Saroj Dahal - 191725

DSA Tutorial.

1. What is the difference between data types and data structure?

Ans.

Data Types	Data Structure.
→ Data types are the types of the value that are assigned to the entites in the programming language.	→ Data structure deals with the schema and organization of data in a system so that it adheres space and time complexity for data.
→ It doesn't make concern for organization & structure of data.	→ It does concern; and is a methodology for efficient organization & structure of data.
→ Data-types itself cannot expand its nature; it acts as same mentioned in the programming language.	→ Any data structure can be expanded and implemented as per our requirement.
→ Example: int, char, float.	→ Example: stack, Queue, List, Graph, Tree etc.
→ They are more relevant to statically typed programming language.	→ They are relevant to any programming languages.

Saroj Dahal - 191725

2. What is abstract data type (ADT), what is the importance of ADT in software development?

Ans. ADT is a useful tool for specifying the logical operations properties of a datatype. It is a mathematical model with a collection of operations defined on the model by specifying the mathematical and logical properties of a datatype or structure, the ADT is a useful guideline to implement and a useful tool for programmer who wish to use datatype correctly.

ADT = data + operations allowed.

3. Write an algorithm to push and pop data on to a stack.

Ans. 1. Start

2. Declare all necessary variables:

i.e. stack_size, TOS = -1, item[stack_size], data.

3. Push operation.

3.1. check for full condition.

If (isfull()) then Display "stack is full".

else:

Read data.

TOS = TOS + 1

item[TOS] = data.

3.2. For next data repeat step 3.

4. POP operation.

4.1. Check for empty condition.

If (isempty()) then Display "stack is empty".

else

data ← item[TOS]

TOS ← TOS-1.

4. Convert the following infix expression to postfix A+(B*(-(D/E^F)*G)*H using stack.

Ans.

Steps	Scanned character	Stack control	Postfix expression
1.	A	empty	A
2.	+	+	A
3.	(	+(	A
4.	B	+(	AB
5.	*	+(*	AB
6.	(	+(*	ABC
7.	-	+(*-	ABC*.
8.	(	+(-(	ABC*
9.	D	+(-(	ABC*D.
10.	/	+(-(/	ABC*D.
11.	E	+(-(/	ABC*DE
12.	^	+(-(/^	ABC*DE
13.	F	+(-(/^	ABC*DEF.
14.	)	+(-	ABC*DEF^/
15.	*	+(-*	ABC*DEF^/
16.	G	+(-*	ABC*DEF^/G
17.	)	+	ABC*DEF^/G*-
18.	*	+*	ABC*DEF^/G*-
19.	H	+*	ABC*DEF^/G*-H
20.	-	-	ABC*DEF^/G*-H*+.

Hence, the postfix form of A+(B*C-(D/E^F)*G)*H is, ABC*DEF^/G*-H*+.

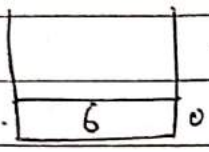
Saroj Dahal - 191725

5. Evaluate the following postfix expression using stack

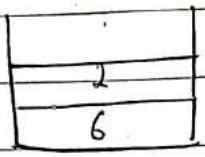
6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Ans. Here, TOS = -1.

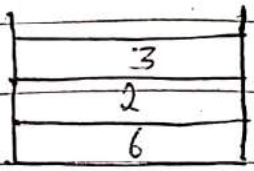
a. Push (6)



b. Push (2)



c. Push (3)



d. Push (+).

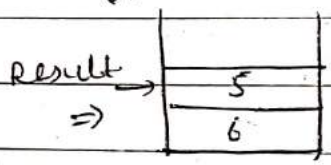
Pop 1 = 3

Pop 2 = 2.

∴ Result = Pop 2 + Pop 1

= 3 + 2

= 5.



e. Push (-)

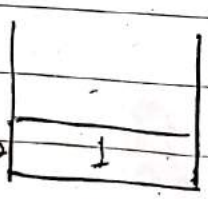
Pop 1 = 5

Pop 2 = 6.

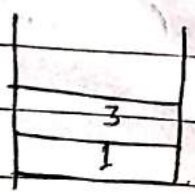
∴ Result = Pop 2 - Pop 1

= 6 - 5

= 1.

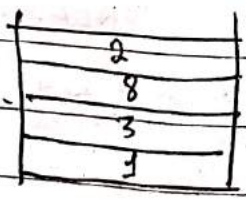


f. Push (3)



g. Push (8)

h. Push (2)



i. Push (/).

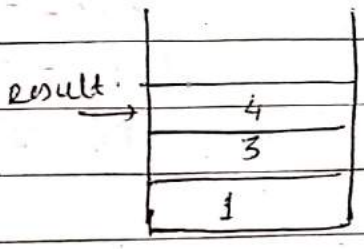
POP1 = 2

POP2 = 8

∴ Result = POP2 / POP1

= 8/2

= 4.



j. Push (+)

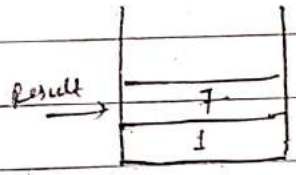
POP1 = 4

POP2 = 3

Result = POP2 + POP1

= 3+4

= 7.



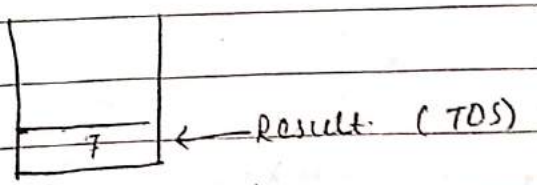
k. Push (*)

POP1 = 7

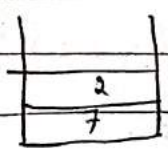
POP2 = 1.

Result = POP2 * POP1

= 7.



l. Push (2)



m. Push (/.)

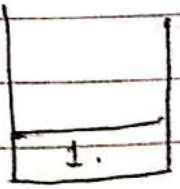
POP1 = 2

POP2 = 7.

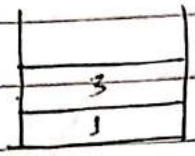
∴ Result = POP2 /. POP1

= 7 /. 2

= 1



n. Push (3)



o. Push (+)

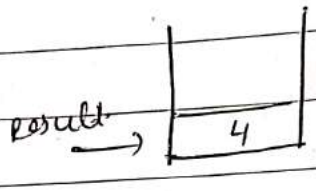
POP1 = 3

POP2 = 1.

∴ Result = POP2 + POP1

= 1+3

= 4.



p. POP. TOS

Result = 4.

6. Write an algorithm to Enqueue() and Dequeue() values in to queue.

Ans.

Enqueue():

1. If Rear = Max - 1 ; display "Queue is full and stop".
2. Read Data.
3. If Rear = front = -1.

Rear ← front ← front + 1.

Q [Rear] ← data.

else,

Rear ← Rear + 1.

Q [Rear] ← data.

Dequeue().

1. If front = -1. Display "Queue is empty and stop."
2. If Rear = front.

item ← Q [front]

Rear ← front ← -1

return item.

else,

item ← Q [front]

front ← front + 1

return item.

3. stop.

7. What are the limitation of linear queue, how this can be corrected using circular queue?

Ans. The limitation / problem in linear queue is storage utilization. Although there gets vacant space after dequeuing process, but because of the value pointed by front make the sense of being full.

To solve this problem of linear queue, we can make use of circular queue where we evaluate the value of rear and front in different manner else than just shifting the value.

In enqueue & dequeue, (or front)

we calculate; front = (Rear+1) % MaxSize

and Rear = (Rear+1) % MaxSize.

8. What is priority queue explain its types.

Ans. A priority queue is a data structure in which each element has been assigned a value called the priority of the element and an element can be inserted or deleted not only from front but at any position of the queue. A priority queue is a collection of elements such that each element has been assigned an implict or explicit priority and such that the order in which elements are deleted follows following rules:

a. An element of highest priority is processed first before any element of lower priority.

b. Two elements with the same priority are processed in the order of arrival in queue.

The priority of element is decided by its value known as implicit priority and number assigned with each element is explict priority.

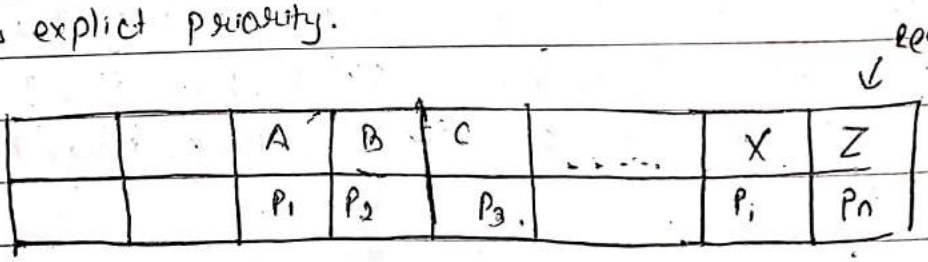


fig: Priority Queue.

Here, an element X of priority Pi may be deleted before an element pointed by front.

Types of priority queue:

a. Ascending priority queue.

b. Descending priority queue.

In Ascending priority queue, elements can be inserted in any order but while deleting, always element having smallest priority number will be deleted whereas in descending priority queue, element can come in any order but while removing an element, the element associated with highest priority number will be removed.

g. What is the major advantage of using dynamic linked list instead of using static list (array)?

Ans. There are several advantage of using dynamic linked list instead of static list (array), and some of the major advantages are:

- The size of array is fixed; so we must always make sure of its max size while working with it; whereas, dynamic linked list has no finite size, and upper limit is rarely reached.
- Inserting & deleting nth element in array is difficult; as we have to re-order the position of other elements; whereas in linked list it is very easy process, as we just have to ~~connect~~ manage the working node to its previous one and next node only; and order lining of data will be same as the intial state.

In gist:

→ Dynamic size.

→ Ease of insertion/deletion.

