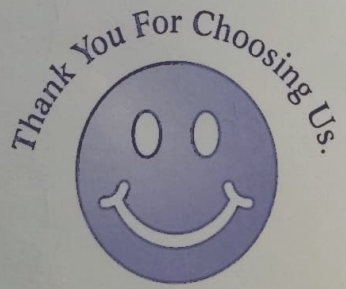
Voice of MD

My Dear Students,

I wish you all, work hard towards your goals, earn hard, enjoy your life, make your parents proud & ultimately contribute in development of Country.



* Serving students for last 40+ years.
* Superior quality in reasonable price.
* Products crafted specially from fully computerised atomatic machine.

INDEX

NAME: ..... STD.: ..... SEC.: .....

ROLL NO.: ..... SUBJECT.: .....

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
		DSA		
		By Aman Bhuva		

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks

Page No. 5
Date / /

Why to learn DSA?

1. Raise level of programming
2. Efficient Programming
3. Able to solve complex problems
4. Campus Placement
5. A Grade Company Placements.

Importance of Structuring Data?

Eg :- Dictionary Book, Maps,

What is a Data Structure?

Data structure is a particular way of storing & organizing data in a computer so that it can be used efficiently.

Classification of Data Structure.

(1) Linear DS

Array; Dynamic array; linked list; stack; queue; deque etc.

(2) Non-linear DS

BST, AVL, B-Tree, B+tree, graph etc

Teacher's Signature........................

Peacock
Page No.
Date / /

Algorithm:-

An algorithm is the step by step, linguistic representation of logic to solve a given problem.

Python to be known

- built-in types
- control statements
- functions
- classes and objects
- __init__()
- Types of variables and functions in class.

Array & List

Array:- collection of similar elements.

Ages are integer values so not
But should belong to same category
to make an array.

Array is not a builtin data structure
so need to be imported.

Teacher's Signature.....

from list :- list to array conversion
to list :- array to list conversion

Peacock
Page No.
Date / /

Creating Array:-

```
from array import *
a1 = array (type code, [, elements ])
```

type code	C type
'b'	signed integer
'B'	unsigned integer
'u'	unicode character
'h'	signed integer
'H'	unsigned integer
'i'	signed integer
'I'	unsigned integer
'l'	signed integer
'L'	unsigned integer
'q'	signed integer
'Q'	unsigned integer
'f'	floating point
'd'	floating point

Array methods:-

append()	pop()
count()	remove()
extend()	reverse()
fromlist()	tolist()
index()	
insert()	

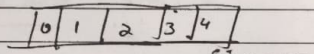
Teacher's Signature.....

List

-> list is a class
-> list is mutable
-> list elements are indexed
-> list is an iterable
-> list can grow (dynamic array)
-> list can contain different type elements

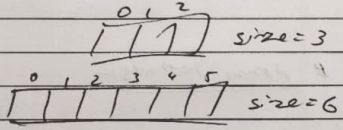
Array

-> collection of same type elements.
-> fixed size
-> indexed



Dynamic Array

-> collection of same type elements
-> resizable
-> indexed



Teacher's Signature.....

Creating list

```
l1 = [10, 20, 30]
l2 = []
l3 = [10, 5.7, 'abc']
```

Methods of list

append()
clear()
count()
extend()
index()
insert()
pop()
remove()
sort()
reverse()

Numpy

```
import numpy as np

a = np.array([1,2,3])
print(a)

b = np.array([[1,2,3],[10,20,30]])
# 2D array
```

Teacher's Signature.....

c = np.array([[1,2,3],[10,20]])

print(c)

↑ c is array with two list type elements.

13

Classes & objects:

Common Noun → Class.
Proper Noun → Object

Object → is something which can store its properties and capable to perform certain no. of actions.

Encapsulation:-

An act of combining properties of methods related to same object is known as ___.

Class → is a group of variables and functions. (description of an object)

↳ is a way to implement Encapsulation.

Attributes

are member variables & member functions.

eg:-

```
x = 5
def f1():
    # some code
```

x & f1 are attributes.

Objects:-

-> instance of a class
-> are of two types
  => class object
  => instance object

t1 = test()
t2 = test()

t1 & t2 are instance object of test class.

Class Object

Test Vs Test()

Test → Class Object
Test() → creates instance Object

One class has exactly one class object but can have any no. of instance objects.

For eg.

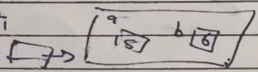
```
Class Test:
    ...
    ...
    ...
t1 = Test()
t2 = Test()
t3 = Test()
```

Test → class object

t1 →, t2 →, t3 → } instance object

__init__ method

```
class Test:
    def __init__(self):
        self.a = 5
        self.b = 6

t1 = Test() -> # __init__(t1)
```



a & b are instance object variables.

Methods:

- → Instance method
- → Static Method
- → Class Method

1) Static method does not access the [illegible] values of class object and instance object hence can be used to print a line or text within the code.

Static Variables & Instance variables.

Static → Class Object → Static Variables

Instance object → instance variables

L-4

Singly Linked List:-

What is a list?

→ List is a linear collection of data items also known as list item.

* List is not a fixed size, it is growable.

Eg:-

Ex 1 → list of marks (int)

30, 32, 20, 35, 41, 38, ——

Ex 2 → list of city names visited (str)

"Hzb", "Patna", "Gaya", "Ranchi", ——

Ex 3 → list of Employees. (Employee)

100	101	102	103	
"Atul"	"Savita"	"Akshay"	"Shivam"	
25000	35000	40000	30000	

What is a Node?

list of marks.

Eg:- 30, 32, 20, 35, 41, 38, ——

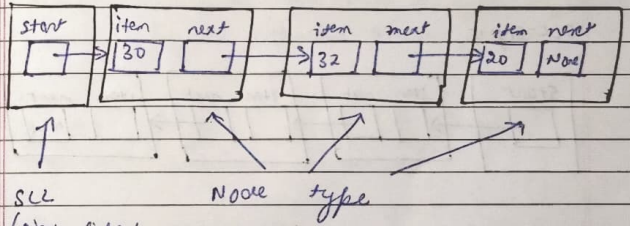
To store this, suppose we have 100 variables.

Teacher's Signature.....

x → 30 y → 32 z → 20

This way is not possible.

So,



Defining a Node :-

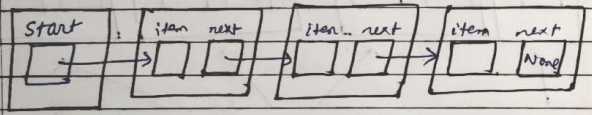
```
class Node:
    def __init__(self, item = None, next = None):
        self.item = item
        self.next = next
```

Teacher's Signature.....

• Singly linked - list.

- SLL is a linear data structure
- It can grow and shrink.
- Each node has a single link to the next one.

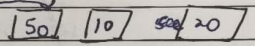
SLL-Object



Operations on Singly linked list :-

→ insertion
→ deletion
→ Is- empty
→ traverse

```
Obj = SLL()
Obj.insert_at_start(10)
Obj.insert_at_last(20)
Obj.insert_at_first(50)
```



Teacher's Signature.....

16

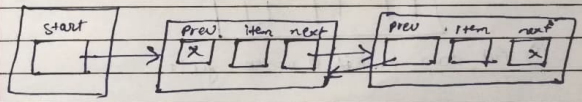
Doubly Linked List :-

- Limitation of SLL

→ SLL can only use link to move in forward direction, it is not possible to access the node which is before any node. So, no backward direction possible.

DLL :-

It is a linear data structure.



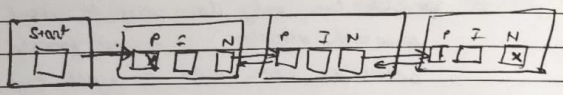
Elementary Operations :-

1. Insertion
2. Deletion
3. Traversing
4. Searching
5. Checking for empty list.

Teacher's Signature.....

Insertion :-

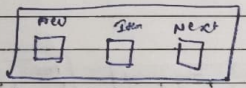
- Logic Build



If we have to insert one more at the first node then?

① Create node.

②



③ Put data in ~~this~~ node first.

n.next = start

start.prev = n

start = n

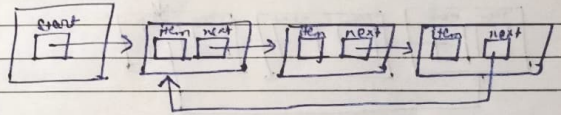
n.item = data

Add

Limitation of SLL.

- Putting "None" in last node "next" the will use memory, to reduce that memory space, circular singly linked list is used.

- It goes like



circular linked list -

- check for last node :-

→ If any node has same refrence value as the start node.

if temp.next == start:

- SLL vs CLL.

↳ at start - Traversing
at last - Traversing

Insertion

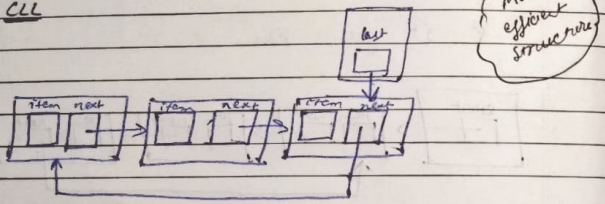
① at start → No Traversing

② at last → Traversing

Deletion :-

	SLL	CLL
①	first - No Traversing	Traversing
②	Last - Traversing	Traversing

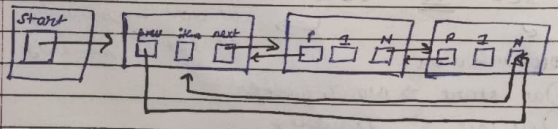
* CLL



Using this

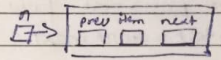
Insertion	Deletion
① start - No Traversing	① start - No Traversing
② Last - No Traversing	② Last - Traversing

Circular Doubly linked list :-



* Insertion :-

① at starting




```
n.item = data
n.next = start
n.prev = start.prev
start.prev.next = n
start.prev = n
start = n
```

② at last

```
n.item = data
n.prev = start.prev
n.next = start
start.prev.next = n
start.prev = n
```

③ after a particular node.

```
n.item = data
n.prev = temp
n.next = temp.next
temp.next.prev = n
temp.next = n.
```

• Deletion :-

① From start. (First Node)

start.prev.next = start.next

start.next.prev = start.prev

start = start.next

② Last Node.

start.prev.prev.next = start

start.prev = start.prev.next

temp.prev.next = temp.next

temp.next.prev = temp.prev

Stacks :-

Stack is a linear data structure.

