.

Python List Methods
append() - Add an element to the end of the list
extend() - Add all elements of a list to the another list
insert() - Insert an item at the defined index
remove() - Removes an item from the list
pop() - Removes and returns an element at the given index
clear() - Removes all items from the list
index() - Returns the index of the first matched item
count() - Returns the count of number of items passed as an argument

sort() - Sort items in a list in ascending order

reverse() - Reverse the order of items in the list

copy() - Returns a shallow copy of the list

Example.py
```
my_list = [3, 8, 1, 6, 0, 8, 4]
# Output: 1
print(my_list.index(8))
# Output: 2 print(my_list.count(8))
my_list.sort()
# Output: [0, 1, 3, 4, 6, 8, 8]
print(my_list)
my_list.reverse()
# Output: [8, 8, 6, 4, 3, 1, 0] print(my_list)
```

append():

This method appends / adds the pass object (v) to the existing list.

Syntax: **listname.append(element)**

Example.py
```
list1=[1,2,3,4]
list1.append(99)
print(list1)
```
Output:
```
[1, 2, 3, 4, 99]
```

Insert():

This method insert the given element at the specified position.

Syntax: **listname. insert(position,element)**

Example.py
```
list1=[1,2,3,4]
list1.insert(1,100)
print(list1)
```
Output:
```
[1, 100, 2, 3, 4]
```

Remove():

Removes the element from the list. If there is no element then it displays error.

Syntax: **listname.remove(element)**

Example.py

list1=[1,2,3,4]

list1.remove(4)

print(list1) Output:

[1, 2, 3]

Extend():

This method appends the contents of the list.

Syntax: **listname1.extend(listname2)**

Example.py

list1=[1,2,3,4]

list1.extend([10,20,30]) print(list1)

Output:

[1, 2, 3, 4, 10, 20, 30]

Sort():

This method sorts the element either alphabetically or numerically.

Syntax: **listname.sort()**

Example.py

list1=[5,2,10,4]

list1.sort()

print(list1) Output:

[2, 4, 5, 10] **Reverse():**

This method reverses the element in the list.**Syntax:** **listname.reverse()**

Example.py

```
list1=[5,2,10,4]
list1.reverse()
print(list1)
```

Output:

```
[4, 10, 2, 5]
```

Count():

This method returns count of how many times elements occur in the list.

Syntax: **listname.count(element)**

Example.py

```
list1=[5,2,10,4,3,5]
print(list1.count(5))
```

Output:

```
2
```

Index():

This method returns the index value of an element.

Syntax: **listname.index(element[,start[,end]]))**

Example.py

```
list1=[5,2,10,4,3]
print(list1.index(10))
```

Output:

```
2
```

Pop():

This method removes the element from the list at the specified index. If the index value is not specified it removes the last element from the list.

Syntax: **listname.pop(index)**

Example.py

```
list1=[5,2,10,4,3]
print(list1.pop(4))
print(list1.pop(7))
```

Output:

```
3
Traceback (most recent call last):
  File "python", line 3, in <module>
IndexError: pop index out of range
```

Del():

Element to be deleted is mentioned using list name and index.

Syntax:

```
dellistname[index]
```

Example.py

```
list1=[5,2,10,4,3]
del list1[3]
print(list1)
```

Output:

```
[5, 2, 10, 3]
```

4.1.5 List Loops

- Python are executed sequentially but branching statements are used to break the sequential pattern.
- Python supplies two different kinds of loops,the while loop and for loop.

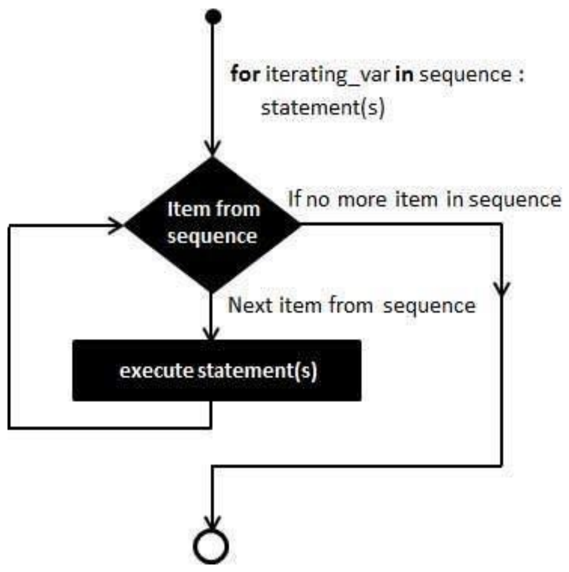
For loop:

- Executes a sequence of statements multiple times.