10. Write an algorithm to insert a node at nth position in doubly circular linked list.

Ans.

Algorithm:

1. temp = (DCLL*) malloc (sizeof (DLL)).
2. temp → info = data.
3. p = first.
4. while (n > 2)
   p = p → next
   n--
5. end while.
6. p → next → prev = temp.
7. temp → next = p → next.
8. p → next = temp.
9. temp → prev = p.

11. Write an algorithm to delete the last node of a linear circular linked list.

Ans.

```
struct singly_circular {
    int data;
    struct singly_circular *next;
}

typedef struct singly_circular SCLL.
SCLL *p, *temp, *q
temp = (SCLL*) malloc(sizeof(SCLL)).
```

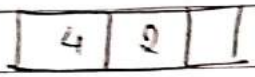
Deletion of last node:

1. P = first.
2. while (P → next → next != first)
   P = P → next.
3. End while.
4. temp = P → next.
5. P → next = first.
6. free (temp).

12. Write an algorithm to add two polynomial using Linked List

Ans: We can add two polynomial using Linked List ; where what we do is, in each node we keep the coefficient, exponent and the next term of the polynomial.

For Ex: $4x^2$ can be represented as

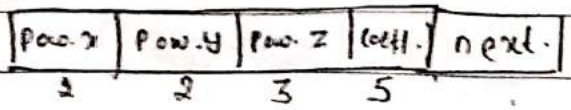


where 4 = coeff.
2 = exponent.

Ex: 2: $3x^4 + 8x^2 + 6x + 8$

For Multivariable polynomial,

Ex: $5x^2y^2z^3$.



polynomial

Let P1 and p2 represent a linked list.

step 1 : Loop around all values of linked list and follow step 2 and 3.

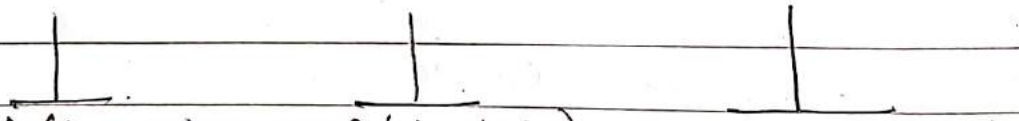
step 2 : If the value of a node's exponent is greater; copy this node to result node and head towards next node.

step 3 : If the value of both node's exponent is same; add the coefficients and then copy the added value with node to the result.

step 4 : Print the resultant node.

13. Write code and algorithm for Tower of Hanoi (TOH).

Ans.



Algorithm :

1. Move the top (n-1) disks from "A" to peg "B" using C as auxiliary.
2. Move the nth disk from peg 'A' to peg 'C'.
3. Move (n-1) disk from peg 'B' to peg 'C' using peg 'A' as auxiliary one.

Program :

```
#include <stdio.h>
#include <conio.h>
```

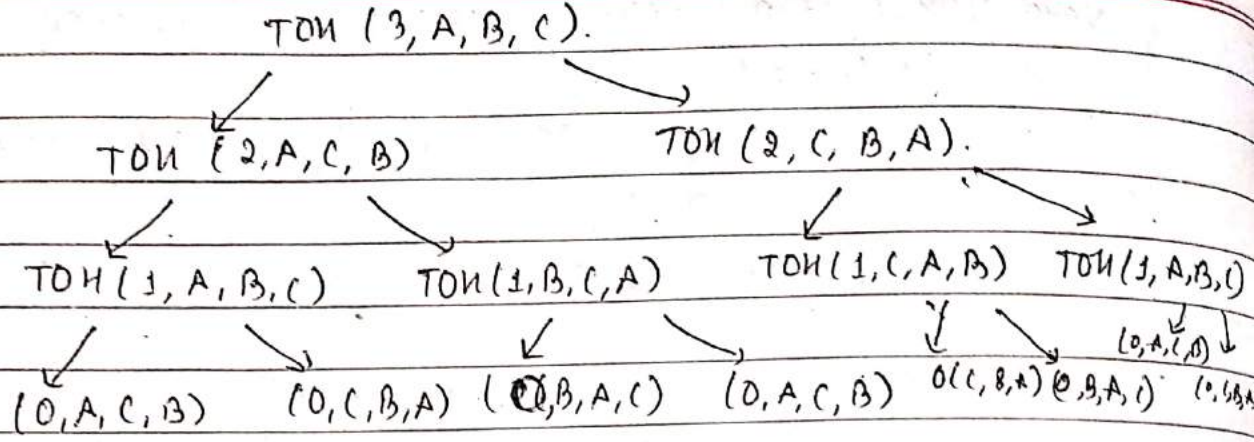
```
void TOH ( int n, char source, char temp, char dest) {
    if (n==1) {
        cout << "Move disk " << n << " from " << source << "to" <<
        dest;
        return;
    }
    TOH (n-1, source, dest, temp);
    cout << "Move disk " << n << source << "to" << dest;
    TOH (n-1, temp, source, dest);
}

void main() {
    TOH (3, 'A', 'B', 'C');
}
```

Output:

```
Move disk 1 from A to C.
Move disk 2 from A to B.
Move disk 1 from C to B.
Move disk 3 from A to C.
Move disk 1 from B to A.
Move disk 2 from B to C.
Move disk 1 from A to C.
```

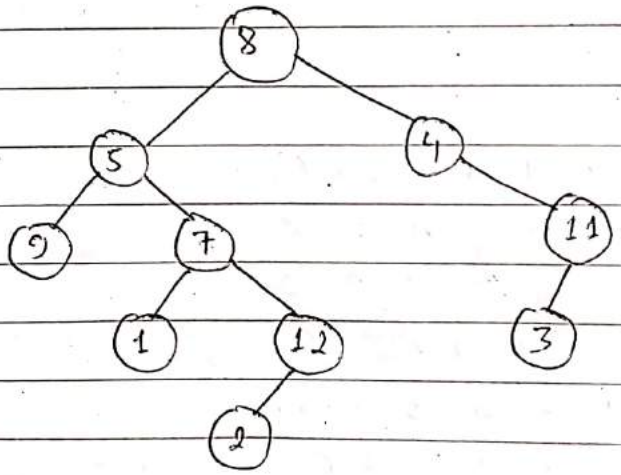
14. Draw recursive tree for TOH where no. of disk n=3.

P.T.O.



Recursion tree on Tower of Hannoi
(for 3 disks).

15. Perform In-order, pre-order and post-order tree traversal.



Pre-order : 8, 5, 9, 7, 1, 12, 2, 4, 11, 3
In-order : 9, 5, 1, 7, 2, 12, 8, 4, 3, 11.
Post-order : 9, 1, 2, 12, 7, 5, 3, 11, 4, 8.

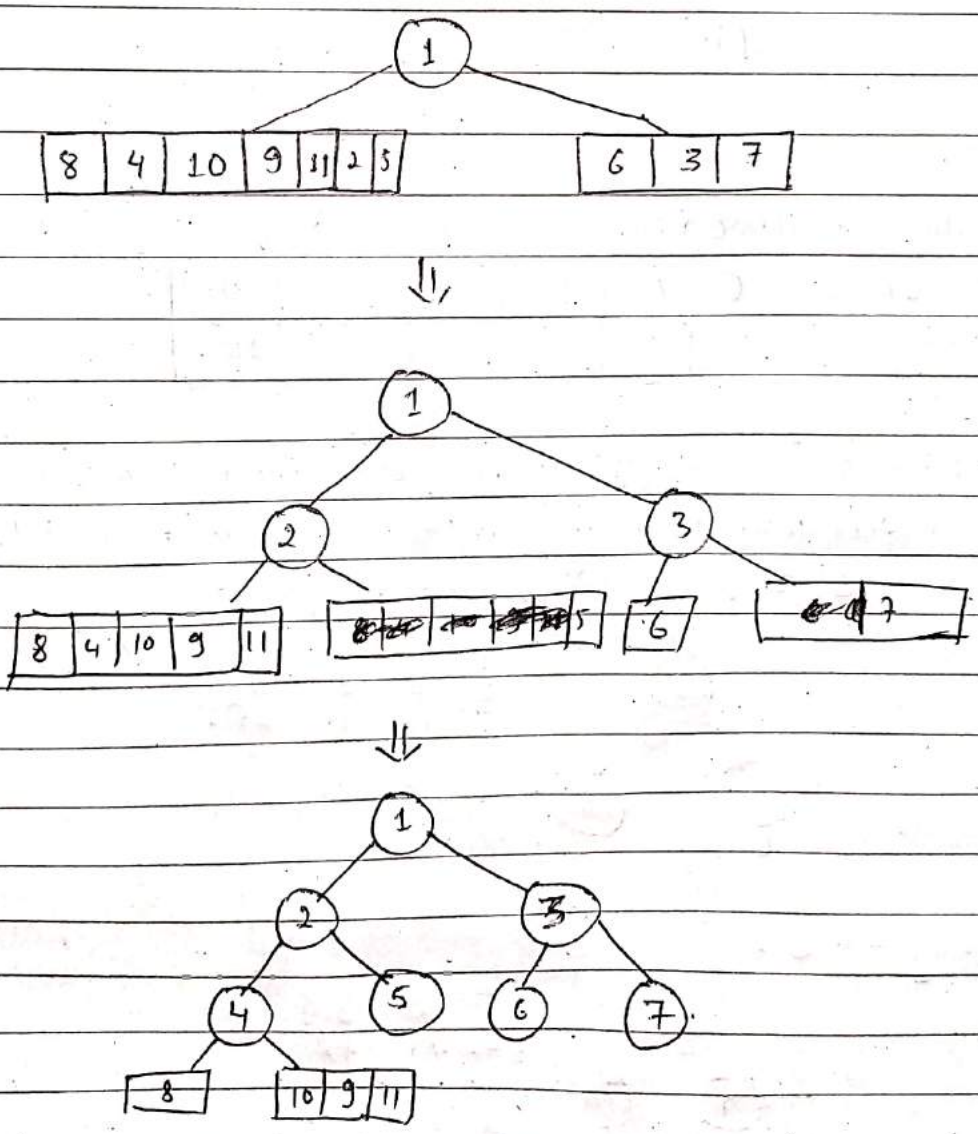
16. Construct the tree from given values;

