Name: Prasad Regmi
Roll No: 221737
Program: BE Software
Shift: Day

Tutorial

1) What is data structure? Compare and constrast bet-n primitive and non-primitive data structure. Classify different data structures as linear and non-linear on the basis of logical organization and memory organization.

→ Data structure is a programming construct for representing or managing collection of data.

Difference:

	Primitive Data Structure	Non-Primitive Data Structure
i)	It stores the data of only one type.	It stores the data of more than one type.
ii)	It contains some value.	It can consist a NULL value.
iii)	It starts with a lowercase character.	It starts with a upper-case character.
iv)	It can be used to call the methods.	It cannot be used to call the methods.
v)	The size depends on the type of data structure.	The size is not fixed.
vi)	E.g.: integer, character, float, etc.	E.g.: Array, Linked List, Stack, etc.

Similarities:

i) Both are used to store and organize data in a computer system.

ii) Both allow manipulation and operations on the stored data.

iii) Both requires memory allocation for storage.

iv) Both are the types of data structure.

Classification:

i) Linear Data Structures

- Logical Organization: Linear data structures organize elements in a sequential manner where each element is connected to its previous and next elements.
- Memory Organization: In memory, Linear data structures allocate memory in a contiguous block.

E.g.: Array, Linked List, Stack, Queue

ii) Non-Linear Data Structures

- Logical Organization: Non-Linear data structures organize elements in a hierarchical manner where each element may connect to multiple elements.
- Memory Organization: In memory, non-Linear data structures allocate memory in a non-contiguous or scattered manner.

E.g.: Tree, Graph, Heap

2) Explain the concept of algorithm analysis, focusing on the Best, Worst, and Average Case Analysis. Describe how these analysis help in understanding the behavior and performance of algorithms. Provide examples to illustrate the difference bet^n best, worst, and average-case scenarios for algorithms.

→ Algorithm analysis is the process of evaluating the efficiency and performance of algorithms. The analysis typically focuses on 3 cases:

i) Best Case:
It refers to the scenario in which an algorithm performs optimally.
It takes the max amount of time to complete its execution or utilizes the max. amount of resources.

ii) Worst Case:
It refers to the scenario in which an algorithm performs the most work or uses most resources

iii) Average Case:
It considers the expected performance of an algorithm over all possible inputs.

These analysis helps in understanding the behavior and performance of algorithms in several ways:

i) Efficiency Evaluation:
Algorithm analysis allows us to evaluate the efficiency of algorithms in terms of time

complexity, space complexity, etc.

ii) Comparative Analysis:
Algorithm analysis enables the comparison of different algorithms solving the same problem.

iii) Performance Prediction:
Algorithm analysis helps predict the performance of algorithms for different inputs sizes.

iv) Algorithm Selection:
Algorithm analysis helps in selecting the most appropriate algorithm based on the specific requirements.

v) Understanding Limitations:
Algorithm analysis helps in understanding the limitations of algorithms, including scenarios where certain algorithms may perform poorly or become impractical.

Example 1:
Consider the sorting algorithm "Bubble Sort"
- Best Case: The array is already sorted.
  It has complexity of $O(n)$.
- Worst Case: The array is sorted in reverse order.
  It has complexity of $O(n^2)$.
- Average Case: There is random distribution of elements in the input list.

~~The~~ ~~The~~ It has complexity of $O(n^2)$.

Example 2:
Consider the searching algorithm "Binary Search"
-Best Case: The element being searched is found at the middle of the array.
It has complexity of $O(1)$.
-Worst Case: The element being searched is not in the array or is located at beginning or end of the array.
It has complexity of $O(\log n)$.
-Average Case: It typically performs close to its worst case because ~~often~~ elements are often not found in the middle of the array.
It has complexity of $O(\log n)$.

3) Justify the statement "Data Structure is the backbone of software ~~engineering~~ programming". How can we represent a polynomial using array? Explain with pros and cons. Explain polynomial as an ADT.

→ The statement is justified by several reasons:
i) Organizing data efficiently:
Data structures provide a means to organize and store data efficiently.

ii) Efficient data retrieval and manipulation:
Data structures allow for efficient retrieval,

insertion, deletion, and ~~manipul~~ modification of data, which are fundamental operations in software programming.

iii) Algorithm Design and Optimization:
Many algorithms are tailored to work with specific data structures according to their capabilities to achieve optimal performance. for Eg., sorting algorithms work differently with arrays compared to linked lists or trees.

iv) Memory Management and Resource Utilization:
Data structures help in managing memory allocation and utilization effectively, minimizing overhead and maximizing the use of ~~arat~~ available resources.

v) Scalability and Performance:
Choosing the right data structures enables software applications to scale efficiently, and maintain acceptable performance levels under varying workloads.

We can represent a polynomial using array by:
Consider a polynomial with single variable,

$4x^4 - 2x + 5$

The array representation of the polynomial is,

0	1	2	3	4
5	-2	0	0	4

Pros:

i) Simplicity:
It is easy to understand and implement.

ii) Efficient Arithmetic Operations:
We can perform arithmetic operation by simply adding, multiplying, etc. corresponding coefficients.

Cons:

i) Fixed Size:
Requires pre-determining of size, which may lead to wasted space.

ii) Inefficiency for Sparse Polynomials:
Not ideal for polynomials with many zero coefficients, as it still allocates space for every term.

An ADT polynomial is the one that shows various operations that can be performed on ~~poly~~ polynomials.

Polynomial as an ADT is given as:

```
class Polynomial{
        int coeff, exponent;
        public:
            Boolean IsZero(poly){ }
            Coefficient Coeff(poly, exponent){ }
            Exponent LeadExp(poly){ }
            Polynomial Remove(poly, exponent){ }
```

```
Polynomial Add (poly1, poly2) { }
Polynomial Mult (poly1, poly2) { }
Polynomial Zero () { }
};
```

4) Explain the basic operations of stack. Write an algorithm for insertion and deletion of an element in a stack using array. Also mention the condition for overflow and underflow.

→ A stack is a linear data structure that follows LIFO (Last In First Out).

The basic operations of stack are:

i) push(): Insert an element into top of the stack
ii) pop(): Remove the element from top of the stack
iii) top(): Returns the element at top of the stack without removing it.
iv) isEmpty(): Returns true if stack is empty else false
v) isFull(): Returns true if stack is full else false
vi) size(): Returns the number of elements in the stack.

Algorithms for insertion of an element in a stack using array:

~~1) Check if the stack is full~~
~~2) Chec t~~

i) If (tos = maxSize - 1) then
   a) Display "Stack is full" / Overflow Condition"
   b) Return

ii) Else increment tos pointer (i.e. (tos ← tos + 1)
iii) Insert new element at top of stack (i.e. S[tos] ← item)
iv) Return

Algorithm for deletion of an element in a stack using array:
i) If (tos = -1) then
   a) Display "Stack is empty"/Underflow Condition"
   b) Return
ii) Else retrieve the element at top of stack (i.e. item ← S[tos])
iii) Decrement the tos pointer (i.e. tos ← tos - 1)
iv) Return item

5) Define infix and postfix expression. In an application, the client request you to process their data in a particular format. The format states that the in the data count of 1000, first data entered in the application is the first that comes out if in the application. Write down the insert and delete mechanism for manipulation of data.

→ Infix expression is a type of expression where the operators are placed between operands.
Postfix expression is a type of expression where the operators are placed after the operands.

The insert and delete mechanism ~~for to insert and d~~ for manipulation of data is given as:

```
#include <stdio.h>
#include <stdbool.h>
#define max 1000
int queue [max];
int rear = -1;
int front = -1;
void enqueue (int data)
{
   if (rear==max-1)
   {
      printf("Queue is full \n");
      return;
   }
   else if(front==-1)
   {
      front=0;
   }
   else
   {
      queue[++rear]=data;
      printf("%d added to queue \n", data);
   }
}
void dequeue()
{
   if(front==-1 || front>rear)
   {
      printf("Queue is empty \n");
      return;
   }
```

```
else if (rear == front)
{
    rear = -1;
    front = -1;
    printf("%d removed from queue \n", queue[front]);
}
else
{
    printf("%d removed from queue \n", queue[front]);
    front++;
}
}
void traverse()
{
  if (front == -1 || front > rear)
  {
    printf("Queue is empty \n");
  }
  else
  {
    for (i= front; i<= rear; i++)
    {
      printf("Queue [%d] = %d ", i, queue[i]);
    }
    printf("\n");
  }
}
int main()
{
```

```
int choice, data;
bool Loop = true;
while (Loop)
{
    printf("1. Enqueue \n 2. Dequeue \n 3. Display \n
            4. Exit \n Enter choice: ");
    scanf("%d", &choice);
    switch (choice)
    {
        case 1:
                printf("Enter data:");
                scanf("%d", &data);
                enqueue (data);
                break;
        case 2:
                dequeue();
                break;
        case 3:
                printf("Current queue: \n");
                traverse();
                break;
        case 4:
                Loop = false;
                break;
        default:
                printf("Invalid Choice");
    }
}
printf("Thank You \n");
return 0;
}
```

6) Convert the given expression into postfix expression showing the content of stack at each step:
(A+B) * (C$D) - E/F

Scanned Character	Stack	Postfix Expression
→ C	C	Empty
A	C	A
+	C+	A
B	C+	AB
)	Empty	AB+
*	*	AB+
(	*(	AB+
C	*(	AB+C
$	*($	AB+C
D	*($	AB+CD
)	*	AB+CD$
-	-	AB+CD$*
E	-	AB+CD$*E
/	-/	AB+CD$*E
F	-/	AB+CD$*EF
Empty	Empty	AB+CD$*EF/-

∴ Converted Postfix Expression = AB+CD$*EF/-

8) Explain the application area of queue. What will happen if front is equal to rear in Linear queue? Demonstrate the concept of enqueue, dequeue and traverse operation with suitable example and supporting algorithm.

→ The application area of queue are:

i) Operating Systems:
Queues are used in OS for managing various tasks such as CPU scheduling, handling interrupts, etc.

ii) Networking:
Queues are used in networking protocols for managing packets.

iii) Print Spooling:
Queues are used in print spooling systems where multiple print jobs are queued up and printed in the order they were received.

iv) Synchronization:
Queues are used for synchronization between multiple processes or threads to ensure that resources are accessed in an orderly manner.

v) Breadth First Search (BFS):
Queues are used in graph traversal algorithms like BFS, where nodes are visited in a level by level manner.

If the front is equal to rear in a linear queue then;
i) Case 1: front = rear = -1
Queue is empty

ii) Case 2: front = rear = 0
Only one element is present in the queue.

iii) Case 3: front = rear = maxSize - 1
Queue is full due to the ~~sing~~ element present in the ~~tas~~ Last index.

9) What is a benefit of circular queue when compared to linear queue? Write enqueue, dequeue and traverse algorithm of circular queue.

→ Benefits of circular queue when compared to linear queue are:

i) Flexible insertion and deletion:
In a circular queue, elements can be added till the queue ~~don't~~ isn't full. But in the linear queue, you can not add more elements once the rear end points to the last index.

ii) Simplicity:
In the linear queue, the element inserted first is the element to be deleted first. That is not the case with the circular queue because the front and rear are not fixed. That allows us to modify the order of insertion and deletion which is extremely helpful.

iii) Memory Efficiency:
Circular queue is more efficient than a linear queue because we can add elements until it's completely filled. ~~Thros~~ While in a linear queue, the vacant spaces after dequeue operations can never be filled.

11) Compare contiguous list with Linked List. Can you implement stack using Linked List? Illustrate.

→

	Contiguous List	Linked List
i)	It stores elements in adjacent memory locations.	Elements are stored in nodes, where each node contains data and a reference to the next node.
ii)	It has fixed size.	It can dynamically grow and shrink.
iii)	Memory allocation for contiguous list is static.	Memory allocation for Linked List is dynamic.
iv)	Insertion and deletion can be inefficient	Insertion and deletion are efficient.
v)	It offers efficient random access to elements.	It does not support direct random access to elements. (Traversal from head or tail is necessary.

Yes, we can implement stack using Linked List by:

In case of SLL

T1: Insert at first node and delete at first node

T2: Insert at Last node and delete at Last node

```
#include <stdio.h>
#include <stdLib.h>
struct singlyLinear
{
    int info;
    struct singlyLinear *next;
};
typedef struct singlyLinear ~~Node;~~ SLL;
SLL *first;
void insertfirst (int value)
{
    ~~SLL *temp;~~
    ~~temp = (SLL*) malloc (sizeof(S~~
    SLL *temp;
    temp = (SLL*) malloc (sizeof (SLL));
    temp -> info = value;
    temp -> next = NULL;
    if (first == NULL)
    {
        first = temp;
        printf ("%d inserted \n", temp -> info);
    }
    else
    {
        temp -> next = first;
        first = temp;
        printf ("%d inserted \n", temp -> info);
    }
}
```

```
void deleteFirst()
{
    if (first == NULL)
    {
        printf("Stack is empty");
        return;
    }
    temp SLL
    SLL *temp;
    temp = first;
    first = first->next;
    free(temp);
}
int main()
{
    insertFirst(10);
    insertFirst(20);
    deleteFirst();
    return 0;
}
```

12) Explain the advantages of doubly linked list over singly Linked list? Write an algorithm to insert a node of the Linked list (all possible cases).

→ The advantages of DLL over SLL are:

i) Bidirectional Traversal

ii) Insertion and Deletion Operations

iii) Backward Iteration (Reversing DLL)

iv) Implementation of Data Structures
v) Flexibility and Functionality

13) What are benefits of using dynamic list over static list? Write an algorithm to insert an item at the specified position of a static list.

→ Benefits of using dynamic list over static list are:

i) Dynamic Size:
Dynamic Lists can grow or shrink in size dynamically as elements are added or removed.

ii) Memory Efficiency:
Dynamic Lists allocate memory as needed, which can lead to better memory utilization.

iii) Insertion and Deletion:
Dynamic Lists typically supports efficient ~~in~~ insertion and deletion operations at arbitrary positions. In constrast, static lists may require shifting elements.

iv) Flexibility:
Dynamic lists can adapt to changing requirements and data sizes without the need for manual adjustments to the list size.

Algorithm to insert an item at the specified position of a static list:

i) Check if the position is valid (i.e., within the range (0, size of list - 1))
ii) If position is not valid, then display error message and return.
iii) If position is valid, check if the list is full (i.e., sizeoflist = max capacity)
iv) If List is full, print error message and return
v) If List is not full, then
   a) Shift elements to right starting from last element upto the position of insertion
   b) Insert element at specified position
vi) Exit

14) Differentiate between singly circular linked list and singly linked list. Implement queue using linked list.

→

	Singly Linked List	Singly Circular Linked List
i)	Each node contains a data element and a reference (or pointer) to the next node.	It is similar to SLL, but last node's pointer points back to the first node (head).
ii)	The last node's pointer points to ~~the~~ NULL.	The last node's pointer does not point to NULL
iii)	Traversal starts from head.	Traversal starts from any node.

iv)	Operations like insertion and deletion can be performed by adjusting pointers accordingly.	Operations are similar to SLL but with adjustments to ensure circular structure is maintained.
v)	It is used in applications where continuous looping or cyclic behavior is not required.	It is used in applications where continuous looping or cyclic behavior is required.

16) How polynomial can be represented using Linked List? Illustrate with example. ~~Write~~

→ Polynomials can be efficiently represented using Linked Lists. In this representation, each term of the polynomial is represented as a node in the Linked List.

E.g.: Consider a polynomial $3x^3 + 2x^2 - 5$ ~~x + 7~~

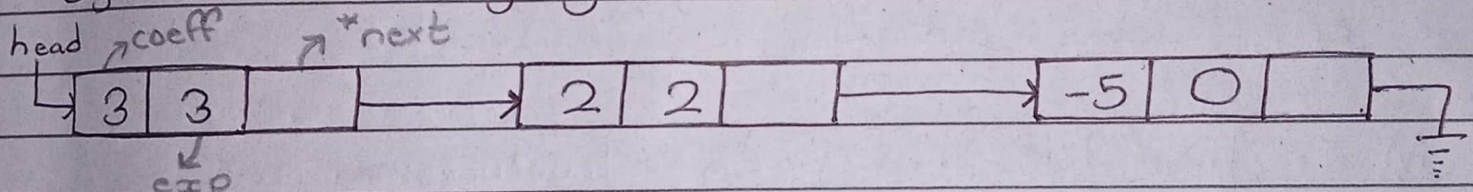


Fig: Representation of polynomial using Linked List

Illustration:

- Each node in the Linked List represents a term of the polynomial.
- Each node contains 3 parts: coefficient, exponent, pointer to the next node.
- The Linked List ~~can~~ is traversed to evaluate the polynomial at a given value of x or to perform arithmetic operations.

17) What are the advantages and disadvantages of Linked List?

→ Advantages:

i) Dynamic Size:
Linked Lists can dynamically grow and shrink in size during program execution, allowing for efficient memory usage.

ii) Insertion and Deletion Operations:
These operations are quite easier in the Linked List as there is no need to shift elements.

iii) No ~~me~~ Memory Wastage:
Linked Lists only use the memory necessary for storing the elements and pointers, eliminating wasted memory due to empty or unused slots.

iv) Implementation of Data Structures:
Linear data structures like stack and queues are often easily implemented using a Linked List.

v) Flexible:
It is flexible because elements in Linked List are not stored in contiguous memory locations.

Disadvantages:

i) Memory Usage:
More memory is regured in the Linked list due to the use of pointer.

ii) Random Access:
Random access is not possible in a Linked list due to its dynamic memory allocation.

iii) Traversal:
Linked List traversal is time consuming as direct access to an element is not possible.

iv) Lower efficiency at times:
For certain operations such as searching, can be slower in Linked List.

v) Complex Implementation:
Linked ~~imp~~ List implementation is complex as compared to other.

10) Differentiate between SCLL and DCLL.

→

SCLL	DCLL
i) SCLL nodes have one pointer pointing to the next node.	DCLL nodes have two pointers, one pointing to next node and the other pointing to the previous node.

