Course: Data Structure and Algorithm.
YEAR: 2022 FALL

Page No.
Date

What is the advantages of using recursive algorithms? Implement the recursive algorithms to solve the Tower of Hanoi problems using C or C++ code.

The advantages of using recursive algorithms:

i) Simplifies complex problems
ii) saves time and space.
iii) Increases code readability.
iv) Enables efficient data processing
v) Facilitates problem solving
vi) Applicable for following task
   ↳ to calculate factorial of a given number
   ↳ to solve ToH problem
   ↳ Stacks evolutions and infix, prefix, postfix evaluations
   ↳ to check validity of expression.

Algorithm to solve Tower of Hanoi problem.

Let us consider that we have N disc and three towers named source, temp and distination.

Step 1: If (N == 1)
   a) move a disc from source to distination.
   b) Exit

Step 2: Move upper N-1 disc from Source to temp using distination as temporary.
   Move (N-1, source, destination, temp)

Step 3: Move largest disc from source to destination

Step 4: Move upper N-1 from temp to destination using source as temporary.
   Move (N-1, temp, source, destination)

Step 5: Exit.

Date

Program:

```
#include<stdio.h>
void TOH (int n, char rod A, char rod C, char rod B)
{
    if(n==1)
    {
    printf("\n Move disk 1 from rod %c to
            rod %c.", rod A, rod C);
    return;
    }
TOH(n-1, rod A, rod B, rod C);
printf("\n Move %d from rod %c to %c",
        rod A, rod C);
TOH(n-1, rod B, rod C, rod A);
}
int main()
{
    int no_of_disc;
    printf("Enter no of disk:");
    scanf("%d", no_of_disc);
    TOH(no_of_disc, 'A', 'C', 'B');
    return 0;
}
```

Construct an AVL tree in which elements are inserted in the following order. 50, 72, 96, 107, 26, 12, 11, 9, 2. Show how the tree would look after the deletion of 26, 50, 16, and 10 respectively. Explain each steps of deletion.

Soln:

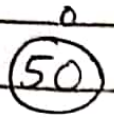
Given data are: 50, 72, 96, 107, 26, 12, 11, 9, 2

Date

Balancing factor = [height(left subtree) - height(right subtree)]

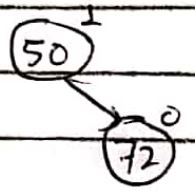
(BF = -1, 0, 1 for balanced tree)

Insert 50.

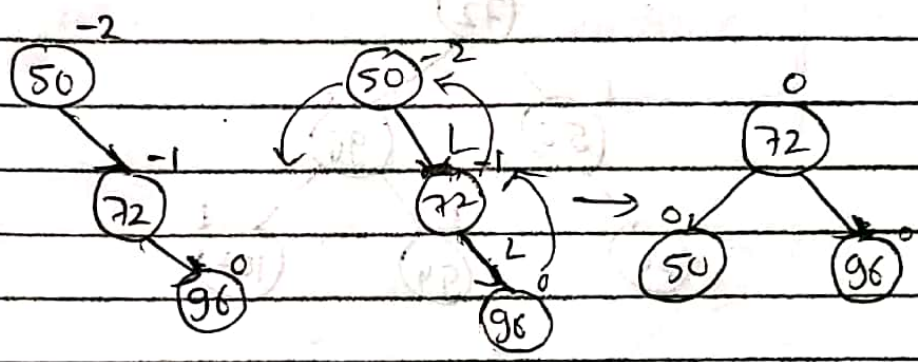


Tree is balanced.

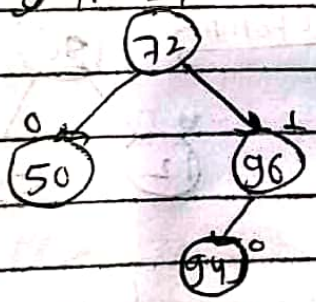
Insert 72:



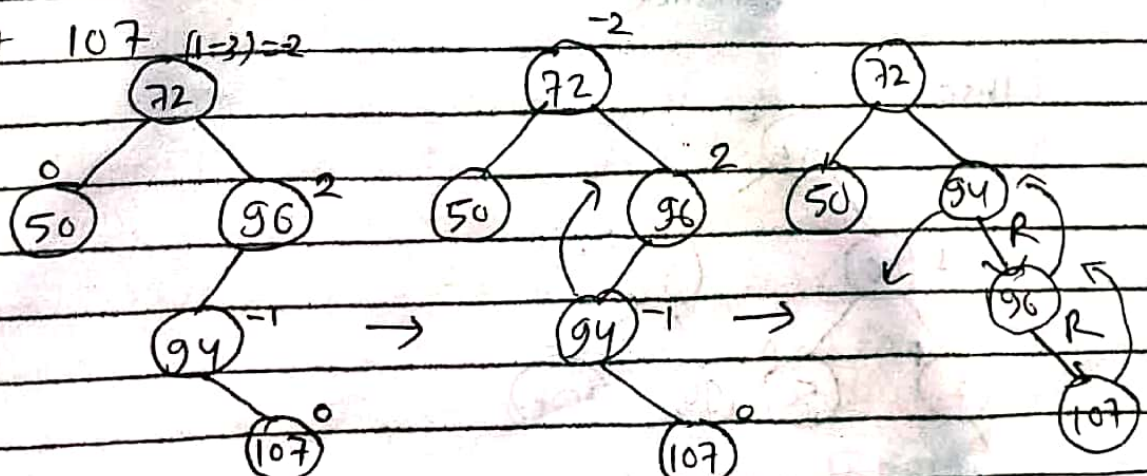
Insert 96:

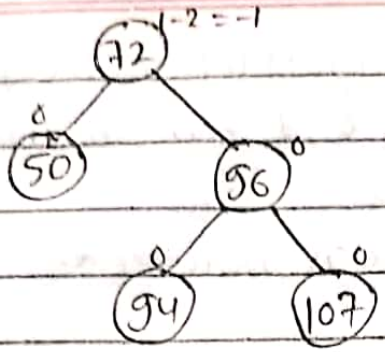


Insert 94: -1



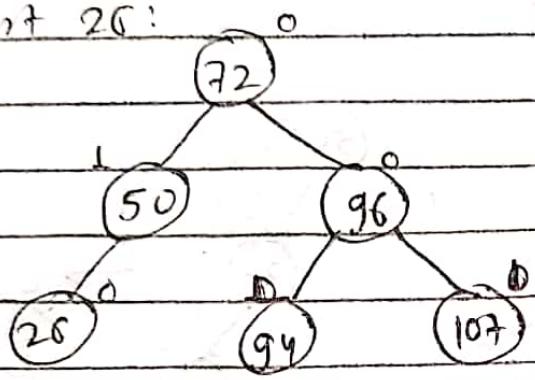
Insert 107 (1-3)=-2



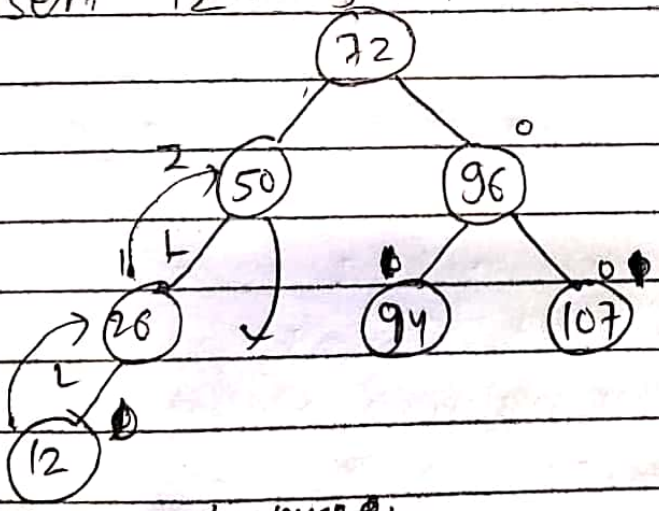


balanced

insert 26:

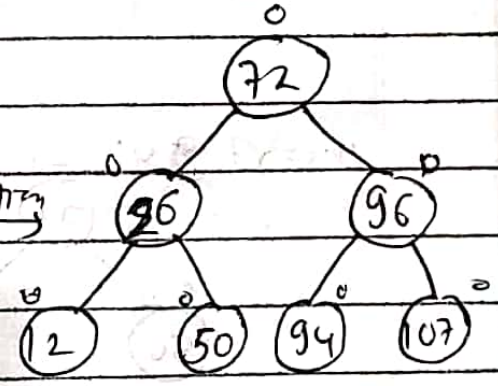


insert 12: 3-2=1



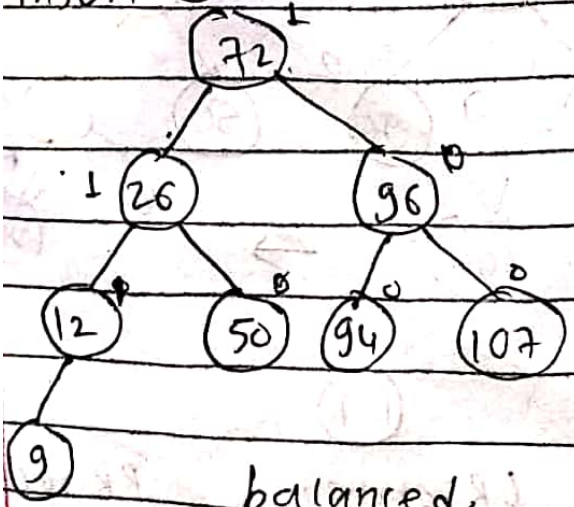
unbalance.

LL Rotation



balanced.

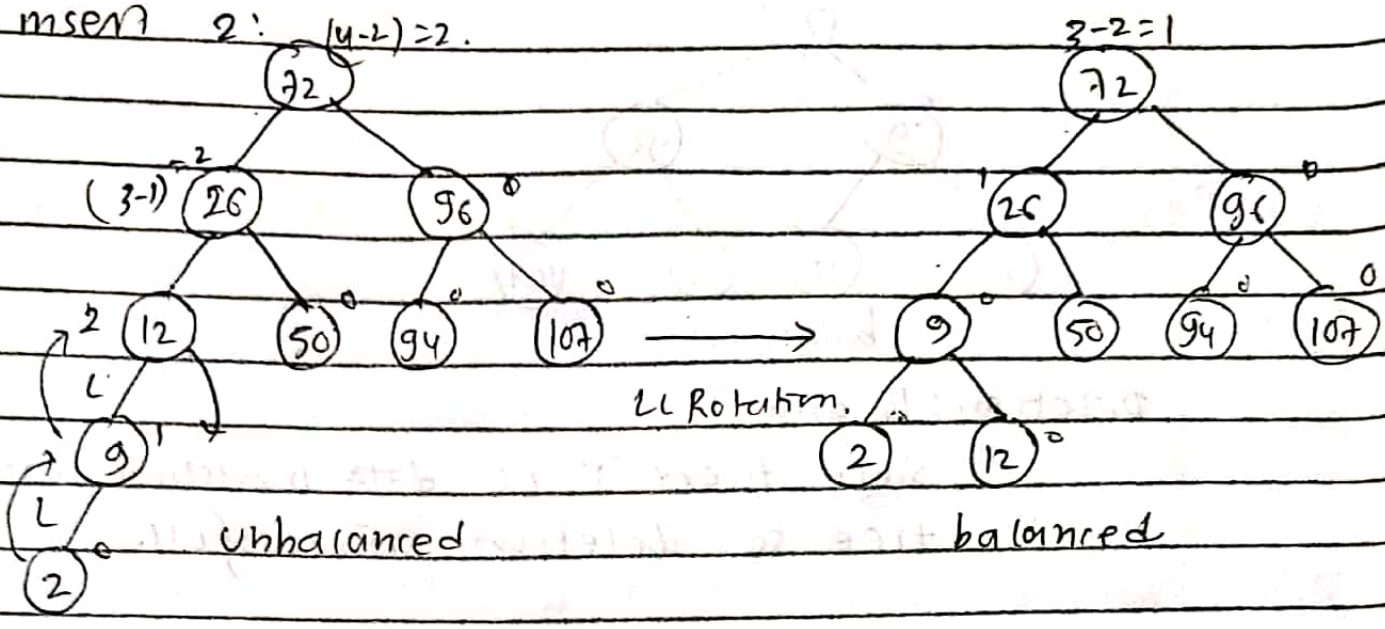
Insert 9:



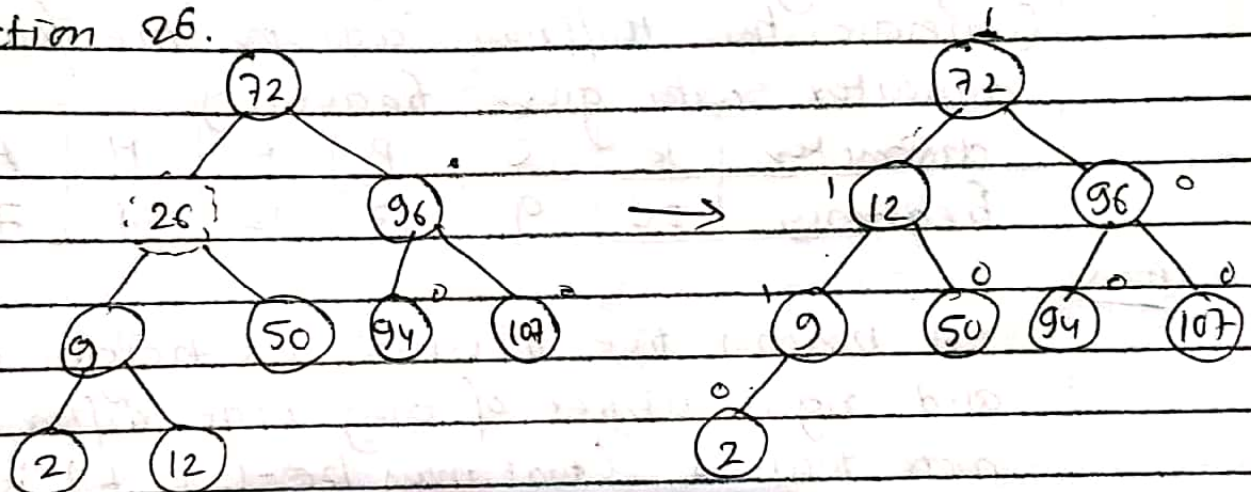
balanced.

Date

insert 2: (4-2)=2.

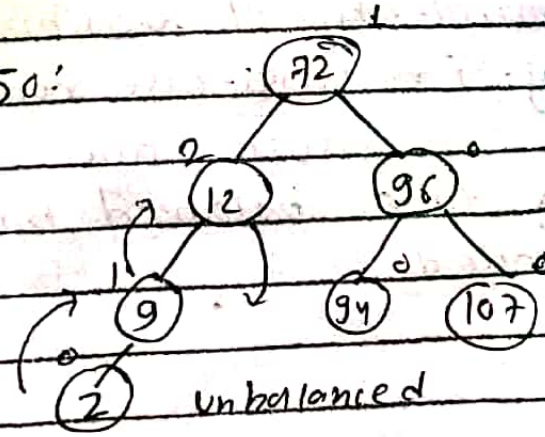


Deletion 26.

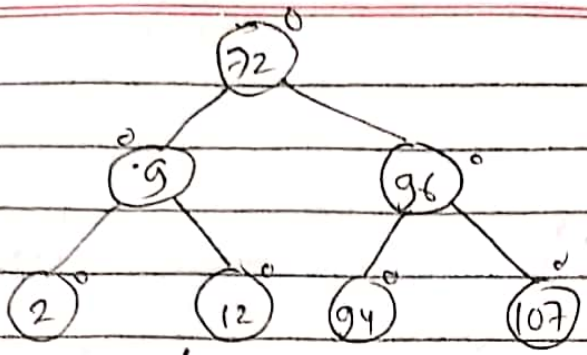


Here, 26 has two children. Among them from left pointer having highest value is ~~the~~ in order successor of 26. (i.e 12 is in order successor). So we replace 26 by 12 and gives left and right pointer address of 26 to 12.

Deletion 50:



Since 50 has no children. So we can delet it simply by giving null value to its parents.



balanced.

Deletion: 16 and 10:

Since there is no ~~data~~ matched data in tree so deletion unsuccessfull.

Why do you need to balance binary search tree? Generate the Huffman code for the following character with given frequency.

character	K	O	P	R	H	A
frequency	20	9	17	12	1	7

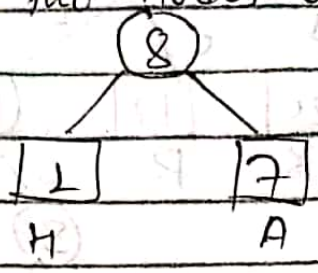
A binary tree in which the height of the left and right subtree of any node differ by not more than 1 (that may be -1, 0, 1) is called balanced binary tree.

While working with Binary search tree, we come across a ~~se~~ scenario where the elements are in sorted order. In such cases, all the array elements are arranged in one side of the root, which leads to an increase in time complexity of searching an element in a array. i.e worst-case complexity of tree. To resolve such issues and decrease the searching time, balanced binary search tree is needed.

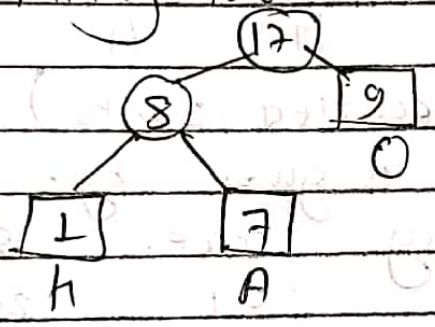
Page No.
Date

Given character | K | O | P | R | H | A
freq^n | 20 | 9 | 17 | 12 | 1 | 7

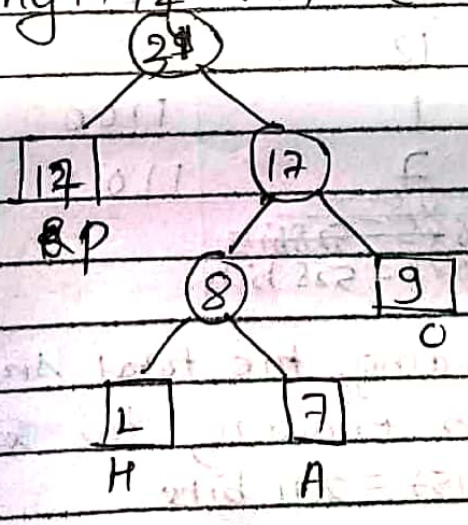
Step 1 : taking two nodes with minimum freq^n i.e 7.