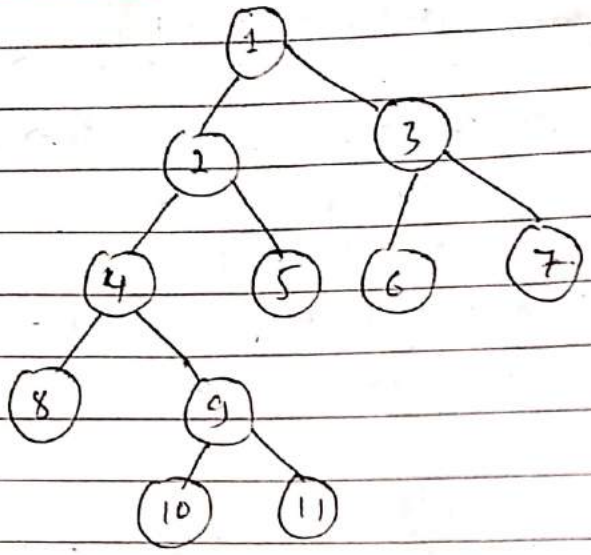
Preorder Traversal : 1, 2, 4, 8, 9, 10, 11, 5, 3, 6, 7. ⇒ (VLR)

Inorder Traversal : 8, 4, 10, 9, 11, 2, 5, 1, 6, 3, 7. ⇒ (LVR)

left | root | right

Ans. From pre-order traversal, 1 is the root node and from inorder traversal, everything on left of 1 is left sub-tree and everything on its right is right subtree.

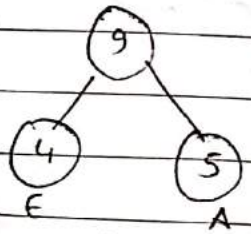




17. Construct Huffman Tree for following values:

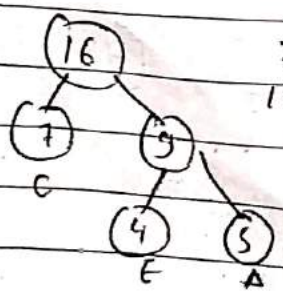
Value	E	A	C	F	D	B.
Frequency	4	5	7	12	15	25.

Ans. ~~Sol~~. As per the Huffman tree construction rule, we add up two small frequency node and attach its data as a child.
Here, 'E' & 'A' has min^m frequency. so;



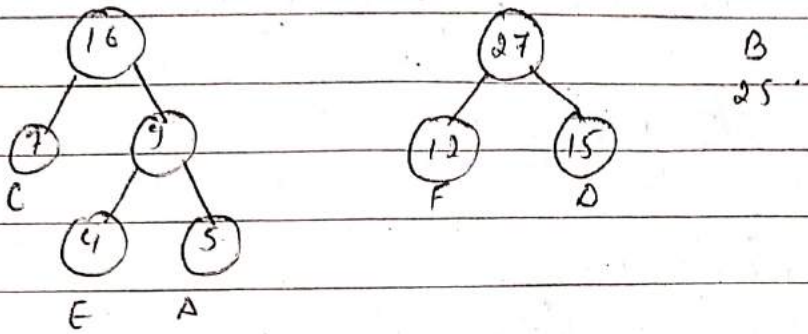
C F D B
7 12 15 25.

Adding 9 & 7,

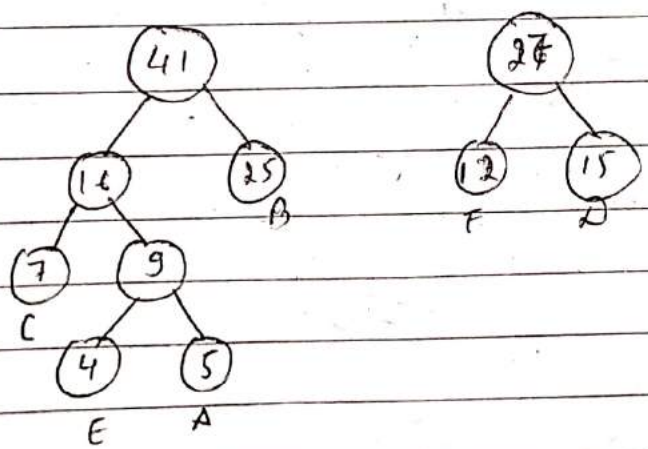


F D B
12 15 25

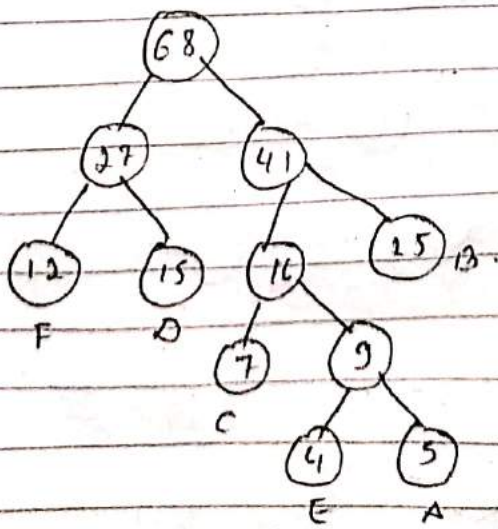
Here, again the min^m weights are 12 and 15.



The min^m weights are 16 and 25.

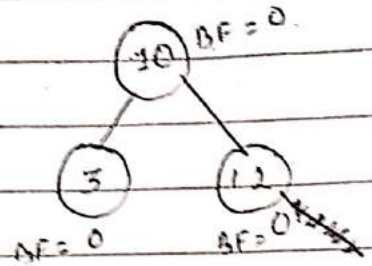


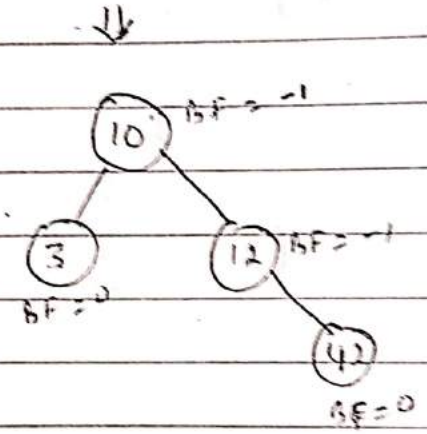
Lastly,



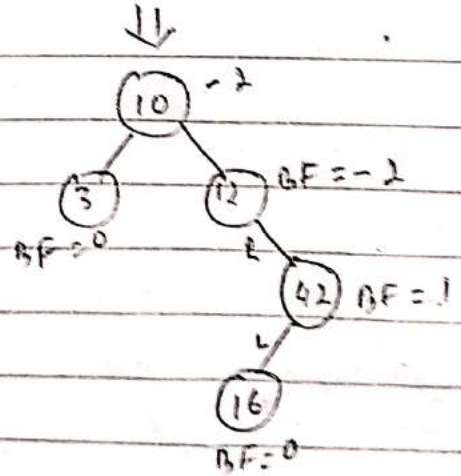
18. Construct AVL tree for following values:
10, 12, 3, 42, 16, 18, 27, 30, 6, 89, 22, 7, 33.

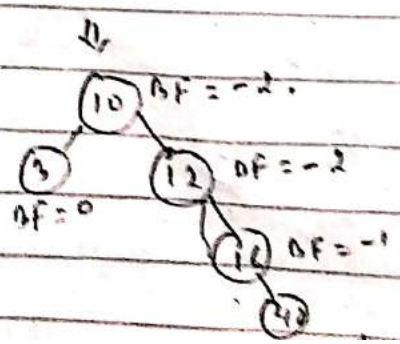
Ans.

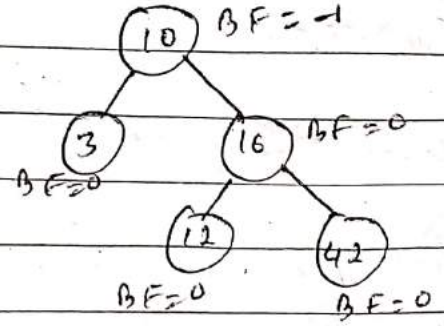




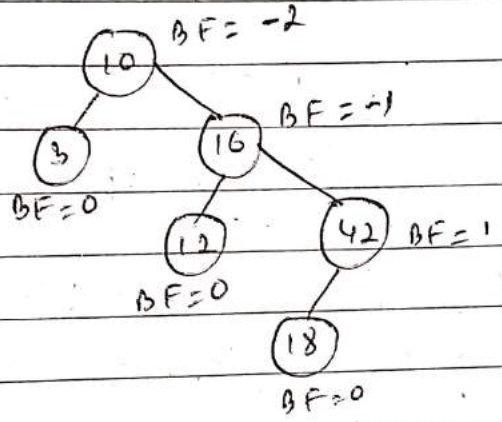
Insert 16.