Working Principle of stack is —

Last In First Out (LIFO).

Operations on Stack.

push()
pop()
• is-empty()
peek()
size()

Teacher's Signature.....

Page No.
Date

• Real world examples of Stack -

① Travelling Bag
② Stacked Plates in Buffet
③ Stack of books
④ Bread Packet

• Programming Examples -

① function call stack
② Evaluating expressions
③ Parenthesis matching
④ Iterative soln of binary tree traversal
⑤ Depth first search
⑥ undo - redo operations.

• Implementation of Stack -

① using list
② by extending list class
③ using singly linked list class
④ by extending singly linked list class
⑤ using linked list concept.

Teacher's Signature.....

Stack using List:-

① to create stack using list.

```
class stack :
    def __init__(self):
        self.items = [ ]
```

② to check if stack is empty.

```
def is_empty(self):
    return len(self.items) == 0
```

③ def push() function to add new data onto stack.

```
def push(self, data):
    self.items.append(data)
```

④ def pop() function to remove top element from stack.

```
def pop(self, data):
    if not self.is_empty():
        return self.items.pop()
    else:
        raise IndexError("Stack is empty")
```

Teacher's Signature.....

⑤ def peek() to return top element from stack.

```
def peek(self):
    if not self.is_empty():
        return self.items[-1]
    else:
        raise IndexError("Stack is empty")
```

⑥ def size() method to return the no. of elements in stack.

```
def size(self):
    return len(self.items)
```

```
s1 = stack()
s1.push(10)
s1.push(20)
s1.push(30)
print s1.peek()
print().
```

Teacher's Signature.....

stack using Inheriting list Class.

① create a stack using extending list class.

class stack (list):

② Define a method is_empty() to check if the stack is empty in stack class.

```
def is_empty(self):
    return len(self) == 0
```

③ def push() method to add data into the stack.

```
def push(self, data):
    self.append(data)
```

④ In stack class, def pop() method to remove top element from the stack.

```
def pop(self):
    if not self.is_empty():
        return super().pop()
    else:
        raise IndexError("stack is empty")
```


⑤ define peek() method to return top element from the stack.

```
def peek(self):
    if not self.is_empty():
        return self[-1]
    else:
        raise IndexError("stack is empty")
```

⑥ define size() method to return size of the stack.

```
def size(self):
    return len(self)
```

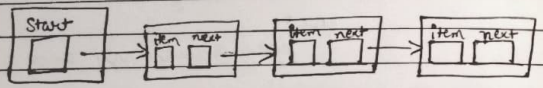
⑦ Implement a way to restrict use of insert() method of list class from stack object.

```
def insert(self, index, data):
    raise AttributeError("No attribute insert")
```

```
s1 = stack()
s1.insert(0,10)
```

Stack using Linked list :-
(Singly Linked List).

* Linked list



Points to keep in Mind

① This linked will be treated as a Stack.

② Stack will LIFO.

③ So, No insertion or deletion of any node from b/w is possible

④ If we insert at last and delete from last, then for every insertion/deletion, we will have to traverse the linked list.

⑤ So, to avoid this we will ~~use~~ insert and delete from starting of linked list to make stack.

Teacher's Signature.....

① Implement stack using SLL.
define __init__() to initialise start reference variable & item_count variable to keep track no. of elements in stack.

```
class Node:
    def __init__(self, item=None, next=None):
        self.item = item
        self.next = next

class Stack:
    def __init__(self):
        self.start = None
        self.item_count = 0
```

② define a method is_empty() to check if the stack is empty in Stack class.

```
def is_empty(self):
    return self.start == None
```

③ define push() method to add data into the stack

```
def push(self, data):
    n = Node(data, self.start)
    self.start = n
    self.item_count += 1
```

Teacher's Signature.....

④ define pop() method to remove top element from stack

```
def pop(self):
    if not self.is_empty():
        data = self.start.item
        self.start = self.start.next
        self.item_count -= 1
        return data
    else:
        raise IndexError("Stack is Empty").
```

⑤ define peek() to return top element from stack.

```
def peek(self):
    if not self.is_empty():
        return self.start.item
    else:
        raise IndexError("Stack is Empty")
```

self.is_empty().

⑥ define size() method to return size of the stack.

```
def size(self):
    return self.item_count.
```

```
S1 = stack()
S1.push(10)
S1.push(20)
S1.push(30)
```

Teacher's Signature.....

```
print("Total elements in the stack =", S1.size())
print("Top element in the stack is", S1.peek())
print("Popped element is", S1.pop())
```

Stack by importing Singly Linked List Module.

① import SLL module in your .py file.

Name in my pc.

```
from SLL import *
```

② define a class stack to implement a stack data structure.
define __init__() method to create SLL object.

```
class stack:
    def __init__(self):
        self.mylist = SLL()
        self.item_count = 0
```

any name.

③ define is_empty() to check if the stack is empty.

```
def is_empty(self):
    return self.mylist.is_empty()
```

④ define push() to add data onto the stack

```
def push(self, data):
    self.mylist.insert_at_start(data).
    self.item_count += 1.
```

Teacher's Signature.....

(5) pop() method to implement top element from stack.

```
def pop(self):
    if not self.is_empty():
        self.mylist.delete_first()
        self.item_count -= 1
```

(6) peek() method to return top element in stack

```
def peek(self):
    if not self.is_empty():
        return self.mylist.item
```

(7) size() method to return size

```
def size(self):
    return self.item_count
```

```
S = Stack()
S.push(10)
S.push(20)
S.push(30)
print("Top element is", S.peek())
S.pop()
print("Top element is", S.peek())
print()
```

Teacher's Signature.....

Implementation of stack by Inheriting linked list.

(1) Import module containing SLL code in your py file.

```
from SLL import *
```

(2) Define a class stack to implement stack by inheriting SLL class.

```
class Stack(SLL):        # this SLL is bij "from SLL"
    def __init__(self):
        self.item_count = 0
```

(3) Define a method is_empty() to check if the stack is empty in Stack class.

```
def is_empty(self):
    return super().is_empty()
```

↳ parent class called directly.

(4) define push() to add new data onto the stack.

```
def push(self, data):
    self.insert_at_start(data)
    self.item_count += 1
```

(5) define pop() to remove top element.

```
def pop(self):
    if not self.isempty():
        self.delete_first()
        self.item_count -= 1
```

Teacher's Signature.....

③ define peek() method to return top element.

```
def peek(self):
    if not self.is_empty():
        return self.items[-1]
    else:
        raise IndexError("Stack Underflow")
```

④

⑤ define size(), to return size

```
def size(self):
    return self.item_count
```

Queue

Queue is a linear data structure

Working principle of Queue is

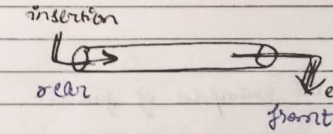
First in First Out (FIFO)

Agar hume class banaya hai toh ek data type banaya hai

ADT → Abstract Data Type

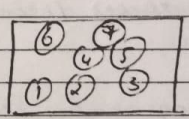
For eg. someone else has created a "Queue Data Structure" or a "Queue Data Type" and we are using it directly, then Queue is called an Abstract Data type.

#



rear → Insertion

front → deletion.



Suppose 1 to 7 are inserted in sequence

So, 1 is to be deleted first.
So, front element is (1)
& rear element is (7)

operations on Queue.

enqueue (Insertion)
dequeue (deletion)
is_empty
get_front
get_rear
size

Real world Examples of Queue.

① Queue for roller coster in an amusement park.

② Shuttlecock (badminton) in a cylindrical box.

Programming Examples.

① Print tasks in a printer
② Admission module to print first come first serve.
③ Breadth first Search
④ Railway reservation

Teacher's Signature.....

Implementation of Queue.

① using list
② by extending list class
③ using singly linked list class
④ by extending singly linked list class
⑤ using linked list concept.

Queue using list :-

① Define a class Queue to implement Queue data structure using list. Define __init__() method to create an empty list object as instance object member of Queue.

```
class Queue:
    def __init__(self):
        self.items = []
        self.front = None
        self.rear = None
```

② Define a method is_empty to check if the Queue is empty or not.

```
def is_empty(self):
    len(self.items) == 0
```

Teacher's Signature.....

③ define enqueue() method to add data at rear end of the queue

```
def enqueue(self, data):
    self.items.append(data)
```

④ define dequeue() method to remove front element from the queue.

```
def dequeue(self):
    if not self.is_empty():
        self.items.pop(0)
    else: raise IndexError("Queue Underflow")
```

⑤ define get_front() method to return front element of the queue.

```
def get_front(self):
    if not self.is_empty():
        return self.items[0]
    else:
        raise IndexError("Queue Underflow")
```

⑥ def get_rear() method to return rear element of the queue.

```
def get_rear(self):
    if not self.is_empty():
        return self.items[-1]
    else:
        raise IndexError("Queue Underflow")
```

Teacher's Signature.....