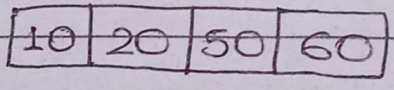
ii)	SCLL supports forward traversal only.	DCLL supports both forward and backward traversal.
iii)	SCLL has lower memory overhead comparatively.	DCLL has higher memory overhead comparatively.
iv)	SCLL has limited functionality comparitively.	DCLL offers more functionality due to its ability to traverse in both directions.

21) Create the B tree of order 5 from the following set of data.

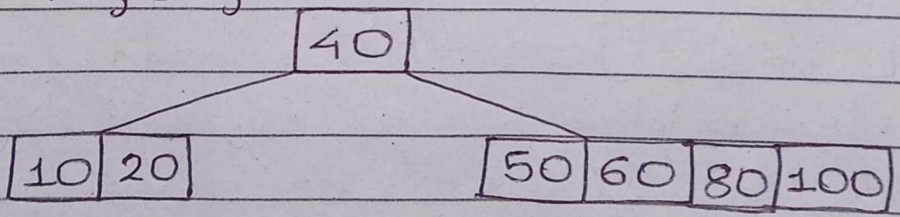
10, 20, 50, 60, 40, 80, 100, 70, 130, 90, 30, 120, 140, 25, 35, 160, 180

→ Solution,

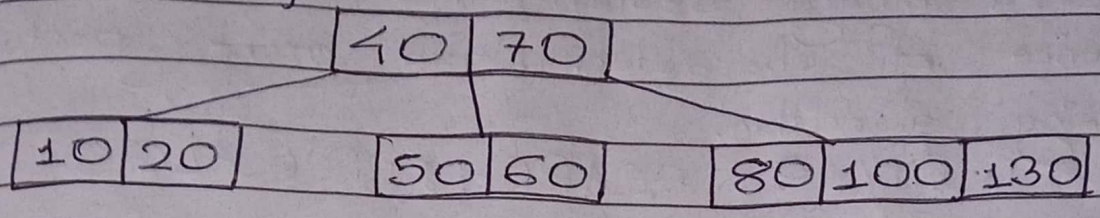
Insert 10, 20, 50, 60



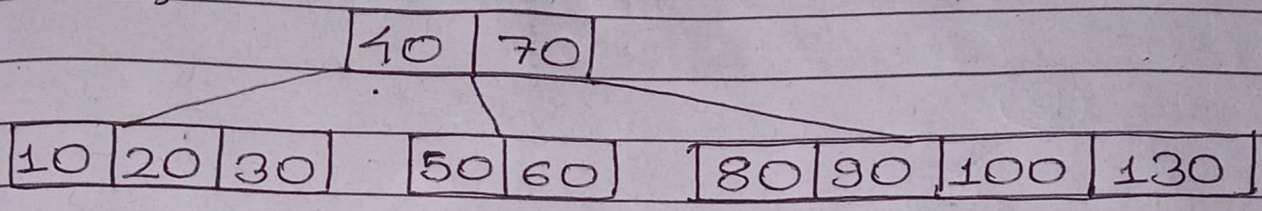
Insert 40, 80, 100



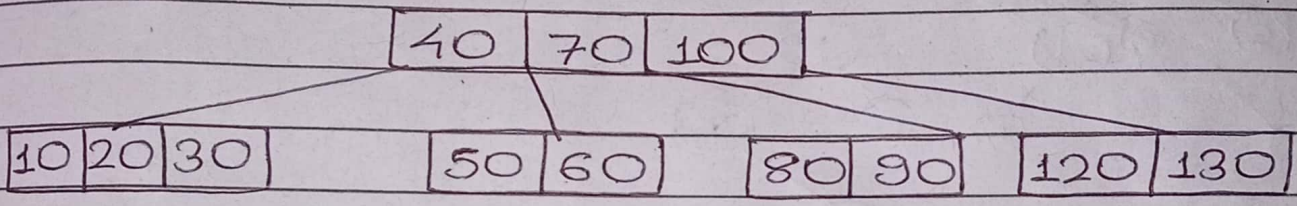
Insert ~~100~~ 70, 130



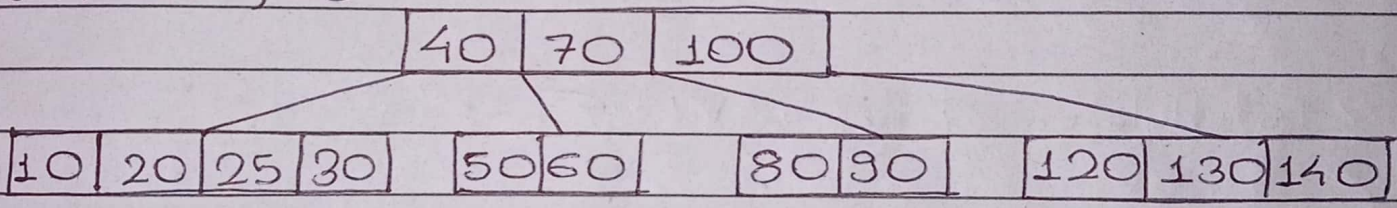
Insert 90, 30.



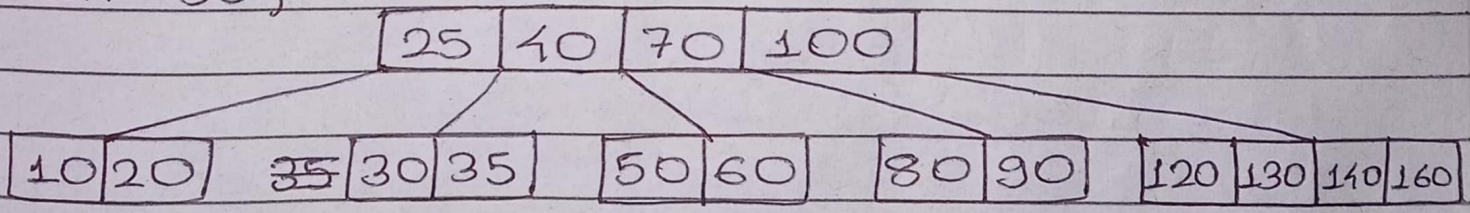
Insert 120



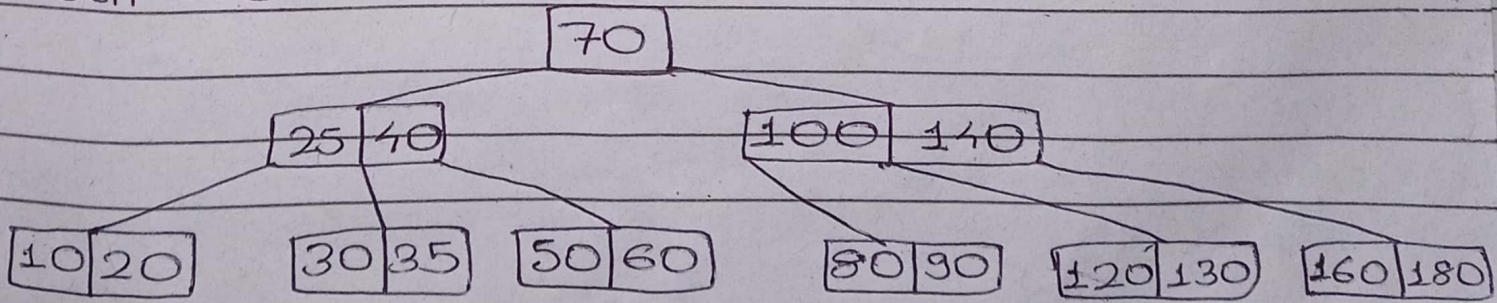
Insert 140, 25



Insert 35, 160



Insert 180

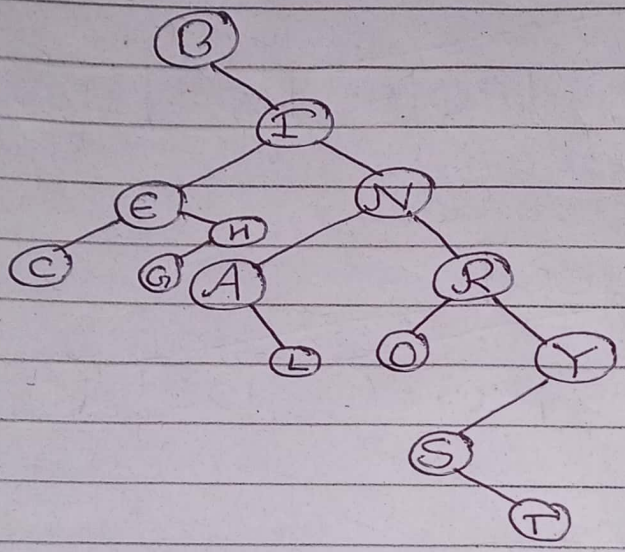


22) Create a binary search tree from the following sequence of character. Also ~~probt~~ perform the balancing algorithm.

BINARY SEARCH TREE ALGO

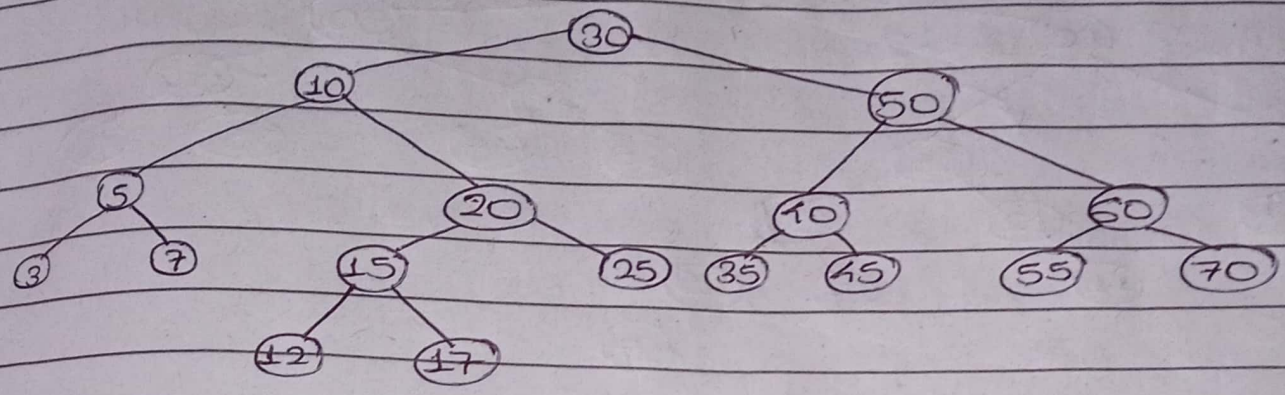
→ Solution,

BST is ~~created be~~ created below:



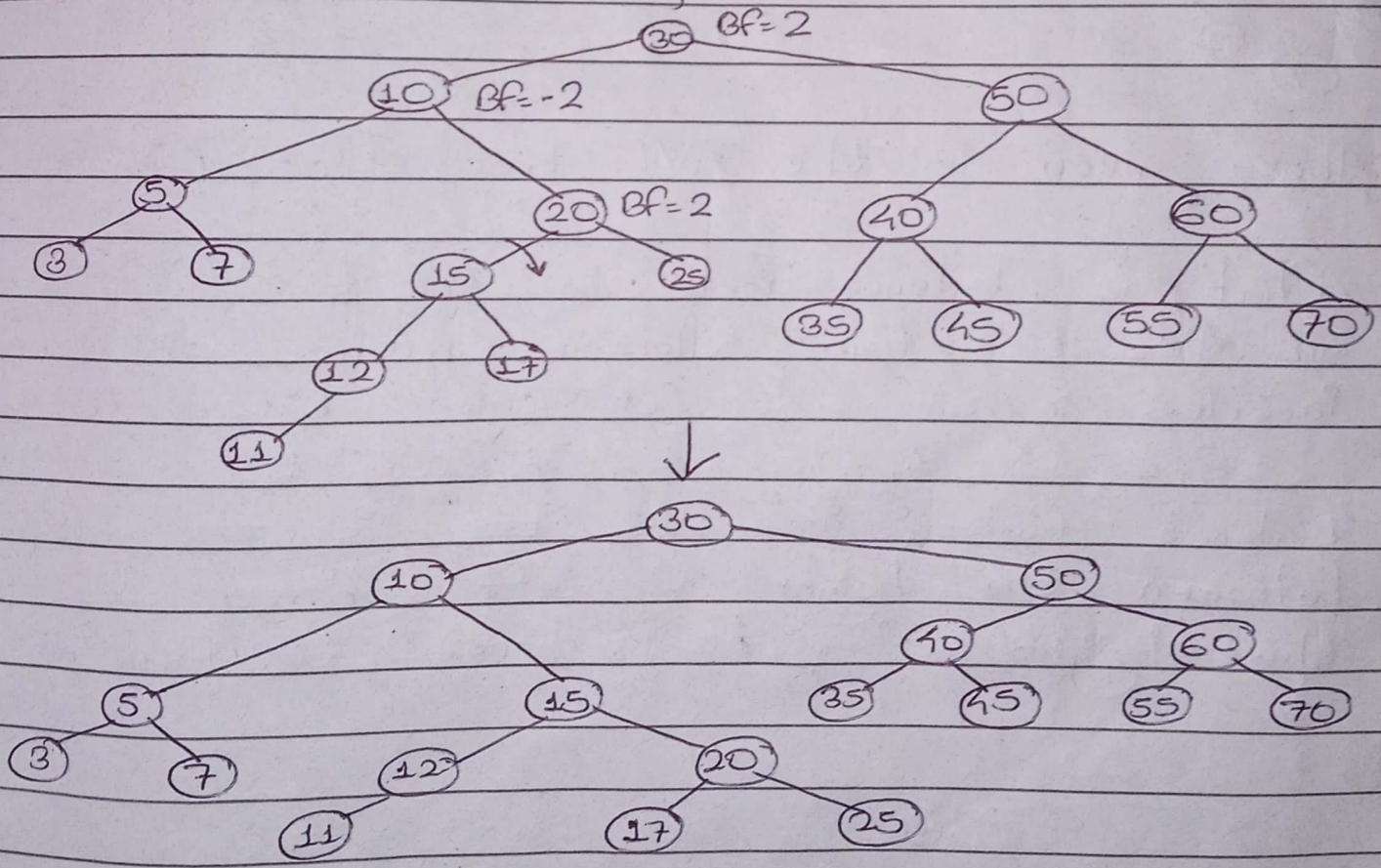
To balance (Using AVL Tree),

23) Show the AVL tree that results when 11 is inserted.



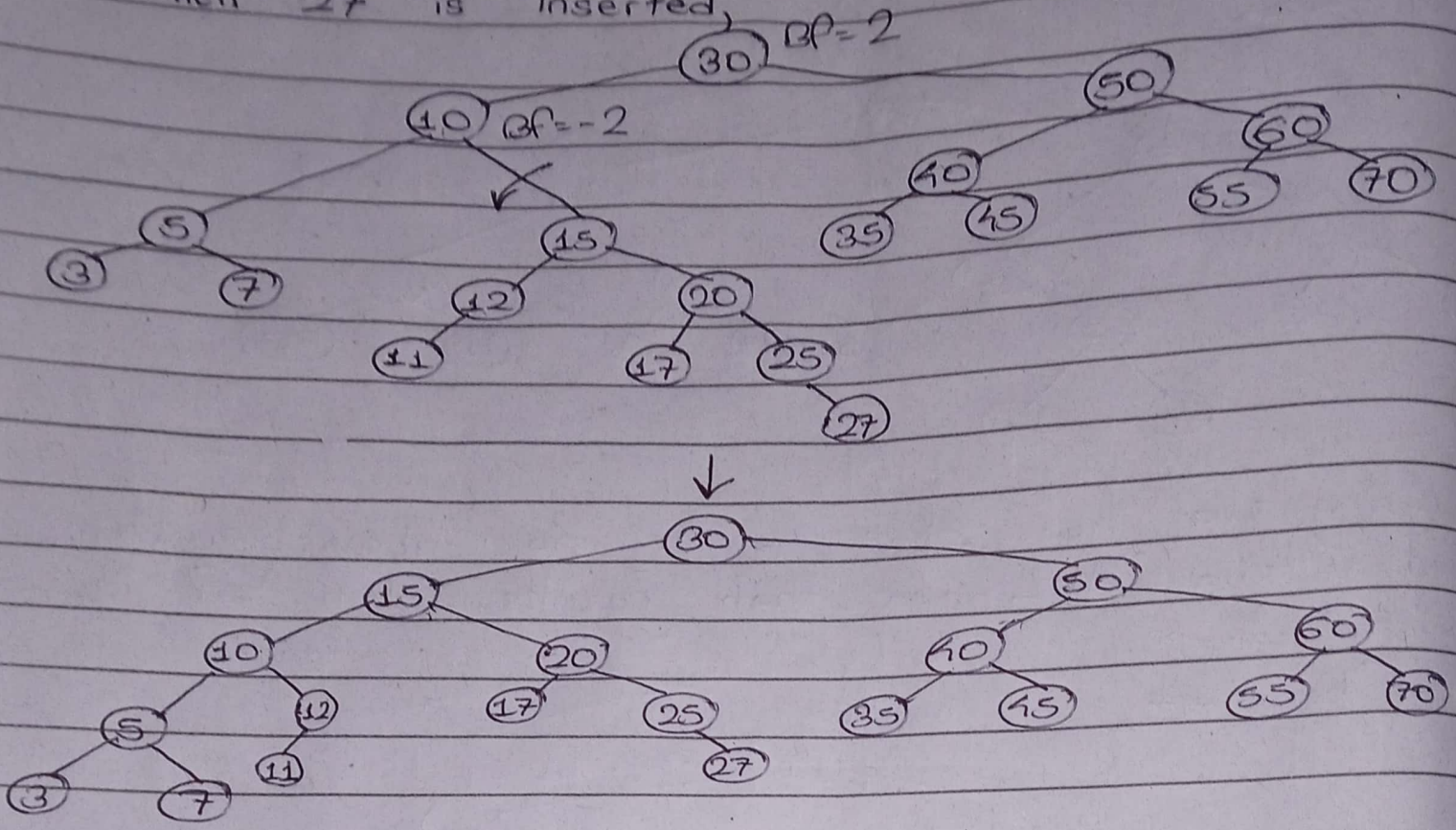
Using the tree that results from above problem, show the AVL tree that results when 27 is inserted.