Syntax:

```
for val in sequence:
Body of for
```

- Val is a variable that takes value of the item inside the sequence on each iteration.
- Loop continues until the last item in the sequence is reached.
- For loop contains initialization & condition part, but increment and decrement need not to be defined in python. **Flowchart**


for iterating_var in sequence :
statement(s)
Item from sequence
If no more item in sequence
Next item from sequence
execute statement(s)

Example.py

(To print square of number from 1 to 5)

```
for i in range(1,6):

print("The square of {0} is {1}".format(i,i**2))
```

Output:

```
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
```

Example.py

```
for letter in "Magic":

print("current letter:",letter)
```

Output:

```
current letter: M
current letter: a current
letter: g current letter:
i
current letter: c
```

while loop:

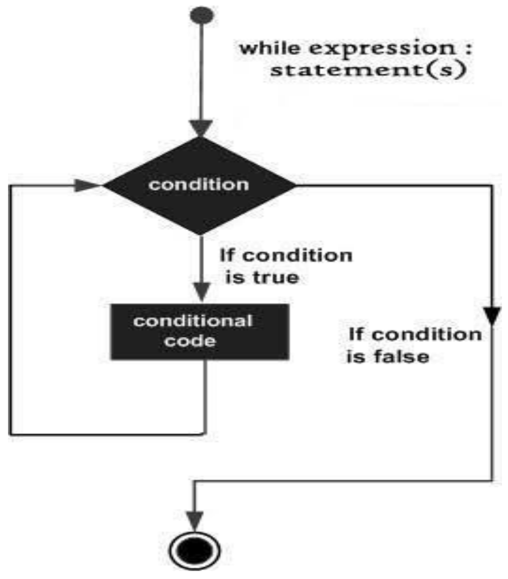
- It is an entry controlled loop
- It executes a block of code repeatedly till the condition becomes true.

Syntax:

```
while expression:

statement(s)
```

Flowchart:



example.py(To print numbers)

```
i=5 while i<50:
print(i)
i=i*3
```

Output:

```
5
15
45
```

Infinite loop

- A loop becomes infinite loop if a condition becomes FALSE. Such a loop is called as infinite loop
- Example: client-server programming(server needs to run continuously so that client can communicate with it) Example.py

```
i=1 while
i==1:
print("Chennai")
```

Output:

```
Chennai
Chennai
Chennai
```

Example.py

4.1.6 Mutability

- Mutability is the ability to change certain types of data without entirely recreating it.
- Using mutable, data types allow programs to operate quickly and efficiently.
- A list is mutable but string is not mutable (or) immutable.
- The “=” operator is used to copy a list to another variable, the list is not actually replicated. Instead both variables point to the same list.

l=[5,6,7,8,9]

l[2]=10 print(l)

Output:

[5, 6, 10, 8, 9]

4.1.7 Aliasing

- When two identifiers refers to the same variable or value,then it is called as aliasing.
- This type of change is known as side effect. The variable one and two both refers to the exact same list object. They are aliases for the same object.

Example.py

one=[10,20,30] two=one

print(one)

print(two)

one[2]=55

print(one)

print(two) Output:

```
[10, 20, 30]
[10, 20, 30]
[10, 20, 55]
[10, 20, 55]
```

- To prevent aliasing a new object is created and the contents of the original can be copied to the new object. Example.py one=[10,20,30] three=[] for i in one:

    three.append(i) Output:

[10, 20, 30]

4.1.8 Cloning List

- It is a process of making a copy of the list without modifying the original list.
- When changes are done, it is applied only in duplicate copy and not reflected to original copy. Example.py

```
def dup(l1,l2):
for i in l1: if i
in l2:
l1.remove(i)
return l1=[1,2,3,4]
l2=[1,2,5,6]
dup(l1,l2)
print("l1:",l1)
```

Output:

```
l1: [2, 3, 4]
```

List Parameters

- A list can also be passed as a parameter to a function.
- If changes are done, then it is notified to main program.
- The list arguments are always passed by reference only.