⑦ define size() method to return size of the queue.

```
def size(self):
    return len(self.items)
```

```
q1 = Queue()
print(q1.get_front())
```

on this case it will raise an unhandled error.

So,

```
q1 = Queue()
try:
    print(q1.get_front())
except IndexError as e:
    print(e.args[0])

q1.enqueue(10)
q1.enqueue(20)
q1.enqueue(30)
q1.enqueue(40)
q1.enqueue(50)

print("Front =", q1.get_front(), "Rear:", q1.get_rear())
try:
    q1.dequeue()
    print("Queue has now", q1.size(), "elements")
except IndexError as e:
    print(e.args[0])
```

Teacher's Signature.....

Queue using Singly linked list.

① Define a class queue to implement queue data structure using Singly linked list concept. Define __init__() method to initialize front & rear reference variable; and item_count variable to keep track of no. of elements in the queue.

```
class queue:
    def __init__(self):
        self.front = None
        self.rear = None
        self.item_count = 0
```

② Define a method, is_empty() to check if the queue is empty in queue class

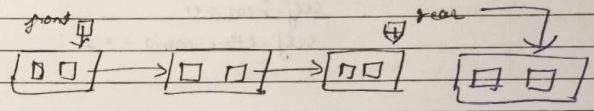
```
def is_empty(self):
    return self.front == None      # Method 1 ->
    return self.rear == None       # Method 2 ->
    return self.item_count == 0    # Method 3 ->
```

③ define enqueue() method.

~~def enqueue(self, data):~~

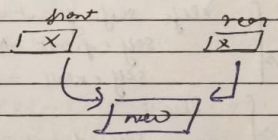
Here two cases will be formed

① If the LL has nodes,



- If a new node is added, then, only rear will point to new.

② If the LL is empty,



- If a new node is added, the front & rear both will point to the new node

```
class Node:
    def __init__(self, item = None, next = None):
        self.item = item
        self.next = next
```

will be created in starting above queue class.

Teacher's Signature.....

Teacher's Signature.....

```
def enqueue (self, data):
    n = Node (data)
    if self.is_empty():
        self.front = n
    else:
        self.rear.next = n
    self.rear = n
    self.item_count += 1
```

④ define dequeue()

```
def dequeue (self):
    if self.is_empty():                          # if it is empty
        raise IndexError("Empty queue")
    elif self.front == self.rear:                # if only one element in it
        self.front = None
        self.rear = None
    else:                                        # if more than one element
        self.front = self.front.next
    self.item_count -= 1
```

⑤ define get_front()

```
def get_front(self):
    if self.is_empty():
        raise IndexError("No data in queue")
    else:
        return self.front.item
```

⑤ define get_rear()

```
def get_rear(self):
    if self.is_empty():
        raise IndexError("No data in queue")
    else:
        return self.rear.item
```

⑥ define size()

```
def size(self):
    return self.item_count
```

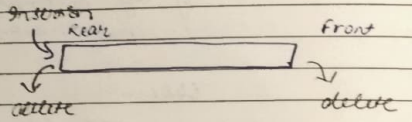
```
q1 = queue()
q1.enqueue(10)
q1.enqueue(20)
q1.enqueue(30)
print(q1.get_front(), q1.get_rear())
q1.dequeue()
print(q1.get_front(), q1.get_rear())
```

Teacher's Signature.....

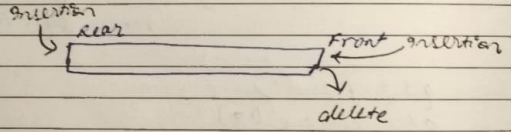
Deque

Variants of queue :-

① Insertion Restricted queue



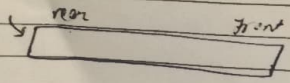
② Deletion Restricted queue



Deque :-

- Deque is another variation of queue.
- Deque is double - ended - queue
- Deque is a linear data structure

- Both ends can be used for insertion & deletion



Teacher's Signature.....

- Some deques may support random access, depending on implementation.

Operations on Deque.

insert_front
insert_rear
delete_front
delete_rear
get_front
get_rear
is_empty
size

Implementation of Deque :

① using list
② by extending list class
③ using doubly linked list class
④ by extending doubly linked class
⑤ using linked list concept

Teacher's Signature.....

Deque using list :-

① Define a class deque to implement deque data structure using list. Define __init__() method to create an empty list object as instance object member of deque.

```
class deque:
    def __init__(self):
        self.items = []
```

② Define a method is_empty() to ———

```
def is_empty(self):
    return len(self.items) == 0
```

③ def insert_front() to add data at front end of deque.

```
def insert_front(self, data):
    self.items.insert(0, data)
```

④ def insert_rear() —— rear ——

```
def insert_rear(self, data):
    self.items.append(data)
```


⑤ def delete_front() —— remove ——

```
def delete_front(self):
    self.items
    if self.is_empty():
        raise IndexError("Deque empty")
    else:
        self.items.pop(0)
```

⑥ def delete_rear() ———

```
def delete_rear(self):
    if self.is_empty():
        raise IndexError("Deque empty")
    else:
        self.items.pop(0)
```

⑦ def get_front()

```
def get_front(self):
    if self.is_empty():
        raise IndexError("  ")
    else:
        return self.items[0]
```

② def get_rear() :-

```
def get_rear(self):
    if self.is_empty():
        raise IndexError(" ")
    else:
        return self.items[-1]
```

③ def size() :- -

```
def size(self):
    return len(self.items)
```

```
d1 = Deque()
d1.insert_front(10)
d1.insert_front(20)
d1.insert_rear(30)
d1.insert_rear(40)
print(d1.get_front(), d1.get_rear())
```

Teacher's Signature.....

Deque using Doubly linked list :-

① Define a class Node with instance object member variables prev, item & next

```
class Node:
    def __init__(self, item=None, prev=None,
                 next=None):
        self.prev = prev
        self.item = item
        self.next = next
```

= ② - ⑩ (same as prev ① - ⑨)

② class Deque :

```
def __init__(self):
    self.front = None
    self.rear = None
    self.item_count = 0
```

③ def is_empty(self):

```
    return self.front == None
    return self.item_count == 0
```

④ def insert_front(self, data):

```
    n = Node(data, None, self.front)
    if self.is_empty():
        self.rear = n
    else:
        self.front.prev = n
    self.front = n
```

Teacher's Signature.....

```
④ def insert_rear(self, data):
    n = Node(data, self.rear)
    if self.is_empty():
        self.front = n
    else:
        self.rear.next = n
    self.rear = n
```

```
⑤ def delete_front(self):
    if self.is_empty():
        raise IndexError("Deque is empty")
    if self.front == self.rear:
        self.front = None
        self.rear = None
    else:
        self.front = self.front.next
        self.front.prev = None
```

```
⑥ def delete_rear(self):
    if self.is_empty():
        raise IndexError("Deque is empty")
    if self.front == self.rear:
        self.front = None
        self.rear = None
    else:
        self.rear = self.rear.prev
        self.rear.next = None
```



```
⑦ def get_front(self):
    if self.is_empty():
        raise IndexError("Deque is empty")
    return self.front.item
```

```
⑧ def get_rear(self):
    if self.is_empty():
        raise IndexError("Deque is empty")
    return self.rear.item
```

```
⑨ def size(self):
    return self.item_count
```

Priority Queue

A priority queue is a collection of elements such that each element has been assigned a priority

-> An element with higher priority is processed first.
-> Two elements with same priority are processed in the order they are added.

- Operations of Priority queue
  - -> push()
  - -> pop()
  - -> is_empty()
  - -> size

Implementation of Priority Queue

1. using ordered list concept
2. using list
3. using heap (will see later)

Priority Queue using list

1. Define a class PriorityQueue to implement Priority queue data structure using list. Provide --init--() method to create a list object.

```
class PriorityQueue :
    def --init-- (self):
        self.items = []
```

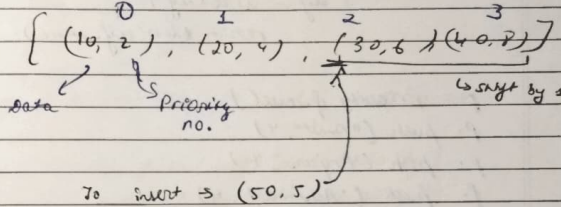
2. Define a push method in PQ to insert a new data with given priority.

```
def push (self, data, priority):
    index = 0
    while index < len(self.items) and
    self.items[index][1] <= priority :
        index += 1
    self.items.insert(index, (priority, data))
```

Teacher's Signature.....

Working Algo

Suppose a list



To insert → (50, 5)

initialize → index = 0 # 2 ;

3. Define a pop method in PQ, which returns the highest priority data stored in the PQ data structure. Raise exception if PQ is empty.

```
def is_empty (self):
    return len(self.items) == 0

def pop(self):
    if self.is_empty():
        raise TypeError("PQ is empty")
    self.items.pop(0)[0]
```

Teacher's Signature.....

⑤ In class PQ, define a method size to return the no. of elements present in the PQ.

```
def size(self):
    return len(self.items)

p = PriorityQueue()
p.push("Amit", 4)
p.push("Arjun", 1)
p.push("Ashwin", 4)
p.push("Ajeet", 4)
p.push("Anant", 4)

while not p.is_empty():
    print(p.pop())
```

Priority Queue using linked list.