→ When 11 is inserted,



∴ Above given is the AVL tree after 11 is inserted.

When 27 is inserted,



∴ Above given is the AVL tree after 27 is inserted.

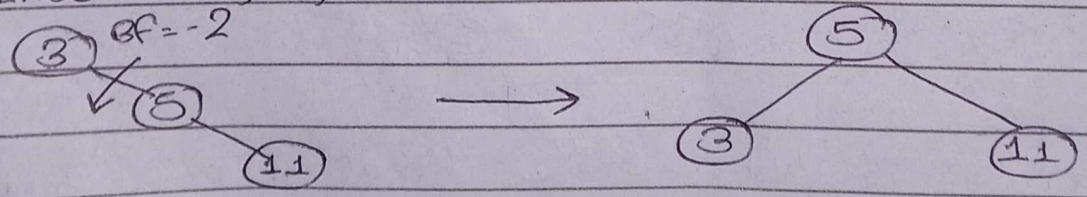
27) What is balance factor in AVL tree? Construct an AVL tree from following sequence of number inserted in an order of empty binary tree.

3, 5, 11, 8, 4, 1, 12, 7, 2, 6, 10

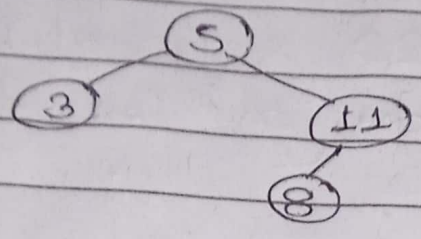
→ Balance factor of a node in AVL tree is the difference between the height of left subtree and the height of right subtree of that node.

AVL tree is constructed as:

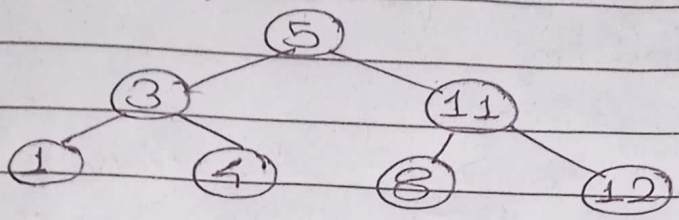
Insert 3, 5, 11



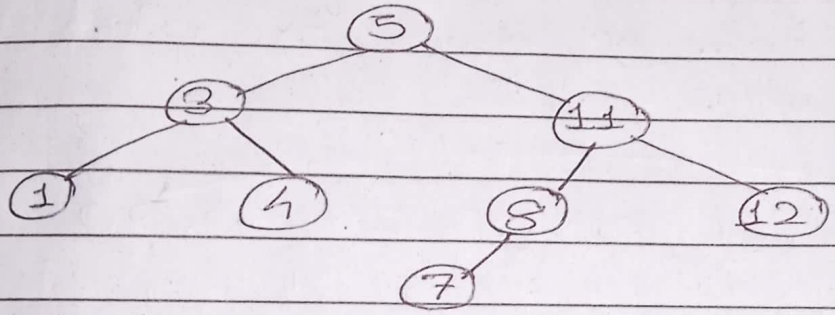
Insert 8



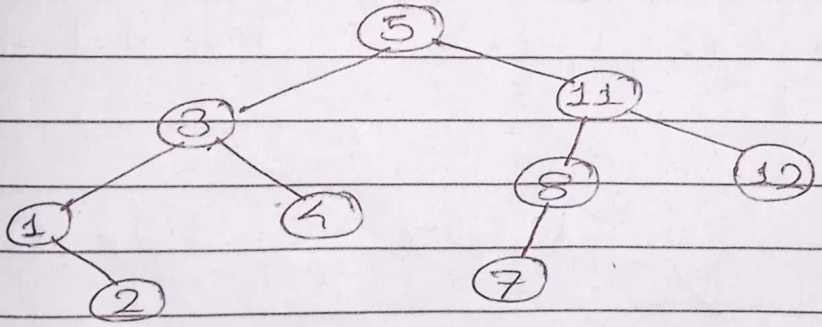
Insert 4, 1, 12



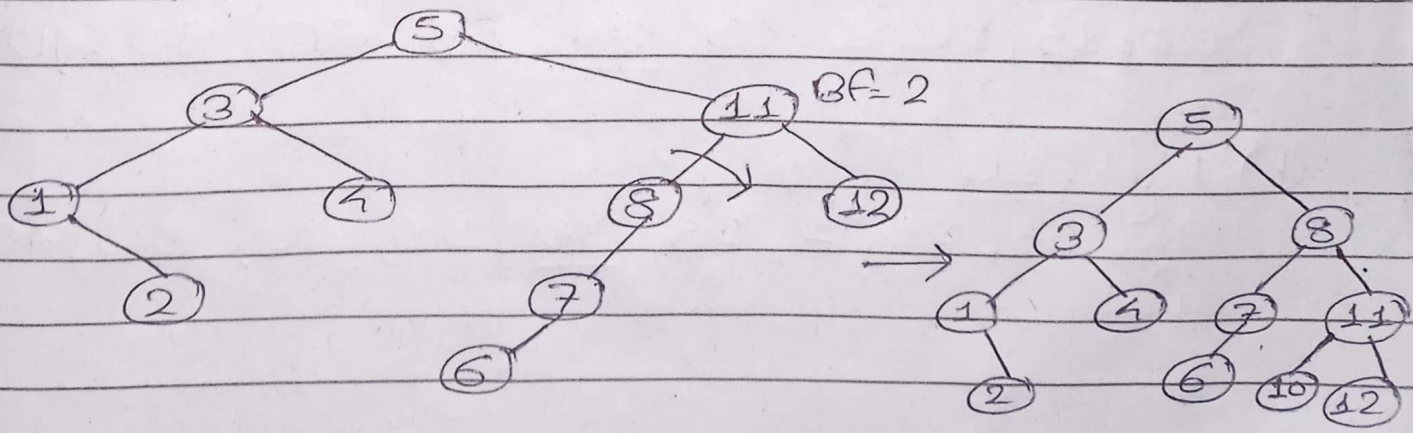
Insert 7



Insert 2



Insert 6, 10



∴ Above given is the required AVL tree.

29) Use the quicksort algorithm to sort 36, 15, 40, 1, 60, 20, 55, 25, 50, and 20 is a stable sorting algorithm? Also explain the partition algorithm.

→ Solution,

Given,

36, 15, 40, 1, 60, 20, 55, 25, 50, 20

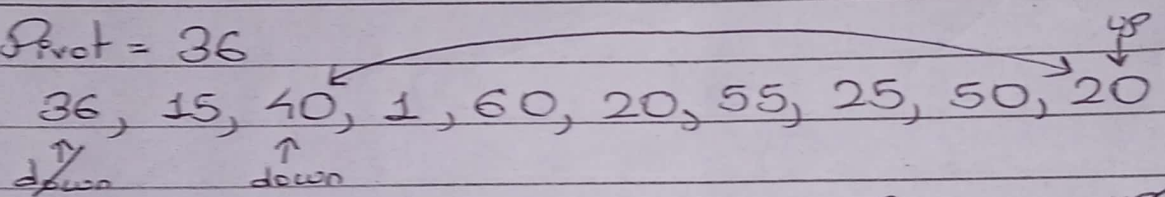
Using quicksort algorithm,

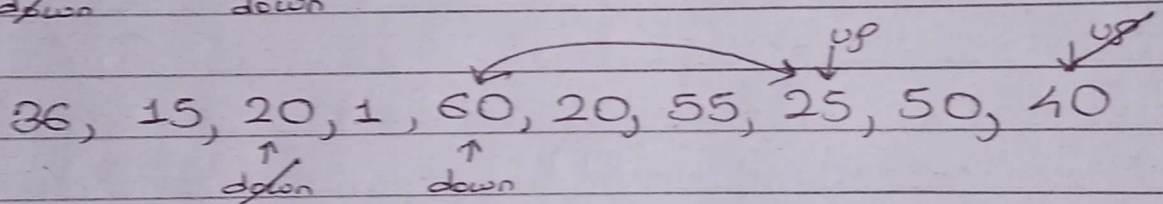
We use two pointers:

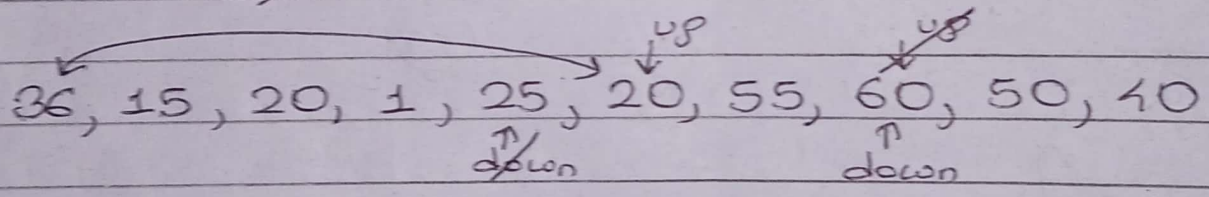
i) down: It moves to right direction unless it gets value greater than pivot

ii) up: It moves to left direction unless it gets value less than or equal to pivot

(a) Pivot = 36

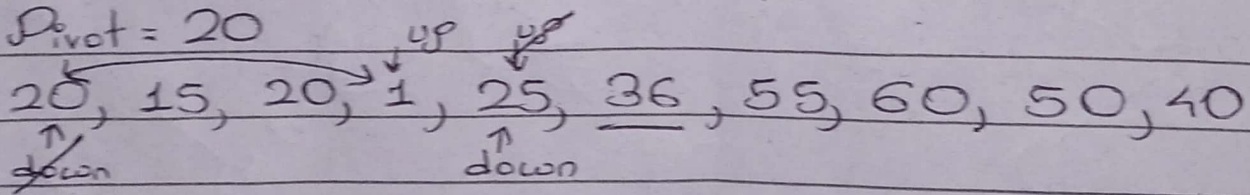






20, 15, 20, 1, 25, 36, 55, 60, 50, 40

(b) Pivot = 20



1, ~~15~~, 20, 20, ~~1~~ 25, 36, 55, 60, 50, 40

(c) Pivot = 1

1, 15, 20, 20, 25, 36, 55, 60, 50, 40

1, 15, 20, 20, 25, 36, 55, 60, 50, 40

(d) Pivot = 15

1, 15, 20, 20, 25, 36, 55, 60, 50, 40

1, 15, 20, 20, 25, 36, 55, 60, 50, 40

(e) Pivot = 55

1, 15, 20, 20, 25, 36, 55, 60, 50, 40

1, 15, 20, 20, 25, 36, 55, 40, 50, 60

1, 15, 20, 20, 25, 36, 50, 40, 55, 60

(d) Pivot = 50

1, 15, 20, 20, 25, 36, 50, 40, 55, 60

1, 15, 20, 20, 25, 36, 40, 50, 55, 60

∴ Hence, the sorted array is:
1, 15, 20, 20, 25, 36, 40, 50, 55, 60

∴ Here, the sorting algorithm is not stable because the two occurances of 20 in the original list ~~is~~ have changed their relative order in the sorted list.

Partition Algorithm:
The ~~partion~~ partition algorithm is a fundamental ~~concept~~ of component of many sorting algorithms, including quicksort. It is used to rearrange the elements of an array around ~~a~~ a pivot element, such that elements smaller than the pivot are placed before it, and elements greater than the pivot are placed after it. The partition algorithm is a key step in the quicksort algorithm, which efficiently sorts the elements of an array.

30) What is binary heap? Sort 13, 25, 17, 22, 5, 8, 3 using max heap. Also write ~~a~~ the sorting algorithms.

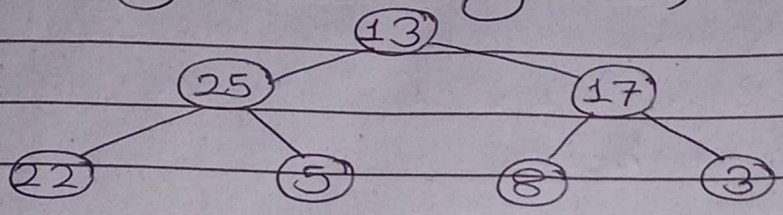
→ Binary heap is a binary tree where the root of any subtree has a higher (or lower based on the type of heap) value than all the nodes in its subtree.

Given,

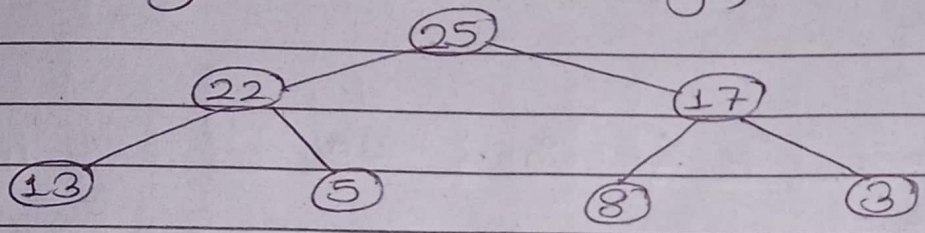
13, 25, 17, 22, 5, 8, 3

Using max heap,

Constructing a binary tree,

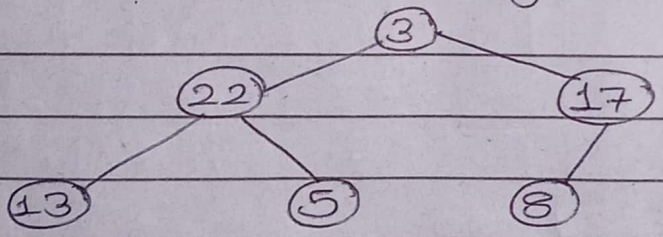


Converting into max heap,



Once the max heap is obtained, then delete the root node and place it at the last position of the array.

In the ~~vacatio~~ vacant position, the last value of the max heap tree is inserted.



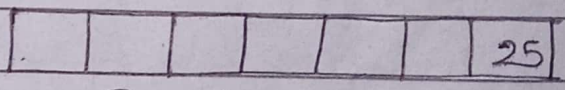
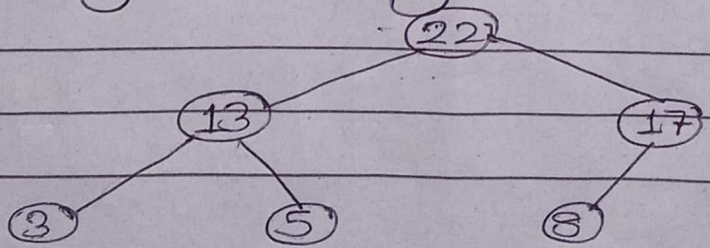
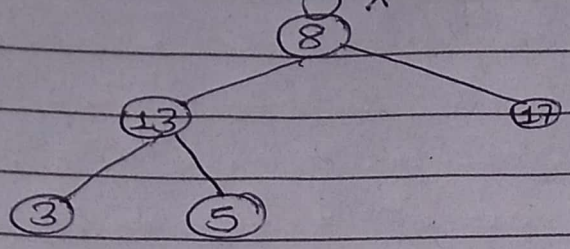


Fig: Array

Eg Similarly, Converting into max heap,



Since max heap is obtained, so



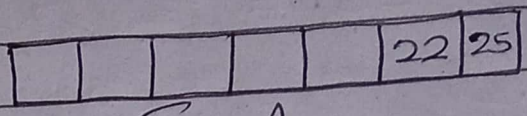
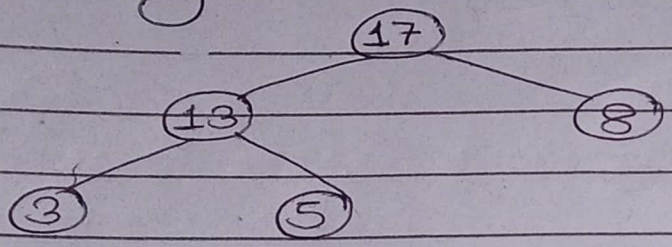
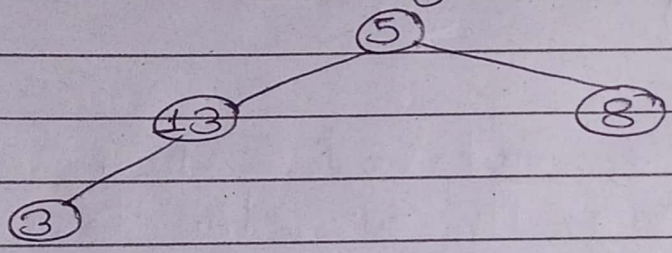


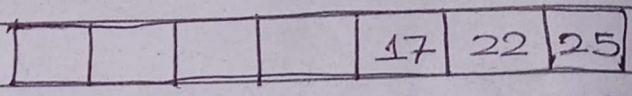
Fig: Array

Converting into max heap,

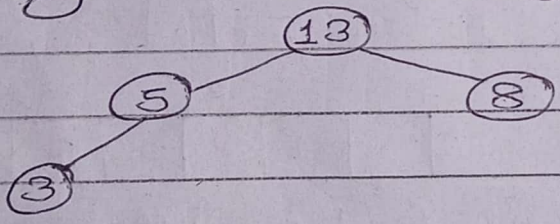


Since max heap is obtained, so

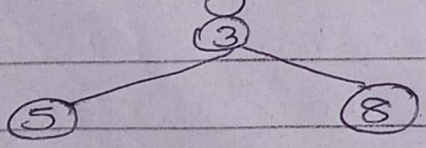


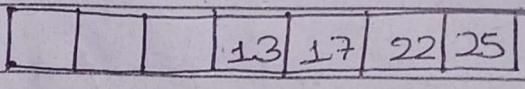


Converting into max heap,

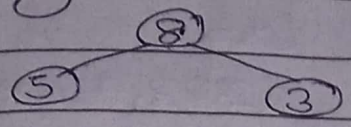


Since max heap is obtained, so

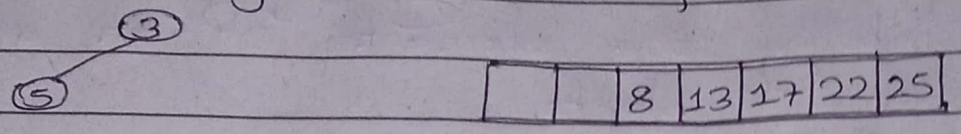




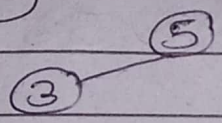
Converting into max heap,



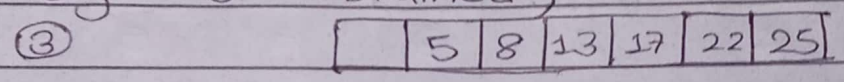
Since max heap is obtained, so



Converting into max heap,



Since max heap is obtained, so



Finally, sorted List is:

3	5	8	13	17	22	25

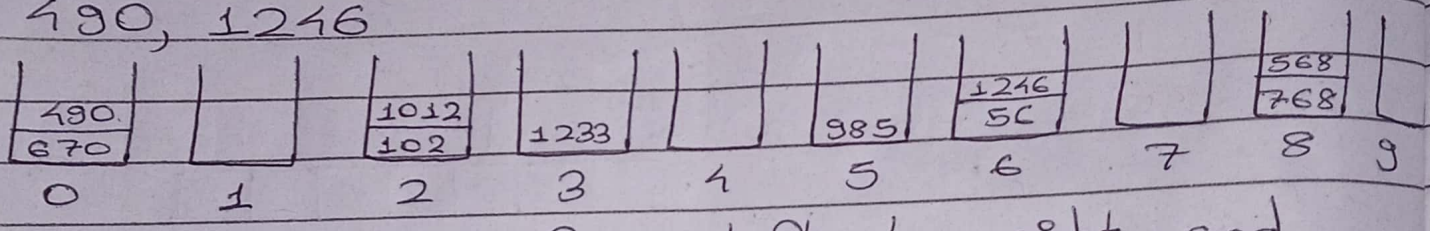
Algorithm:

1) Construct a binary tree with given List of ~~elements~~ elements.
2) Transform the binary tree into max heap
3) Delete the root node and ~~put the~~ place it into the sorted List.
4) In the vacant position (~~from wh~~ root node position which is deleted), place the Last value of max heap tree.
5) Repeat above steps until max heap is empty.
6) Display the sorted List.