Example.py

```
def delete(t):
del t[0]
letter=['a','b','c','d']
delete(letter)
print(letter)
```

Output:

```
['b', 'c', 'd']
```

```
t1=[1,2] t1.append(3)
print(t1)
```

Example.py
t3=t1+[4]

print(t3) <u>Output:</u>

[1, 2, 3]
[1, 2, 3, 4]

<u>Example.py</u> def

tail(t): return t[1:]

letter=['a','b','c','d']

print(tail(letter))

<u>Output:</u>

['b', 'c', 'd']

4.2 TUPLES

- A tuple consists of number of values separated by commas and enclosed within parentheses.
- Tuples are similar to lists.
- Tuples are immutable sequences. The elements in the tuple cannot be modified.
- The difference between tuple and lists are the tuple cannot be modified like lists and tuples use parentheses, whereas list use square brackets. <u>Example.py</u> fruits=(‘apple’, ‘orange’) print(fruits)

<u>Output:</u>

(‘apple’, ‘orange’)

4.2.1 Creating and Accessing tuple

Tuples can be created by putting different values separated by commas and enclosed within parenthesis. <u>Example.py</u>

#Tuple Example

no=(1,2,3) print(no)

#Nested Tuple

n1=(“python”,[‘list’,’in’,’tuple])

print(n1[0]) print(n2[1]) #mixed

tuple m1=(3,”aaa”)

nos,value=m1 print(nos)

print(value) Output:

(1,2,3) python

[‘list’,’in’,’tuple’]

3 aaa

4.2.2 Operations in tuples

Two operators + and * is allowed in tuples. + concatenates the tuples and * repeats the tuple elements a given number of times. Example.py no=(1,2,3) rep=(‘a’,’b’)*2 new=no+rep print(new) print(rep) Output:

```
(1,2,3,’a’,’b’,’a’,’b’)
(‘a’,’b’,’a’,’b’)
```

4.2.3 Slicing and Indexing of tuples

Slicing of tuples is similar to lists. [m:n] where m is the inclusive starting point and n is the exclusive ending point. Example.py

```
t1=(1,2,3,4,(10,20,30))
print(t1[0])
print(t1[2:3])
print(t1[2:]) print(t1[-
2]) Output:
```

```
1
3
(3,4,(10,20,30))
4
```

4.2.4 Deleting and updating tuples

Tuples are immutable. The elements cannot be changed. The entire tuple can be deleted using del. Example.py

```
n1=(‘c’,’o’,’m’,’p’,’u’,’t’,’e’,’r’)
n1[1]=’O’ del
n1[0]
```

Output:

```
Error:’tuple’ object does not support item assignment and item deletion.
```

4.2.5 Tuple functions

Functions in tuple are also similar to list data type. The methods of tuple are also similar to list.

S.no	Functions	Description
1.	len(tuple)	Gives the total length of the tuple
2.	max(tuple)	Returns item from the tuple with max value.
3.	min(tuple)	Returns item from the tuple with min value.
4.	tuple(seq)	Converts list or string into tuple.
5.	sum(tuple)	Returns the sum of all elements.
6.	sorted(tuple)	Sorts the tuple in ascending order.

Example.py

```
t3=(1,2,35,4,15,6,7)

l1=[10,20,30] s1="computer"

print(len(t3)) print(max(t3))

print(min(t3))

print(tuple(l1))

print(tuple(s1))

print(sum(t3))

print(sorted(t3))
```

Output:

```
7
35
1
(10, 20, 30)
('c', 'o', 'm', 'p', 'u', 't', 'e', 'r')
70
[1, 2, 4, 6, 7, 15, 35]
```

4.2.6 Tuple Assignment

- Swapping of values in two variables need three statements and one temporary variable as below. temp=a a=b b=temp
- Tuple assignment provides easiest way of swapping.The number of variables in left and right of assignment operator must be equal. Example.py a=100 b=345 c=450 a,b=b,a print("a=",a,"&","b=",b) a,b,c=c,a,b

```
print("a=",a,"&","b=",b,"c=",c) Output:
a= 345 & b= 100 a= 450
& b= 345 c= 100
```

4.2.7 Tuple with return value(Functions can return tuples)

```
Example.py def
swap(a,b,c):
returnc,b,a
a=100 b=345
c=765
print("a,b,c:",a,",",b,",",c)
a,b,c=swap(a,b,c)
print("a,b,c:",a,",",b,",",c) Output:
a,b,c: 100 , 345 , 765 a,b,c:
765 , 345 , 100
```

4.3 ADVANCED LIST PROCESSING: LIST COMPREHENSION

- List comprehension is used to construct lists in an easy and natural way.
- It create a list with a subset of elements from another list by applying condition.
- The list comprehension makes code simpler and efficient.
- The execution is much faster than for loop.