① Define Node class with instance member variables items, priority and next.

```
class Node:
    def __init__(self, items=None, priority=None,
                 next=None)
        self.item = item
        self.next = next
        self.priority = priority
```

② Def PQ.

```
class PriorityQueue:
    def __init__(self):
        self.start = None
        self.item_count = 0
```

③ def push in PQ to add new data.

```
def push(self, data, priority):
    n = Node(data, Priority)
    if not self.start or priority < self.start.
                                    priority:
        n.next = self.start.
        self.start = n.
    else:
        temp = self.start
        while temp.next and temp.next.
                        priority <= priority:
            temp = temp.next
        n.next = temp.next
        temp.next = n.
    self.item_count += 1
```

④ def pop in PQ ——— .

⑤ def is_empty() ——.

```
def is_empty(self):
    return self.start == None
```

Teacher's Signature.....

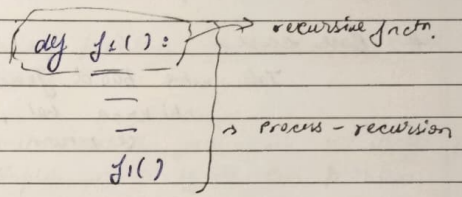
```
def pop(self):
    if self.is_empty():
        raise IndexError("PQ is empty")
    data = self.start.item
    self.start = self.start.next
    return data
```

⑥
```
def size(self):
    return self.item_count
```

```
p = PriorityQueue()
p.push("A", 4)
p.push("B", 5)
p.push("C", 1)
p.push("D", 3)
p.push("E", 6)
p.push("F", 2)

while not p.is_empty():
```


Recursion

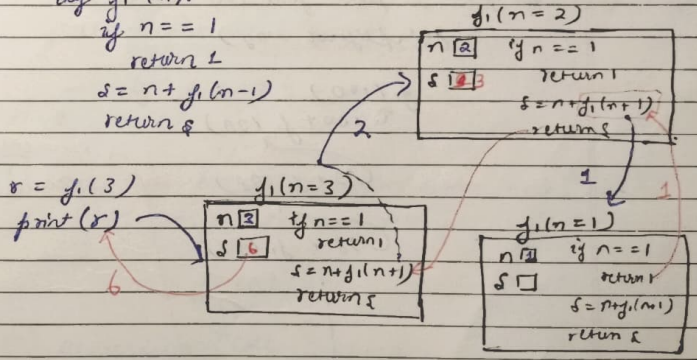
Function calling itself is called recursion.



Recursion trees:-

Taste Dry Run

```
def f1(n):
    if n == 1:
        return 1
    s = n + f1(n-1)
    return s
r = f1(3)
print(r)
```



(This an example of sum of 1st n natural nos.)

Page No.
Date / /

Recursive Case :-
Jisme hum function ko call kar rahe hain fir se.. to solve a problem

Base case :-
Jab hume baad function ko call nhi krna hai, direct wahan se return lele.

If base case is not there then it will go in ∞ loop.

For eg :- To print first 100 natural nos.
~~(Simplified way)~~

f(100)
↳ 100 + f(99)
↳ 99 + f(98)
↳ 98 + f(97).
⋮
3 + f(2)
2 + f(1)
↓
1

→ Recursive cases

→ Base case, at the last, it will return 1.

Teacher's Signature........................

Page No.
Date / /

- Recursive statements me problem ko problem ke terms me solve krte to hai to make it smaller.

In recursion, problem is solved in terms of problem itself.

Each time recursive fnctn call to itself for little simpler version of the Og problem.

HOW TO Approach recursive function?

Q. Write a recursive function to calculate sum of first n natural nos.

⇒ def f(n) :

① Maan lete hai ki function bana hua hai, tab hi to call krenge.

⇒ f(n) → 1 + 2 + 3 + 4 + --- + n.

② Recursive Case.

⇒ f(n-1) + n → 1 + 2 + 3 + --- + n-1 + n

Ye kaise kaam krega ye humein nhi sochna hai

Teacher's Signature........................

③ Base case

n == 1 -> 1.

∴ fact →

```
def f1(n):
    if n == 1
        return 1
    return n + f1(n-1)
```

Problems based on Recursion :-

① Print first N natural nos.

```
def printN(n):
    if n > 0:
        printN(n-1)
        print(n, end=' ')

printN(10).
```

② Print first N natural nos. in reverse order.

```
def printNrev(n):
    if n > 0:
        print(n, end=' ')
        printNrev(n-1)

printNrev(10).
```

③ print first N odd natural nos.

```
def printNodd(n):
    if n > 0:
        printNodd(n-1)
        print(2*n-1, end=' ')

printNodd(10)
```

④ print first N even natural nos.

```
def printNeven(n):
    if n > 0:
        printNeven(n-1)
        print(2*n, end=' ')

printNeven(10)
```

Teacher's Signature.....

⑤ print first N odd natural nos. in rev order.

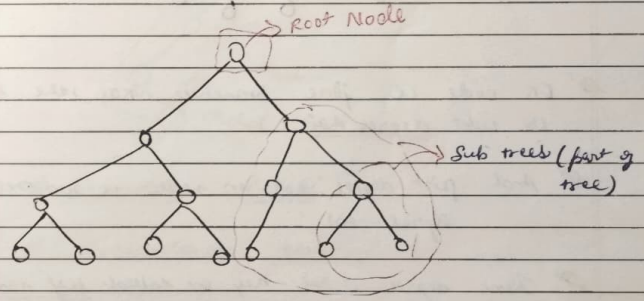
```
def printNoddrev(n):
    if n>0:
        print(2n-1, end=' ')
        printNoddrev(n-1)

printNoddrev(10)
```

⑥ same for even ↗

Teacher's Signature.....

Tree Data Structure.

A tree is defined as a finite set of one or more data items (called nodes) such that there is a special node called the root node of the tree and the remaining nodes are partitioned into $n \geq 0$ disjoint subsets, each of which is itself a tree, and they are known as subtrees.

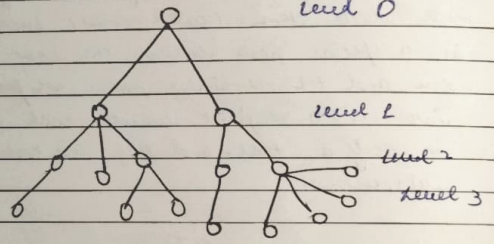


- Tree can't form a cycle.
- It is Hierarical Data Structure

Teacher's Signature.....

Degree of the tree :-

Suppose a tree as:-



- Ek node se jitne branches nikal rahe hain wo uski degree hai
- And pure degree hai us node me, wo tree ki degree hai.
- Jisme degree 0 hai, they are called leaf nodes or terminal nodes

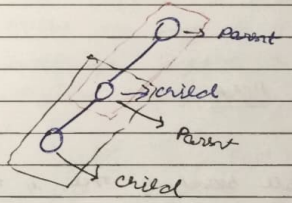
Level Number:-

To root node hai that is level 0. Uske baad ke nodes at level 1 and similarly.

- Generations :-

Level 0 → 1st Gen
Level 1 → 2nd Gen
Level 2 → 3rd Gen
Level 3 → 4th Gen

- Parent child :-

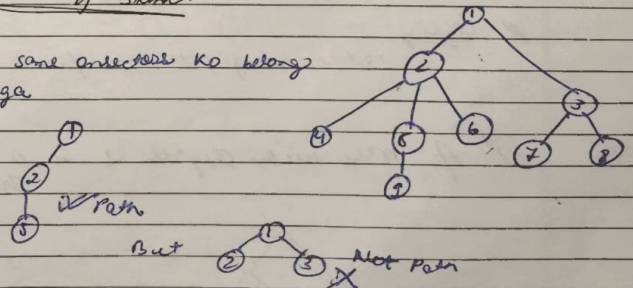


Ancestors :- Purvaj

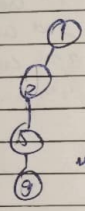
Descendants :- Vanshaj

- Path of Branch :-

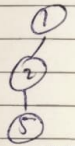
Path same ancestors ko belongs krega



∴ Ek branch ka end leaf node pe honge chahiye.



✓ Branch



✗ Not a Branch.

Height or Depth

↓

Largest possible branch of tree is the height.

(Total no. of nodes) + 1

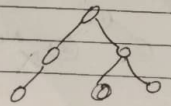
or

Total no. of generations

Siblings :-

nodes belonging to same parent.

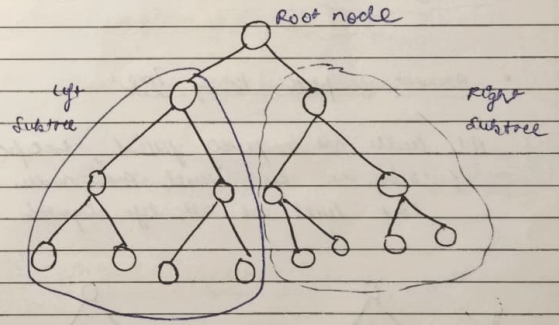
A tree with degree 2 is a Binary Tree.



Binary Tree

A binary tree is defined as a finite set of elements, called nodes, such that

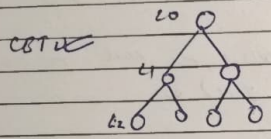
- T is empty (called the Null Tree or empty Tree) or
- Tree contains a distinguished node R, called the root of T, and the remaining nodes of T form an ordered pair of disjoint binary trees T_1 & T_2.



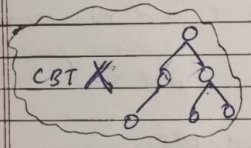
Any node in the binary tree has either 0, 1 or 2 child nodes.

• Complete Binary Tree:

All levels are completely filled

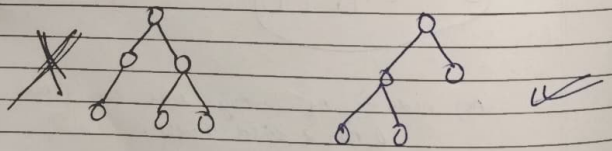


L_0 - 1 node
L_1 - 2 node
L_2 - 4 node
L_3 - 8
L_4 - 16
L_5 - 32
L_6 - 64
L_7 - 128
L_n - 2^n nodes.