31) What is bucket sort? Perform radix sort from the following sequence of unordered list:
102, 1012, 985, 768, 56, 1233, 568, 670, 490, 1246

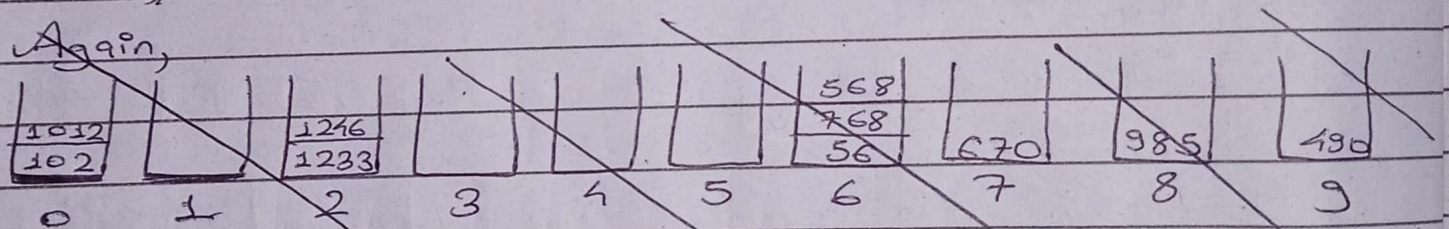
→ Bucket sort is an sorting algorithm that works by distributing the elements of an array into a number of blocks.

Given,
102, 1012, 985, 768, 56, 1233, 568, 670, 490, 1246



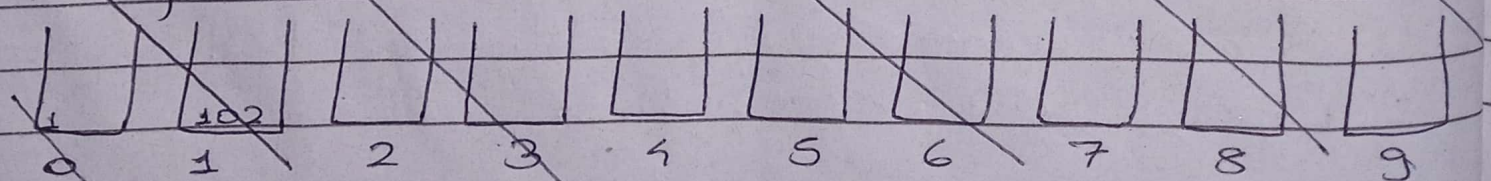
Collecting values from left to right and bottom to top,
670, 490, 102, 1012, 1233, 985, 56, 1246, 768, 568,

Again,

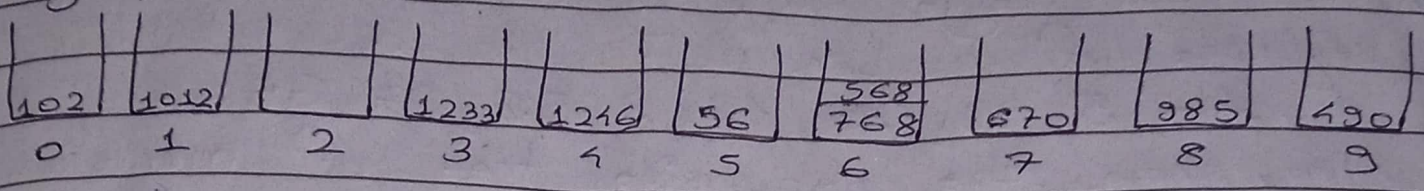


Collecting values from left to right and bottom to top,
102, 1012, 1233, 1246, 56, 768, 568, 670, 985, 490

Then,



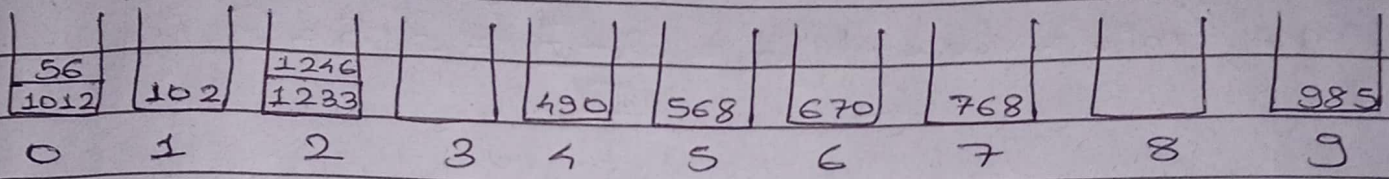
Again,



Collecting values from left to right and bottom to top,

102, 1012, 1233, 1246, 56, 768, 568, 670, 985, 490

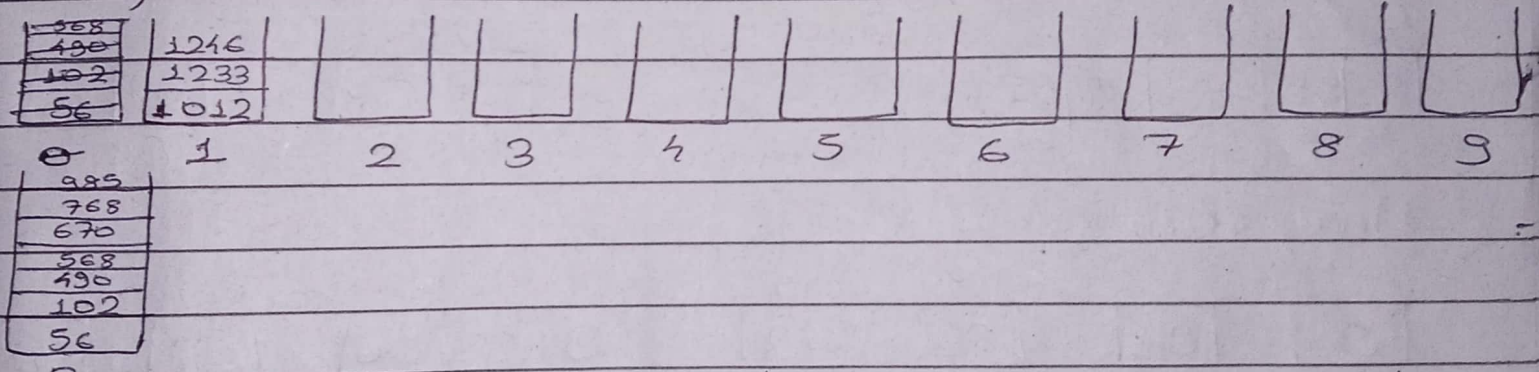
Then,



~~10~~ Collecting values from left to right and bottom to top,

1012, 56, 102, 1233, 1246, 490, 568, 670, 768, 985

Now,



Collecting values from left to right and ~~bottom to top~~ bottom to top,

56, 102, 490, 568, 670, 768, 985, 1012, 1233, 1246

∴ Above given is the sorted list.

32) Trace the merge sort algorithm for following sequence of number.

~~Ans~~ 5, 99, 25, 42, 9, 56, 27, 35

Also, show the recursive tree of merge sort taking reference of above sequence.

→ Given,

Array List is:

A =

5	99	25	42	9	56	27	35
0	1	2	3	4	5	6	7

Since, $mid = \frac{low + high}{2} = \frac{0+7}{2} = 3.5 = 3$

5	99	25	42
0	1	2	3

9	56	27	35
4	5	6	7

Since, $mid = \frac{0+3}{2} = 1$

Since, $mid = \frac{4+7}{2} = 5$

5	99
0	1

25	42
2	3

9	56
4	5

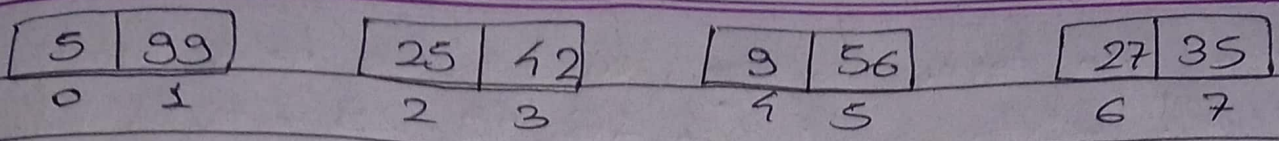
27	35
6	7

Here, ~~mid~~ again dividing,

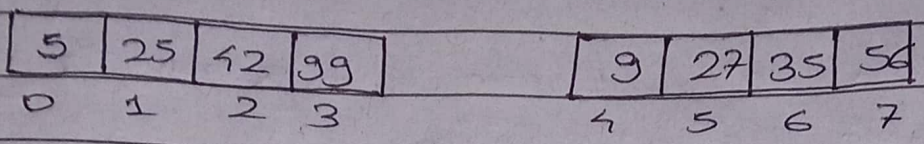
5	99	25	42	9	56	27	35
0	1	2	3	4	5	6	7

Here, all array elements are divided into ~~single~~ sub part array. So they become ~~are~~ sorted in their respective part.

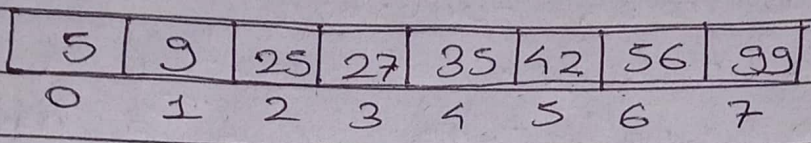
Now, merging these array elements ~~where~~ we get bigger sorted subarrays.



Similarly, we get on merging,

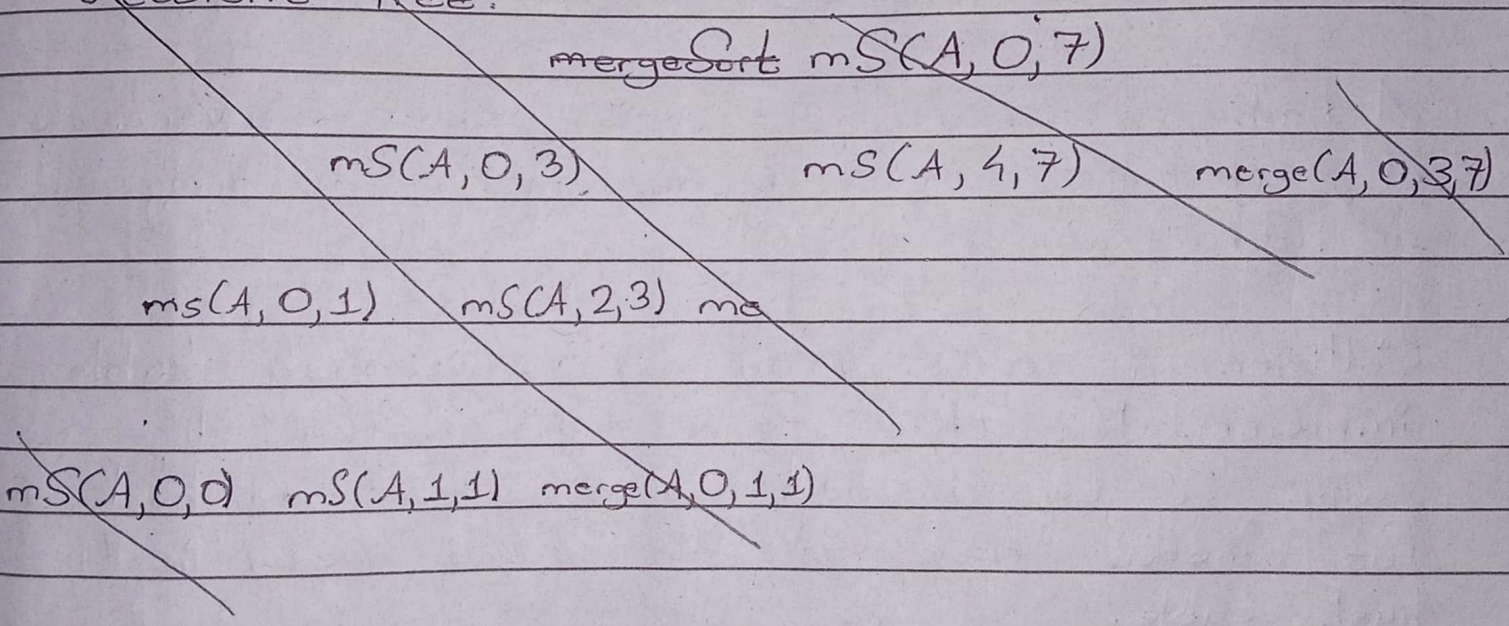


Finally, we get,



∴ Above given is the sorted list.

Recursive tree:



33) Define the terms hashing, load factor, hash function and collision. Given input {1, 16, 49, 25, 64, 0, 81, 4, 9} and a hash function $h(x) = x \bmod 10$. Show the resulting:

i) Hash table using linear and quadratic probing
ii) Hash table using chaining
iii) Hash table using double hashing (define your own double hash function)

→ Hashing is a technique of mapping a large chunk of data into small tables using a hashing function.

Load factor is the ratio of total number of elements stored in the table to the size of the table.

A hash function is a function that converts data of arbitrary size (such as keys) into array indices or addresses.

Collision is a common issue in hash table implementation that occurs when more than one value to be hashed by a particular hash function hash to the same index in the hash table.