Example.py

```
x=[i for i in range(10)] print(x)

x1=[i for i in range(10) if i%2==0]

print(x1) x2=[i*2 for i in

range(10)] print(x2)

vowels=('a','e','i','o','u')

w="hello" x3=[ch for ch in w if ch

in vowels] print(x3)
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[0, 2, 4, 6, 8]

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

['e', 'o']
```

4.4 DICTIONARIES

- Dictionary is one of the compound data type like string, list and tuple.
- In dictionary the index can be immutable data type.
- It is a set of zero or more ordered pairs(key, value) such that:
  - The value can be any type.
  - Each key may occur only once in the dictionary ○ No key may be mutable. A key may not be a list or tuple or dictionary and so on.
- A dictionary is set of pairs of value with each pair containing a key and an item and is enclosed in curly brackets.

Example:

dict={} #empty dictionary d1={1:'fruit',2:'vegetables',3:'cereals'}

4.4.1 Creation and Accessing dictionary

- There are two ways to create dictionaries.
  - It is created by specifying the key and value separated by a colon(:) and the elements separated by commas and entire set of elements must be enclosed by curly braces.
  - dict() constructor that uses a sequence to create dictionary.

<u>Example.py</u>

d1={1:'fruit',2:'vegetables',3:'cereals'}

d2=dict({'name':'aa','age':40})

print(d1[1]) print(d2) <u>Output:</u> fruit

{'name': 'aa', 'age': 40}

4.4.2 Deletion of elements

- The elements in the dictionary can be deleted by using del statements.
- The entire dictionary can be deleted by specifying the dictionary variable name.

Example.py del d1[1]#remove

entry with key '1' print(d1)

d1.clear()#remove all entries in dict

print(d1) Output:

```
{2: 'vegetables', 3: 'cereals'}

{}
```

4.4.3 Updating elements

- In dictionary the keys are always immutable.
- The values are mutable that can be modified. New-key value pair can be inserted or deleted from the dictionary. Example.py

```
d3={'name':'abc','dept':'ece'}

d3['name']='xxxx' print(d3)
```

Output:

```
{'name': 'xxxx', 'dept': 'ece'}
```

4.4.4 Dictionary methods

The methods are accessed as **dictionary variable. Method name()**

S.No	**Method Name**	**Syntax**	**Description**
1.	Clear	dict.clear()	Removes all elements in dictionary
2.	Copy	dict.copy()	Returns a shallow copy(alias) of the dictionary
3.	Fromkeys	dict.fromkeys(seq,value)	Create a new dictionary with keys from seq and value.
4.	Get	dict.get(key[,value])	For key K,returns the value or default if key not in dictionary
5.	Items	dict.items()	Returns a list of(key,value) tuple pairs
6.	keys	dict.keys()	Returns a list of dictionary keys
7.	Pop	Dict.pop(key[,default])	Remove element with key K and return its value. Returns default id K is not found.
8.	Popitem	dict.popitem()	Removes and return element(key,value) from the last
9.	Setdefault	dict.setdefaule(key[,default value])	Set value for key K to default if key is not already
10.	Update	dict.update([other])	Adds dictionary D’s key – value pairs

11.	values	dict.values()	Returns list of values.

Example.py
```
h1={'name':'abc','sub':['maths','pspp','chemisty'],1:[100,200,300]}
#copy method temp=h1.copy() print("copy of dict",temp)
#fromkeys method print("fromkeys method in
dict",h1.fromkeys(('dob','age'),10))
#get method print("get method in
dict",h1.get(1,'NA')) print("get method in
dict",h1.get('age','NA'))
#items method print("items in
dict",h1.items())
#keys method print("keys in
dict",h1.keys()) #pop method print("pop
method in dict",h1.pop('name'))
#popitem method print("popitem method in
dict",h1.popitem())
#set default method print("set default method in
dict",h1.setdefault('age',20))
#update method d={1:"one"}
d2={'name':'xxx'} d.update(d2)
print("update method in dict",d)
#values method print("values in
dict",h1.values())
#clear method h1.clear()
print("clear method in dict",h1)
```

