Name: **Class:** IV Semester

Register Number:

Course Name & Code:

Rubrics for Assessment of Activity (Qualitative Assessment)

SI No	**Dimension**	**Beginner**	**Intermediate**	**Good**	**Advanced**	**Expert**	**Students Score**
		2	**4**	**6**	**8**	**10**	
1	Python Installation	Satisfactory	Average	Good	Very Good	Excellent	
2	Develop Algorithm	Satisfactory	Average	Good	Very Good	Excellent	
3	Program writing	Satisfactory	Average	Good	Very Good	Excellent	
4	Code-exe-test-debug	Satisfactory	Average	Good	Very Good	Excellent	
5						Average Marks	

Signature of the Student **Course Coordinator**

1. **Design a Data structure for handling Student Records- Designing a Solution, Implementation (Using Basic DS)**

   Students data stored in list or Array. 175XY001, Anitha, anitha@email.com, 7777788888 175XY002, Bharath, bharatha@email.com, 7777788889 175XY003, Chandhan, chandhan@email.com, 7777788887 175XY004, Haneef, haneef@email.com, 7777788880 175XY005, Joel, joel@email.com, 7777788808 175XY006, Vidhya, vidhya@email.com, 7777788888
   Accessing the students data from students.txt file and create a basic data structure using list or array.

```
# function to return the records of students in array or list.
defstudents_list(file_name):   students = list()   with
open(file_name) as f:     for line in f.readlines():       reg_no, name, email, ph
one = line.split(',')       students.append([reg_no.strip(), name.strip(), email.st
rip(), phone.rstrip('\   return students
students = students_list('sample_data/students.txt')
```

```
students [['175XY001', 'Anitha', 'anitha@email.com', '7777788888'],
['175XY002', 'Bharath', 'bharatha@email.com', '7777788889'], ['175XY003',
'Chandhan', 'chandhan@email.com', '7777788887'], ['175XY004', 'Haneef',
'haneef@email.com', '7777788880'], ['175XY005', 'Joel', 'joel@email.com',
'7777788808'], ['175XY006', 'Vidhya', 'vidhya@email.com', '7777788888']]
```

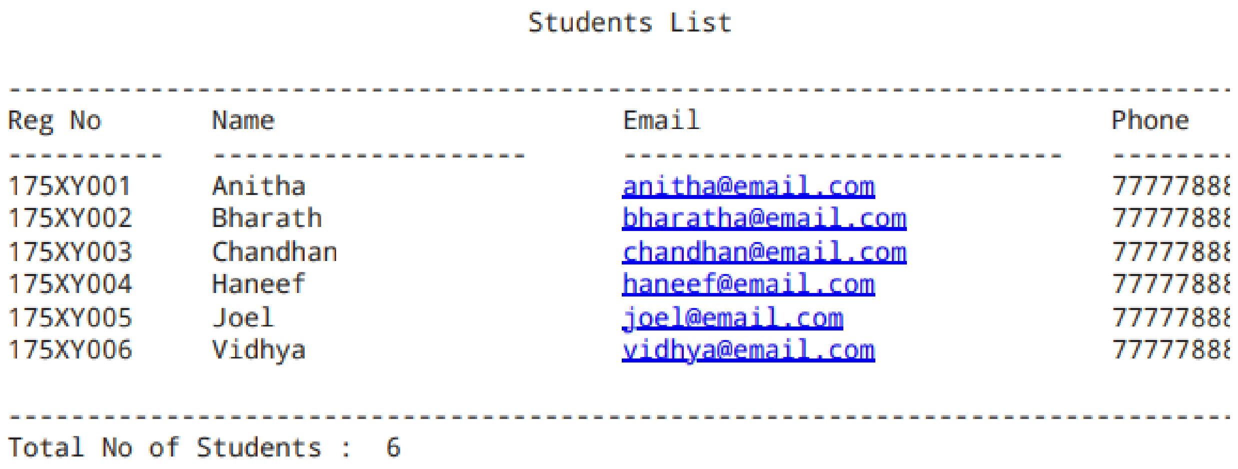
```
students.sort(key= lambda stu :stu[1] )
```

```
students [['175XY001', 'Anitha', 'anitha@email.com', '7777788888'],
['175XY002', 'Bharath', 'bharatha@email.com', '7777788889'], ['175XY003',
'Chandhan', 'chandhan@email.com', '7777788887'], ['175XY004', 'Haneef',
'haneef@email.com', '7777788880'], ['175XY005', 'Joel', 'joel@email.com',
'7777788808'], ['175XY006', 'Vidhya', 'vidhya@email.com', '7777788888']]
```

   Printing the students report

```
# Function to print the students report.
```

```
defreport_printing(students):   print("Students List".center(80))   print()   pri
nt('-'*80)   print(f"{'Reg No':12} {'Name':25} {'Email':30}
{'Phone':10} ")   print(f"{'-'*10:12} {'-'*20:25} {'-'*28:30} {'-'*10:10} ")

for student in students:     print(f"{student[0]:12} {student[1]:25}
{student[2]:30} {student[3]:10}

 print()
print('-'*80)
 print("Total No of Students : ", len(students))
report_printing(students)
```