Given,

Keys = {1, 16, 49, 25, 64, 0, 81, 4, 9}
Hash function, $h(x) = x \bmod 10$

We know,

i) Using Linear probing,

$h'(x) = h(x) + i$, $i = 1, 2, 3, 4, 5, \ldots\ldots$

Key	Hash Value	Stored Index	Probing
1	$h(1) = 1 \% 10 = 1$	1	1
16	$h(16) = 16 \% 10 = 6$	6	1
49	$h(49) = 49 \% 10 = 9$	9	1
25	$h(25) = 25 \% 10 = 5$	5	1
64	$h(64) = 64 \% 10 = 4$	4	1
0	$h(0) = 0 \% 10 = 0$	0	1
81	$h(81) = 81 \% 10 = 1$ (Collision) $h'(81) = 1 + 1 = 2$	2	2
4	$h(4) = 4 \% 10 = 4$ (Collision) $h'(4) = 4 + 1 = 5$ $h''(4) = 4 + 2 = 6$ $h'''(4) = 4 + 3 = 7$	7	4
9	$h(9) = 9 \% 10 = 9$ (Collision) $h'(9) = 9 + 1 = 10$ $= 10 \% 10 = 0$ $h''(9) = 9 + 2 = 11$ $= 11 \% 10 = 1$ $h'''(9) = 9 + 3 = 12$ $= 12 \% 10 = 2$ $h''''(9) = 9 + 4 = 13$ $= 13 \% 10 = 3$	3	5

49	9
	8
4	7
16	6
25	5
64	4
9	3
81	2
1	1
0	0

Fig: Hash Table

Using quadratic probing,

$h'(x) = h(x) + i^2$, $i = 1, 2, 3, 4, 5, \ldots\ldots$

Key	Hash Value	Stored Index	Probing
1	$h(1) = 1 \% 10 = 1$	1	1
16	$h(16) = 16 \% 10 = 6$	6	1
49	$h(49) = 49 \% 10 = 9$	9	1
25	$h(25) = 25 \% 10 = 5$	5	1
64	$h(64) = 64 \% 10 = 4$	4	1
0	$h(0) = 0 \% 10 = 0$	0	1
81	$h(81) = 81 \% 10 = 1$ (Collision) $h'(81) = 1 + 1^2 = 2$	2	2
4	$h(4) = 4 \% 10 = 4$ (Collision) $h'(4) = 4 + 1^2 = 5$ $h''(4) = 4 + 2^2 = 8$	8	3
9	$h(9) = 9 \% 10 = 9$ (Collision) $h'(9) = 9 + 1^2 = 10$ $= 10 \% 10 = 0$ $h''(9) = 9 + 2^2 = 13$ $= 13 \% 10 = 3$	3	3

49	9
4	8
	7
16	6
25	5
64	4
9	3
81	2
1	1
0	0

Fig: Hash Table

Page: .....

ii) Using chaining,

$h(1) = 1 \% 10 = 1$

$h(16) = 16 \% 10 = 6$

$h(49) = 49 \% 10 = 9$

$h(25) = 25 \% 10 = 5$

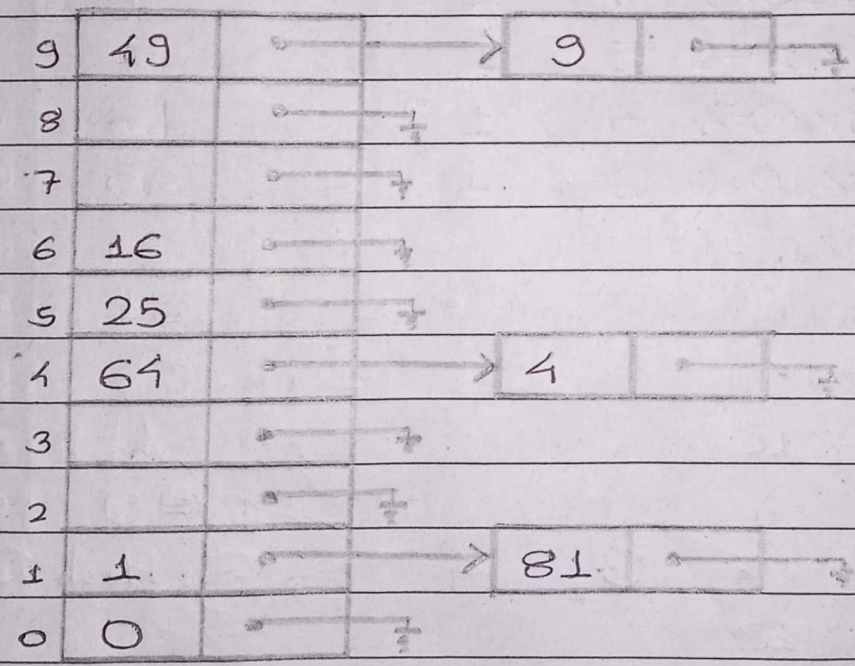
$h(64) = 64 \% 10 = 4$

$h(0) = 0 \% 10 = 0$

$h(81) = 81 \% 10 = 1$ (Collision)

$h(4) = 4 \% 10 = 4$ (Collision)

$h(9) = 9 \% 10 = 9$ (Collision)



iii) Using double hashing,

$h'(x) = (h_1(x) + i * h_2(x)) \bmod 10, \; i = 1, 2, 3, 4, 5$

$h'(x) = (h_1(x) + i * h_2(x)) \bmod 10, \; i = 1, 2, \ldots$

where,

$h_1(x) = x \bmod 10$

$h_2(x) = x \bmod 7$

Key	Hash Value	Stored Index	Probing
1	$h(1) = 1 \% 10 = 1$	1	1
16	$h(16) = 16 \% 10 = 6$	6	1
49	$h(49) = 49 \% 10 = 9$	9	1
25	$h(25) = 25 \% 10 = 5$	5	1
64	$h(64) = 64 \% 10 = 4$	4	1
0	$h(0) = 0 \% 10 = 0$	0	1
81	$h(81) = 81 \% 10 = 1$ (Collision)		
	$h'(81) = (1 + 1 * (81 \% 7)) \% 10 = 5$		
81	$h''(81) = (1 + 2 * (81 \% 7)) \% 10 = 9$	3	4
	$h'''(81) = (1 + 3 * (81 \% 7)) \% 10 = 3$		
4	$h(4) = 4 \% 10 = 4$ (Collision)	8	2

49	9
4	8
9	7
16	6
25	5
64	4
81	3
-	2
1	1
0	0

Fig: Hash Table

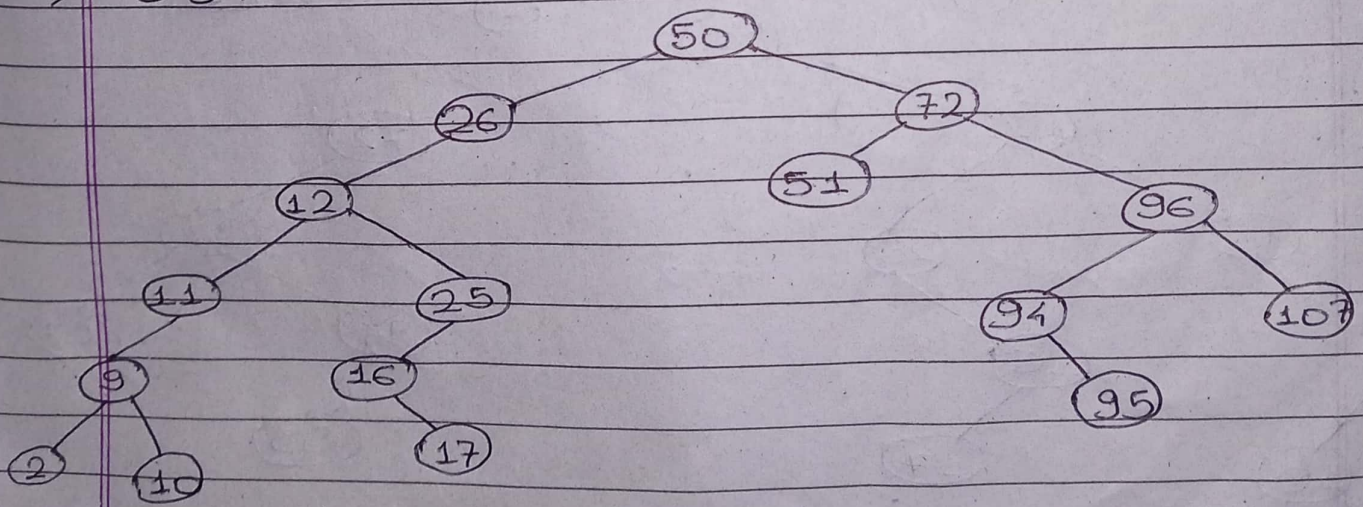
	$h'(4) = (4 + 1*(4 \% 7)) \% 10 = 8$		
	$h(9) = 9 \% 10 = 9$ (Collision)		
	$h'(9) = (9 + 2*(9 \% 7)) \% 10 = 3$		
9	$h''(9) = (9 + 3*(9 \% 7)) \% 10 = 5$	7	4
	$h'''(9) = (9 + 4*(9 \% 7)) \% 10 = 7$		

34) Draw the binary search tree whose elements are inserted in the following order.

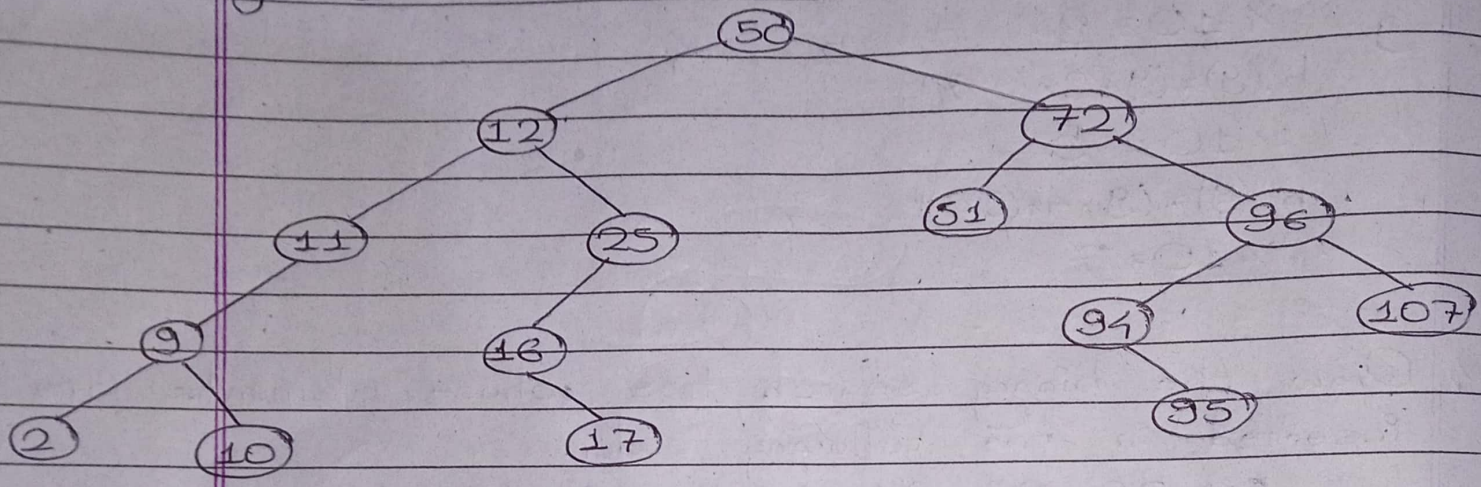
50, 72, 96, 94, 107, 26, 12, 11, 9, 2, 10, 25, 51, 16, 17, 95

Show how the tree would look after the deletion of 26, 50, 16, and 10 respectively. Explain each steps of deletion.

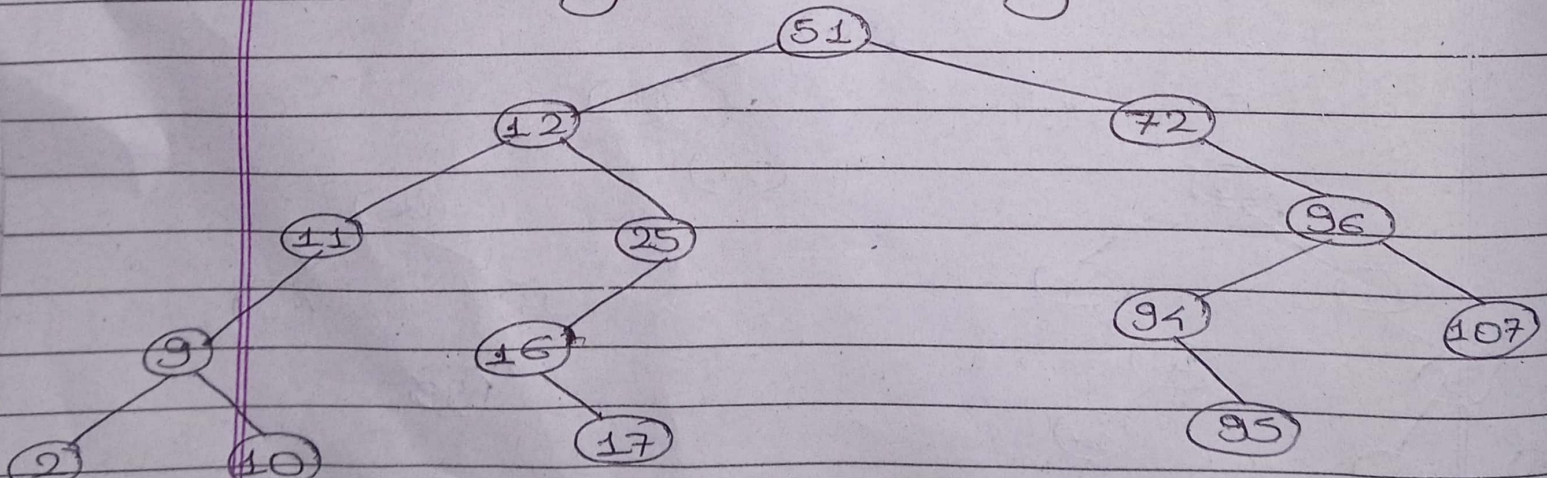
→ BST is constructed as:



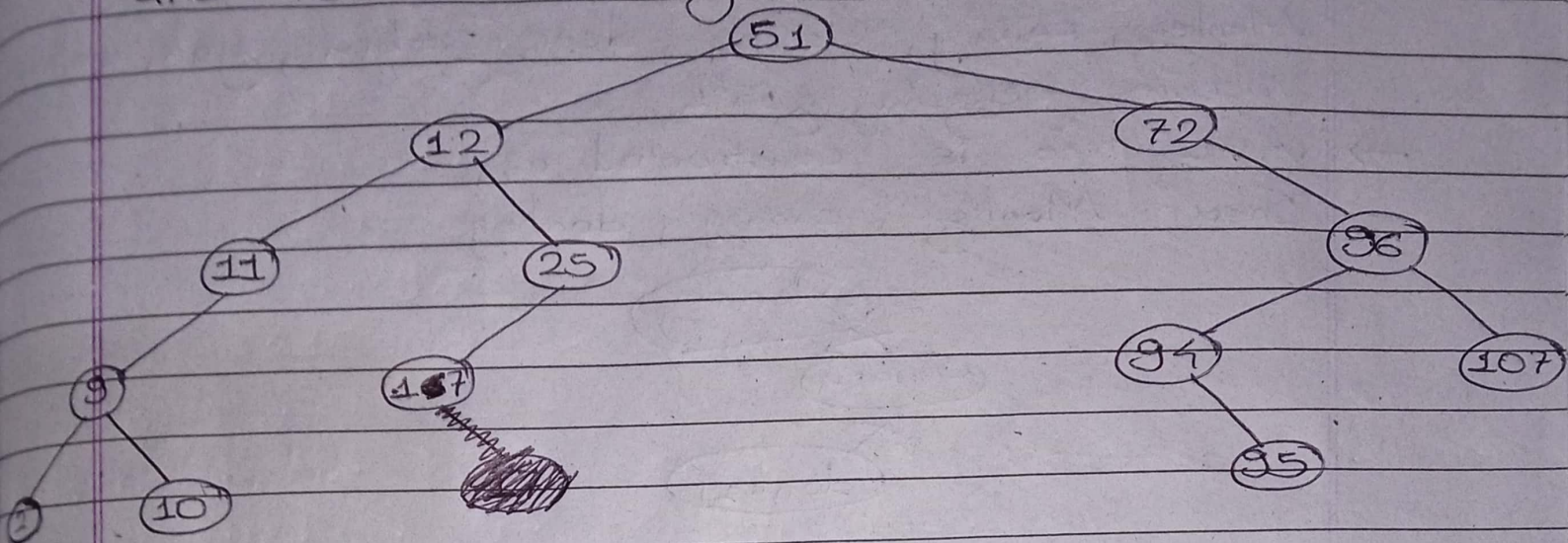
Delete 26:
Since the node to be deleted has a single child, so we delete the ~~node~~ node from the tree and redirect the pointer to its subtree.



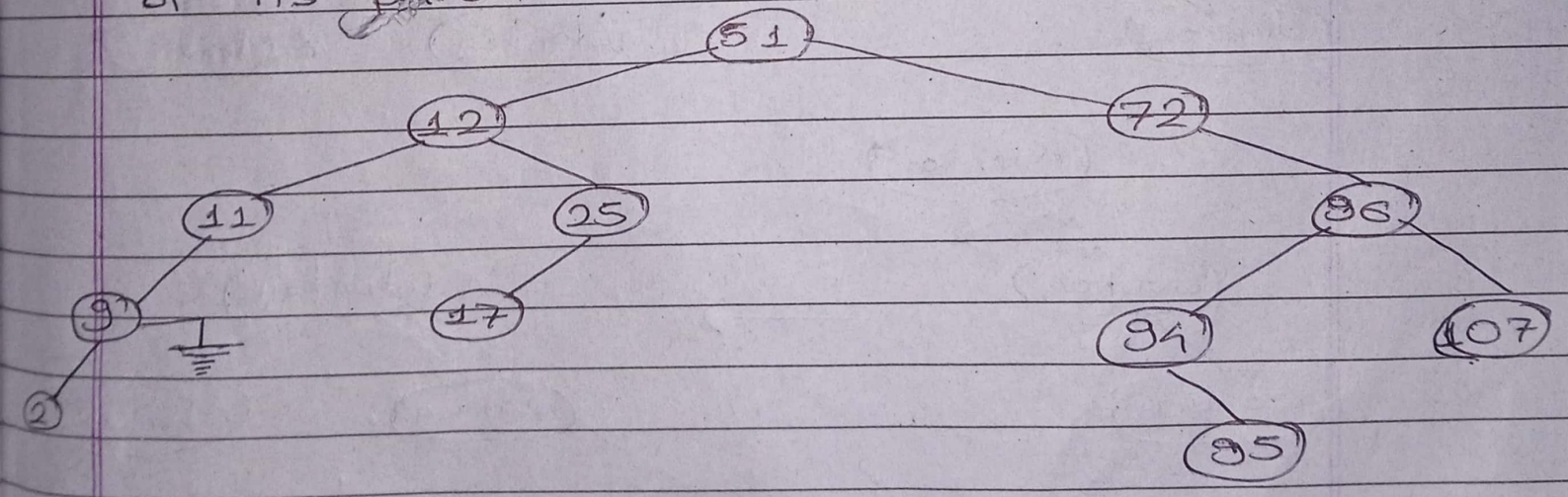
Delete 50:
Since the node to be deleted has two children, so we replace the node with the smallest key from its right subtree.



Delete 16:
Since the node to be deleted has a single child, so we delete the node from the tree and redirect the pointer to its subtree.