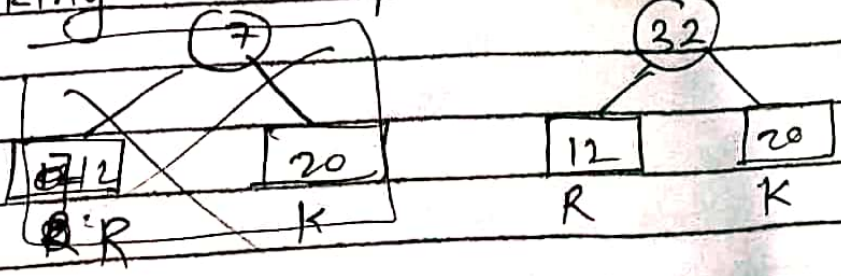
Step 2: taking two nodes with minimum freq^n 8 & 9



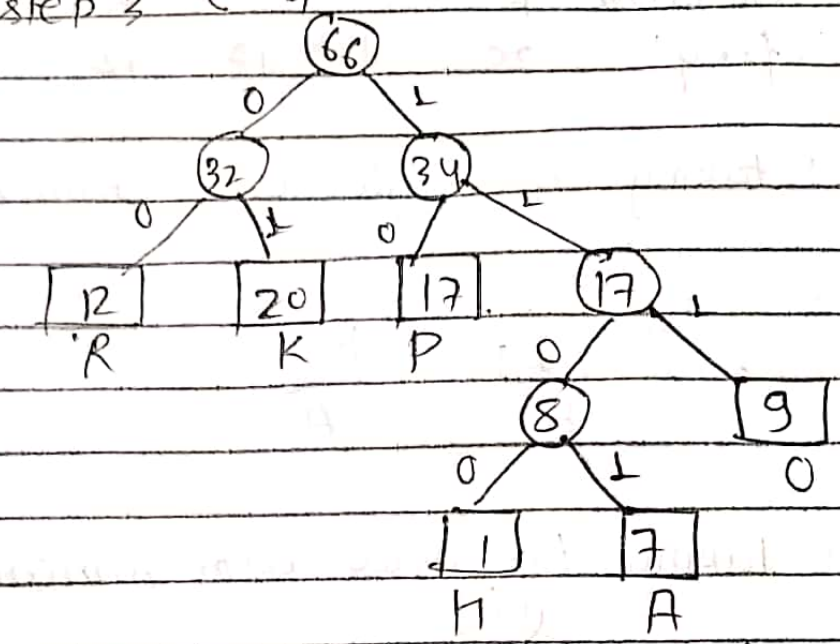
Step 3: taking 17 ~~17~~ & 17 ~~(i.e R=12 & 17 (step 2))~~



Step 4: taking 20 & ~~17~~ 12



from step 3 & 4



(If each character occupied 8 bits)

The Huffman's code is given by:

character	frequency	code.	Size.
K	20	01	20x2 = 40
O	9	111	9x3 = 27
P	17	10	17x2 = 34
R	12	00	12x2 = 24
H	1	1100	1x4 = 4
A	7	1101	7x4 = 28
6x8 = 48 bits	~~66x8 = 528bits~~ 66x8 = 528 bits		total = 157 bits.

Without encoding, the total size of string was 528 bits. After encoding the size is reduced to 48+66+157 = 271 bits.

5a) Use insertion sort to sort the following data.
24, 11, 49, 35, 98, 72, 34, 44.

Soln:

24	11	49	35	98	72	34	44

Initially comparing 1st two elements. ie 24 > 11
So 24 is not at correct position for ascending order.
Now swap 24 and 11
i.e.

11	24	49	35	98	72	34	44

Now moving to next two elements and compare them following above steps.
24 < 49 that means both elements are already in ascending order.

11	24	49	35	98	72	34	44

49 > 35 swaping.

11	24	35	49	98	72	34	4

49 < 98. are in correct order.

11	24	35	49	98	72	34	4

98 > 72 swaping.

11	24	35	49	72	98	34	4

98 > 34 swap is done.

11	24	35	49	72	34	98	4

Here 34 is less than 72, 49 & 35
So 35 inserted to correct position by shifting other elements one position right.

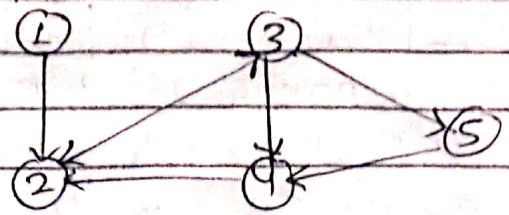
11	24	34	35	49	72	98	4

comparing last two elements.
98 > 4, 72 > 4, 49 > 4, 35 > 4, 34 > 4, 24 > 4
11 > 4 so shifting all other element to one position right to insert 4 to correct position.

4	11	24	34	35	49	72	98

sorted

6a) What do you mean by transitive closure of a graph? Find the transitive closure of given graph.



Ans The adjacency matrix A, of graph G, is the matrix with elements a_{ij} such that $a_{ij}=1$ implies there is an edge from i to j. If there is no edge then $a_{ij}=0$.

Let A be the adjacency matrix of D and T be the adjacency matrix of transitive close of D. T is called the reachability matrix of digraph (directed graph) D due to the property that $\forall \cdot T_{ij}=1$ if and only if v_j can be reached from v_i in D by a sequence of arcs (edges).

In simple word the defination is as.

~~Franz~~ The reachability matrix to reach from vertex u to vertex v of a graph. OR

A path exist between two vertices i, j if

1. there is an edge from i to j ; or
2. there is a path from i to j going through vertex 1 or 2 through vertex 1, 2 and 3 or going through any of the other vertices.

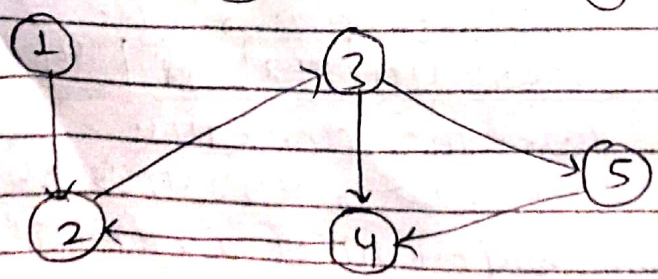


Fig: a) Digraph G

Page No.
Date

$A =$

	1	2	3	4	5
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	0	1	0	0	0
5	0	0	0	1	0

Fig. b) Adjacency matrix.