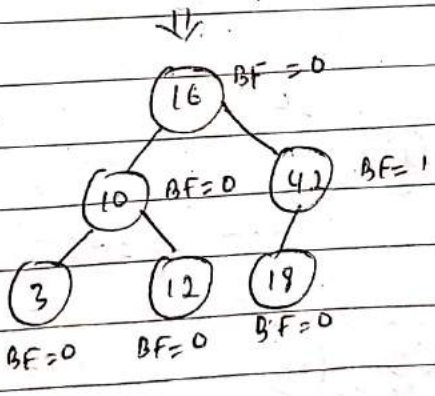




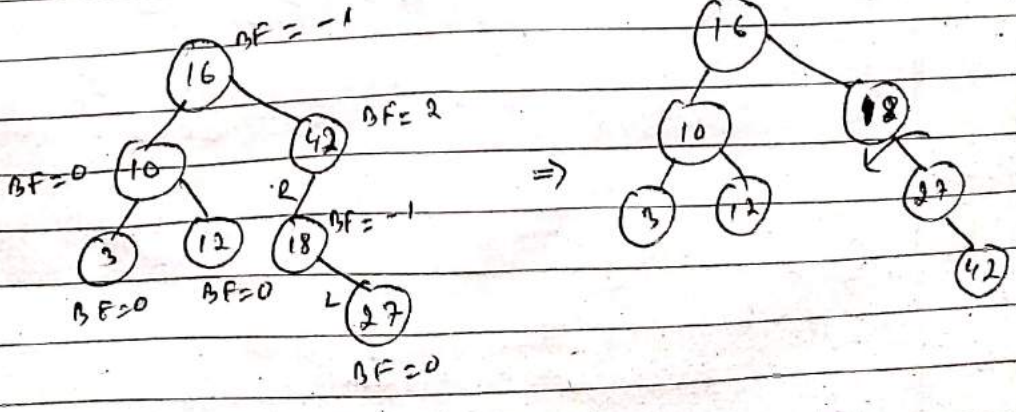


Insert 18.

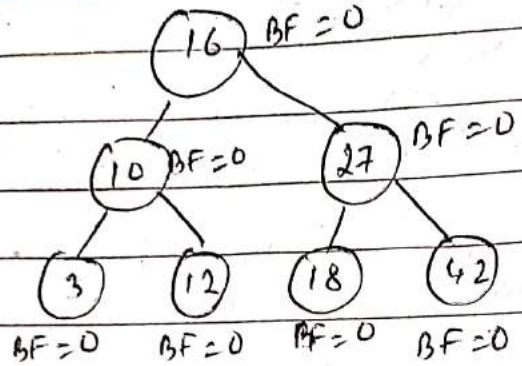




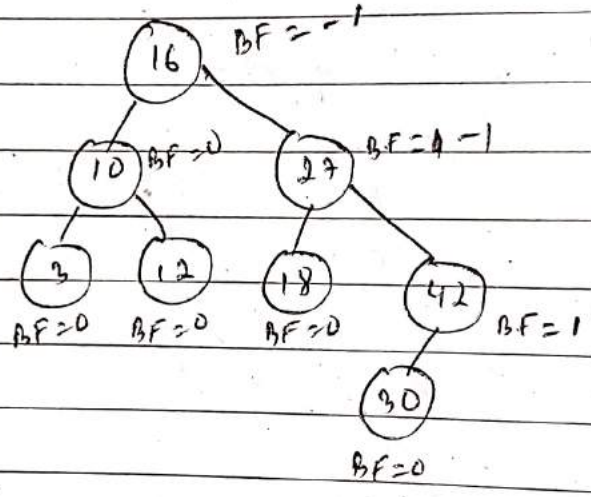
Insert 27.



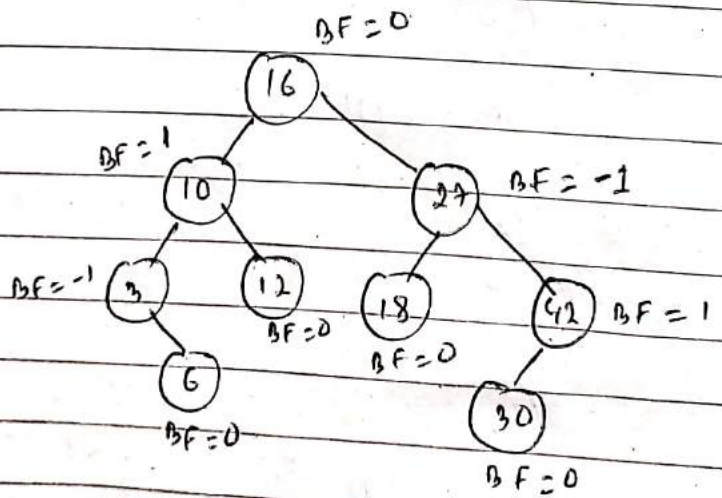
Saroj Dahal - 191725



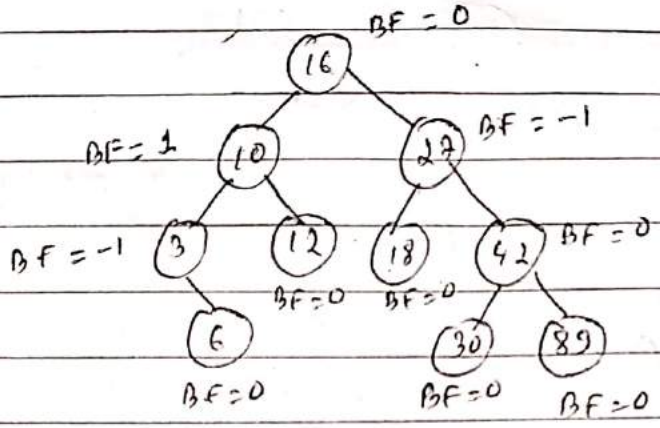
Insert 30.



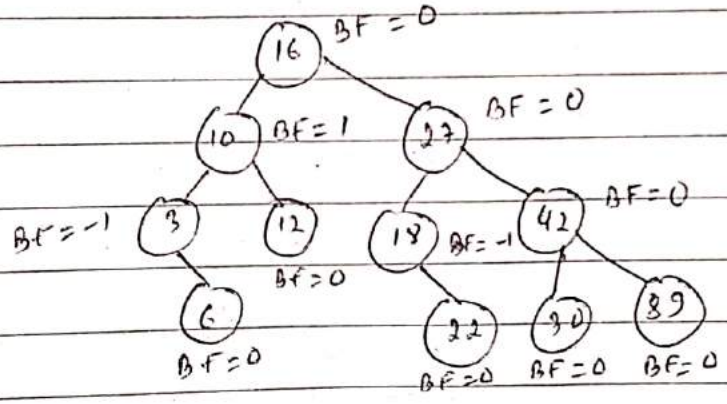
Insert 6



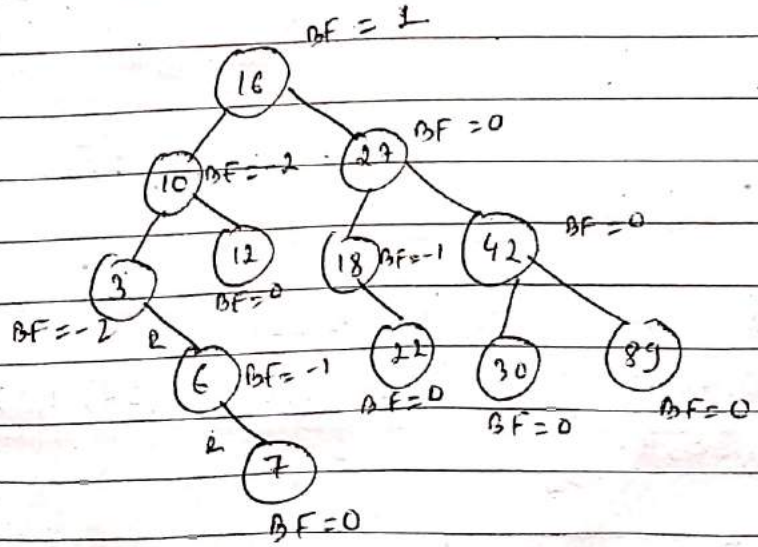
Insert 89

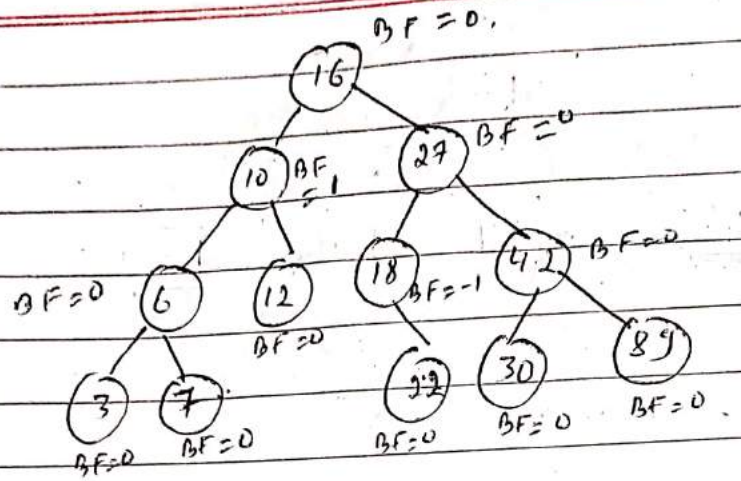


Insert 22

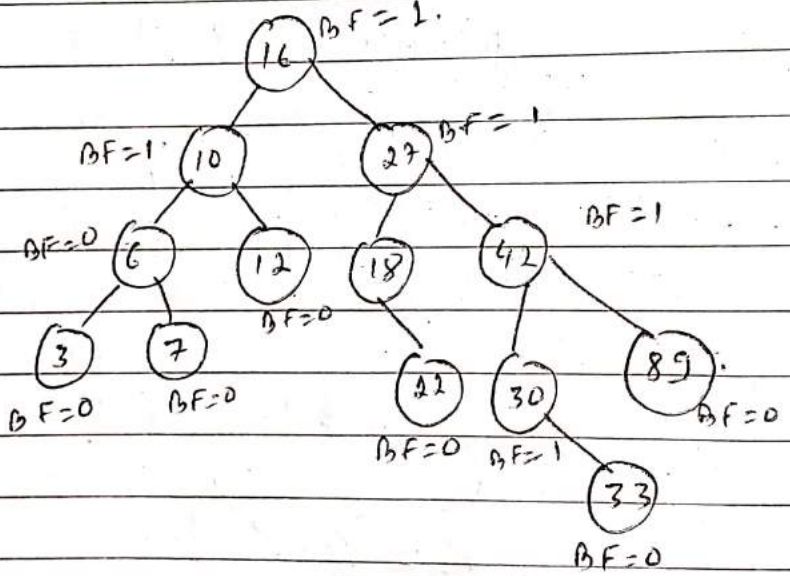


Insert 7.