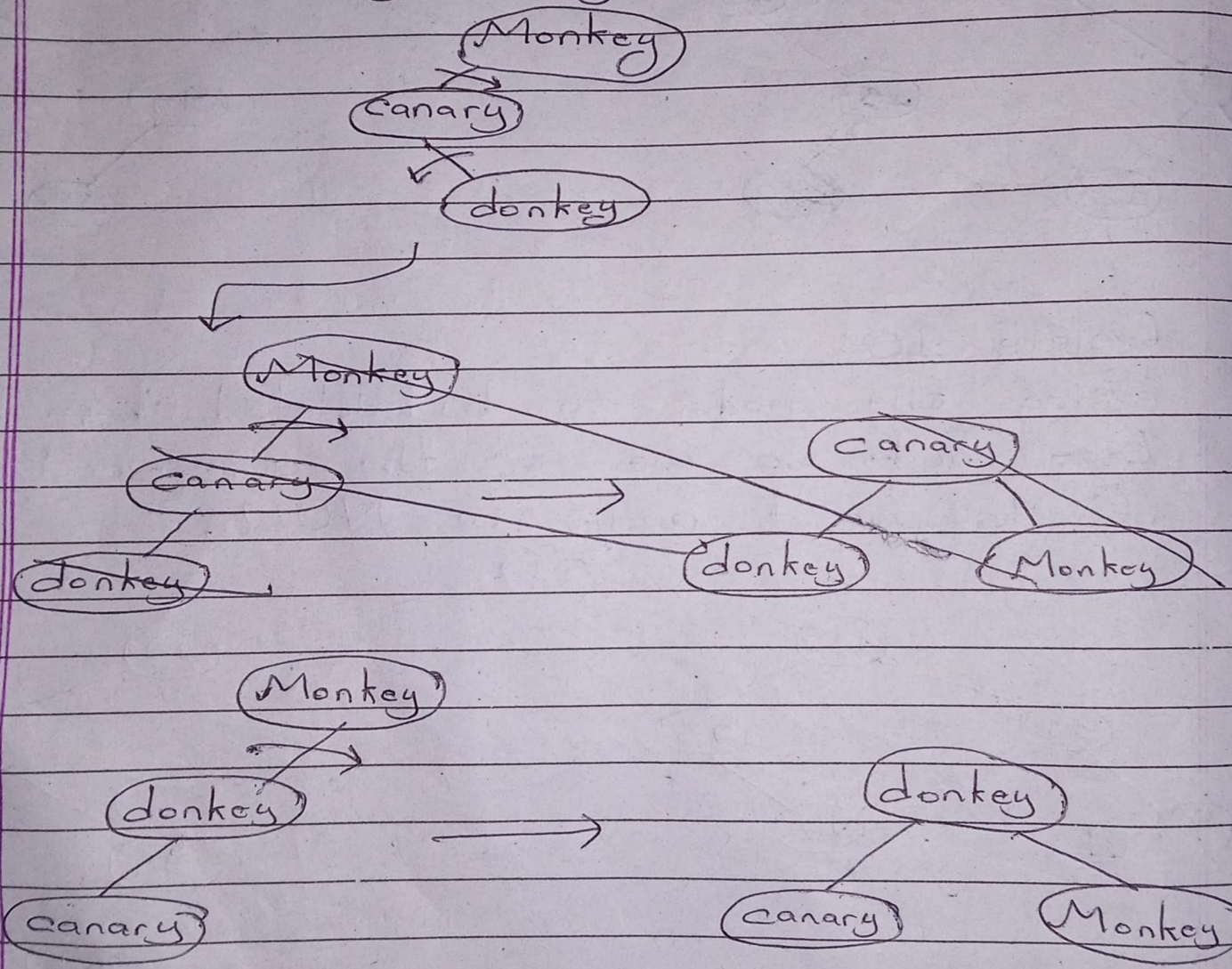
Delete 10:
Since the node to be deleted ~~has~~ is a leaf node, so we set the corresponding link of its parent node to NULL.



35) The key of each node in a binary search tree is a short character string. Show AVL tree after the following words were inserted (in the order indicated):
Monkey, canary, donkey, deer, zebra, yak, walrus, vulture, penguin, quail

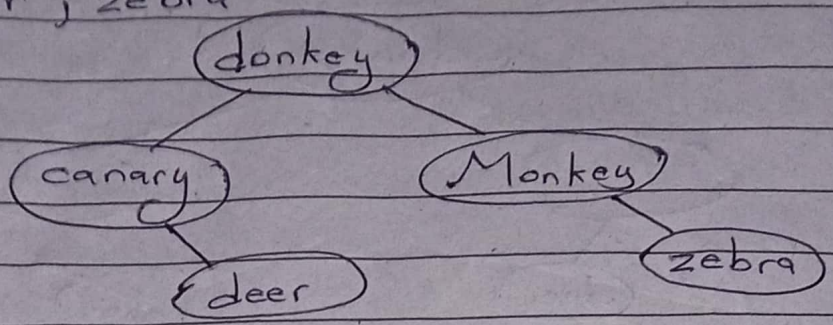
→ AVL tree is constructed as,

Insert Monkey, canary, donkey

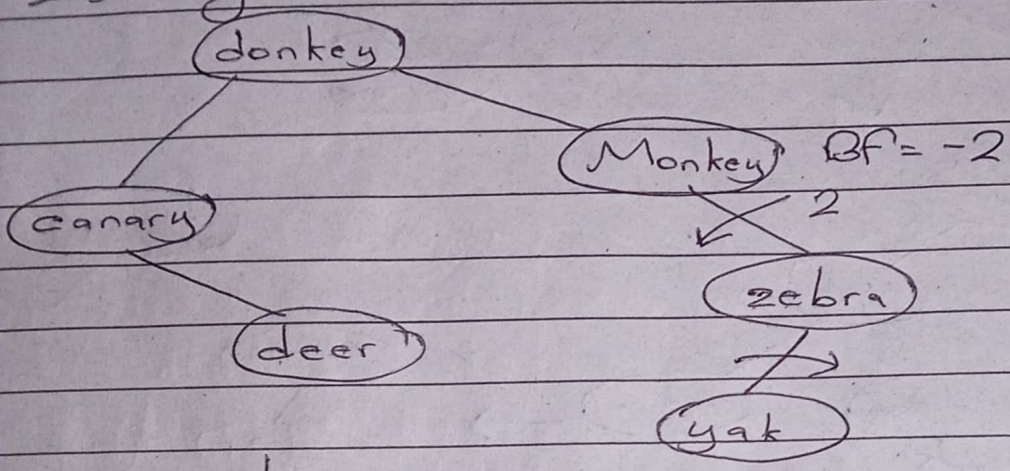


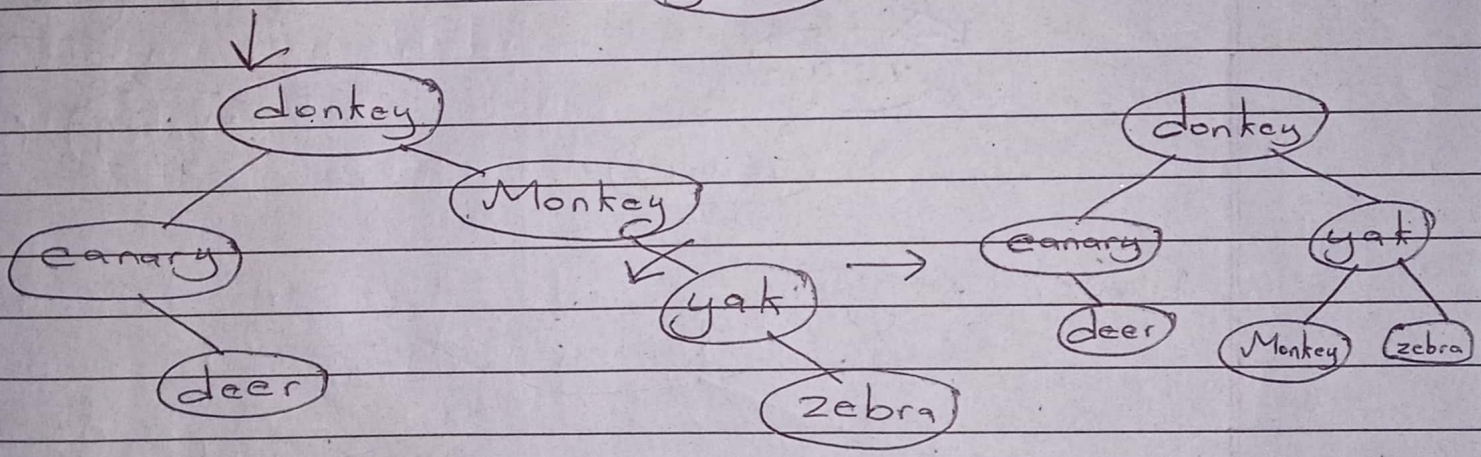
Page: .....