Output:
```
copy of dict {'name': 'abc', 'sub': ['maths', 'pspp', 'chemisty'], 1: [100, 200, 300]}
fromkeys method in dict {'dob': 10, 'age': 10} get method in dict [100, 200,
300] get method in dict NA
items in dictdict_items([('name', 'abc'), ('sub', ['maths', 'pspp', 'chemisty']), (1, [100, 200, 300])])
keys in dictdict_keys(['name', 'sub', 1]) pop method in dictabc popitem method in dict (1, [100,
```

200, 300]) set default method in dict 20 update method in dict {1: 'one', 'name': 'xxx'} values in dictdict_values([['maths', 'pspp', 'chemisty'], 20]) clear method in dict {}

4.4.5 Dictionary functions

S.No	Function Name	Syntax	Description
1.	Length	len(dict)	Gives the number of elements in dictionary.
2.	string	str(dict)	Produce a printable string representation of a dictionary
3.	Type	type(variable)	Returns a type of variable passed
4.	Del	del dict(k)	Deletes the element with key K from dict dictionary

Example.py

```
h1={'name':'abc','sub':['maths','pspp','chemisty']}
print("length of dict is:",len(h1)) print("string
representation of  dict is:",str(h1)) print("type of
variable of dictis:",type(h1)) del h1['name']
print("deletion of element of dict is:",h1)
```

Output:

```
length of dict is: 2 string representation of  dict is: {'name': 'abc', 'sub': ['maths',
'pspp', 'chemisty']} type of variable of dict is: <class 'dict'> deletion of element
of dict is: {'sub': ['maths', 'pspp', 'chemisty']}
```

Difference between List,Tuples and Dictionary

S.No	List	Tuple	Dictionary
1.	A list is mutable	A tuple is immutable	A dictionary is mutable
2.	Dynamic	Fixed	Values can be of any type and repeat but keys must be immutable
3.	Enclosed in square brackets [ ] and their elements and size can be changed	Enclosed in parenthesis ( ) and cannot be updated	Enclosed in curly braces { } and consists of key,value
4.	Homogenous	Heterogenous	Homogenous
5.	Eg: l1=[10,20]	l1=(10,20)	d1=(1:"a",2:"b")

6.	Access: l1[0]	l1[0]	d1[1]
7.	Can contain duplicate elements	Can contain duplicate elements. Faster when compared to lists	Can contain duplicate elements but cannot contain duplicate keys
8.	Slicing can be done	Slicing can be done	Slicing cannot be done
9.	List is used if a collection of data that does not need random access. Data can be modified frequently	Tuple can be used when data cannot be changed. Tuple is used in combination with a dictionary (ie)tuple must represent a key	It is used in logical association between key value pair. Data is being modified constantly.

4.5 ILLUSTRATIVE PROGRAMS

<u>Sorting:</u>

- Sorting is the process of arranging elements in the list according to their values in ascending (or) descending order.
- A sorted list is called an ordered list. The types of sorting are ○ Selection sort ○ Insertion sort ○ Merge sort ○ Quick sort ○ Bubble sort

4.5.1 Selection sort

- Selection sort makes only one exchange for every pass through the list. In this selection sort looks for either largest (or) smallest value as it makes a pass and after completing the pass, places it in proper location.
- After the first pass, the smallest item is in the correct place. After the second pass, the next smallest is in place
- The iteration is repeated for (n-1) items.
- The time complexity of selection sort is (O(n2)).

Working of selection sort: Intial list

16	19	11	15	10	12	14

First iteration

16	19	11	15	10	12	14
10	19	11	15	16	12	14

Second iteration

10	19	11	15	16	12	14

10	11	19	15	16	12	14

Third iteration

10	11	19	15	16	12	14
10	11	12	15	16	19	14

Fourth iteration

10	11	12	15	16	19	14
10	11	12	14	16	19	15

Fifth iteration

10	11	12	14	16	19	15
10	11	12	14	15	19	16

Sixth iteration

10	11	12	14	15	19	16
10	11	12	14	15	16	19

Final list

10	11	12	14	15	16	19

Example.py

```
print("---------------------------")
print("Selection Sort") print("-------------
-------------") a = [44, 22, 11, 55, 33]
print("\nThe unsorted elements are : ", a)
i = 0   while (i < len(a)):
   #smallest element in the sublist
smallest = min(a [ i : ])
   #index of smallest element
index_of_smallest = a.index(smallest)
```