• Almost complete Binary Tree:-

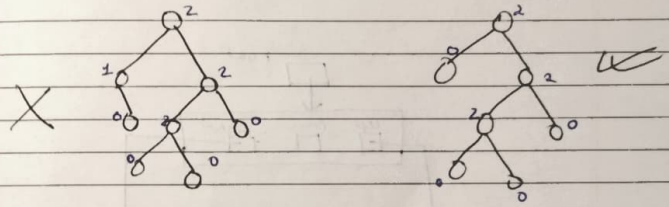
All levels are completely filled, except possibly the last level and nodes in the last level are all left aligned



Teacher's Signature.....

• Strict Binary Tree :-

Each node of a strict binary tree will have either 0 or 2 children.

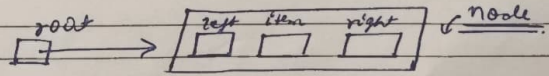


Representation of Binary Tree:-

→ There are 2 possible representation.

1. Array Representation
2. Linked Representation (by default)

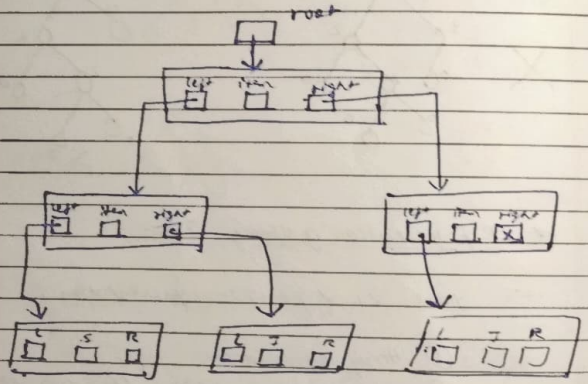
Linked Representation



- root is a node pointer
- when root contains NULL, tree is empty.

Teacher's Signature.....

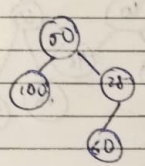
```
class Node:
    def __init__(self, item=None, left=None, right=None):
        self.item = item
        self.left = left
        self.right = right
```



Teacher's Signature.....

Discussion

* How to insert an item in a BT?
* How to traverse a BT?



Now where to insert (75)

⇒ 75 can be inserted anywhere
⇒ There is no rule for insertion in a Binary Tree.
⇒ Therefore code is not done for a Binary Tree. Or there is no correct rule in a Binary Tree.

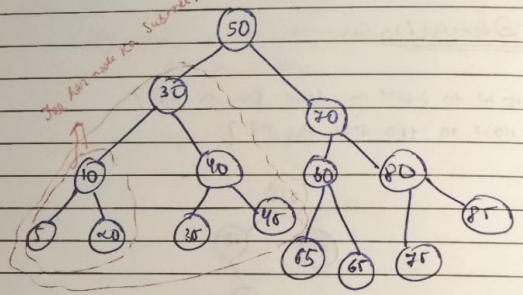
Binary Search Tree

A binary search tree is the most imp. data structure, that enables one to search for and find an element with an avg. running time.

$$f(n) = O(\log_2 n)$$

* Duplicate values are not allowed in BST (By default)

Teacher's Signature.....



Kisi bhi node mein jo value hai, uss uske left subtree ki sabhi value se badi aur right subtree ki sabhi values se chhoti hai.

Binary Search Tree is a binary tree with the value of node N is greater than every value in the left subtree of N and is less than every value in the right subtree of N.

Unless, explicitly said, BST doesn't allow duplicate values.

If by any chance there is a duplicate value, we it will go to the right subtree.

implementation.

1. node
2. Insertion
3. Traversing
4. Search
5. Deletion.

Node:-

```
class Node:
    def __init__(self, item = None, left = None
                 right = None)
        self.item = item
        self.left = left
        self.right = right
```

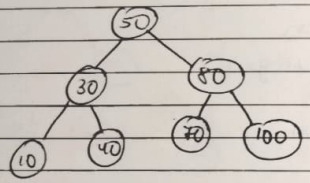
Insertion.

First we will create a class named BST.

```
bst = BST()
```

Kisi bhi value jo insert karna hoga, usse bhi humara ek node banayenge, with item containing the value, and left and right containing None.

Traversing



Preorder

root → T_L → T_R

⇒ 50, 30, 10, 40, 80, 70, 100

Inorder

T_L → root → T_R

⇒ 10, 30, 40, 50, 70, 80, 100 (Inorder)

- This traversing gives values in sorted order.

Postorder

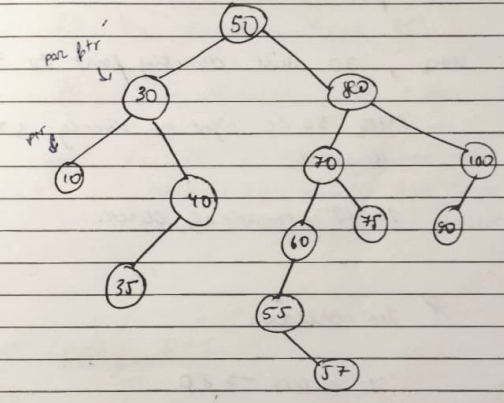
T_L → T_R → root.

⇒ 10, 40, 30, 70, 80, 100, 50

Teacher's Signature.....

Deletion :-

1. No child
2. Single child
3. Two children



No Child :-

To delete → 10

2 reference variable → (ptr.)

1. Parent ptr → 30
   ptr → 10

Then we will keep None in the left of 30

Teacher's Signature.....

Page No.
Date : / /

Ø single child.

To delete → 40

par ptr → 30
ptr → 40

Aur, 30 will directly point to 35.

Matlb 30 ke right me directly 35 daal dunga.

To 40 automatically delete.

Ø Two children.

To delete → 80

Two cases → ① Immediate successor
② Immediate predessor.

Yaa, yaa hum hume 55 in inorder traversing.

10, 30, 35, 40, 50, 55, 57, 60, 70, 75, 80, 90, 100

Predessor

Successor.

Teacher's Signature.....

Page No.
Date : / /

Toh we will choose any one, either successor or predessor and put it in place of the node to be deleted.

Jaise,

80 Ka predesser hoga 75.

To, 50 directly point krega 75 ko.

and jo 75 jo pehle se ek aur node hai, usko delete kar dunga (single child, ya two child se).

Coding BST

① Define a class Node, with instance variable left, item, right. The variables L & R are used to refer L & R child node. The item variable is used to hold data item.

class Node:
def __init__(self, item=None, left=None, right=None)

self.item = item
self.left = left
self.right = right

Teacher's Signature.....

② Define a class BST, Make __init__() method to create root instance variable to hold the reference of root node.

```
class BST:
    def __init__(self):
        self.root = None
```

③ In class BST, define insert method to store new data item in BST.

```
def insert(self, data):
    self.root = self.rinsert(
        self.root, data)
```

Yahan pe hum recursion method ka use karenge kyunki BT ka subtree bhi BT hi hota.

```
def rinsert(self, root, data):
    if root is None:
        return Node(data)
    if data < root.item:
        root.item = self.rinsert(root.left, data)
    elif data > root.item:
        root.right = self.rinsert(root.right,
                                  data)
    return root.
```

④ Define a search method to find a given item in BST & return node reference.

```
def search(self, data):
    return self.rsearch(self.root, data)

def rsearch(self, root, data):
    if root is None or root.item == data:
        return root
    if data < root.item:
        return self.rsearch(root.left, data)
    else:
        return self.rsearch(root.right, data)
```

Yahe nam root nhi, kuch, just Locating var. hai.

⑤ Def a method to implement inorder traversal.

```
def inorder(self):
    result = []
    self.rinorder(self.root, result)
    return result

def rinorder(self, root, result):
    if root:
        self.rinorder(root.left, result)
        result.append(root.item)
        self.rinorder(root.right, result)
```

Teacher's Signature.....

Teacher's Signature.....

(6) define a method to implement preorder.

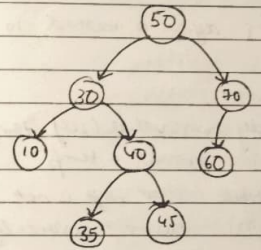
```
def preorder(self):
    result = []
    self.rpreorder(self.root, result)
    return result

def rpreorder(self, root, result):
    if root:
        result.append(root.item)
        self.rpreorder(root.left, result)
        self.rpreorder(root.right, result)
```

(7) def a method to implement postorder.

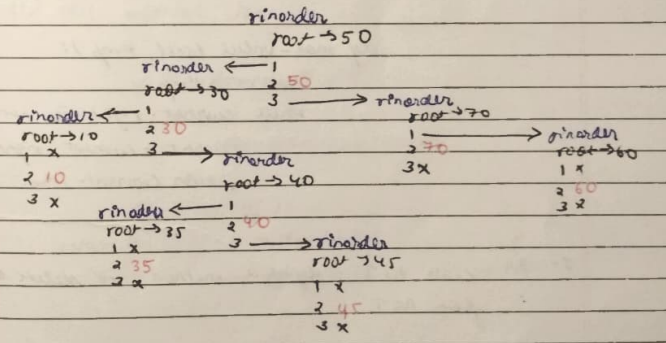
```
def postorder(self):
    result = []
    self.rpostorder(self.root, result)
    return result

def rpostorder(self, root, result):
    if root:
        self.rpostorder(root.left, result)
        self.rpostorder(root.right, result)
        result.append(root.item).
```

Teacher's Signature.....