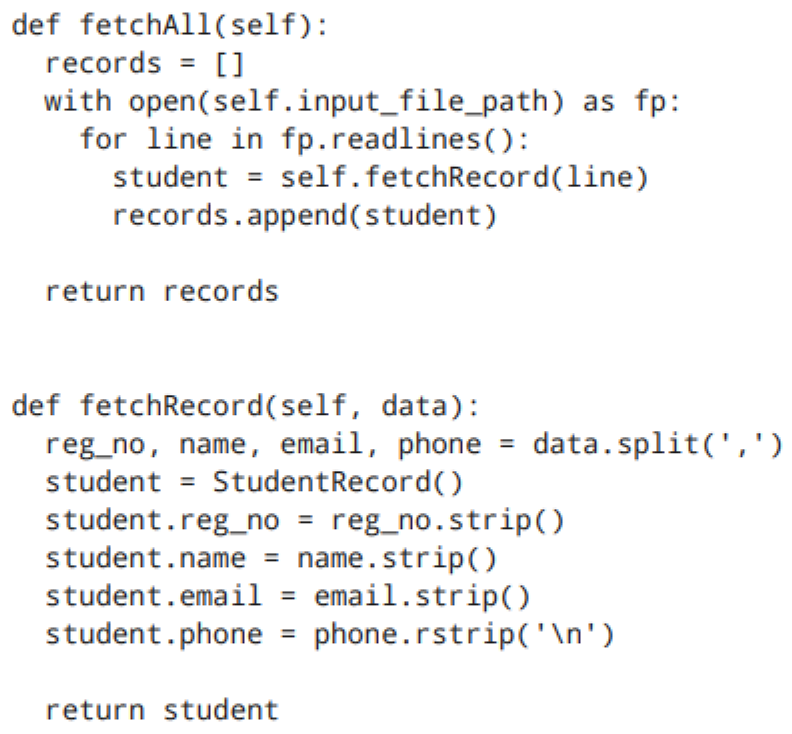


```
                              Students List

----------------------------------------------------------------------------
Reg No       Name                      Email                          Phone
----------   --------------------      ----------------------------   --------
175XY001     Anitha                    anitha@email.com               7777788
175XY002     Bharath                   bharatha@email.com             7777788
175XY003     Chandhan                  chandhan@email.com             7777788
175XY004     Haneef                    haneef@email.com               7777788
175XY005     Joel                      joel@email.com                 7777788
175XY006     Vidhya                    vidhya@email.com               7777788

----------------------------------------------------------------------------
Total No of Students :   6
```

2. Design a data structure for handling student records using ADT.

Implementation of the StudentFileReader ADT using a text file as the input source in which each line represents student data, seperated by comma.

```
class StudentRecord:
def __init__(self):
 self.reg_no = None
 self.name = None
   self.email = None
self.phone = None

class StudentFileReader:
def __init__(self, input_file_path):
  self.input_file_path = input_file_path
```



```
def fetchAll(self):
  records = []
  with open(self.input_file_path) as fp:
    for line in fp.readlines():
      student = self.fetchRecord(line)
      records.append(student)

  return records


def fetchRecord(self, data):
  reg_no, name, email, phone = data.split(',')
  student = StudentRecord()
  student.reg_no = reg_no.strip()
  student.name = name.strip()
  student.email = email.strip()
  student.phone = phone.rstrip('\n')

  return student
```

```
def printRecords(students):
  print("Students List".center(80))
  print()
  print('-'*80)
  print(f"{'Reg No':12} {'Name':25} {'Email':30} {'Phone':10} ")
  print(f"{'-'*10:12} {'-'*20:25} {'-'*28:30} {'-'*10:10} ")

  for student in students:
    print(f"{student.reg_no:12} {student.name:25} {student.email:30} {student.phone

  print()
  print('-'*80)
  print("Total No of Students : ", len(students))




reader = StudentFileReader('sample_data/students.txt')
students_list = reader.fetchAll()


printRecords(students_list)
```

```
                                 Students List

--------------------------------------------------------------------------------
Reg No       Name                      Email                          Phone
----------   --------------------      ----------------------------   --------
175XY001     Anitha                    anitha@email.com                7777788
175XY002     Bharath                   bharatha@email.com              7777788
175XY003     Chandhan                  chandhan@email.com              7777788
175XY004     Haneef                    haneef@email.com                7777788
175XY005     Joel                      joel@email.com                  7777788
175XY006     Vidhya                    vidhya@email.com                7777788

--------------------------------------------------------------------------------
Total No of Students :   6
```

3. Optimize your solution (Bubble sort, selection sort and Insertion sort)

Bubble sort Algorithm

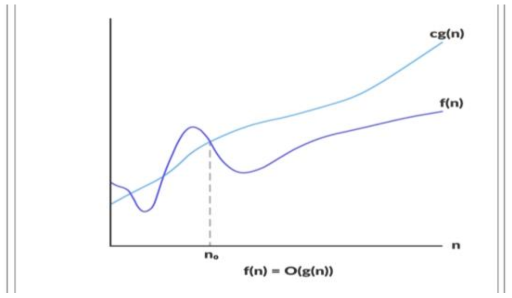
```
Step 1: [Import time and Define a function for Bubble sort]
        import time
        start=time.time()
        defbubblesort(a):
            n = len(a)
        fori in range(n-2):
        for j in range(n-2-i):
        if a[j]>a[j+1]:
        temp = a[j]
        a[j] = a[j+1]
        a[j+1] = temp
Step 2: [Create array and do the operation]
        alist = [34,46,43,27,57,41,45,21,70]
Step 3: [Output operation ]
        print("Before sorting:",alist)
        bubblesort(alist)
        end=time.time()
        print(f"Runtime of the program is {end - start}")
```

Python Code

```
import time
start=time.time()
defbubblesort(a):
 n = len(a)
fori in range(n-2):
for j in range(n-2-i):
if a[j]>a[j+1]:
temp = a[j]
a[j] = a[j+1]
a[j+1] = temp
alist = [34,46,43,27,57,41,45,21,70]
print("Before sorting:",alist)
bubblesort(alist)
end=time.time()
print(f"Runtime of the program is {end - start}")
```

Output:

```
Before sorting: [34, 46, 43, 27, 57, 41, 45, 21, 70]
After sorting: [21, 27, 34, 41, 43, 45, 46, 57, 70]
Runtime of the program is 0.06250619888305664
```



The bubble sort algorithm is a reliable sorting algorithm. This algorithm has a worst-case time complexity of O(n2). The bubble sort has a space complexity of O(1).