Insert deer , zebra

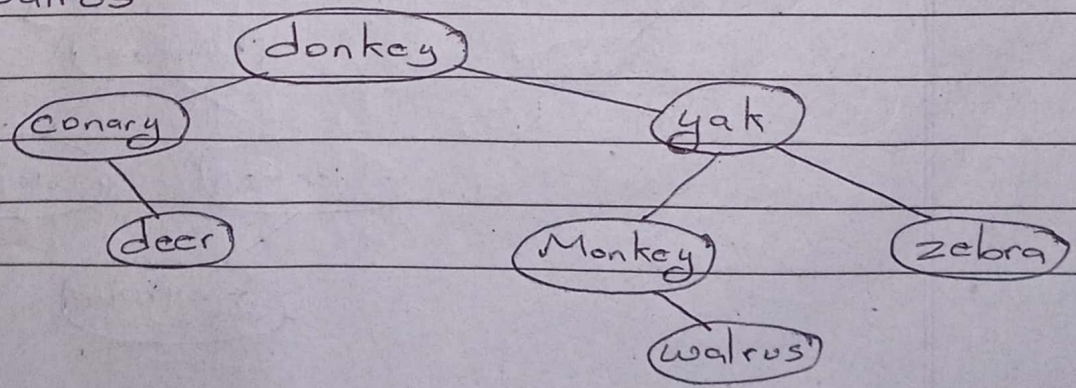


Insert yak

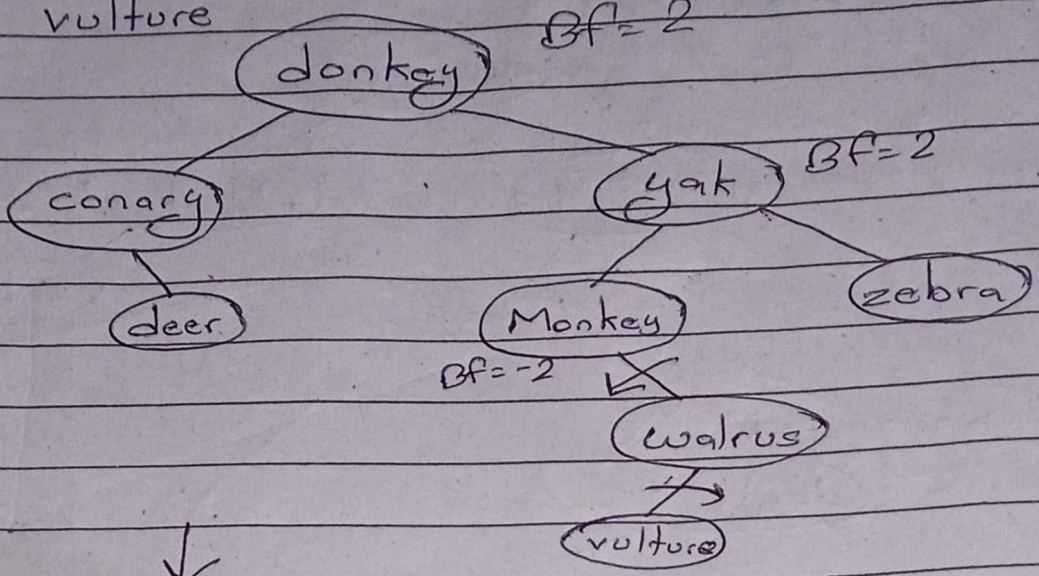


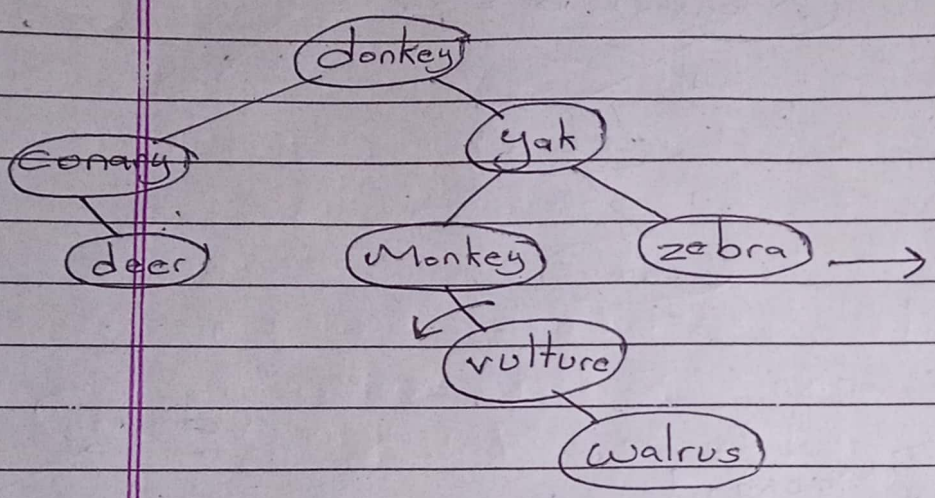


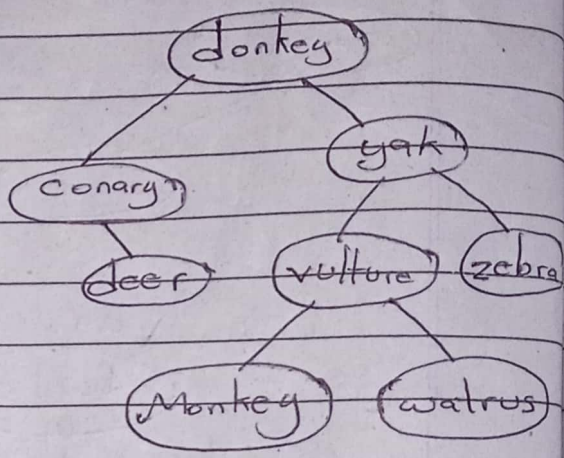
Insert walrus



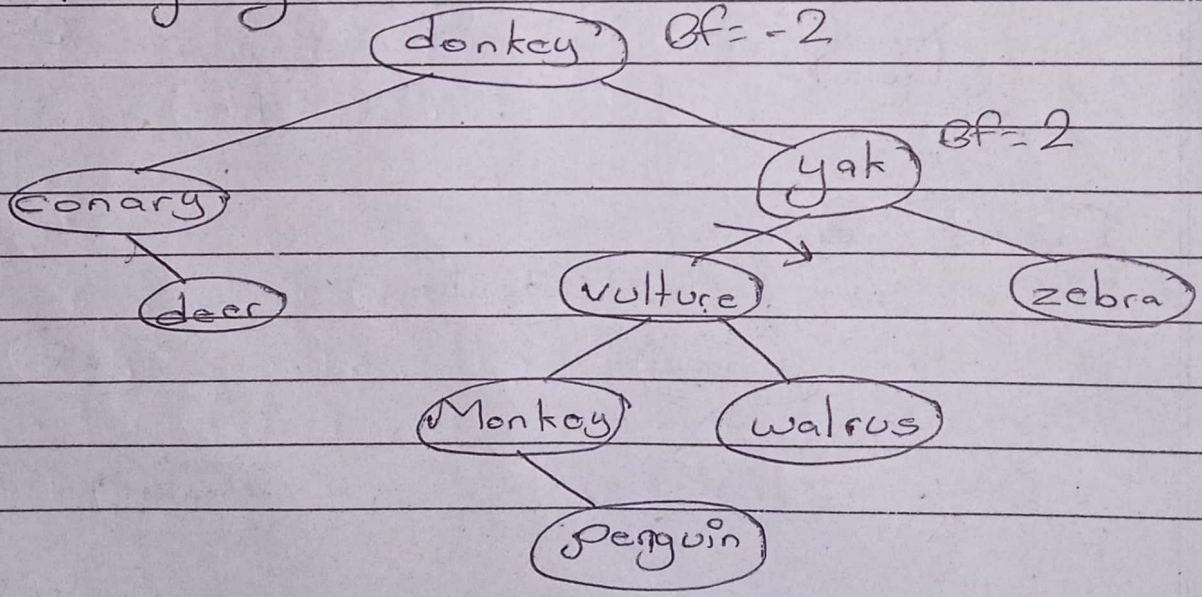
Insert vulture



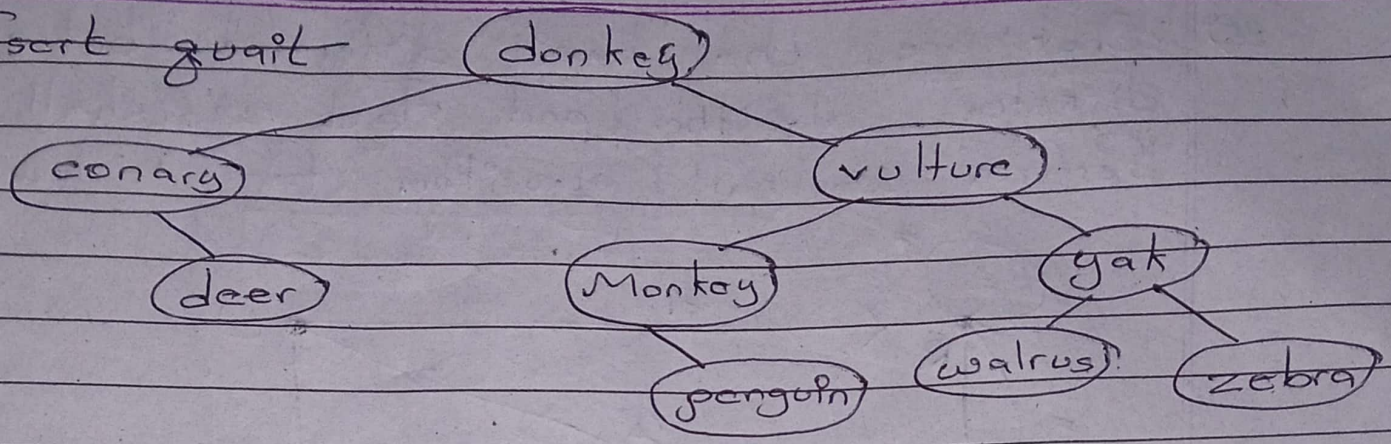




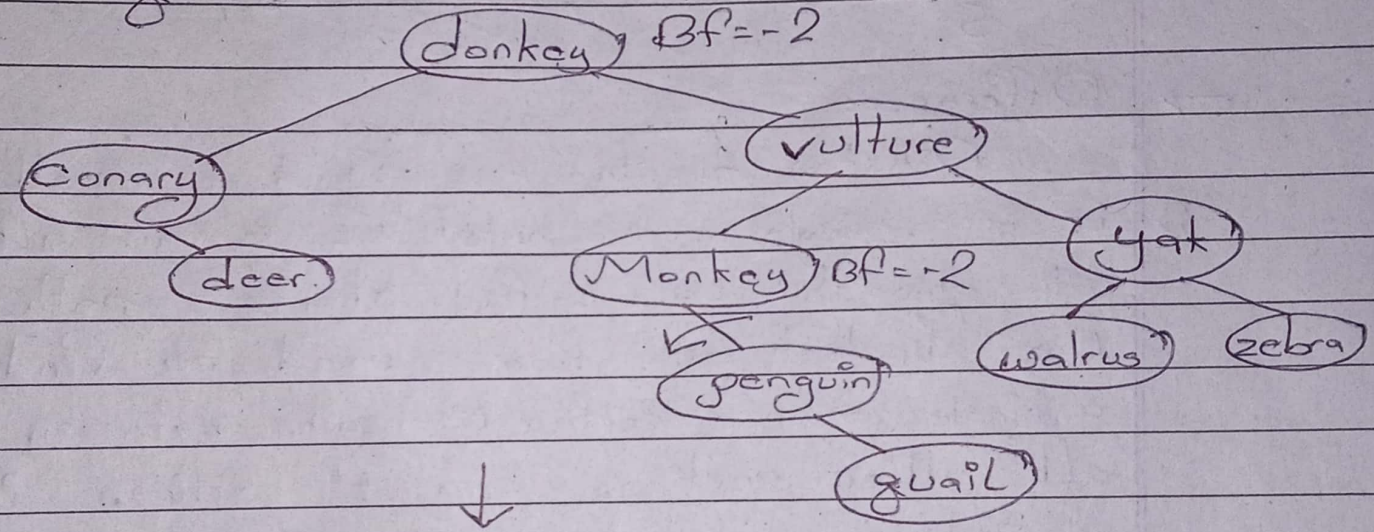
Insert penguin

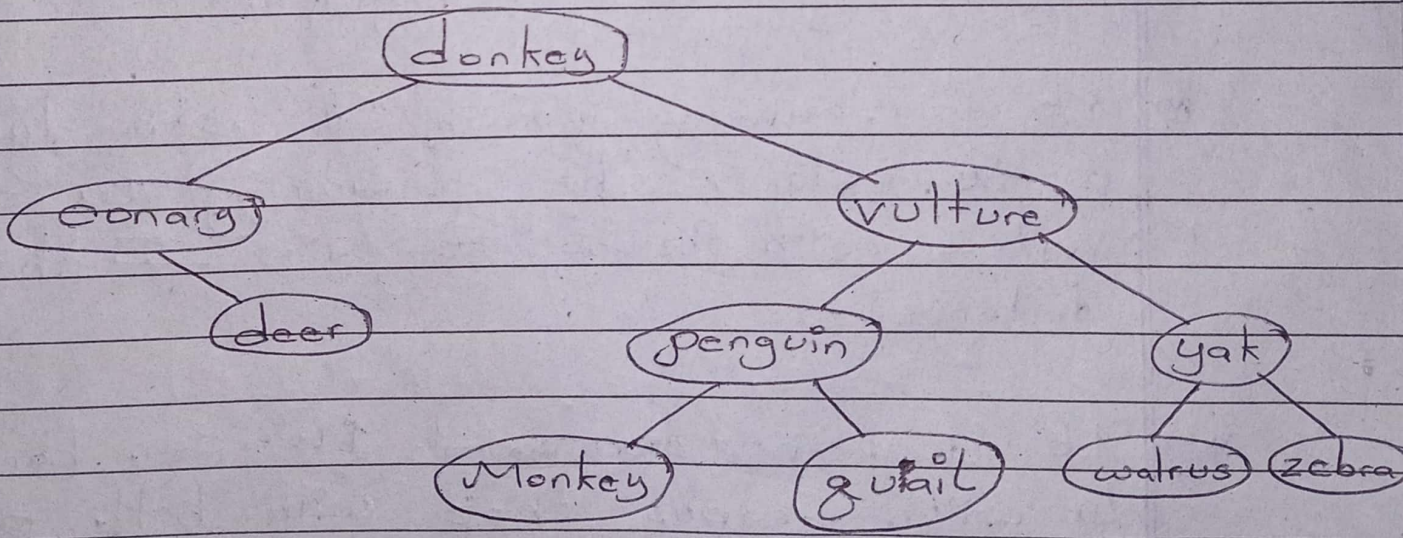


~~Insert quail~~



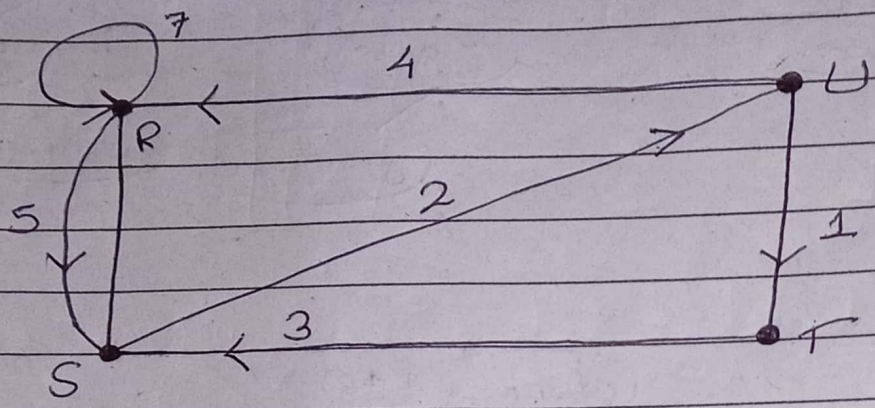
Insert quail





Page: .....

36) What are the basic difference between dijkstra's algorithm and floyd warshall algorithm? Perform warshall algorithm for following graph.



→ Difference:

	Dijkstra's Algorithm	Floyd Warshall Algorithm
i)	It is a single source shortest path algorithm used to find the shortest paths from a single source vertex to all other vertices in a weighted graph.	It is an all-pairs shortest path algorithm used to find the shortest paths between all pairs of vertices in a weighted graph.
ii)	It is a greedy approach, as it selects the vertex with the shortest distance.	It uses dynamic programming to compute all pairs ~~of~~ shortest paths.
iii)	It does not work correctly with graphs that have negative edge weights.	It can handle graphs with both positive and negative edge weights.

Date :
Page :

iv)	It's time complexity is $O(E + V * \log(V))$.	It's time complexity is $O(V^3)$.
v)	It's space complexity is $O(V)$.	It's space complexity is $O(V^2)$.
vi)	It uses a ~~priority~~ priority queue or min-heap.	It uses a two-dimensional array.

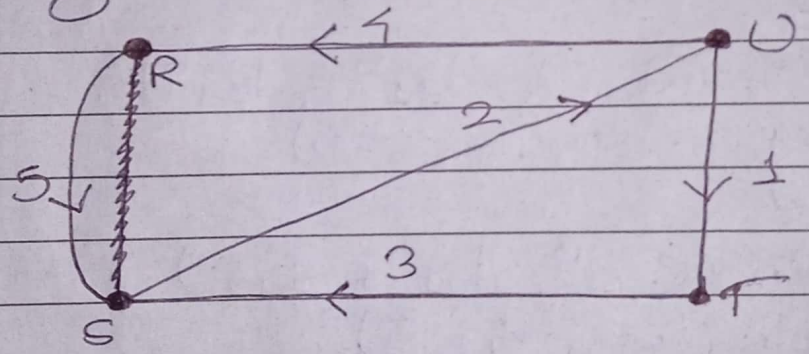
We know,

Using Floyd Warshall Algorithm,

$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

where, i = row, j = column, k = iteration no.

Removing self loop,



So,

$D^0 =$

	R	S	T	U
R	0	5	∞	∞
S	∞	0	∞	2
T	∞	3	0	∞
U	4	∞	1	0

Page:

Iteration 1

Freeze row 1 and col 1, k = 1

$D^1 =$

	R	S	T	U
R	0	5	∞	∞
S	∞	0	∞	2
T	∞	3	0	∞
U	4	9	1	0

$d_{23}^1 = \min(d_{23}^0, d_{21}^0 + d_{13}^0)$
$= \min(\infty, \infty + \infty)$
$= \infty$

$d_{24}^1 = \min(d_{24}^0, d_{21}^0 + d_{14}^0)$
$= \min(2, \infty + \infty)$
$= 2$

$d_{32}^1 = \min(d_{32}^0, d_{31}^0 + d_{12}^0)$
$= \min(3, \infty + 5)$
$= 3$

$d_{34}^1 = \min(d_{34}^0, d_{31}^0 + d_{14}^0)$
$= \min(\infty, \infty + \infty)$
$= \infty$

$d_{42}^1 = \min(d_{42}^0, d_{41}^0 + d_{12}^0)$
$= \min(\infty, 4 + 5)$
$= 9$

$d_{43}^1 = \min(d_{43}^0, d_{41}^0 + d_{13}^0)$
$= \min(1, 4 + \infty)$
$= 1$

Similarly,

Page: .....

Iteration 2

Freeze row 2 and col 2, k = 2

$D^2 =$

	R	S	T	U
R	0	5	∞	7
S	∞	0	∞	2
T	∞	3	0	5
U	4	9	1	0

Iteration 3

Freeze row 3 and col 3, k = 3

$D^3 =$

	R	S	T	U
R	0	5	∞	7
S	∞	0	∞	2
T	∞	3	0	5
U	4	4	1	0

Iteration 4

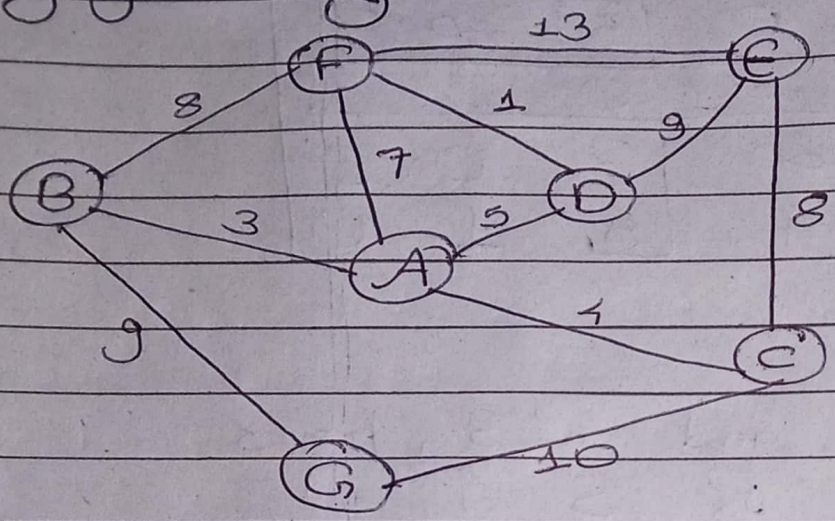
Freeze row 4 and col 4, k = 4

$D^4 =$

	R	S	T	U
R	0	5	8	7
S	6	0	3	2
T	9	3	0	5
U	4	4	1	0

∴ Above given matrix helps to find the shortest paths between all pairs of vertices in a weighted graph.

37) What is minimum spanning tree? Find minimum spanning tree for the given graph using Kruskal's and prim's algorithm.



→ Minimum spanning tree is a spanning tree that has the minimum weight among all the possible spanning trees.

~~Using Kruskal's Algorithm,~~

Here,

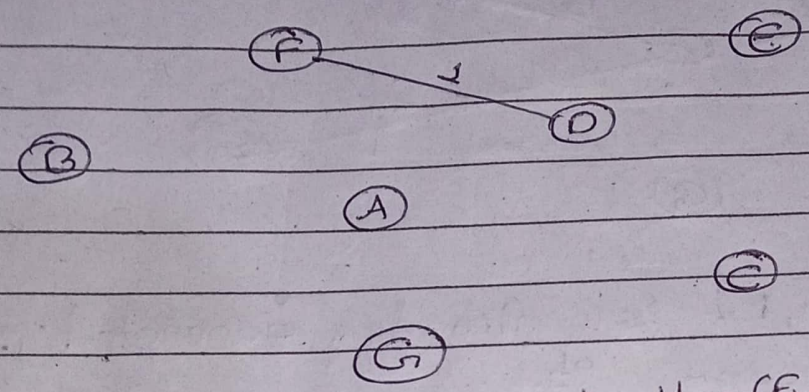
List of edges:

(F, D) = 1
(B, A) = 3
(A, C) = 4
(A, D) = 5
(A, F) = 7
(C, E) = 8
(B, F) = 8
(B, G) = 9
(D, E) = 9
(G, C) = 10
(F, E) = 13

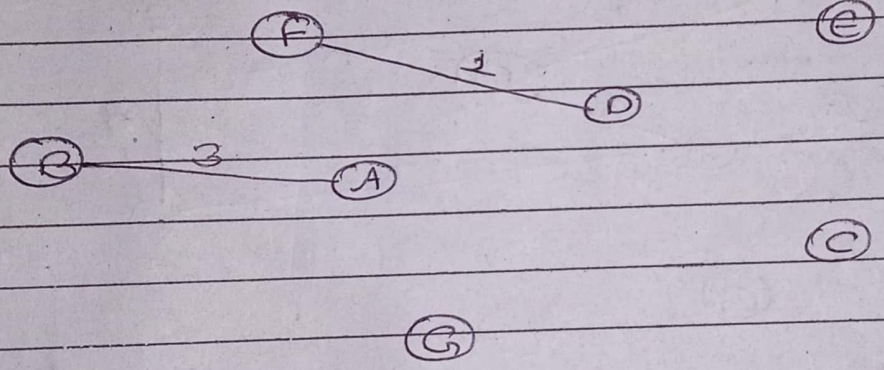
Page: .....

Using Kruskals Algorithm,

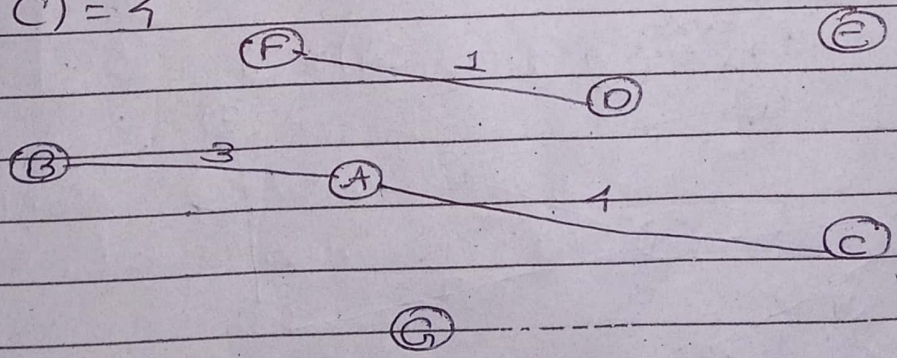
i) Choose minimum edge, cost (F, D) = 1



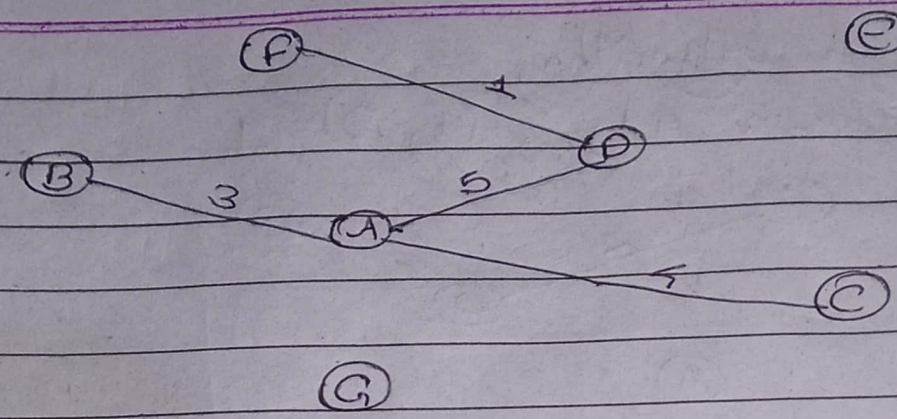
ii) Choose minimum edge, ~~greater than (F, D)~~ greater (B, A) = 3



iii) Similarly, choose min edge, cost greater than (B, A), (A, C) = 4

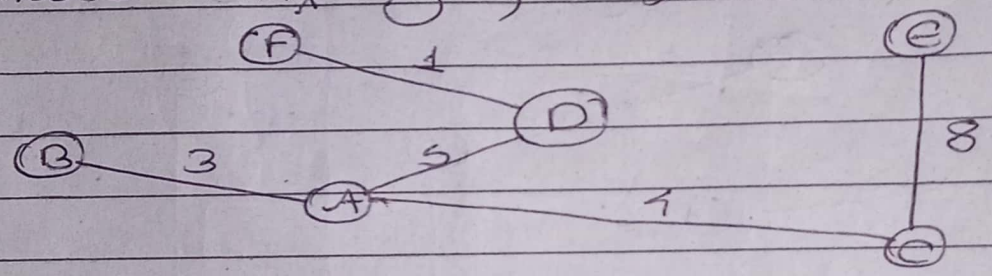


iv) Choose min, cost edge greater than (A, C), (A, D) = 5



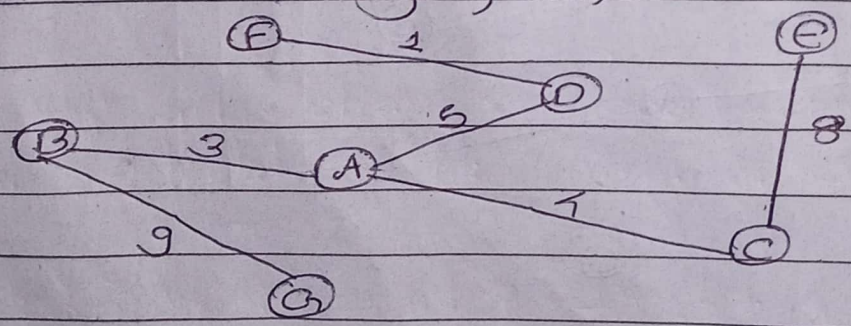
G

v) Since, (A, F) is already connected, we don't choose it.
Choose $\min_{A}^{cost}$ edge, (C, E) = 8



G

vi) Since (B, F) is already connected, we don't choose it.
Choose $\min_{A}^{cost}$ edge, (B, G) = 9



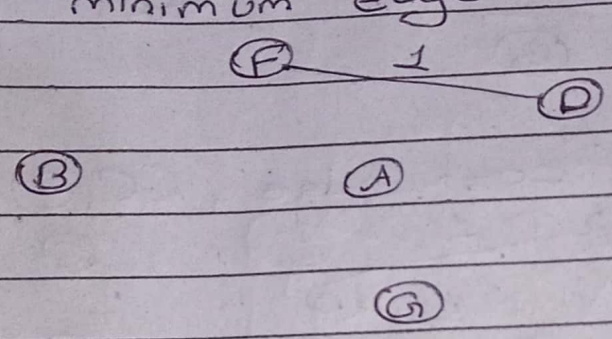
Since all the vertices are connected, we can calculate the minimum cost.