$T =$

	1	2	3	4	5
1	1	1	1	1	1
2	0	1	1	1	1
3	0	1	1	1	1
4	0	1	1	1	1
5	0	1	1	1	1

fig (c) : Transitive ~~clove~~ closure

b) What is the pupose of Krushkal's algorithm? Explain with a suitable example.

Ans ~~Krushkal's~~ Kruskal's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of the graph which

i) form a tree that includes every vertex

ii) has the minimum sum of weights among all the trees that can be formed from the graph.

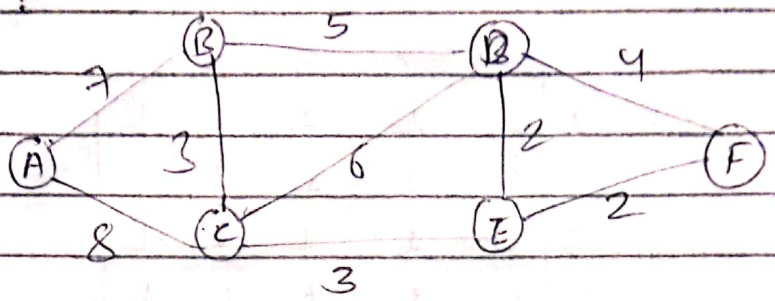
It is a greedy algorithms which find the local optimum in the hopes of finding a global optimum. This algorithm starts from an edge with the lowest weight and keep adding edges until a minimum spanning tree is built.

Date

Steps for implementing kruskal's algorithm are:

1. Sort all edges from low to high.
2. Take the edge with the lowest weight & add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
3. Keep adding edges until we reach all vertices.

Example:

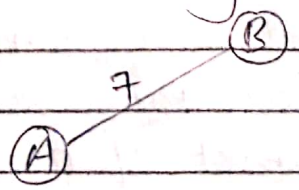


No of vertex (n) = 6

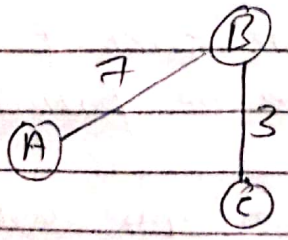
no of edges must be (n-1)

= 6 - 1 = 5.

Step 1: ~~Edge~~ Vertex A has path connection to ~~edge~~ vertex B & C taking only shortest or lowest weight



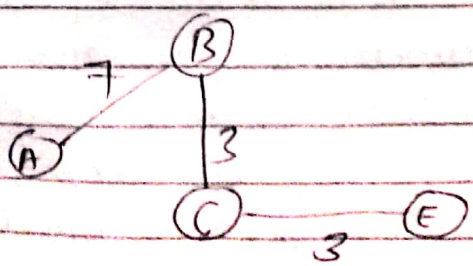
Step 2: From B. BC → 3, BD → 5,



taking BC only.

Step 3: CE → 3 ✓

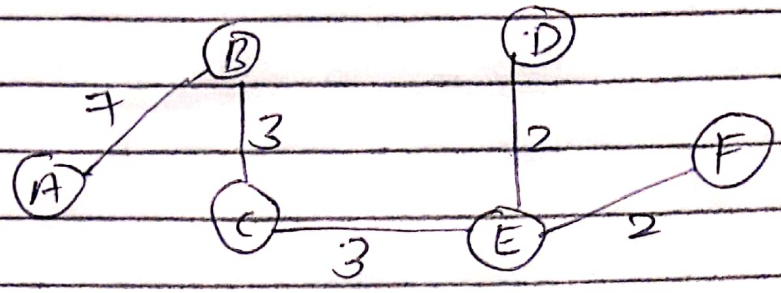
CD → 6. ×



Date

Step 4: ED → 2, EF = 2, ~~DF → 4~~ DF → 4

Here shortest path from ~~edge~~ vertex E to D is 2 and for ~~edge~~ vertex F, EF → 2 and DF → 4 taking ED and EF.



Here no loop is created and minimum spanning tree is formed having ~~edge~~ vertex n = 6 and no of edge ~~=~~ (n-1) = 5.

YEAR: 2021 Spring.

Page No.
Date

3b) In which condition recursive algorithm are suitable? Explain with problem of printing Fibonacci series.

Ans Recursion - problem provides a clean and simple way to write code. Some problems are inherently recursive like tree traversals, Tower of Hanoi, etc. For such problems, it is preferred to write recursive code. Recursive function are simpler to solve complex problems. Therefore, it is better idea to use recursive function to solve complex problem because to write the iterative function for some complex function yeild in very complex function.

Application of recursion

- used to calculate factorial of a given numbers
- used to solve TOH problem
- to check validity of expression
- stacks evaluation and infix, prefix, postfix evaluations.

If time complexity is not an issue and shortness of code is, then recursion would be the best to implement which involves calling the same function again and hence, has a very small length of code.

Fibonacci series is a sequence of numbers where every number after the first two numbers is a sum of proceeding numbers.

Eg:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 ....

Date

```
Procedure Recursive-Fibonacci(n)
  int f0, f1;
    f0 = 0
    f1 = 1
  if(n <= 1);
    return n
    else
  return Recursive-Fibonacci(n-1) + Recursive-fibonacci(n-2);
  END
```

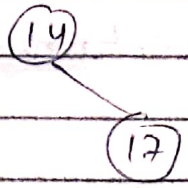
```
# include <stdio.h>
  int fibo(int);
  int main()
   { int i;
     for(i=0; i<10; i++)
     {
      printf("%d ", fibo(i));
     }
   }
   int fibo(int n)
      {
       if(n == 0)
        return 0;
       if(n == 1)
        return 1;
       else
       return fibo(n-1) + fibo(n-2);
   }
```

Date

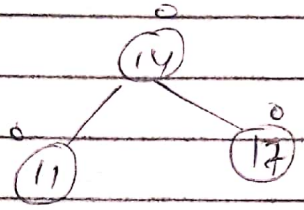
4 b) construct an AVL tree by inserting following data: 14, 17, 11, 7, 53, 4, 13, 12, 8, 60.
Also delete the item 8, 7, 14, 17 from the constructed tree showing every step.