Selection Sort Algorithm

Step 1: [Import time & create function of selection sort]

```
import time
start=time.time()
defselectionsort(a):
 n = len(a)
fori in range(n-2):
min = i
for j in range(i+1,n-1):
if a[j]<a[min]:
temp = a[j]
a[j] = a[min]
a[min] = temp
```

Step 2: [Define array & execute the operation]

```
alist = [34,46,43,27,57,41,45,21,70]
print("Before sorting:",alist)
selectionsort(alist)
print("After sorting:",alist)
end=time.time()
print(f"Runtime of the program is {end - start}")
```

Python Code

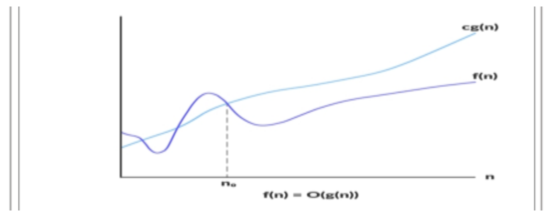
```
import time
start=time.time()
defselectionsort(a):
 n = len(a)
fori in range(n-2):
```

```
min = i
for j in range(i+1,n-1):
if a[j]<a[min]:
temp = a[j]
a[j] = a[min]
a[min] = temp
alist = [34,46,43,27,57,41,45,21,70]
print("Before sorting:",alist)
selectionsort(alist)
print("After sorting:",alist)
end=time.time()
print(f"Runtime of the program is {end - start}")
```

Output:

```
Before sorting: [34, 46, 43, 27, 57, 41, 45, 21, 70]
After sorting: [21, 27, 34, 41, 43, 45, 46, 57, 70]
Runtime of the program is 0.06238508224487305
```

Time Complexities:



The time complexity of the bubble sort is O(n2)
Worst Case Complexity: O(n2) : If we want to sort in ascending order and the array is in descending order then the worst case occurs.

Best Case Complexity: O(n2) : If the array is already sorted, then there is no need for sorting.
Average Case Complexity: O(n2) : It occurs when the elements of the array are in random order.

Space Complexity: Space complexity is O(1) because an extra variable (temp) is used for swapping.

Insertion sort Algorithm

Step 1: [Define insertion sort long with importing time]

```
import time
start=time.time()
definsertionsort(a):
 n = len(a)
fori in range(1,n-1):
    k = a[i]
    j = i-1
while j>=0 and a[j]>k:
a[j+1] = a[j]
       j=j-1
a[j+1] = k
```

Step 2: [Create an array]

```
alist = [34,46,43,27,57,41,45,21,70]
print("Before sorting:",alist)
```

Step 3: [Output operation]

```
insertionsort(alist)
print("After sorting:",alist)
end=time.time()
```

```
          print(f"Runtime of the program is {end - start}")
```

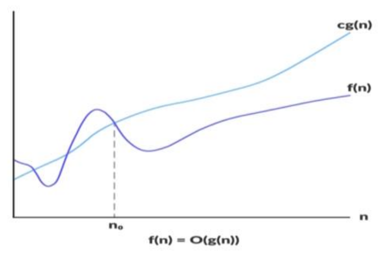
Python code

```
import time
start=time.time()
definsertionsort(a):
 n = len(a)
fori in range(1,n-1):
     k = a[i]
     j = i-1
while j>=0 and a[j]>k:
a[j+1] = a[j]
         j=j-1
a[j+1] = k
alist = [34,46,43,27,57,41,45,21,70]
print("Before sorting:",alist)
insertionsort(alist)
print("After sorting:",alist)
end=time.time()
print(f"Runtime of the program is {end - start}")
```

Output:

```
Before sorting: [34, 46, 43, 27, 57, 41, 45, 21, 70]
After sorting: [21, 27, 34, 41, 43, 45, 46, 57, 70]
Runtime of the program is 0.07800817489624023
```

Time Complexities:



The time complexity of the bubble sort is O(n2)

- Worst Case Complexity: O(n2) : If we want to sort in ascending order and the array is in descending order then the worst case occurs.
- Best Case Complexity: O(n) : If the array is already sorted, then there is no need for sorting.
- Average Case Complexity: O(n2) : It occurs when the elements of the array are in random order.
- Space Complexity: Space complexity is O(1) because an extra variable (temp) is used for swapping.