Working prev ques:-



[10, 30, 35, 450, 45, 50, 60, 70]



Teacher's Signature.....

Deletion in BST:-

1. In class BST, define a method to find min value item node.

```
def min_value(self, temp):
    current = temp
    while current.left is not None:
        current = current.left
    return current.item
```

2. ——— max value ———

```
def max_value(self, temp):
    current = temp
    while current.right is not None:
        current = current.right
    return current.item
```

3. In class BST, define a method to delete a node from BST.

```
def delete(self, data):
    self.root = rdelete(self.root, data)
def rdelete(self, root, data):
    if root is None:
        return root
    if data < root.item:
        root.left = self.rdelete(root.left, data)
    elif data > root.item:
        root.right = self.rdelete(root.right, data)
    else:
        if root.left is None:
            return root.right
        elif root.right is None:
            return root.left
        root.item = self.min_value(root.right)
        self.rdelete(root.right, root.item)
    return root
```

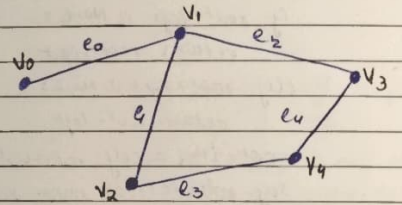
(1) def. method size.

In class BST def

```
def size(self):
    return len(self.inorder()).
```

Graph

- Graph is a non linear data structure



$V = \{ v_0, v_1, v_2, v_3, v_4 \}$ → 5 vertices
$E = \{ e_0, e_1, e_2, e_3, e_4 \}$ → 5 Edges.

$e_0 = [v_0, v_1]$
$e_1 = [v_1, v_2]$
} It means ki both direction me jaa sakte hain, v_0, se v_1 bhi and v_1 se v_0 bhi.

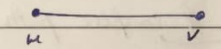
$e_0 = (v_0, v_1)$ Iska matlb sirf v_0 se v_1 jaa sakte hain.

$\therefore G = (V, E)$

- A graph consists of two things
  - → A set V of elements called nodes.
  - → A set E of edges such that each edge e in E is identified with a unique (unordered) pair $[u, v]$ of nodes in V, denoted by $e = [u, v]$
  - → We indicate the parts of the graph by writing $G = (V, E)$

- Adjacent nodes –

  Agal Bagal wala nodes.



  If $e = [u, v]$ then u & v are Adjacent nodes

- Degree of node

  Ek node se kitne edges nikal rahi hain.

  If $deg(u) = 0$, then u is called isolated node.

Teacher's Signature.....

• Path.

A path of length n from a node u to a node v is defined as a sequence of n+1 nodes.

$$P = (v_0, v_1, v_2, \ldots v_n)$$

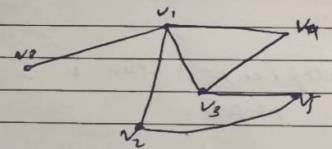
• The path is said to be closed if $v_0 = v_n$.

(Jisme koi v node repeat na ho)

• simplex or complex path.

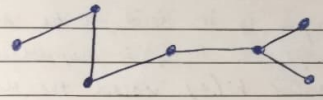
(Some node repeat ho sake again)

The path is said to be simple if all the nodes are distinct, with the exception that v_0 may equal to v_n otherwise it is complex path.

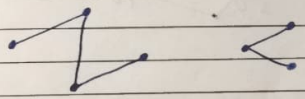


Teacher's Signature.....

Connected Graph



Disconnected Graph

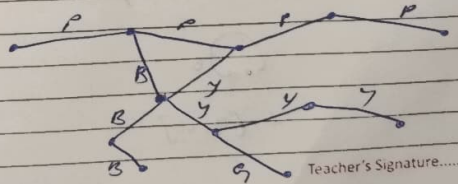


Tree Graph.

A connected graph T without any cycles is called a tree graph or free tree or simply a tree

Labelled Graph:

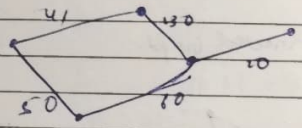
A graph is to be labelled if its edges are assigned data.



Teacher's Signature.....

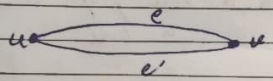
Weighted Graph.

A graph G is said to be weighted if each edge e in G is assigned a non-numerical value w(e) called the weight or length of e.



Multiple Edges

Distinct edge e & e' are called multiple edges if they connect the same end points, that is, if e = [u, v] and e' = [u, v]



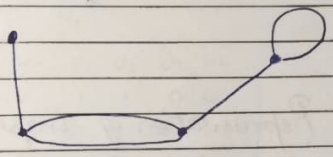
Loop

An edge is called loop if it has identical end pts.

e = [u, u]

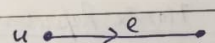
Multigraph

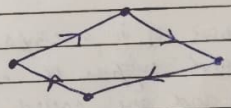
Multigraph is a graph consisting of multiple edges and loops.



Directed Graph

e = (u, v) → ordered pair

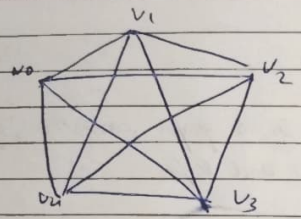




Complete Graph.

A simple graph in which there exists an edge b/w every pair of vertices is called a complete graph.

Also known as universal graphs or clique.



Representation of Graph

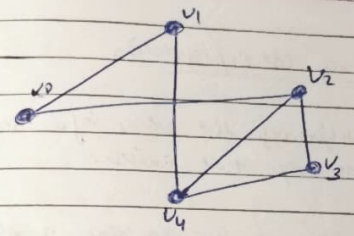
(1) Adjacency Matrix Representation
(2) List Representation.

Adjacency Matrix Representation.

Suppose G is a simple Graph with n nodes, and suppose the nodes of G have been ordered and are called $v_1, v_2, v_3, \ldots, v_n$. Then the adjacency matrix

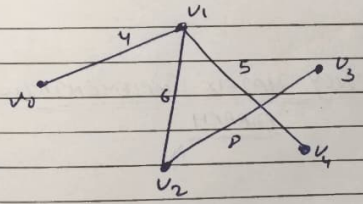
$A = (a_{ij})$ of the graph G is the $n \times n$ matrix —

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$



	v_0	v_1	v_2	v_3	v_4
v_0	0	1	1	0	0
v_1	1	0	0	0	1
v_2	1	0	0	1	1
v_3	0	0	1	0	1
v_4	0	1	1	1	0

Weighted Graph



	v_0	v_1	v_2	v_3	v_4
v_0	0	4	0	0	0
v_1	4	0	6	0	5
v_2	0	6	0	8	0
v_3	0	0	8	0	0
v_4	0	5	0	0	0

Page No.
Date / /

Adjacency List Representation

The adjacency list stores info about only those edges that exists.

The adjacency list contains a directory (dict) containing adjacency list for each vertex.

Isme jaise jo 0 hai use nhi use krenge, bss jo connected hai uska dictionary bana lete hai.

+ ye space bhi save krega.

ADJACENCY MATRIX IMPLEMENTATION OF GRAPH

① Write a class "Graph" to implement Adjacency Matrix of a simple and undirected graph.

```
class Graph:
    def __init__(self, vno):
        self.vertex_count = vno
        self.adj_matrix = [[0]*vno for
                e in range(vno)]
```

Teacher's Signature.......................

Page No.
Date / /

Isme
Pehle ek list banaye aur usko no. of elements se multiply kr diye,
So let vno be 5.
Then, list will be [0 0 0 0 0]
Now, for loop will run, and will look like

$$\begin{bmatrix} [0 & 0 & 0 & 0 & 0] \\ [0 & 0 & 0 & 0 & 0] \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

② def __init__ method to initialize vertex_count of adj_matrix (list of list)

③ def add_edge() method to add an edge in the graph with given weight

```
def add_edge(self, u, v, weight=1):
    if 0<=u<self.vertex_count
        and 0<=v<self.vertex_count:
        self.adj_matrix[u][v] = weight
        self.adj_matrix[v][u] = weight
    else:
        print("Invalid vertex")
```

Teacher's Signature.......................

④ def remove_edge() method to remove edge from the graph.

```
def remove_edge (self, u, v):
    if 0<=u<self.vertex_count and 0<=v<
            self.vertex_count:
        self.adj_matrix[u][v] = 0
        self.adj_matrix[v][u] = 0
    else:
        print("Invalid vertex")
```

⑤ In class graph, def has_edge() method to check whether two given vertices are connected by an edge or not.

```
def has_edge(self, u, v):
    if 0<=u<self.vertex_count and
            0<=v<self.vertex_count:

        return self.adj_matrix[u][v] != 0

    else:
        print("Invalid vertex")
```

⑥ def print_adj_matrix() method to print AM.

```
def print_adj_matrix(self):
    for row_list in self.adj_matrix:
print(" ".join(map(str, row_list)))

g = Graph(5)
g.add_edge(0,1)
g.add_edge(1,2)
g.add_edge(1,3)
g.add_edge(2,3)
g.add_edge(3,4)

g.print_adj_matrix().
```

Join of Map Method used.

① Map

Map (str, [0, 1, 0, 0, 0])
str(0) → "0"
str(1) → "1"
str(0) → "0"
str(0) → "0"
str(0) → "0"

l = ["0", "1", "0", "0", "0"]

yaha jo v diya hoga
wo beech me aayega.
In this case space
hai to space
aaya.

② Join
" ".join(l) → "0 1 0 0 0"

Teacher's Signature........

Teacher's Signature........