By inserting 14

inserting 17

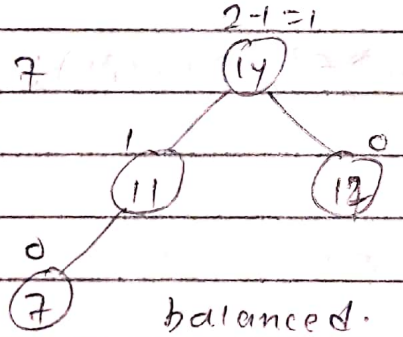


inserting 11



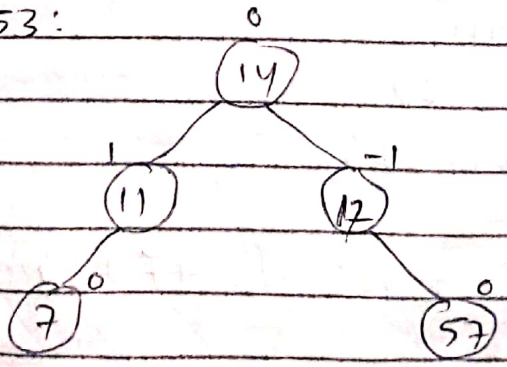
balance No

inserting 7



balanced.

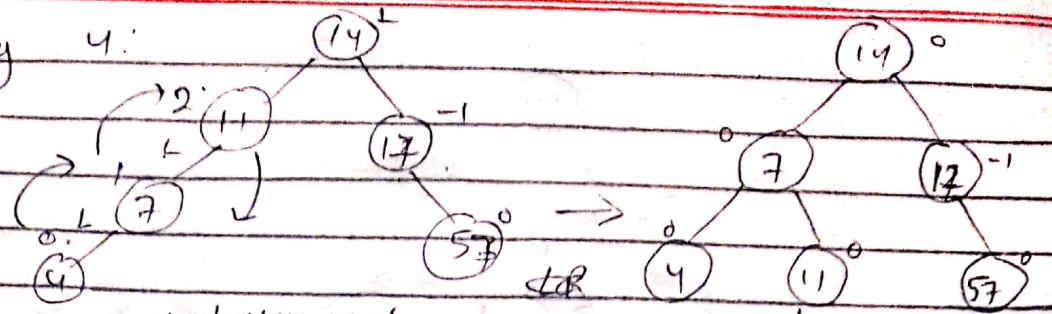
insert 53:



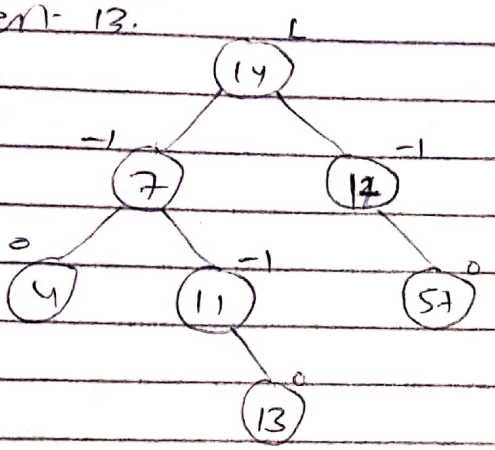
balanced.

Page No.
Date

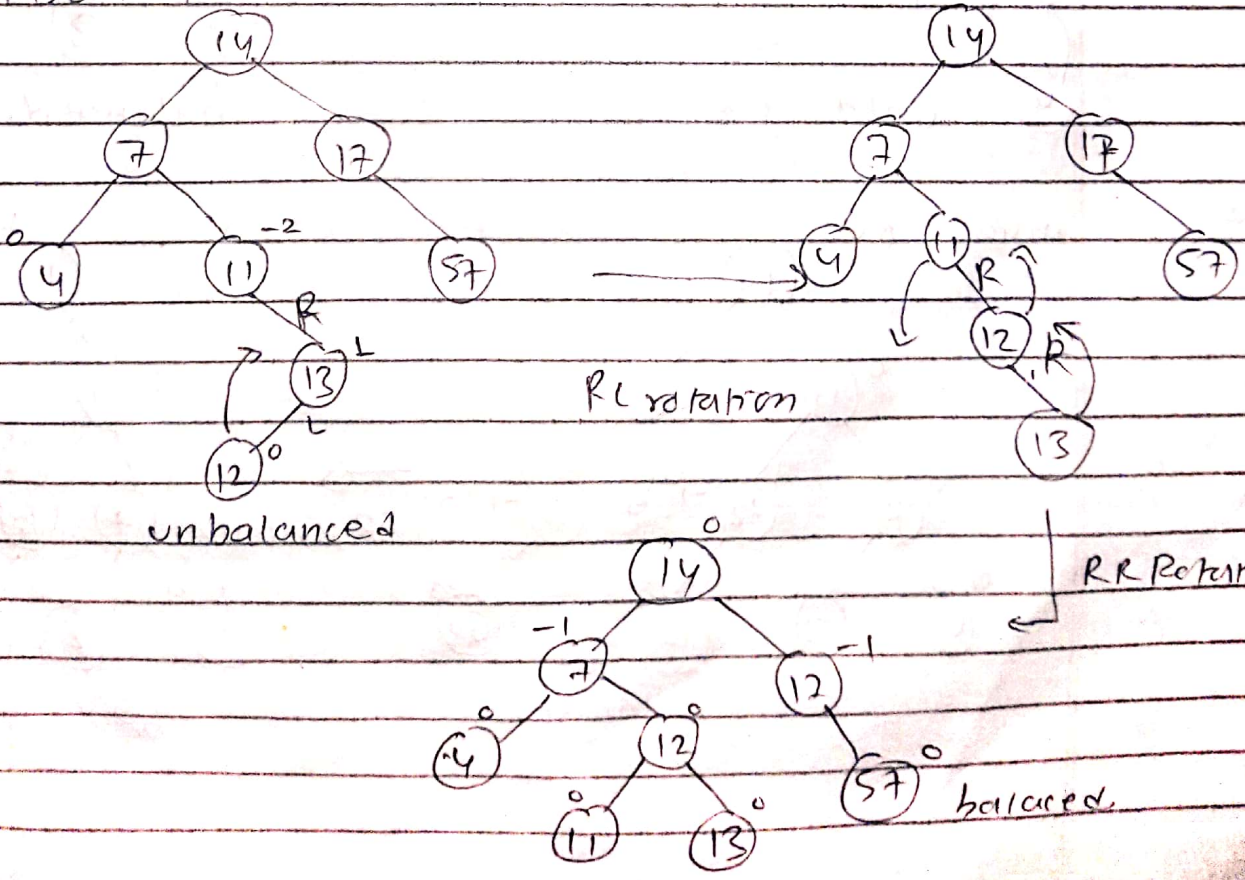
Inserting 4:



Insert 13.