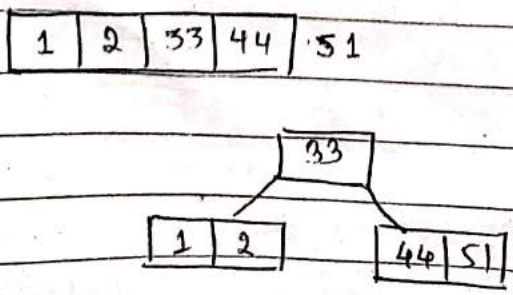
Insert 33.

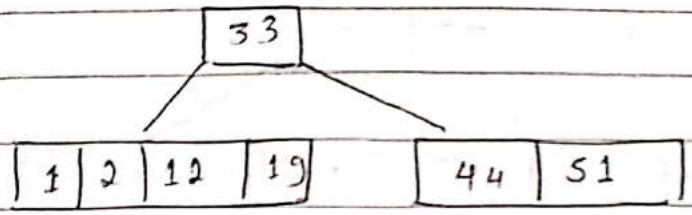


19. Construct a B-tree of order 5.

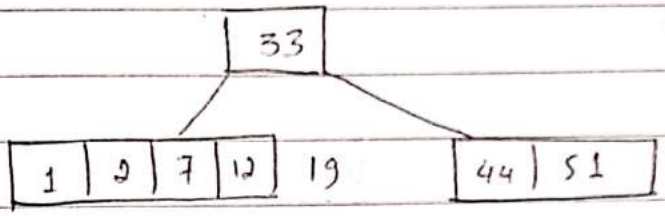
1, 9, 33, 44, 51, 12, 19, 7, 6, 82, 91, 45, 62, 11, 13, 70, 20, 29, 39, 59, ~~69~~, 83, 35, ~~4~~.

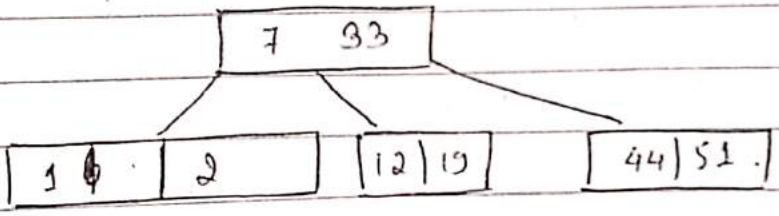
Ans.



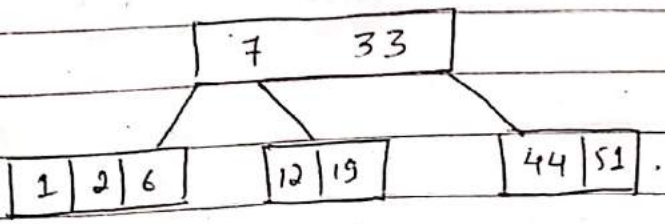


Insert 7.

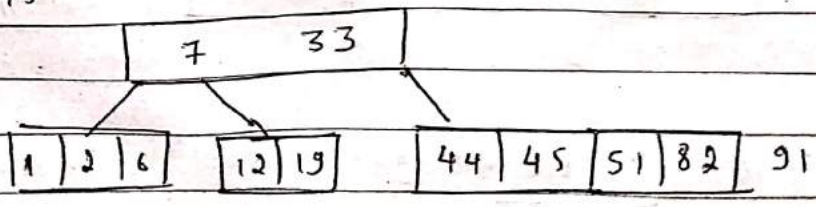




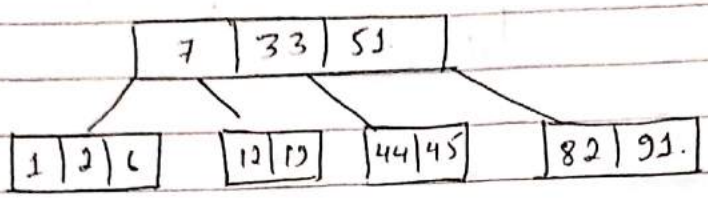
Insert 6.



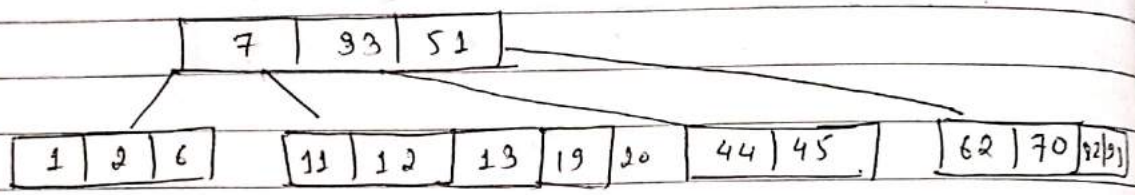
Insert 82, 91, 45

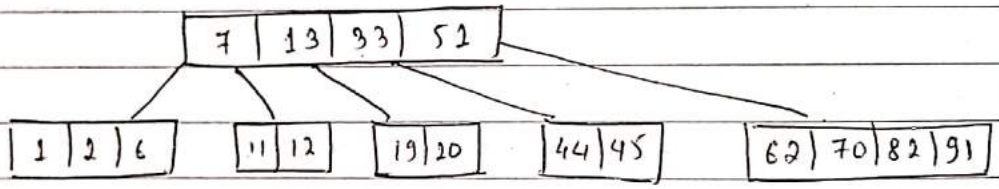


Saroj Dahal - 191725



Insert 62, 11, 13, 70, 20

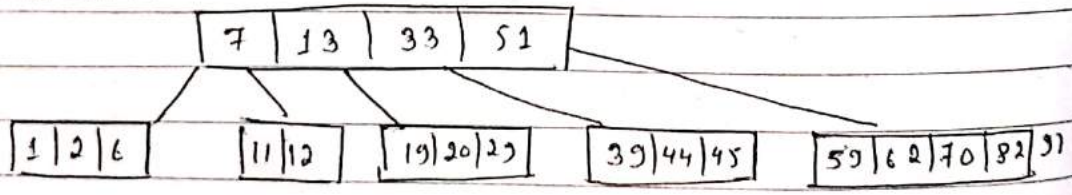




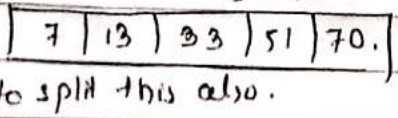
20.

Insert 29, 39, 59.

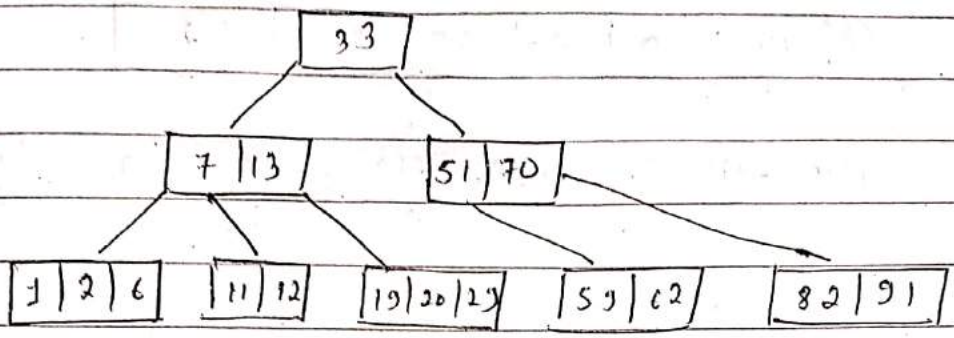
ANS.



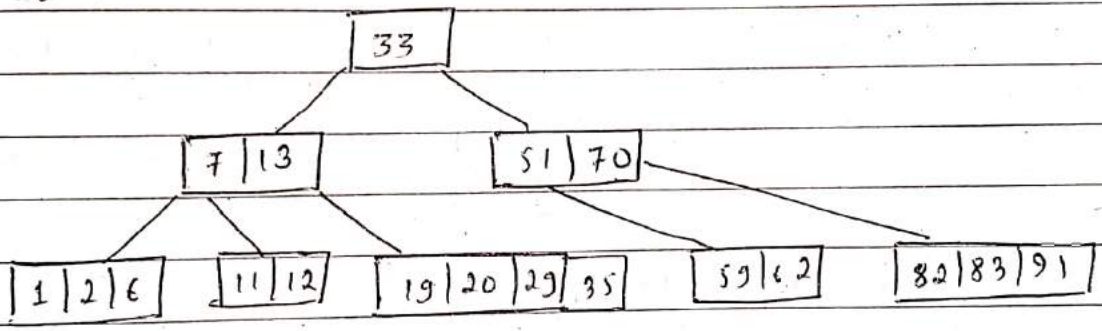
At the top node we will get,



So, we have to split this also.



Insert 35, 35



20. Use Binary search technique to find a key = 40 in this following sequence.

1, 8, 12, 13, 24, 30, 34, 40, 50, 60, 80, 90, 92.

Ans. In array form, we can present sequence as;

	0	1	2	3	4	5	6	7	8	9	10	11	12
Array;	1	8	12	13	24	30	34	40	50	60	80	90	92

$$mid = \frac{start + end}{2} = \frac{0+12}{2} = 6. \Rightarrow 34.$$

Since, key > mid. i.e. 40 > 34.

So, we neglect the value from start to mid and will find in the index greater than mid.

7	8	9	10	11	12.
40	50	60	80	90	92.

mid = $\frac{\text{start} + \text{end}}{2} = \frac{7+12}{2} = \frac{19}{2} \approx 9.$

Here, key < mid.

i.e. 40 < 60.

So, we discard the search from index greater than mid. and will search in index less that of mid index.

7	8
40	50

mid.