4. Implement Radix sort

Program to demonstrate Radix Sort algorithm

```
def counting_sort(arr, exp1):
  n = len(arr)
  # The output array elements that will have sorted arr
  output = [0] * (n)
  # initialize count array as 0
  count = [0] * (10)
  # Store count of occurrences in count[]
  for i in range(0, n):
    index = arr[i] // exp1
    count[index % 10] += 1
  for i in range(1, 10):
    count[i] += count[i - 1]
  # Build the output array
  i = n - 1
  while i >= 0:
    index = arr[i] // exp1
    output[count[index % 10] - 1] = arr[i]
    count[index % 10] -= 1
    i -= 1
  # Copying the output array to arr[],
  # so that arr now contains sorted numbers
  i = 0
  for i in range(0, n):
    arr[i] = output[i]
def radix_sort(arr):
  max1 = max(arr)
  # Do counting sort for every digit.
  exp = 1
  while max1 / exp > 1:
    counting_sort(arr, exp)
    exp *= 10
data = [3, 7, 0, 23, 9, 11, 2]
radix_sort(data)
print(data)
```

Output:

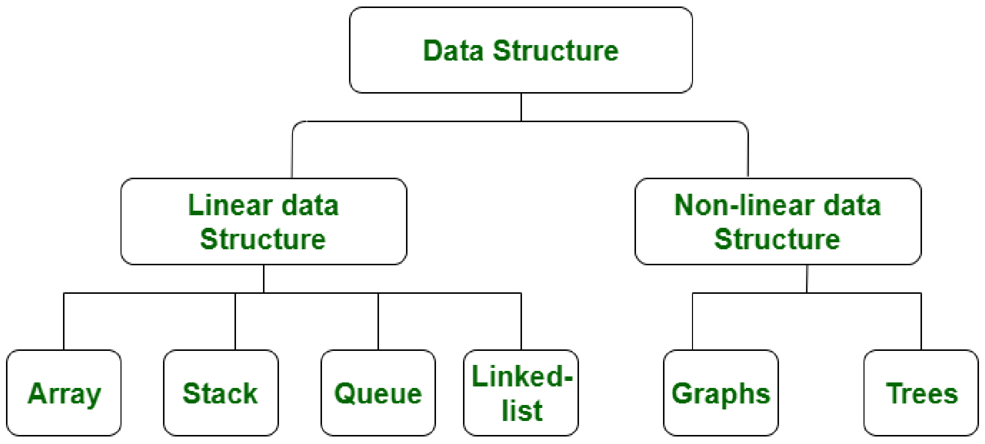
```
[0, 2, 3, 7, 9, 11, 23]
```

5. Prepare report on nonlinear data structures.

Non-linear Data Structure:
Data structures where data elements are not arranged sequentially or linearly are called non-linear data structures. In a non-linear data structure, single level is not involved. Therefore, we can't traverse all the elements in single run only. Non-linear data structures are not easy to implement in comparison to linear data structure. It utilizes computer memory efficiently in comparison to a linear data structure. Its examples are trees and graphs.

1. Trees
A tree data structure consists of various nodes linked together. The structure of a tree is hierarchical that forms a relationship like that of the parent and a child. The structure of the tree is formed in a way that there is one connection for every parent-child node relationship. Only one path should exist between the root to a node in the tree. Various types of trees are present based on their structures like AVL tree, binary tree, binary search tree, etc.

2. Graph
Graphs are those types of non-linear data structures which consist of a definite quantity of vertices and edges. The vertices or the nodes are involved in storing data and the edges show the vertices relationship. The difference between a graph to a tree is that in a graph there are no specific rules for the connection of nodes. Real-life problems like social networks, telephone networks, etc. can be represented through the graphs.



Difference between Linear and Non-linear Data Structures:

S.NO	**Linear Data Structure**	**Non-linear Data Structure**
1.	In a linear data structure, data elements are arranged in a linear order where each and every element is attached to its previous and next adjacent.	In a non-linear data structure, data elements are attached in hierarchically manner.
2.	In linear data structure, single level is involved.	Whereas in non-linear data structure, multiple levels are involved.
3.	Its implementation is easy in comparison to non-linear data structure.	While its implementation is complex in comparison to linear data structure.
4.	In linear data structure, data elements can be traversed in a single run only.	While in non-linear data structure, data elements can't be traversed in a single run only.
5.	In a linear data structure, memory is not utilized in an efficient way.	While in a non-linear data structure, memory is utilized in an efficient way.
6.	Its examples are: array, stack, queue, linked list, etc.	While its examples are: trees and graphs.
7.	Applications of linear data structures are mainly in application software development.	Applications of non-linear data structures are in Artificial Intelligence and image processing.

6. Design and implement sparse matrix representation using linked list

A matrix is a two-dimensional data object made of m rows and n columns, therefore having total m x n values. If most of the elements of the matrix have 0 value, then it is called a sparse matrix.

Why to use Sparse Matrix instead of simple matrix?

Storage: There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.

Computing time: Computing time can be saved by logically designing a data structure traversing only non-zero elements..

Example:

0 0 3 0 4

0 0 5 7 0

0 0 0 0 0

0 2 6 0 0

Representing a sparse matrix by a 2D array leads to wastage of lots of memory as zeroes in the matrix are of no use in most of the cases. So, instead of storing zeroes with non-zero elements, we only store non-zero elements. This means storing non-zero elements with triples- (Row, Column, value).

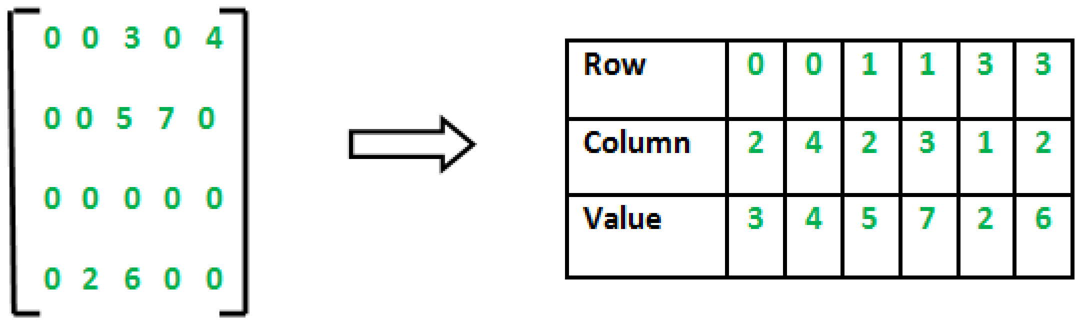
Sparse Matrix Representations can be done in many ways following are two common representations:

Array representation

1. Linked list representation
Method 1: Using Arrays:
2D array is used to represent a sparse matrix in which there are three rows named as

- Row: Index of row, where non-zero element is located
- Column: Index of column, where non-zero element is located

- Value: Value of the non zero element located at index – (row,column)



$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix} \Rightarrow$$

Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

```
# Python Program for Representation of
# Sparse Matrix into Linked List

# Node Class to represent Linked List Node
class Node:

        # Making the slots for storing row,
        # column, value, and address
        __slots__ = "row", "col", "data", "next"

        # Constructor to initialize the values
        def __init__(self, row=0, col=0, data=0, next=None):

                self.row = row
                self.col = col
                self.data = data
                self.next = next
```

```
# Class to convert Sparse Matrix
# into Linked List
class Sparse:

        # Initialize Class Variables
        def __init__(self):
                self.head = None
                self.temp = None
                self.size = 0

        # Function which returns the size
        # of the Linked List
        def __len__(self):
                return self.size

        # Check the Linked List is
        # Empty or not
        def isempty(self):
                return self.size == 0

        # Responsible function to create
        # Linked List from Sparse Matrix
    def create_new_node(self, row, col, data):

            # Creating New Node
            newNode = Node(row, col, data, None)

            # Check whether the List is
            # empty or not
            if self.isempty():
                    self.head = newNode
            else:
                    self.temp.next = newNode
            self.temp = newNode

            # Incrementing the size
            self.size += 1

    # Function display the contents of
    # Linked List
    def PrintList(self):
            temp = r = s = self.head
            print("row_position:", end=" ")
            while temp != None:
                print(temp.row, end=" ")
                temp = temp.next
        print()
        print("column_postion:", end=" ")
        while r != None:
                print(r.col, end=" ")
                r = r.next
        print()
        print("Value:", end=" ")
        while s != None:
                print(s.data, end=" ")
                s = s.next
        print()

# Driver Code
if __name__ == "__main__":

        # Creating Object
        s = Sparse()


  # Assuming 4x5 Sparse Matrix
  sparseMatric = [[0, 0, 3, 0, 4],
                                  [0, 0, 5, 7, 0],
                                  [0, 0, 0, 0, 0],
                                  [0, 2, 6, 0, 0]]
  for i in range(4):
          for j in range(5):

                  # Creating Linked List by only those
                  # elements which are non-zero
                  if sparseMatric[i][j] != 0:
                          s.create_new_node(i, j, sparseMatric[i][j])

  # Printing the Linked List Representation
  # of the sparse matrix
  s.PrintList()
```

Design and implement simple application that require DLL data structure.

Output:

Row_position: 0 0 1 1 3 3

Column_position: 2 4 2 3 1 2

Value: 3 4 5 7 2 6

7. Design and implement simple application that require DLL data structure.

It is **used in the navigation systems where front and back navigation is required**. It is used by the browser to implement backward and forward navigation of visited web pages that is a back and forward button.

Traversal/navigation in Doubly Linked List

Forward Traversal

A doubly linked list can be traversed in the forward direction by iterating from the first element to the last. We get the value of the next data element by simply iterating with the help of the reference address.

Forward Traversing Method

```
#Forward Traversal through Doubly Linked List
class Node:
def __init__(self, value = None):
self.value = value
self.next = None
self.prev = None
classDoublyLinkedList:
def __init__(self):
self.head = None
self.tail = None
defprintForwardList(self):
ifself.head == None:
print("The linked list does not exist.")
```

```
else:

temp_node = self.head

whiletemp_node:

print(temp_node.value)

temp_node = temp_node.next

#Initially, we have a linked list (1,3,5,7,9) called "dll".

dll.printForwardList()
```

Output:

```
1

3

5

7

9
```

Reverse Traversal

A doubly linked list can be traversed in the reverse direction by iterating from the last element to the first. We get the value of the previous data element by simply iterating with the help of the reference address.

Reverse Traversing Method

```
#Reverse Traversal through Doubly Linked List

class Node:

def __init__(self, value = None):
```

```
self.value = value
self.next = None
self.prev = None
classDoublyLinkedList:
def __init__(self):
self.head = None
self.tail = None
defprintReverseList(self):
ifself.head == None:
print("The linked list does not exist.")
else:
temp_node = self.tail
whiletemp_node:
print(temp_node.value)
temp_node = temp_node.prev
#Initially, we have a linked list (1,3,5,7,9) called "dll".
dll.printReverseList()
```

Output

```
9
7
5
3
1
```

8. Implement and demonstrate evaluating postfix expression.

```
# Python program to evaluate value of a postfix expression

# Class to convert the expression
classEvaluate:

    # Constructor to initialize the class variables
    def__init__(self, capacity):
        self.top =-1
        self.capacity =capacity
        # This array is used a stack
        self.array =[]

    # check if the stack is empty
    defisEmpty(self):
        returnTrueifself.top ==-1
     else
           False
    # Return the value of the top of the stack
    defpeek(self):
        returnself.array[-1]
    # Pop the element from the stack
    defpop(self):
        ifnotself.isEmpty():
            self.top -=1
            returnself.array.pop()
        else:
            return"$"
    # Push the element to the stack
    defpush(self, op):
        self.top +=1
        self.array.append(op)
    # The main function that converts given infix expression
    # to postfix expression
    defevaluatePostfix(self, exp):
        # Iterate over the expression for conversion
        fori inexp:
            # If the scanned character is an operand
            # (number here) push it to the stack
            ifi.isdigit():
                self.push(i)

            # If the scanned character is an operator,
            # pop two elements from stack and apply it.
            else:
                val1 =self.pop()
                val2 =self.pop()
                self.push(str(eval(val2 +i +val1)))

        returnint(self.pop())
# Driver program to test above function
exp ="231*+9-"
obj =Evaluate(len(exp))
print("postfix evaluation: %d"%(obj.evaluatePostfix(exp)))
```

output :

```
postfix evaluation: -4
```

9. Presentation on run time stack.

In Python How Recursion Works on Run Time

Consider the code snippet below which has two functions foo and main. A special variable __name__ which is fundamentally set by the python interpreter before execution and its value is set to __main__ when executing as a main program.