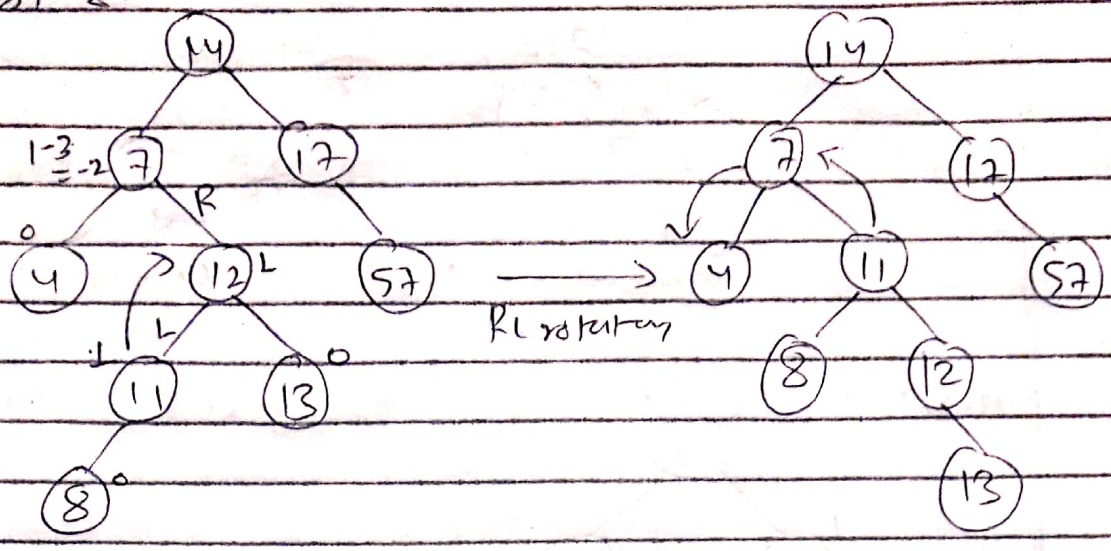


Insert 12.

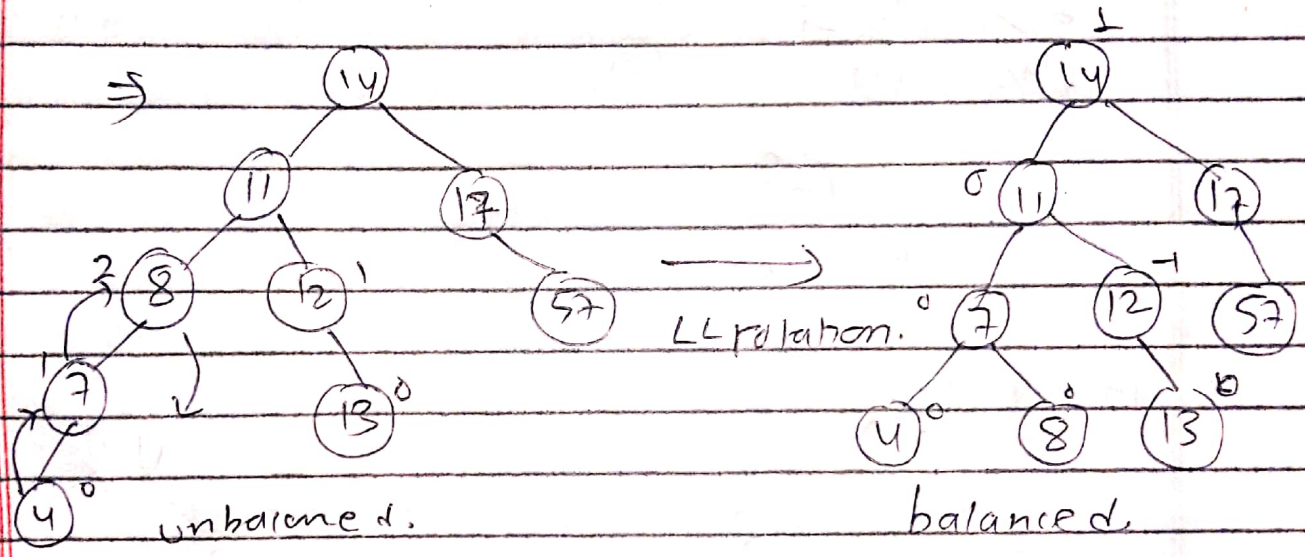


Date

insert 8.



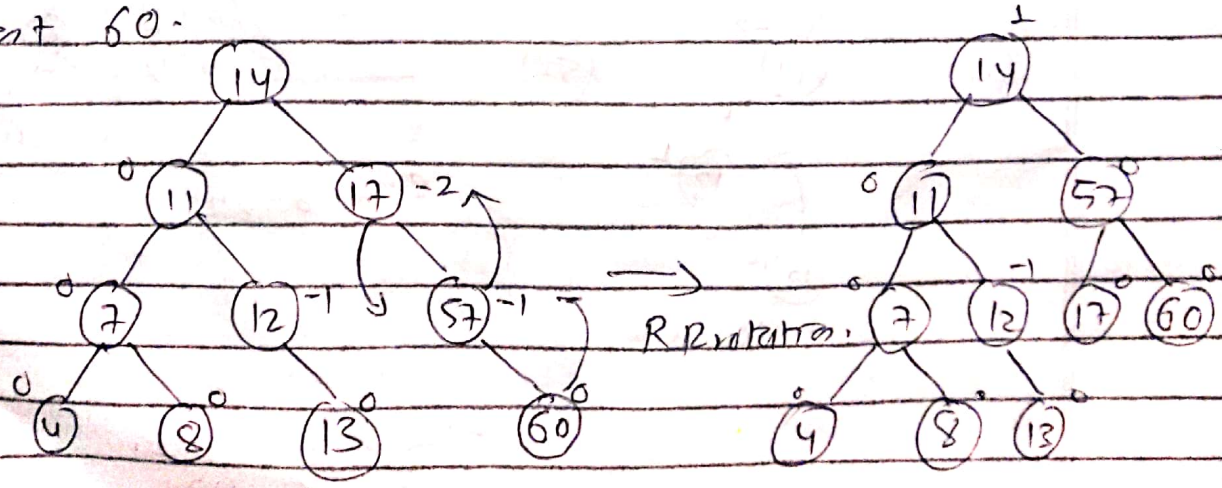
Unbalanced.



unbalanced.

balanced.

insert 60.