Also, mid = $\frac{7+8}{2} \approx 7.$

Here, key = mid.

i.e. 40 = 40. Hence our required key is found.

21. Input the following data in to hash table of size 10.

Input sequence : 1, 27, 6, 87, 47, 7, 8, 17, 37, 67.
Hash Function : key mod 10.
Collision Handling : H(key) + f(i) where f(i) = i.

Here, 1 % 10 = 1, 47 % 10 = 7. 37 % 10 = 7
27 % 10 = 7 7 % 10 = 7. 67 % 10 = 7.
6 % 10 = 6 8 % 10 = 8
87 % 10 = 7. 17 % 10 = 7.

0	1
1	67.
2	37.
3	17.
4	8
5	7.
6	6
7	27
8	87.
9	47.

For, 87 % 10 = 7.
∴ H(key) + f(i)
= 7 + 1 [for i = 1]
= 8

For, 47 % 10 = 7
∴ H(key) + f(i)
= 7 + 2 [for i = 2]
= 9.

For, 7 % 10 = 7
∴ H(key) + f(i)
= 7 - 2 [for i = -2]
= 5.

For, 8 % 10 = 8.
∴ H(key) + f(i)
= 8 - 4 [for i = -4].
= 4

For, 17 % 10 = 7.
∴ H(key) + f(i)
= 7 - 4 [for i = -4]
= 3.

For, 37 % 10 = 7
∴ H(key) + f(i)
= 7 - 5 [for i = -5]
= 2

For, 67 % 10 = 7.
∴ H(key) + f(i)
= 7 - 6 [for i = -6]
= 1.

Saroj Dahal - 191725

Input sequence : 1, 27, 6, 87, 47, 7, 8, 17, 37, 67.

Hash function : key mod 10.

Collision Handling : H(key) + f(i) where, f(i) = i*i.

Here,

1 % 10 = 1.

27 % 10 = 7.

6 % 10 = 6.

87 % 10 = 7. ; (7+4) % 10 = 1 ; (7+9) % 10 = 6 ; (7+16) % 10 = 3

47 % 10 = 7 ; (7+25) % 10 = 2.

7 % 10 = 7 ; (7+36) % 10 = 3 ; (7+49) % 10 = 6 ; (7+64) % 10 = 1 ;

(7+81) % 10 = 8 ;

~~(7+100) % 10 = 7 ;~~

~~(7+121) % 10 = 8 ;~~

~~(7+144) % 10 = 3.~~

8 % 10 = 8 ; (8+1) % 10 = 9

17 % 10 = 7. ; (-

37 % 10 = 7

67 % 10 = 7

Input sequence = 1, 27, 6, 87, 47, 7, 8, 17, 37, 67.
Hash function = key mod 10
Collision handling = H1(key) + f(i) where, f(i) = i * H2(key).
and H2(key) = 7 - (key mod 7).

Here,

1 % 10 = 1

27 % 10 = 7

6 % 10 = 6

87 % 10 = 7 $\Rightarrow$ (7 + f(0)) = 7 + (0 * H2(87)) = 7

$\Rightarrow$ (7 + f(1)) = 7 + (1 * H2(87))

= 7 + (1 * (7 - (87 mod 7))

= 7 + (1 * (7 - 3))

= 7 + (1 * 4)

= 11 % 10 = 1.

$\Rightarrow$ (7 + f(2)) = 7 + [2 * H2(87)]

= 7 + [2 * (7 - (87 mod 7))]

= 7 + [2 * (7 - 3)]

= 7 + [2 * 4]

= 7 + 8

= 15 % 10 = 5.

Hy, 47 % 10 = 7 $\Rightarrow$ (7 + f(0)) = 7 + [0 * H2(47)]

= 7 + 0

= 7.

$\Rightarrow$ (7 + f(1)) = 7 + [1 * H2(47)]

= 7 + [7 - (47 mod 7)]

= 7 + [7 - 5]

= 9 % 10 = 9

Saroj Dahal - 191725

Date
Page

$7 \%. 10 = 7. \Rightarrow [7 + f(1)] = [7 + 1 * H2(7)]$
$= [7 + (7 - (7 \bmod 7))]$
$= [7 + (7 - 0)]$
$= 14 \%. 10 = 4$

$8 \%. 10 = 8$

$17 \%. 10 = 7 \Rightarrow [7 + f(1)] = [7 + 1 * H2(17)]$
$= [7 + (7 - (17 \bmod 17))]$
$= [7 + (7 - 3)]$
$= 7 + 4$
$= 13 \%. 10 = 3$

$37 \%. 10 = 7 \Rightarrow [7 + f(1)] = [7 + 1 * H2(37)]$
$= [7 + (7 - (37 \bmod 7))]$
$= [7 + (7 - 2)]$
$= 12 \%. 10 = 2$

$67 \%. 10 = 7 \Rightarrow [7 + f(1)] = [7 + 1 * H2(67)]$
$= [7 + (7 - (67 \bmod 7))]$
$= [7 + (7 - 4)]$
$= 7 + 3$
$= 10 \%. 10 = 0$

67	0
1	1
37	2
17	3
7	4
87	5
6	6
27	7
50	8
47	9

22. Sort the following sequence of number using quick sort and Heap sort.

12, 2, 24, 56, 19, 17, 13, 90, 44, 61, 27, 45.

Ans. Quick sort

12, 2, 24, 56, 19, 17, 13, 90, 44, 61, 27, 45.

Let Pivot = 56.

1st pass: 12, 2, 24, (56), 19, 17, 13, 90, 44, 61, 27, 45.
↑d ↑d ↑up.

12, 2, 24, (56), 19, 17, 13, 45, 44, 61, 27, 90
↑d ↑d ↑up ↑up

12, 2, 24, (56), 19, 17, 13, 45, 44, 27, 61, 90
↑up ↑d

12, 2, 24, 27, 19, 17, 13, 45, 44, 56, 61, 90.

Let Pivot = 13

12, 2, 24, 27, 19, 17, (13), 45, 44, 56, 61, 90.
↑d ↑d ↑up

13, 2, 24, 27, 19, 17, 12, 45, 44, 56, 61, 90.

Let Pivot = 27.

13, 2, 24, (27), 19, 17, 12, 45, 44, 56, 61, 90
↑d ↑up ↑d ↑up

13, 2, 24, 12, 19, 17, 27, 45, 44, 56, 61, 90

Let pivot = 24.

13, 2, (24) 12, 19, 17, 27, 45, 44, 56, 61, 90

↑d ↑up ↑d ↑up

13, 2, 17, 12, 19, 24, 27, 45, 44, 56, 61, 90.

Let pivot = 44.

13, 2, 17, 12, 19, 24, 27, 45, (44) 56, 61, 90.

↑d ↑up ↑d ↑up

13, 2, 17, 12, 19, 44, 45, 27, 56, 61, 90.

Let pivot = 19.

13, 2, 17, 12, (19) 44, 45, (27) 56, 61, 90.

↑d ↑up ↑d ↑up

Let pivot = 13.

(13), 2, 17, 12, 19, 44, 45, 27, 56, 61, 90

↑d ↑d ↑up ↑up

(13) 2, 12, 17, 19, 44, 45, 27, 56, 61, 90

↑d ↑up ↑d

12, 2, 13, 17, 19, 44, 45, 27, 56, 61, 90

Let pivot = 2,

12, (2) 13, 17, 19, 44, 45, 27, 56, 61, 90.

↑d ↑up ↑up

2, 12, 13, 17, 19, 44, 45, 27, 56, 61, 90.

Let pivot = 44.

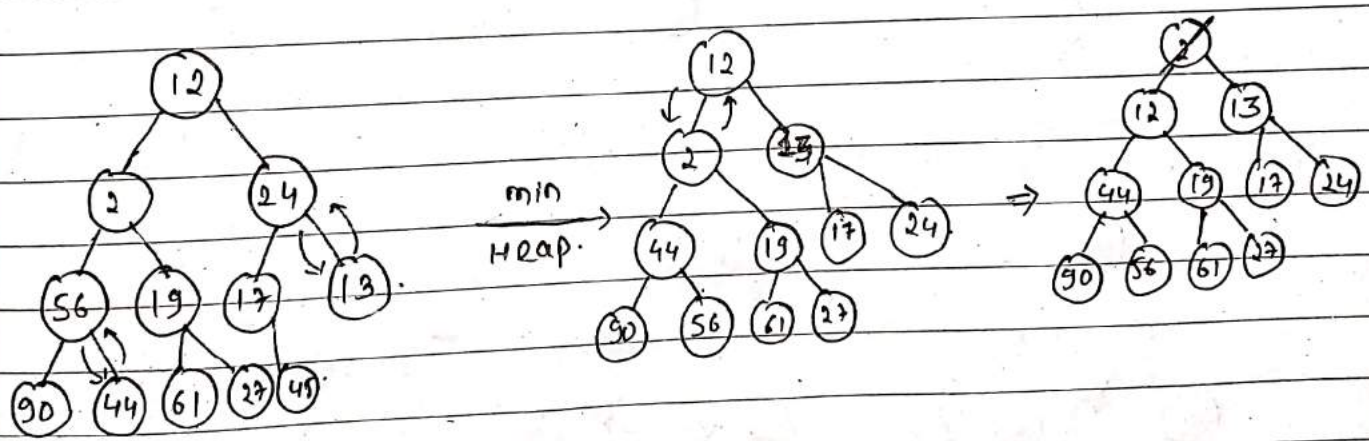
2, 12, 13, 17, 19, (44), 45, 27, 56, 61, 90.

2, 12, 13, 17, 19, (44), 27, 45, 56, 61, 90.