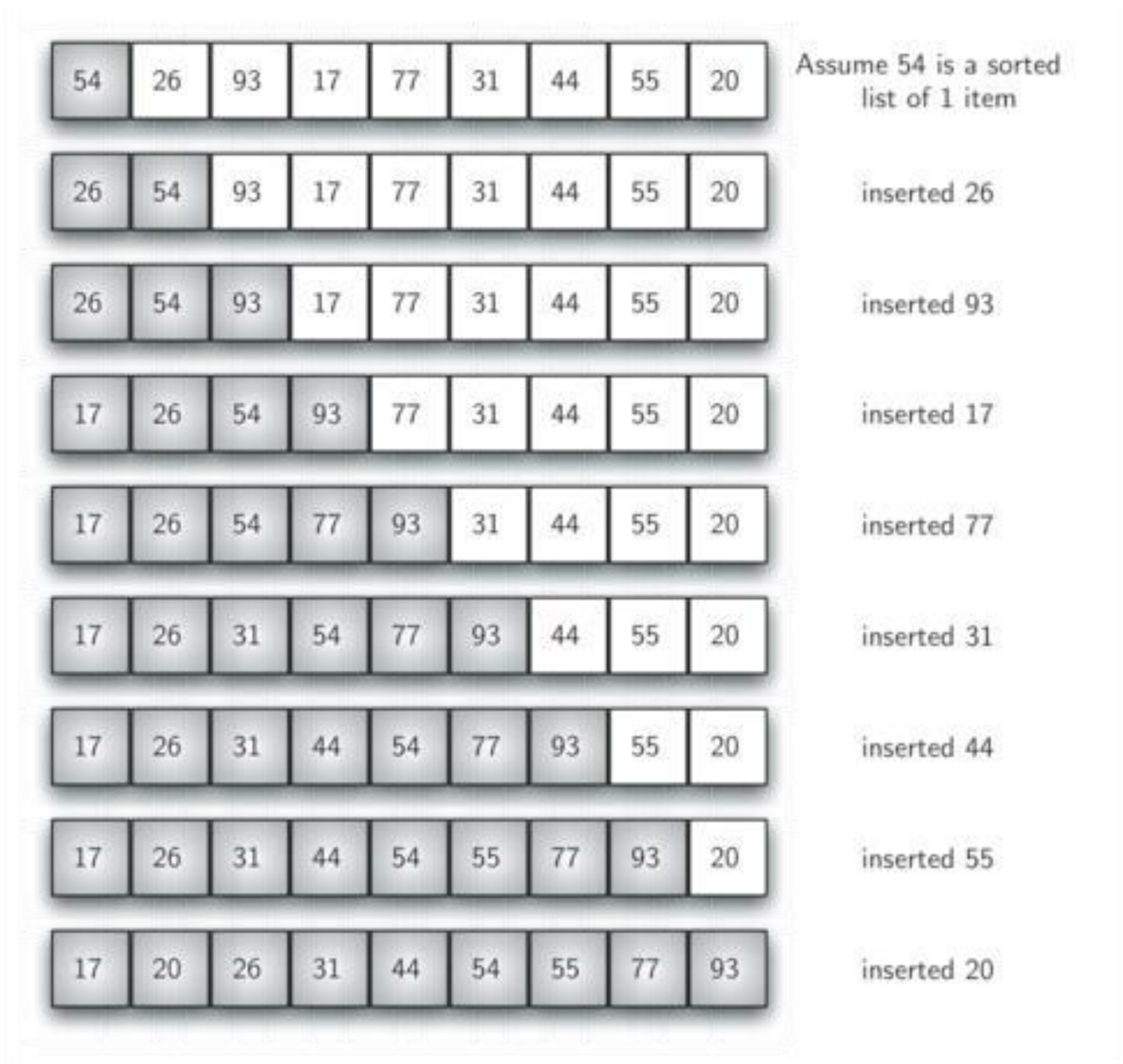
```
        #swapping
        (a[i], a[index_of_smallest] )= (a[index_of_smallest], a[i] )
    i = i+1
    print ("\nThe sorted list is :", a)
```
<u>Output:</u>

```
---------------------------
Selection Sort
--------------------------
The unsorted elements are :
[44, 22, 11, 55, 33] The
sorted list is :
 [11, 22, 33, 44, 55]
```

4.5.2 Insertion sort

- Insertion sort is a sorting algorithm in which the elements are transferred one at a time to the right position.
- It is suited for sorting small set of data, through which new elements can be inserted into a sorted sequence.