Adjacency list Implementation of Graph

(1) Write a class Graph to implement list representation of a graph data structure

```
class Graph
    def __init__(self, vno):
    self.vertex_count = vno
    self.adj_list = {v:[] for v in range
                     (vno)}
```

(2) def __init__() method to initialise instance object variables, vertex_count and a dict adj_list where key is vertex number and value is a list of adjacent vertices.

(3) define add_edge() method to add an edge in the graph with given vertices and weight.

```
def add_edge(self, u, v, weight=1):
    if 0<=u<self.vertex_count and
       0<=v<self.vertex_count:
    self.adj_list[u].append((v,weight))
    self.adj_list[v].append((u, weight))
    else:
        print("Invalid")
```

(4) def remove_edge() —

```
def remove_edge(self, u, v):
    if 0<=u<self.vertex_count and
    0<=v<self.vertex_count:
    self.adj_list[u] = [(vertex, weight) for
    vertex, weight in self.adj_list[u] if
     vertex != v]
    self.adj_list[v] = [(vertex, weight) for
    vertex, weight in self.adj_list[v] if
     vertex != u].
    else:
        print("Invalid vertices").
```

(5) def has_edge().

```
def has_edge(self, u, v):
    if 0<=u<self.vertex_count and 0<=
       v<self.vertex_count:
    return any(vertex==v for vertex,
       x in self.adj_list[u])
    else:
        print("Invalid")
```

(6) define print_adj_list() method.

```
def print_adj_list(self):
    for vertex, x in self.adj_list.items():
        print("V", vertex, ":", x)
```

Teacher's Signature.....

```
g = Graph(5)
g.add_edge(0,1)
g.add_edge(1,2)
g.add_edge(1,3)
g.add_edge(2,4)
g.add_edge(3,4)
g.print_adj_list().
```

BFS of DFS

Traversing :-

(Matlb hum har ek node ko visit kar rhe hain)

There are two standard ways of traversing a graph.

1. BFS (Breadth First Search)
2. DFS (Depth First Search)

BFS

Traversing graph has only issue that graph may have cycle - You may revisit a node.

Teacher's Signature.....

⇒ To avoid processing a node more than once, we divide the vertices into two categories -

1. Visited
2. Not visited

⇒ A boolean visited array is used to mark the visited vertices.

BFS uses a queue data structure for traversal.

Traversing begin from a node called source node.

Logic for BFS (Pseudo Code)

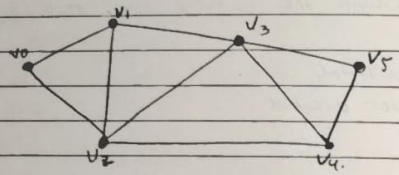
```
BFS(G, s)
    let q be the queue
    q.insert(s)
    v[s] = True
    while (!q.is_empty())
    {
      n = q.getFront()
      q.del()
      for all the neighbours u of n
        if v[u] == False
          q.insert(u)
          v[u] = True
```

Teacher's Signature.....

Page No. 5
Date / /

Record

* Levels:-



$L_0 - v_0$
$L_1 - v_1$
$L_1 - v_2$
$L_2 - v_3$
$L_2 - v_4$
$L_3 - v_5$

∴ nodes at a particular level is accessed at a time and the sequence continues, hence is called BFS.

DFS

(Depth First Search)

It uses Stack Data Structure.

Page No. 7
Date / /

Record

* Logic for DFS (Pseudo code)

```
DFS (G, s)
    Let stack be the stack
    stack.push(s)
    v[s] = True
    while (!stack.isEmpty())
    {
      n = stack.peek()
      stack.pop()
      for all the neighbours u of n
        if v[u] == False
          stack.push(u)
          v[u] = True.
```

* Levels

(For same graph)

$L_0 - v_0$
$L_1 - v_2$
$L_2 - v_4$
$L_3 - v_5$
$L_2 - v_3$
$L_1 - v_1$

∴ called DFS as it goes from level 0 to 1 to 2 to 3 then again to 2 and then 1.

Sorting

Arranging data elements in some logical order is known as sorting.

We are studying internal sorting,
- Some data aur sorting some memory space me data hai.

External sorting:- Some data secondary storage me ~~data~~ data hai, part of data ram me aata hai sort hota hai, fir baaki data aata hai.

When elements are nos., sorting means arranging in ascending order (by default).

When elements are strings, sorting means arranging strings in dictionary order (alphabetical) order. (by default).

Various sorting Algorithms

1. Bubble Sort
2. Modified Bubble Sort
3. Selection Sort
4. Insertion Sort
5. Quick sort
6. Merge Sort
7. Heap Sort.

Bubble Sort:-

0	1	2	3	4	5
24	58	11	67	92	43

(1) (0,1) (1,2) (2,3) (3,4) (4,5)
24 11 58 67 43 92

(2) (0,1) (1,2) (2,3) (3,4)
11 24 58 43 67 92

(3) (0,1) (1,2) (2,3)
11 24 43 58 67 92

(4) (0,1) (1,2)
11 24 43 58 67 92

(5) (0,1) → sorted
11 24 43 58 67 92

Modified Bubble Sort

n elements
Round - k

$0 < k < n$

no swapping in k^{th} round means elements are sorted now, no more rounds to be executed.

Selection Sort -

Jaise ek normal human sort krega. Mean ke hain ek series of data or elements diya hua hai, aur hmko sort krna hai, to hum dekhenge ki sabse chota kaun ka hai, usko nikal ke rkh denge, fir jo bacha usme se chota khojenge aur use age rkh denge.

Teacher's Signature.....

0	1	2	3	4	5	6	7	8
38	90	47	69	52	88	71	18	20
18	90	47	69	52	88	71	38	20
18	20	47	69	52	88	71	38	80
18	20	38	69	52	88	71	47	90
18	20	38	47	52	88	71	69	90
18	20	38	47	52	88	71	69	90
18	20	38	47	52	69	71	88	90

Insertion Sort -

0	1	2	3	4	5	6	7	8	9
50	20	37	91	64	18	43	59	72	81
20	50	37	91	64	18	43	59	72	81
20	37	50	91	64	18	43	59	72	81
20	37	50	91	64	18	43	59	72	81
20	37	50	64	91	18	43	59	72	81
18	20	37	50	64	91	43	59	72	81
18	20	37	43	50	64	91	59	72	81
18	20	37	43	50	59	64	91	72	81
18	20	37	43	50	59	64	72	91	81
18	20	37	43	50	59	64	72	81	91

Hme ye assume krke chlte hai ki pehla element already sorted hai.

Teacher's Signature.....

① Write a py function to implement bubble sort.

```
def bubble_sort(data_list):
    for x in range(1, len(data_list)):
        for i in range(len(data_list)-x):
            if data_list[i] > data_list[i+1]:
                data_list[i], data_list[i+1] =
                data_list[i+1], data_list[i]

l = [34, 67, 12, 89, 25, 50]
bubble_sort(l)
print(l)
```

② Write a py function to implement modified bubble sort.

```
def modified_bubble_sort(data_list):
    flag = False
    for x in range(1, len(data_list)):
        flag = False
        for i in range(len(data_list)-x):
            if data_list[i] > data_list[i+1]:
                data_list[i], data_list[i+1] =
                data_list[i+1], data_list[i]
                flag = True
        if not flag:
            break

m = [34, 67, 12, 89, 25, 50]
modified_bubble_sort(m)
print(m)
```

Teacher's Signature.....

③ Write a py function to implement selection sort.

```
def selection_sort(list1):
    n = len(list1)
    for i in range(n):
        min_index = i
        for j in range(i+1, n):
            if list1[j] < list1[min_index]:
                min_index = j
        list1[i], list1[min_index] = list1[min_index],
                                     list1[i]

mm = [34, 67, 12, 89, 25, 50]
selection_sort(mm)
print(mm)
```

④ Write a py function to implement insertion sort.

```
def insertion_sort(list1):
    for i in range(1, len(list1)):
        temp = list1[i]
        j = i-1
        while j>=0 and temp < list1[j]:
            list1[j+1] = list1[j]
            j -= 1
        list1[j+1] = temp

myList = [34, 67, 12, 89, 25, 50]
insertion_sort(0)
print(0)
```

Teacher's Signature.....

Quick Sort

Time

→ Sabse pehle ye dekhte hai ki total kitne index hain

→ Uske baad pehla jo no. hai like jo index 0 pe hai, usko uski position (matlb uski index position pe shift karna hai)

↳ For this we'll no. ko use jgh pe shift krenge ki uske left side wale saare nos. usse chote ho and right side wale saare bade.

→ Uske baad jo left and right me jo new list bna, usme individually same process follow krenge, to list sort ho jayega

⇒ Working Algo

let's we have, 0-9 index.

∴ quick (0, 9)
left = 0
Right = 9
loc = 0 → ye wo hai jis no. ko change krna hai, for eg. agar wo no. 0 se 4 index pe jayega to loc 4 ho jayega

Teacher's Signature.....

Pseudo code :-

```
while l[loc] < l[right]
        right -= 1
Swap
while l[left] < l[loc]
        left += 1
Swap.
```

(1) Write a python program to perform quick sort.

```
def quick-sort(list1):
    if len(list1) <= 1:
        return list1
    else:
        pivot = list1[0]
        lesser = [x for x in list1[1:] if x <= pivot]
        greater = [x for x in list1[1:] if x >= pivot]
        return quick-sort(lesser) + [pivot] + quick-sort(greater)

p = [42, 87, 15, 73, 56, 67, 11, 45, 78, 10]
p = quick-sort(p)
print("quick-sort :-", p)
```

Teacher's Signature.....

Merge Sort :-

Merge sort me pehle jo list q elements aai usko pehle two parts me divide krte hai.

Pehle middle index nikalte hai

$$= \frac{\text{Lowest index} + \text{highest index}}{2}$$

Uske around jo different list ban jayega. Frr us list me v same process krenge and usko sort krenge.