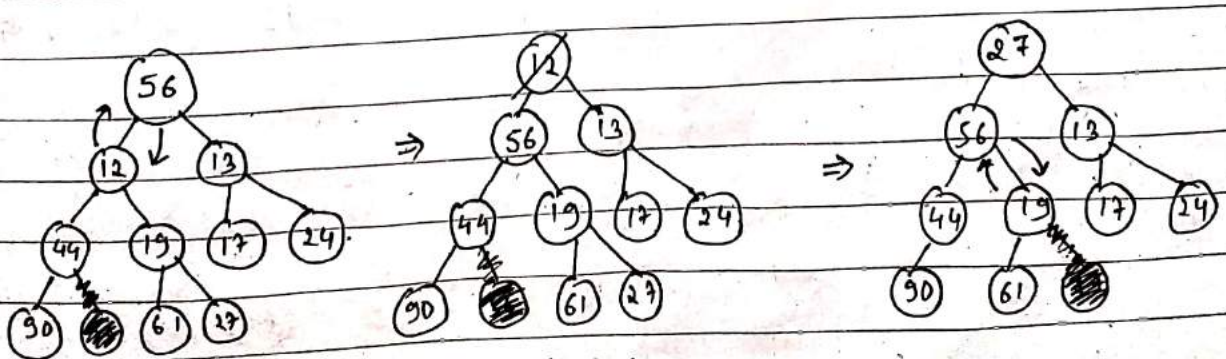
2, 12, 13, 17, 19, 27, 44, 45, 56, 61, 90.

In this way, the list is sorted.

Heap sort :

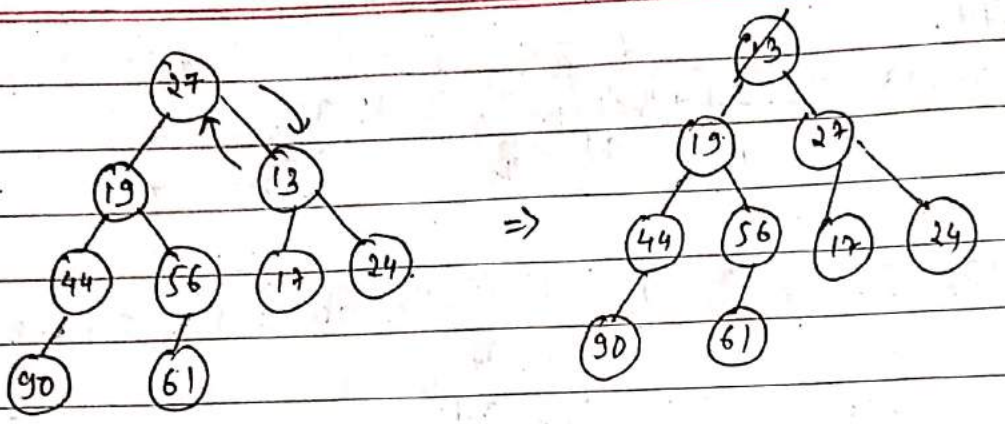


⇒ Sorted list = 2,

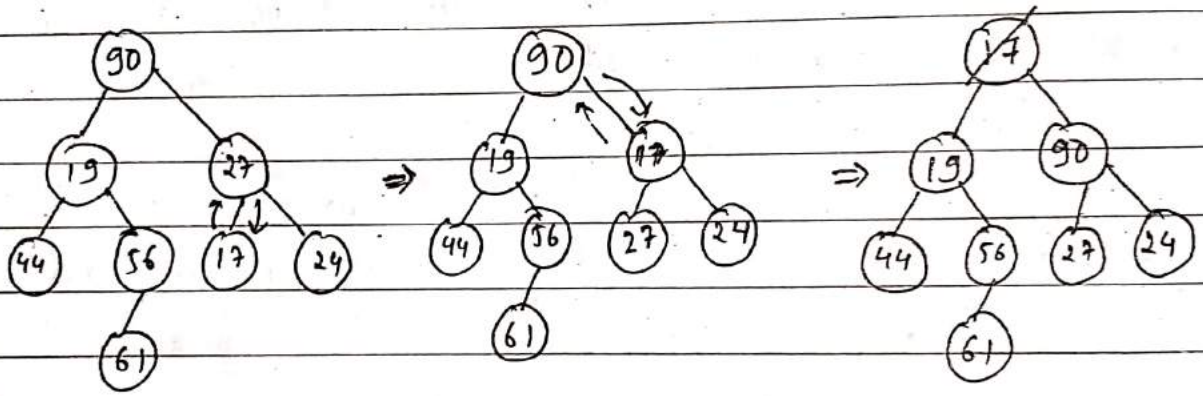


Sorted list : 2, 12,

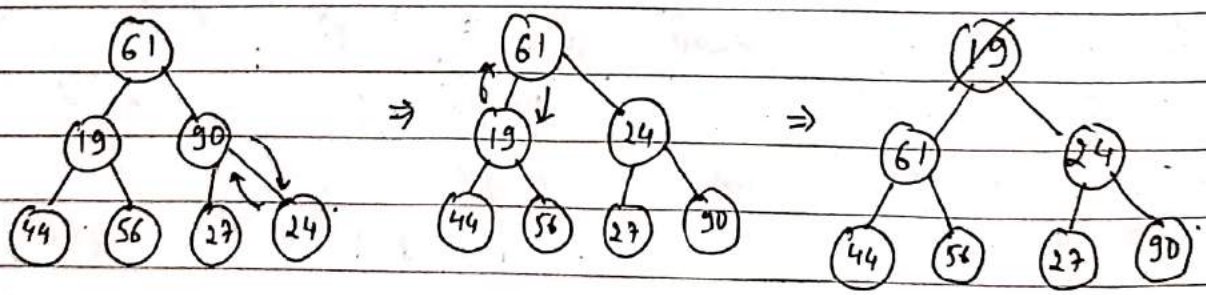
Saroj Dahal - 191725



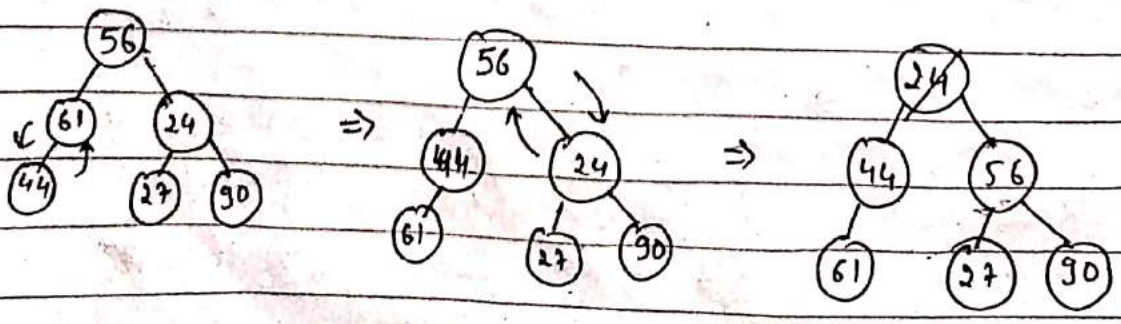
Sorted list :
2, 12, 13



Sorted list : 2, 12, 13, 17

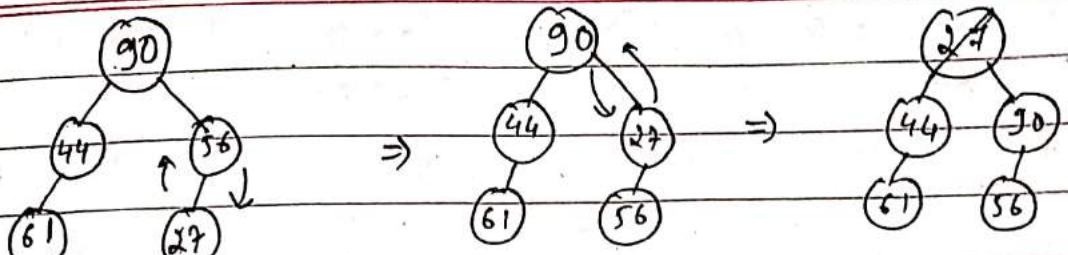


Sorted list : 2, 12, 13, 17, 19.

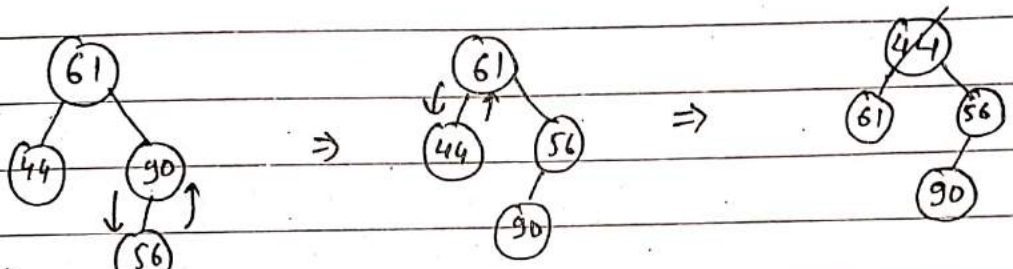


Sorted list : 2, 12, 13, 17, 19, 24

Saroj Dahal - 191725

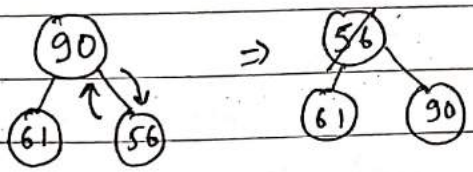


Sorted list : 2, 12, 13 17, 19, 24, 27.



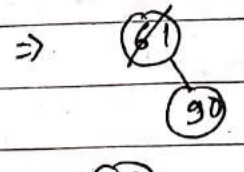
Sorted list :

2, 12, 13, 17, 19, 24, 27, 44



Sorted list :

2, 12, 13, 17, 19, 24, 27, 44, 56



∴ Sorted list is,

2, 12, 13, 17, 19, 24, 27, 44, 56.

23. What is sorting explain the difference between following:
a) Internal and External sorting.
b) Stable and Unstable sort.

Ans: Sorting: The arrangement of data / elements either in increasing order (ascending) or decreasing (descending) order is called sorting.