In python, if a function does not end with a return statement or has a return statement without any expression, the special value None is returned.

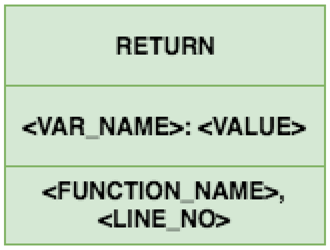
```
#!/usr/bin/python

deffoo(msg):

print'{} foo'.format(msg)

defmain():

msg = 'hello'

foo(msg)

if __name__ == '__main__':

main()

# end
```

To determine the output of the above code, we need to know in which order Python executes lines of code. We can run it and get the output, but that’s not the point here.

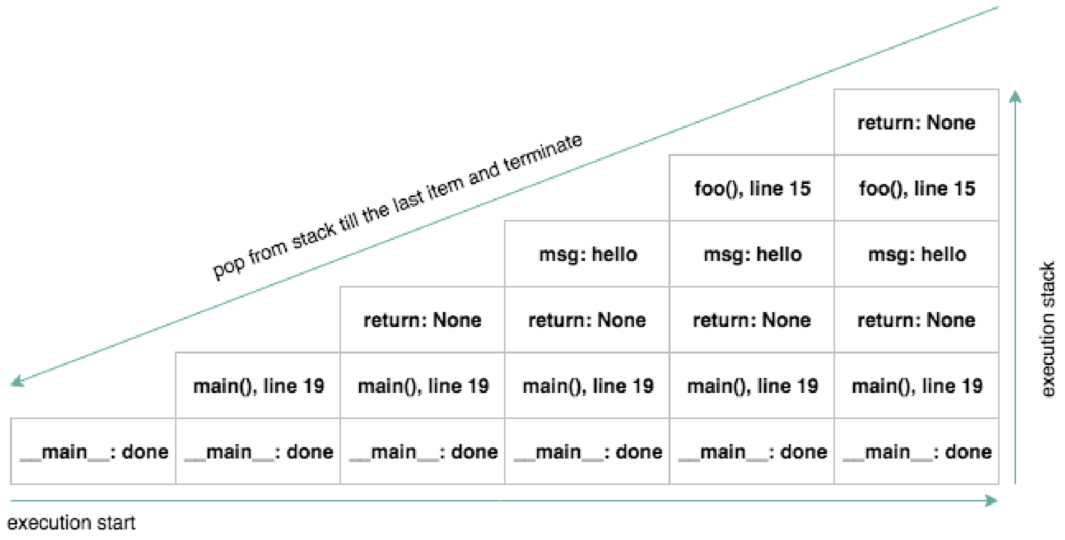
Python interpreter executes the code from level 0 indentation. In the case above main() method will be executed as its at level 0 indentation. That's the starting point of program execution and execution progress is kept in run-time stack.

What is run time stack?

This is the simple run time stack in which call frames are added on to stack and they are popped one by one until the it reaches the last frame in the stack and then execution is terminated.



This is the run time stack for our simple example program.



Each block in the stack is called call frame aka activation records. They are stored in the memory to keep track of function call in progress and allocated memory is released when function returns.

Call frame is added on top of stack when function is called and removed when function returns. It contains the function name that was called and where to pick up from(line number) when function returns.

Parameters and local variables defined inside the function are also push onto stack and popped when function returns.When calling a function with parameters, the parameters becomes local variable on the stack that are initialized with actual parameter value.

So even if two functions have same local variable name they are different as they appear differently on to the stack. Same function call from different places possibly have different value for the same variables. Understanding this concept is important to understand how recursion works.

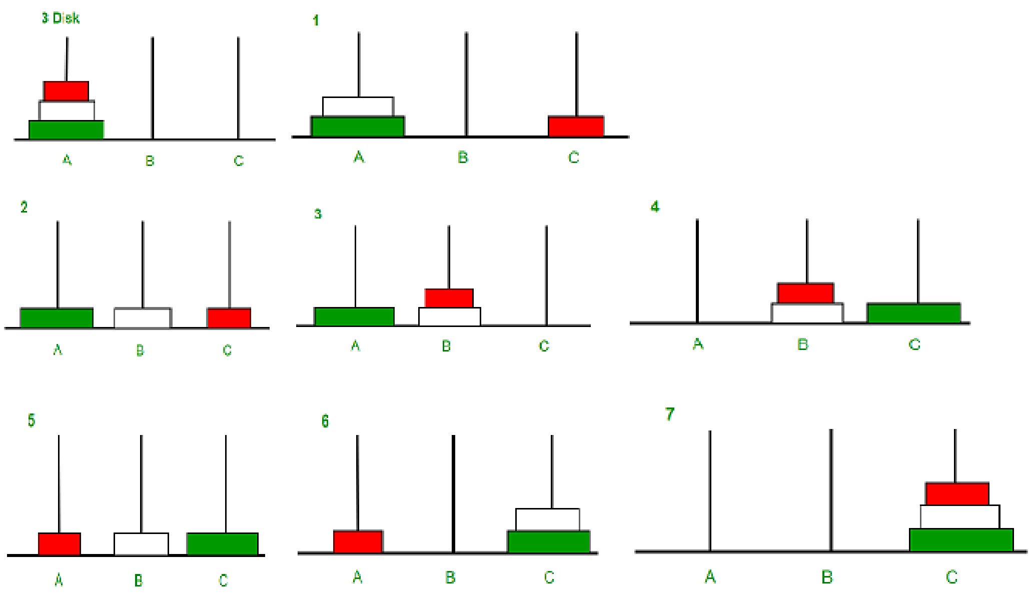
Recursive Applications- Recursive Binary Search, Towers of Hanoi.