Uske baad saare sorted list ko merge kar denge.

To jo list aayega wo sorted hoga.

Q Write a python program to perform merge sort.

```
def merge_sort(list1):
    if len(list1) > 1:
        mid = len(list1)//2
        leftlist = list1[:mid]
        rightlist = list1[mid:]

        merge_sort(leftlist)
        merge_sort(rightlist)
```

Teacher's Signature.....

```
        i = j = k = 0
        while i < len(leftlist) and j < len(rightlist):
            if leftlist[i] < rightlist[j]:
                list1[k] = leftlist[i]
                i += 1
            else:
                list1[k] = rightlist[j]
                j += 1
            k += 1
        while i < len(leftlist):
            list1[k] = leftlist[i]
            i += 1
            k += 1
        while j < len(rightlist):
            list1[k] = rightlist[j]
            j += 1
            k += 1

mylist = [75, 25, 34, 42, 89, 12, 97]
merge_sort(mylist)
print(mylist)
```

Teacher's Signature.....

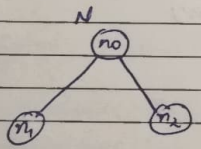
Heap :-

- Heap is a data structure.
- Used in sorting algorithm known as Heap Sort.
- Heap are of two types -
  - → Max heap (default)
  - → Min Heap

Properties -

① The value at node N is greater than or equal to value at each childrens of Node N

② Heap must be an almost complete Binary tree

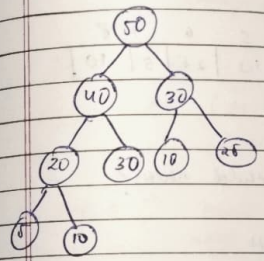


$n_0 \geq n_1$ and $n_0 \geq n_2$

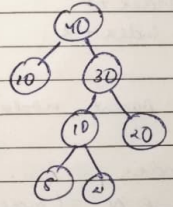
- This heap is max heap.

For min heap → $n_0 \leq n_1$ & $n_0 \leq n_2$

Examples -



Max - heap

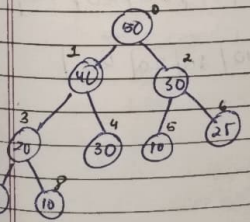


Not a heap

Reason :- Not almost complete Binary Tree

Representation of Heap.

Unless otherwise stated, heap is maintained in memory by a linear array (list)



(Heap)

0	1	2	3	4	5	6	7	8
50	40	30	20	30	10	25	5	10

How to find parent or child node?

0	1	2	3	4	5	6	7	8
50	40	30	20	30	10	25	5	10

→ How to find index of child node?

index is index of node N.

index of left child = 2 * index + 1
index of right child = 2 * index + 2

→ How to find index of parent node?

index of node N = index
index of parent node of N = $\frac{(index - 1)}{2}$

Insertion

40	70	10	90	60	30	50	20	80

	0	1	2	3	4	5	6	7	8
H	90	80	50	70	10	10	30	20	40

Step 1 → Ek element (pehla element ko bich me retrayenge and index 0 pe rakhenge.)

Step 2 → next element ko append karte jayenge heap me (to keep it as a almost complete Binary tree).

Step 3 → insert karne ke baad dekhenge ki jo heap bna uska uska parent node ka value bada hai chota.
→ Bada hai to theek.
→ Nrna chota hai to Swap.

Deletion.

→ Heap me deletion hamesha top most element ka hota hai

→ Uske liye pehle jo element ko delete krna hai usko ek variable me store kiye
→ Jo we delete consider hoga
→ Uske baad jo sbse niche element hai usko ek alware temp variable me rkh denge.
→ Uske baad upar se compare krna start krenge. and jahan needed swapping krenge
→ And hence we delete ho jayega
→ Fir further deletion ke liye same.

Teacher's Signature.....

Heap Sort :-

Delete values from the heap (max-heap) and store them in an array from right to left. As a result, at the end of deleting all the elements of heap, array becomes sorted.

Heap can be used to implement Priority Queue

(i) Def a class Heap to implement Heap data structure with --init-- method to create empty heap list.

```
class Heap :
    def __init__(self):
        self.heap = [ ]
```

(2) In class Heap, def a method to create heap from a given list of elements.

```
def createHeap(self, list1):
    for e in list1:
        self.insert(e)
```


(3) def a function insert to insert given element in the heap at appropriate position.

```
def insert(self, e):
    index = len(self.heap)
    parentIndex = (index - 1)//2
    while index > 0 and self.heap[parentIndex] < e:
        if index == len(self.heap):
            self.heap.append(self.heap[parentIndex])
        else:
            self.heap[index] = self.heap[parentIndex]
        index = parentIndex
        parentIndex = (index-1)//2
    if index == len(self.heap):
        self.heap.append(e)
    else:
        self.heap[index] = e
```

(4) define a top method, which means to return top element of the heap. Raise an exception if Heap is empty.

```
def top(self):
    if len(self.heap) == 0:
        raise EmptyHeapException()
    return self.heap[0]

class EmptyHeapException(Exception):
    def __init__(self, msg="Empty Heap"):
        self.msg = msg
```

```
def __str__(self):
    return self.msg
```

④ define a class EmptyHeap exception to describe custom exception
(solved in ⑦)

⑤ Def a method delete.

```
def delete(self):
    if len(self.heap) == 0:
        raise EmptyHeapException()
    if len(self.heap) == 1:
        return self.heap.pop()
    max_value = self.heap[0]
    temp = self.heap.pop()
    index = 0
    leftChildIndex = 2*index + 1
    rightChildIndex = 2*index + 2

    while leftChildIndex < len(self.heap)
        if rightChildIndex < len(self.heap):
            if self.heap[leftChildIndex] >
                self.heap[rightChildIndex]:
                if self.heap[leftChildIndex]
                        > temp:
                    self.heap[index] = self.
                        heap[leftChildIndex]
                    index = leftChildIndex
                else:
```

Teacher's Signature.....

```
                else:
                    if self.heap[rightChildIndex] > temp:
                        self.heap[index] = self.heap[right
                                ChildIndex]
                        index = rightChildIndex
                    else:
                        break
            else: # No right child
                if self.heap[leftChildIndex] > temp:
                    self.heap[index] = self.heap[leftChildIndex]
                    index = leftChildIndex
                else:
                    break
        leftChildIndex = 2*index + 1
        rightChildIndex = 2*index + 2
    self.heap[index] = temp
    return max_value.
```

⑥ In class Heap, def heapSort to sort given list.

```
def heapSort(self, list):
    self.createHeap(list)
    try:
        def heapSort(self, list=1):
            self.createHeap(list1)
            list2 = []
            try:
                while True:
                    list2.insert(0, self.delete())
            except EmptyHeapException:
                pass
            return list2
```

list 1 = [34, 56, 12, 78, 54, 16, 32, 96, 89]

h = Heap()

list 1 = h.heapsort(list 1)

print(list 1)

Searching

Linear Search:

	0	1	2	3	4	5	6	7	8
list 1 →	34	41	29	62	48	35	96	11	25

To search → 11

Algo :-

```
for e in list 1:
    if e == item
        return list.index(e)
return None
```

Time complexity = O(n) {Worst case}

Teacher's Signature.....

Binary Search.

	0	1	2	3	4	5	6	7	8
list 1 →	34	41	29	62	48	35	96	11	25

To search → 11

Step 1 → To sort the list

Step 2 → We will find the middle element of the sorted list.

using $m = \frac{l+r}{2}$

then, we will use

Algo :-

```
if item == list1[m]:
    return m
if item < list1[m]
    binarySearch(list1, 0, m-1)
else:
    binary search(list1, r, 9)
```

recursion used

Time Complexity = $O(\log_2 n)$

Teacher's Signature.....

Hashing

Why Hashing?

Hashing is designed to solve the problem of needing to efficiently find or store an item in a collection.

If we have a list of 1,00,000 words of English and we want to search a given word in the list, it would be inefficient to successively compare the word with all 1,00,000 words until we find a match.

Hashing

It is a technique of mapping keys and values into the hash table by using a hash function.

If implemented correctly, then one can achieve O(1) time complexity in searching an item in the collection of elements.

Teacher's Signature.....

- Efficiency of mapping depends on the efficiency of the hash function used.

Key terms-

1. Hash Table: It is the data structure used to store elements.
2. Hash function: It is a function to map (hash) key-values to memory address.
3. Hashing: It is a method for storing and retrieving records from a database.

Example

Suppose 100 students records need to be stored in an array of 100 memory slots. We hashing to perform efficient access of student data.

Now, let a basic standard as

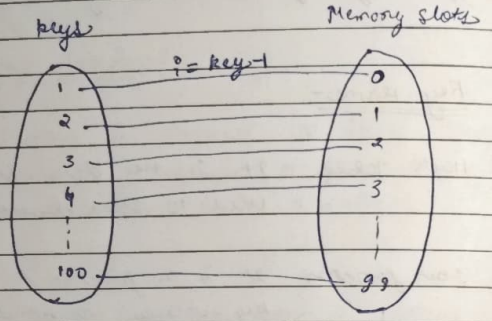
Roll No → 1 to 100
(Key)

i = HF(Key)

```
def HF(key):
    return key-1;
```

Teacher's Signature.....

(Roll Nos).



O Har alg hash function ke liye hum khud se koi v logic develop kar skte hain.

→ Hashing is a method of storing and retrieving records from data structure

→ It lets you insert, delete and search records based on search key value

→ When properly implemented these operations can be performed in constant time.

→ This is far better than O(log n) average cost required to do a binary search on a sorted array

Teacher's Signature.....

Teacher's Signature.....