Difference betⁿ internal and external sorting is;

In internal sorting all the data to sort is sorted in memory at all times while sorting is in progress. Whereas, in external sorting data is stored outside memory (like on disk) and only loaded into memory in small chunks. External sorting is usually applied in cases when data can't fit into memory entirely.

So in internal sorting we can do shell sort, bubble sort - just by accessing whatever array elements we want at whatever moment we want. ~~Sam~~ whereas, in external sorting we cannot do that, as our data are not entirely on memory; so for external sorting we have to deal with loading and unloading chunks of data in optimal manner.

Difference betⁿ stable and unstable sort is,

If the sorting algorithm maintains the relative order of numbers / records then it is known as stable sort. If it doesn't retain the order then it is unstable sort.

Lets take an example and understand stable and unstable sorting algorithm.

Saroj Dahal - 191725

Date
Page

Lets say our inputted data in array is;

3, 2, 4, 5, 7, 6, 4

Note: One 4 is in a different color, to denote that the bold 4 was entered first and then the non-bold 4 was entered.

And, lets do sorting in asending order. On sorting the array there can be two outputs:

1st output: 2, 3, 4, 4, 5, 6, 7.

2nd Output: 2, 3, 4, 4, 5, 6, 7.

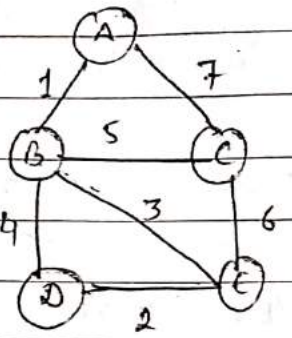
Well, there is a slight difference in the output, the thing is, the bold value 4 once is shifted at first place in first output, whereas in the second time; that same value is placed beside another four.

According to the input, the bold4 was entered first and then the non-bold 4 was entered. Hence, the first output is a stable sorting algorithm whereas the latter one is an unstable sorting algorithm.

Example of stable sorting algorithm: Bubble sort, Insertion sort, Merge sort, Counting sort etc.

Example of unstable sorting algorithm: Selection sort, Heap sort, Quick sort etc.

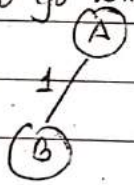
24 Find the MST for following graph using Prims and Kruskal Algorithm.



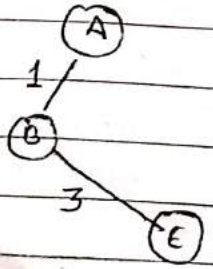
By

Ans. Prim's Algorithm

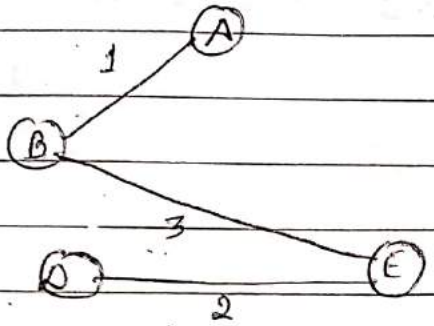
Lets take vertex A. It's adjacent nodes are B & C. Since AB has min^m cost, so we go with it.



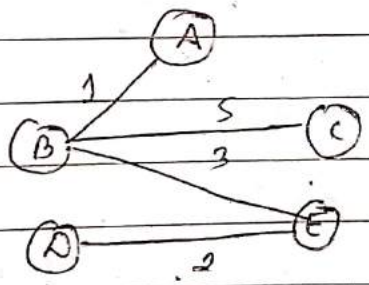
From vertex B, we have BC (node) of weight 5, BE edge of cost 3, BD edge of cost 4. Since the min^m cost is 3. so we will go along with edge BE.



From vertex E, we have DE edge of weight 2, CE edge of weight 6; so we will go along min^m cost i.e. through edge DE.



Here we cannot connect edge BD, as it will make cyclic; which voilate the rule of spanning tree. So, lets take vertex B. From vertex B we have one possible edge BC, which again has min^m weight as compared to other, so we will take this one.

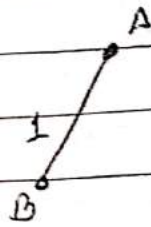


Now, if we connect any of the edges, then it will form cyclic. So, this is our obtained min^m tree from prims algorithm.

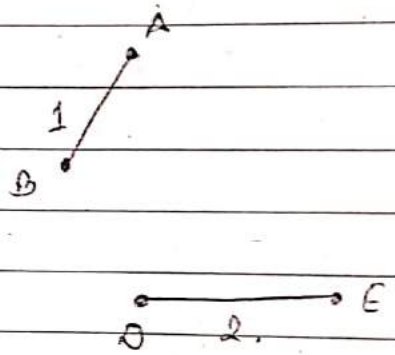
By kruskal Algorithm

In kruskal Algorithm, we connect the nodes in ascending order but while connecting the nodes, we make sure that it is not forming any cycle or loop. If we found the edge to form cycle, we discard that

Here, the min^m cost is 1. So we first construct AB edge.

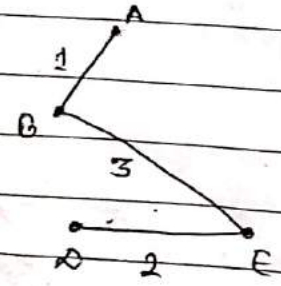


the second min^m cost is 2, which is denoted by edge DE. so we construct that path of MST.



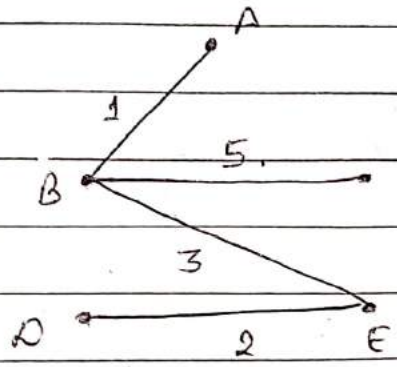
The tree can be disconnected in its different steps in krustal algorithm but at the last part, all ~~of~~ them will be connected without forming any cycle.

Now, lets take the min^m cost, which is 3 and add it into our tree.



Our, another min^m cost is 4 which is of BD edge, but if we add that edge in our tree then it forms cycle so we discard that and move to the next one.

Our next min^m cost is 5. So, lets connect the edge BC.

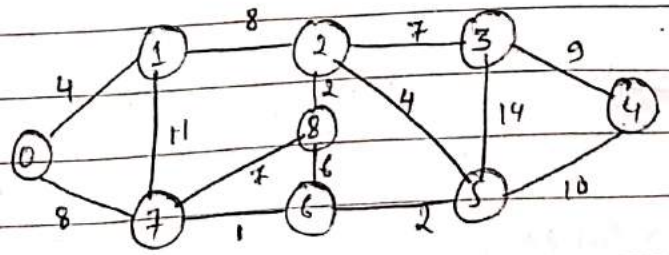


At last we have two edges CE and AC of cost 6 and 7, but if we connect that edges on the tree then we will form cyclic so we discard that one.

Hence, In this way MST can be obtained by using kruskal Algorithm.

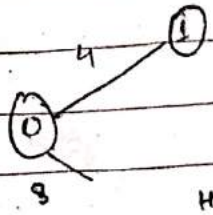
25. Find the shortest path in following graph using Dijkstra Algorithm.

→



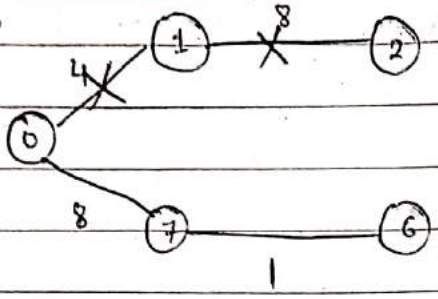
let's take vertex '0' as the source and vertex '4' as the destination.

Here,

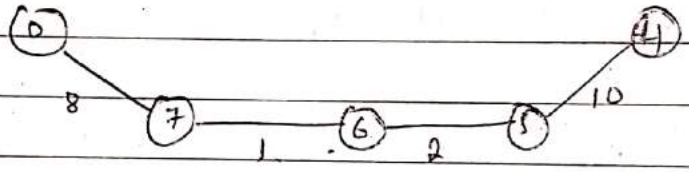


Here, 4 is the shortest path.

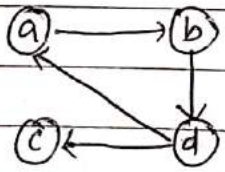
But,



Similarly, to reach destination at vertex '4', the smallest / shortest path will be from vertex '6' to vertex '5' and then to vertex '4'; else than other routes.



26. Find transitive closure of following graph.



Ans: It's Reachability matrix is,

	a	b	c	d.
a	1	1	1	1
b	0	1	1	1.
c	0	0	1	0
d	1	0	1	1.

∴ Transitive closure is.

{ (a,a), (a,b), (a,c), (a,d), (b,b), (b,c), (b,d), (c,c), (d,a), (d,c), (d,d) }.

27. What is Big O notation and how it is useful to measure an efficiency of an algorithm.

Ans. "Big O Notation" is a way to express the speed of algorithms and it is represented by O(n), here n is the amount of data the algorithm is working with.

The value expressed in terms of Big O notation help us to measure efficiency of algorithm especially when we are working with larger lumps of data. The efficiency generally indicates the time and space complexity. And, the larger the value that we obtain from the Big O Notation, the more it takes space and time for the algorithm. So, the lower the value, the more efficient is algorithm.

28. Explain divide and conquer startegy algorithm.

Ans. Divide and conquer is an algorithm design paradigm. A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a soln to the other problem:

The divide and conquer technique is the basis of efficient algorithms for many problems, such as sorting (e.g. quicksort, merge sort), etc.

Date
Page

The name "divide and conquer" is sometimes applied to algorithms that reduce each problem to only one sub-problem, such as binary search algorithm for finding a record in a sorted list.