Python Program for Tower of Hanoi

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1) Only one disk can be moved at a time.
2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3) No disk may be placed on top of a smaller disk.

Note: Transferring the top n-1 disks from source rod to Auxiliary rod can again be thought of as a fresh problem and can be solved in the same manner.

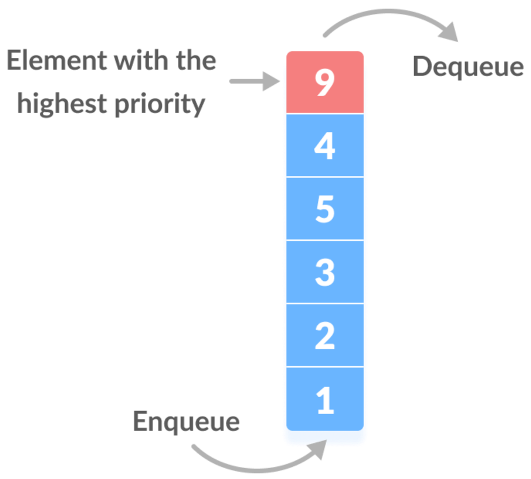

3 Disk
A
B
C
1
A
B
C
2
A
B
C
3
A
B
C
4
A
B
C
5
A
B
C
6
A
B
C
7
A
B
C

10. Design and implement priority queue data structure.

A priority queue is a special type of queue in which each element is associated with a priority value. And, elements are served on the basis of their priority. That is, higher priority elements are served first.

Assigning Priority Value

The element with the highest value is considered the highest priority element. However, in other cases, we can assume the element with the lowest value as the highest priority element.
We can also set priorities according to our needs.



Removing Highest Priority Element

Applications of priority Queue

- CPU priority scheduling
- Solving Dijkstra's shortest path algorithm
- Solving the Prim's Minimum Spanning Tree model
- Data compression using Huffman codes

Implement priority queue
Algorithm

```
Step 1: [Create class Priority queue]
        class PriorityQueue(object):
            def __init__(self):
                self.queue = []
            def __str__(self):
                return ' '.join([str(i) for i in self.queue])
            def isEmpty(self):
                return len(self.queue) == 0
            def insert(self, data):
                self.queue.append(data)
            def delete(self):
                try:
                    max_val = 0
Step 2: [Checking the Condition]
          for i in range(len(self.queue)):
              if self.queue[i] > self.queue[max_val]:
                  max_val = i
          item = self.queue[max_val]
          del self.queue[max_val]
          return item
      except IndexError:
          print()
          exit()
Step 3: [Inserting in Queue]
        if __name__ == '__main__':
            myQueue = PriorityQueue()
            myQueue.insert(12)
            myQueue.insert(1)
            myQueue.insert(14)
            myQueue.insert(7)
Step 4: [Output Operation]
            print(myQueue)
            while not myQueue.isEmpty():
                print(myQueue.delete())
```

Python Code

```
class PriorityQueue(object):
    def __init__(self):
        self.queue = []
    def __str__(self):
        return ' '.join([str(i) for i in self.queue])
    def isEmpty(self):
        return len(self.queue) == 0
    def insert(self, data):
        self.queue.append(data)
    def delete(self):
        try:
            max_val = 0
            for i in range(len(self.queue)):
                if self.queue[i] > self.queue[max_val]:
                    max_val = i
            item = self.queue[max_val]
            del self.queue[max_val]
            return item
        except IndexError:
            print()
            exit()
if __name__ == '__main__':
    myQueue = PriorityQueue()
    myQueue.insert(12)
    myQueue.insert(1)
    myQueue.insert(14)
    myQueue.insert(7)
    print(myQueue)
    while not myQueue.isEmpty():
        print(myQueue.delete())
```

Output:

```
12 1 14 7
14
12
7
1
```

11. Prepare a Report on balanced trees

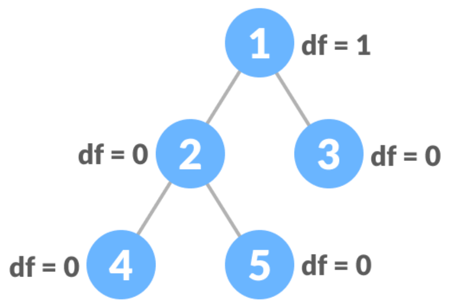
A balanced binary tree, also referred to as a height-balanced binary tree, is defined as a binary tree in which the height of the left and right subtree of any node differs by not more than 1.