Date :
Page :

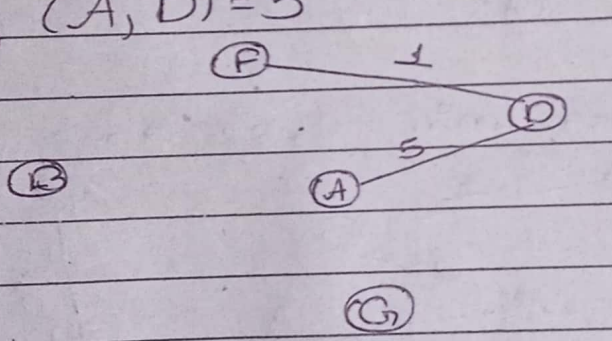
∴ Minimum Cost = 1 + 3 + 1 + 5 + 8 + 9
= 30

Using Prim's Algorithm,

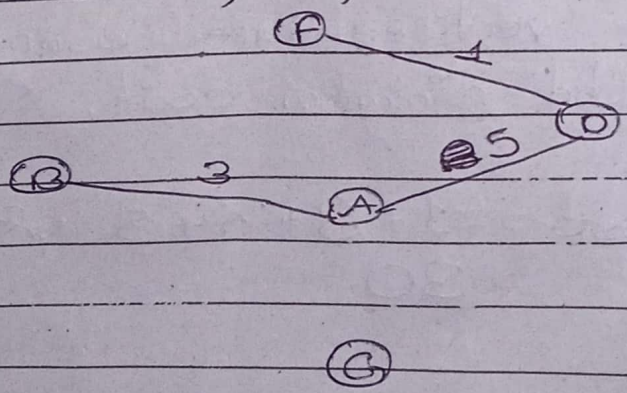
i) Choose minimum ~~edge~~ cost edge, (F, D) = 1



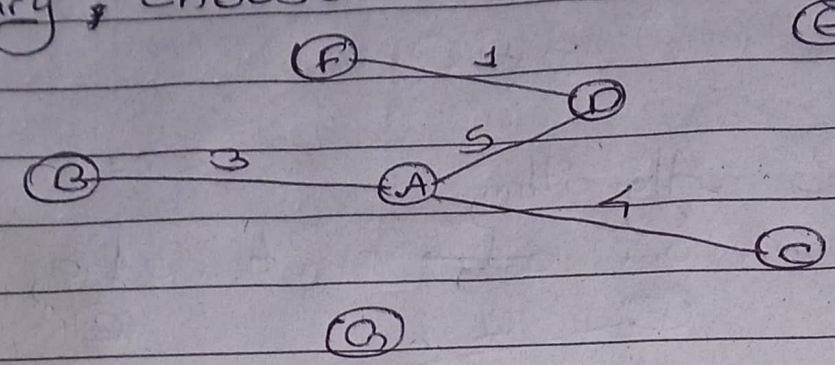
ii) Choose min cost edge connected to vertex F or D, (A, D) = 5



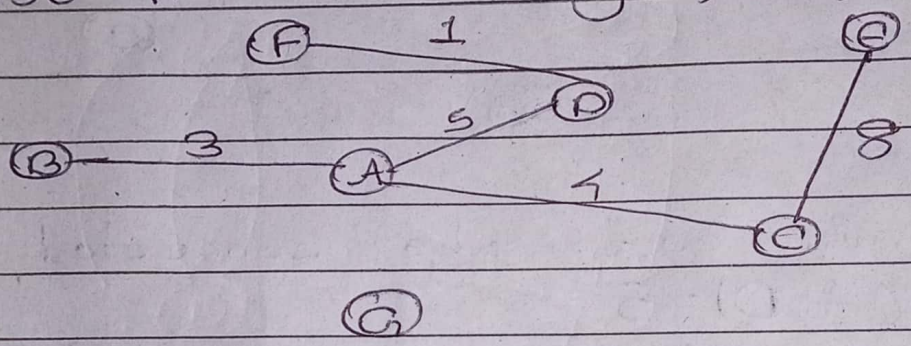
iii) Choose min cost edge connected to vertex A, or D or F, ~~(A, B) = 3~~ (B, A) = 3



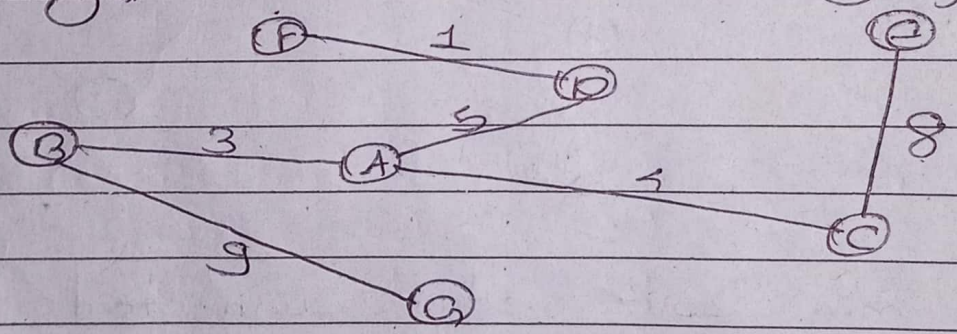
iv) Similary, choose min cost edge, (A, C) = 4



v) Choose min cost edge, (C, E) = 8



vi) Similarly, choose min cost edge, (B, G) = 9



Since, all the vertices are connected, we can calculate the minimum cost.

∴ Minimum Cost = 1 + 3 + 4 + 5 + 8 + 9
= 30

Date: .....
Page: .....

38) Draw the graphs using following representations.

a) $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

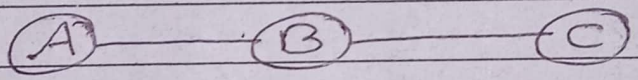
→ Since, matrix elements are either 0 or 1, so it must be matrix representation of undirected graph. or directed graph ~~too~~ with no weight value

Let A, B, C be vertices of the graph.

So,

$$\text{Matrix} = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \end{array}$$

Graph is given as,



b) $\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$

→ Since matrix elements are either 0 or 1, so it must be matrix representation of graph with no weight value.

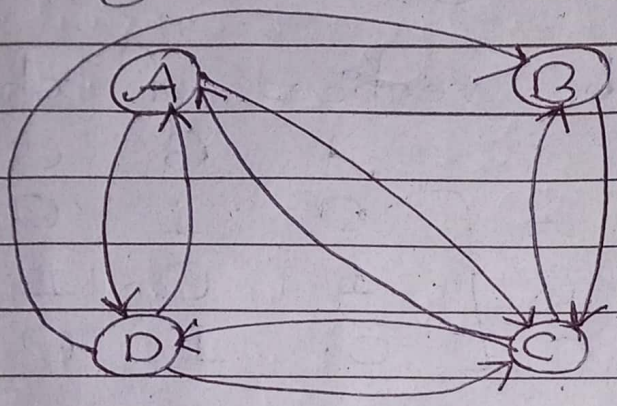
Let A, B, C, D be vertices of the graph.

So,

Matrix =

	A	B	C	D
A	0	0	1	1
B	0	0	1	0
C	1	1	0	1
D	1	1	1	0

Graph is given as,



c)

0	1	3	0	4
1	2	1	3	0
3	1	1	0	1
0	3	0	0	2
4	0	1	2	3

→ Since matrix elements are provided with weights so it must be matrix representation of weighted graph.

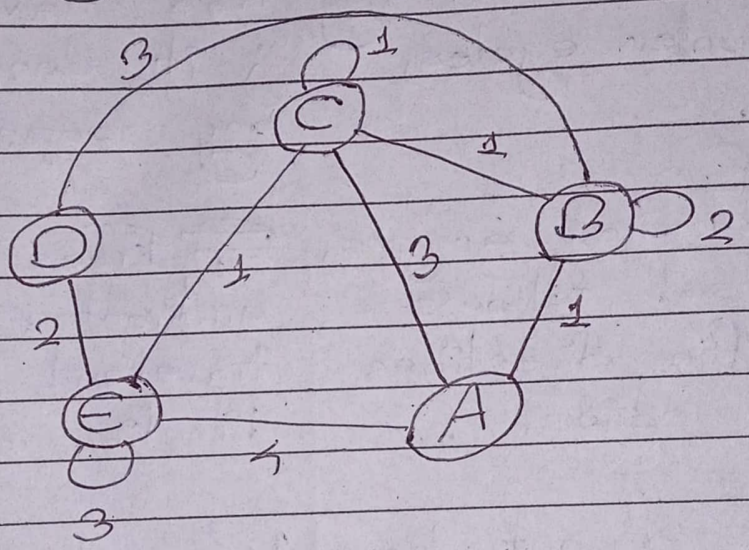
Let A, B, C, D, E be vertices of the graph.

So,

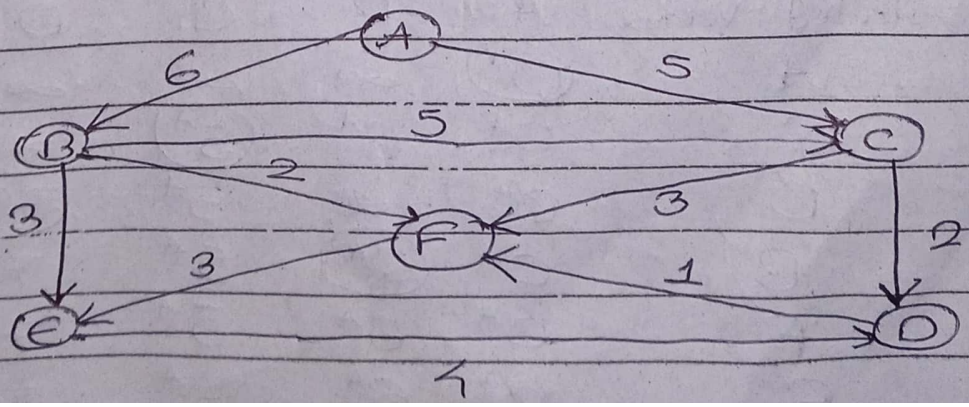
Matrix =

	A	B	C	D	E
A	0	1	3	30	4
B	1	2	1	3	0
C	3	1	1	0	1
D	0	3	0	0	2
E	4	0	1	2	3

Graph is given as,



39) What is the difference between traversing in a graph and traversing in a tree? Perform dijkstra's algorithm and find the shortest path from vertex A to any other vertex from following graph.



→

	Traversing in a graph	Traversing in a tree
i)	Graph traversal moves bet-n any pair of vertices through edges even across disconnected components.	Tree traversal moves bet-n nodes along ~~unique paths etc~~ edges in a connected structure.
ii)	Graph traversal may encounter cycles, loops.	Tree traversal is acyclic ; it does not contain any loops or cycles.
iii)	In directed graphs, traversal follows specific directions along edges.	~~In~~ Trees are typically undirected, allowing traversal in any direction along edges.
iv)	Graph do not have root node, traversal may start from any node.	Trees have a root node from which traversal starts.

Sol-n, Let us assume source vertex is A.

Selected Vertex	A	B	C	D	E	F
A	(0^A)	∞	∞	∞	∞	∞
C	0^A	6^A	(5^A)	∞	∞	∞
D	0^A	6^A	5^A	(7^C)	∞	8^C
B	0^A	(6^A)	5^A	7^C	∞	8^D
F	0^A	6^A	5^A	7^C	9^B	(8^B)
E	0^A	6^A	5^A	7^C	(9^F)	8^B

From A,

$d(B) = \min(d(B), d(A) + d(A,B))$

$= \min(\infty, 0 + 6)$

$= 6$

$d(C) = \min(d(C), d(A) + d(A,C))$

$= \min(\infty, 0 + 5)$

$= 5$

From C,

$d(D) = \min(d(D), d(C) + d(C,D))$

$= \min(\infty, 5 + 2)$

$= 7$

$d(F) = \min(d(F), d(C) + d(C,F))$

$= \min(\infty, 5 + 3)$

$= 8$

From D,

$d(F) = \min(d(F), d(D) + d(D,F))$

$= \min(8, 7 + 1)$

$= 8$

From B,

$d(F) = \min(d(F), d(B) + d(B,F))$

$= \min(8, 6 + 2)$

$= 8$

$d(E) = \min(d(E), d(B) + d(B,E))$

$= \min(\infty, 6 + 3)$

$= 9$

From F,

$d(E) = \min(d(E), d(F) + d(F, E))$

$= \min(9, 8 + 3)$

$= 9$

Shortest Paths:

A → B

A, B ⇒ 6

A → C

A, C ⇒ 5

A → D

A, C, D ⇒ 7

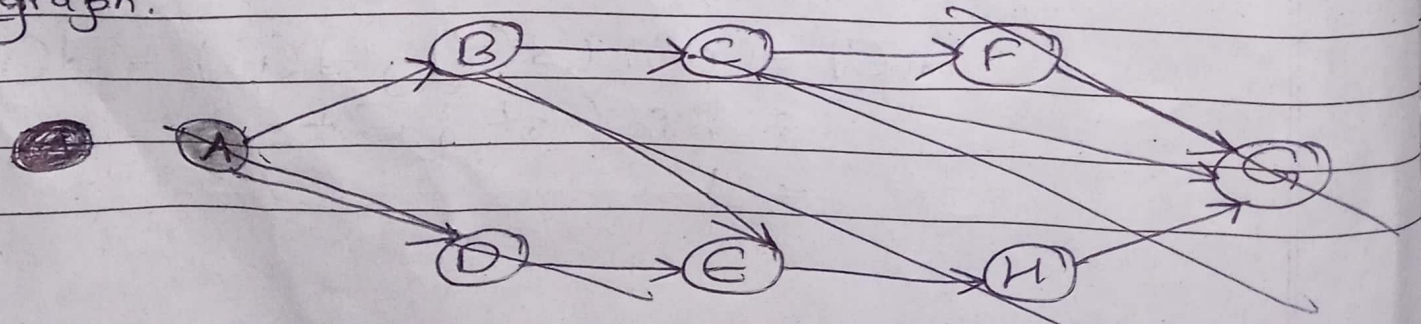
A → E

~~A, F, E ⇒ A, C, F, E ⇒ 9~~ A, B, E ⇒ ~~8~~ 9

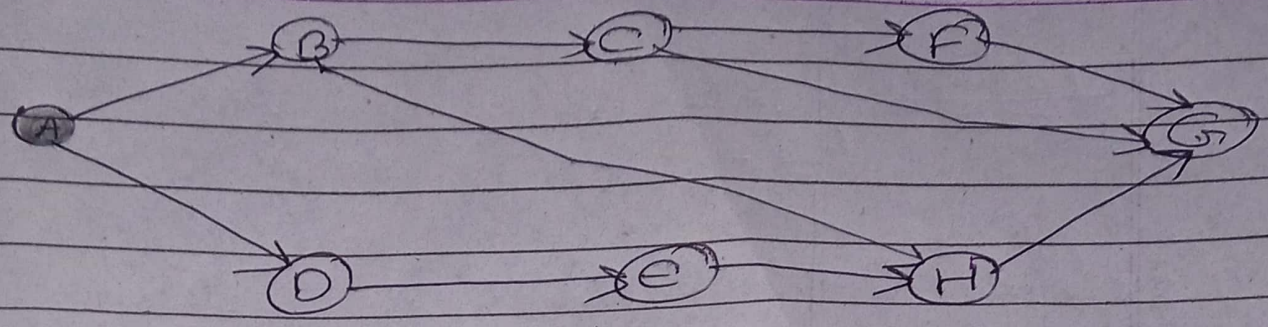
A → F

A, ~~E~~ E B, F ⇒ 8

10) Differentiate graph vs tree. What is topological sorting? Perform topological sort from below graph.



Date : .....
Page: .....



→ Difference:

	Graph	Tree
i)	It can have more than one path between nodes.	It can only have one path between vertices.
ii)	It can have loops.	Loops are not allowed.
iii)	It does not have a root node.	It has exactly one root node.
iv)	It follows network model.	It follows ~~heir~~ hierarchical model.
v)	Traversal techniques are breadth first search and depth first search.	Traversal techniques are pre-order, in-order and post-order.

Topological sorting is a linear ordering of its vertices such that for every directed edge (u,v) from vertex u to vertex v, u comes before v.