Working of Insertion sort:



<u>Example.py</u>

print("---------------------------") print("Insertion Sort")
print("--------------------------") def insertionSort(A): for i
in range(1,len(A)): current_value = A[i] position
=i while(position > 0 and A[position - 1] >
current_value):

```
            A[position] = A[position - 1]
    position = position- 1
    A[position] = current_value        A =
    [44, 22, 11, 55, 33] print("\nThe unsorted
    elements are : ", A) insertionSort(A)
    print ("\nThe sorted list is :", A)
```

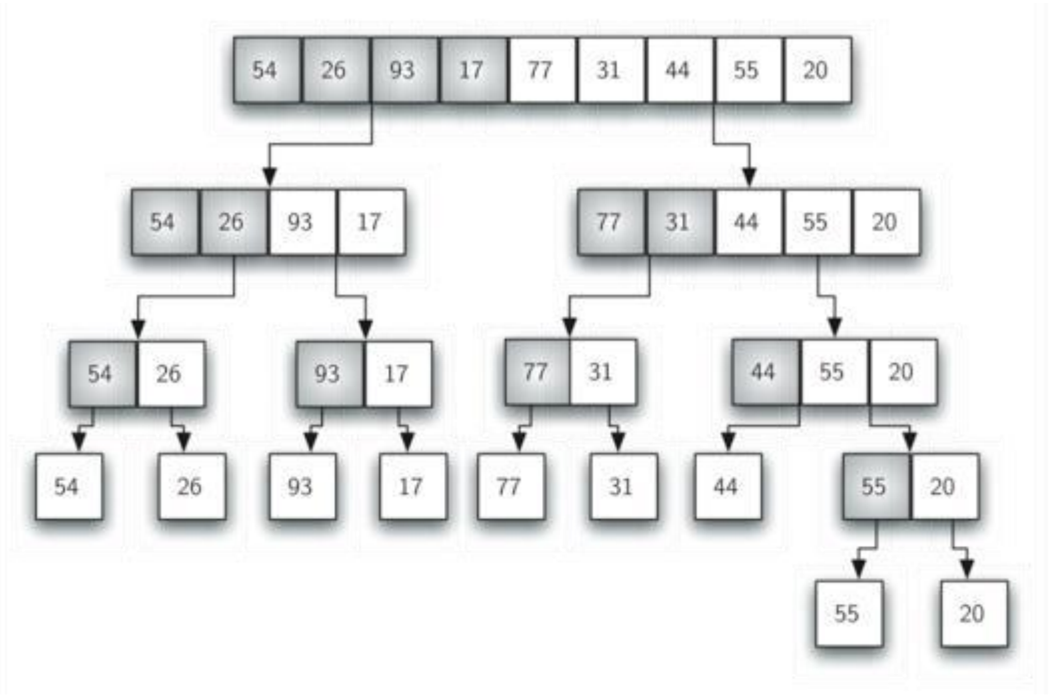
Output:

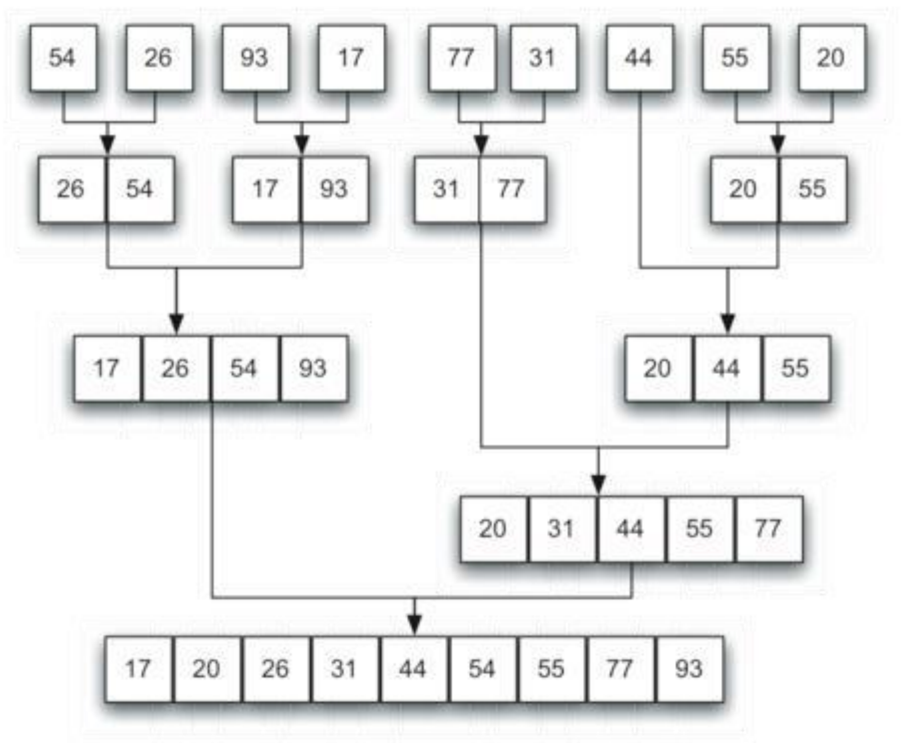
```
---------------------------
Insertion Sort
--------------------------
The unsorted elements are :  [44, 22, 11, 55, 33]
The sorted list is : [11, 22, 33, 44, 55]
```

4.5.3 Merge sort

- Merge sort use divide and conquer strategy to improve the performance of sorting algorithms. It is a recursive algorithm that continually splits a list into half.
- Merging is the process of taking two smaller sorted lists and combining them together into a single sorted new list.

Working of Merge sort:





Example.py

```
print("---------------------------") print("Merge Sort")
print("--------------------------") def mergeSort(nlist):
print("Splitting : ",nlist)     if (len(nlist) > 1):
mid = len(nlist)//2         left_half = nlist [ : mid ]
right_half = nlist [ mid : ]         mergeSort(left_half)
mergeSort(right_half)                  i = j = k = 0
while (i < len(left_half) and j < len(right_half)):
if( left_half [ i ]  < right_half [ j ] ):
                nlist [ k ] = left_half [ i ]
i = i + 1            else:
                nlist[ k ] = right_half [ j ]
j = j + 1              k = k + 1
```

```
        while (i < len(left_half)):
nlist [ k ] = left_half [ i ]
                        i
= i + 1
k = k + 1

        while(j < len(right_half)):
nlist [ k ] = right_half [ j ]             j
= j + 1             k = k + 1

        print("Merging : ",nlist) nlist = [44,
22, 11, 55, 33] print("\nThe unsorted
elements are :" ,nlist) mergeSort(nlist)
print ("\nThe sorted list is :", nlist)
```

Output:

```
---------------------------
Merge Sort
--------------------------
The unsorted elements are : [44, 22, 11, 55, 33]
Splitting :  [44, 22, 11, 55, 33]
Splitting :  [44, 22]
Splitting :  [44]
Splitting :  [22]
Merging :  [22, 44]
Splitting :  [11, 55, 33]
Splitting :  [11]
Splitting :  [55, 33]
Splitting :  [55]

Splitting :  [33]

Merging :  [33, 55]

Merging :  [11, 33, 55]

Merging :  [11, 22, 33, 44, 55]

The sorted list is : [11, 22, 33, 44, 55]
```

4.5.4 Histogram

- A graphical representation similar to a bar chart in structure, that organizes a group of data points into user specified ranges.
- The histogram condenses a data series into an easily interpreted visual by taking many data points and grouping them into logical ranges of values or bins.

Steps involved in constructing the histogram:

1. To construct a histogram, the first step is to identify the range of values.
2. Divide the entire range of values into a series of intervals
3. Count how many values fall into each interval.
4. The range of values is usually specified as consecutive non-overlapping intervals of a variable.
5. The range of values must be adjacent and are often(but not required to be) of equal size.

Example.py

```
def
histogram(items):
for n in items:
output=''
times=n
while(times>0):
output+='*'
times=times-1
print(output)
histogram([5,3,8,4,6,10])
```

Output:
```
*****
***
********
****
******
**********
```