Following are the conditions for a height-balanced binary tree:

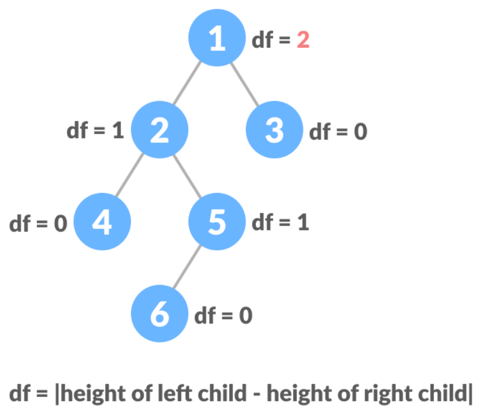
1. difference between the left and the right sub tree for any node is not more than one

2. the left subtree is balanced

3. the right subtree is balanced



Balanced Binary Tree with depth at each level



Unbalanced Binary Tree with depth at each level

# Checking if a binary tree is height balanced in Python

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = self.right = None
class Height:
    def __init__(self):
        self.height = 0
def isHeightBalanced(root, height):
    left_height = Height()
    right_height = Height()
    if root is None:
        return True
    l = isHeightBalanced(root.left, left_height)
    r = isHeightBalanced(root.right, right_height)
    height.height = max(left_height.height, right_height.height) + 1
    if abs(left_height.height - right_height.height) <= 1:
        return l and r
    return False
height = Height()
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)

root.left.right = Node(5)

if isHeightBalanced(root, height):

    print('The tree is balanced')

else:

    print('The tree is not balanced')
```

Output:

```
The tree is balanced
```

12. Implement expression evaluation tree.

Program to build and evaluate an expression tree using Python

1. LEFT = 0 RIGHT = 1.

2. Define a function evaluate() . This will take root. if value of root is a numeric value, then. return integer representation of value of root. ...

3. Define a function buildTree() . This will take postfix. root := null. stack := a new list.

4. Return root.

```
# Python program to evaluate expression tree

class node:

    def __init__(self, value):

        self.left = None

        self.data = value

        self.right = None

def evaluateExpressionTree(root):

    # empty tree

    if root is None:

        return 0

    # leaf node

    if root.left is None and root.right is None:

        return int(root.data)

    # evaluate left tree
```

```
        left_sum = evaluateExpressionTree(root.left)

        # evaluate right tree

        right_sum = evaluateExpressionTree(root.right)

        # check which operation to apply

        if root.data == '+':

                return left_sum + right_sum

        elif root.data == '-':

                return left_sum - right_sum


        elif root.data == '*':

                return left_sum * right_sum

        else:

                return left_sum // right_sum

# Driver function to test above problem

if __name__ == '__main__':

        # creating a sample tree

        root = node('+')

        root.left = node('*')

        root.left.left = node('5')

        root.left.right = node('-4')

        root.right = node('-')

        root.right.left = node('100')

        root.right.right = node('20')
```

```
    print (evaluateExpressionTree(root))
    root = None
    # creating a sample tree
    root = node('+')
    root.left = node('*')
    root.left.left = node('5')
    root.left.right = node('4')
    root.right = node('-')
    root.right.left = node('100')
    root.right.right = node('/')
    root.right.right.left = node('20')
    root.right.right.right = node('2')
    print (evaluateExpressionTree(root))
```

Output:

```
60
110
```

13. Prepare a report on hashing and analyze time complexity.

The hash() method returns the hash value of an object if it has one. Hash values are just integers that are used to compare dictionary keys during a dictionary look quickly.Example

```
text = 'Python Programming'

# compute the hash value of text

hash_value = hash(text)

print(hash_value)
```

```
Output: -966697084172663693
```

hash() Syntax

The syntax of hash() method is:

hash(object)

hash() Parameters

The hash() method takes a single parameter:

- object - the object whose hash value is to be returned (integer, string, float)

hash() Return Value

The hash() method returns the hash value of an object.

Example 1: How hash() works in Python?

```
# hash for integer unchanged

print('Hash for 181 is:', hash(181))

# hash for decimal

print('Hash for 181.23 is:',hash(181.23))

# hash for string
```

print('Hash for Python is:', hash('Python'))

Output

Hash for 181 is: 181

Hash for 181.23 is: 530343892119126197

Hash for Python is: 2230730083538390373

Example 2: hash() for immutable tuple object?

hash() method only works for immutable objects as tuple.

# tuple of vowels

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

print('The hash is:', hash(vowels))

Output

The hash is: -695778075465126279

How does hash() work for custom objects? As stated above, hash() method internally calls __hash__() method. So, any objects can override __hash__() for custom hash values. But for correct hash implementation, __hash__() should always return an integer. And, both __eq__() and __hash__() methods have to be implemented.

Below are the cases for correct __hash__() override.

__eq__()	__hash__()	Description
Defined (by default)	Defined (by default)	If left as is, all objects compare unequal (except themselves)
(If mutable) Defined	Should not be defined	Implementation of hash able collection requires key's hash value be immutable

Not defined	Should not be defined	If __eq__() isn't defined, __hash__() should not be defined.
Defined	Not defined	Class instances will not be usable as hashable collection. __hash__() implicity set to None. Raises TypeError exception if tried to retrieve the hash.
Defined	Retain from Parent	__hash__ = <ParentClass>.__hash__
Defined	Doesn't want to hash	__hash__ = None. Raises TypeError exception if tried to retrieve the hash.

Example 3: hash() for Custom Objects by overriding __hash__()

```
class Person:

    def __init__(self, age, name):

        self.age = age

        self.name = name

    def __eq__(self, other):

        return self.age == other.age and self.name == other.name

    def __hash__(self):

        print('The hash is:')

        return hash((self.age, self.name))

person = Person(23, 'Adam')

print(hash(person))
```

Output

```
The hash is:

3785419240612877014
```