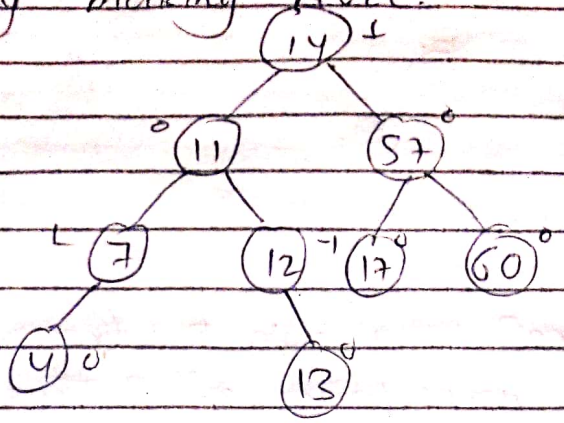


unbalaced

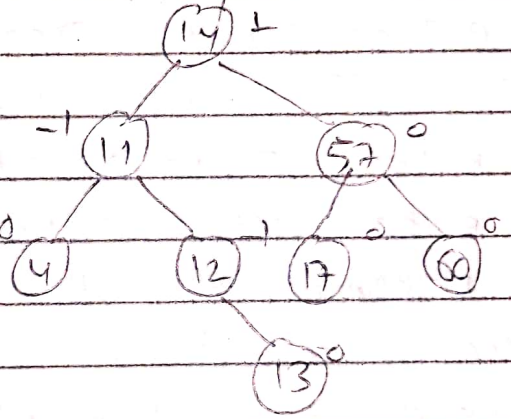
balanced.

Date

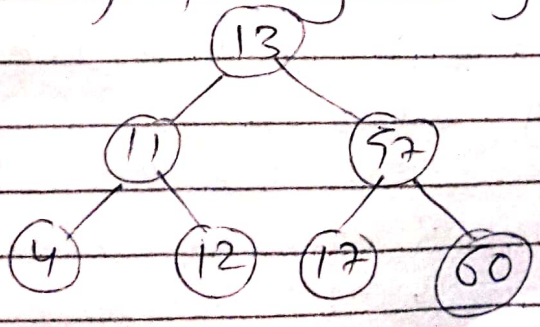
Delete: Since 8 has no children, it can be deleted by making NULL.



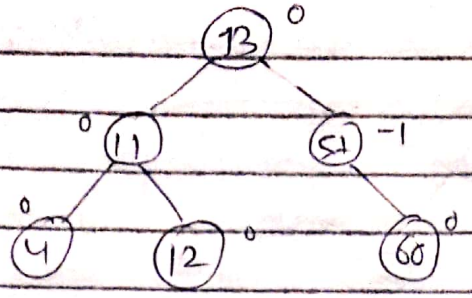
Deleting 7: Sine 7 have only one child so it can be deleted and its left pointer child is pointer to parents node 11.



Deleting 14: Since 14 have children on both sides. Among them left pointer having highest value is in order successor of 14. So 14 is ~~d~~ can be deleted by replacing it by 13.



Delete 17: Sine 17 have no children. It can be simply deleted.



balanced.

4 b) What is B-Tree? What are the features of B-Tree? Discuss on the importance of B-Tree.

Ans B-Tree is a self balanced tree in which every node contains multiple keys and has more than two children. The number of keys in a node and number of children for a node depends on the order of B-Tree. Every B-Tree has an order.

The B-Tree of order m has following properties:

i) All leaf nodes must be at same level

ii) All nodes except root must have at least $\lceil m/2 \rceil - 1$ keys and maximum of $m-1$ keys.

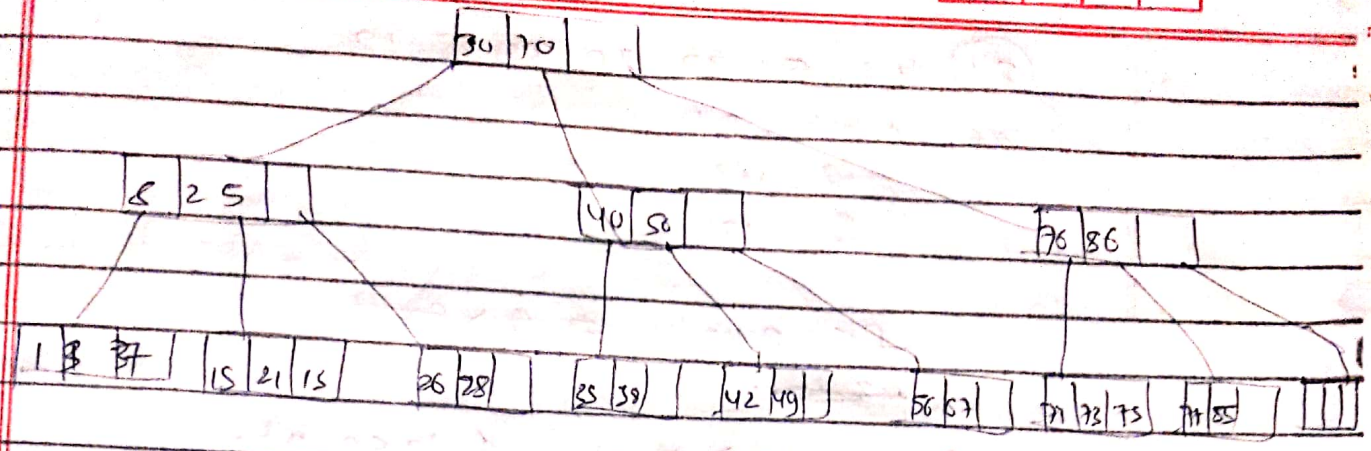
iii) All non leaf nodes except root (i.e all internal nodes) must have at least $m/2$ children.

iv) If the roots node is a non leaf node, then it must have at least 2 children.

v) A non leaf node with $n-1$ keys must have n number of children

vi) All the key values in a node must be in ascending order.

For example, B-Tree of order 4 contains a maximum $(m-1) = (4-1) = 3$ key values in a node and maximum of 4 children for a nodes.



Importances of B-Tree.

1) B-Tree facilitates ordered sequential access, which works more effectively than a hash table.
2) B-Tree enables iterations over items similar to what a binary tree supports. The iterations arrange the items in the proper order (ascending or descending) as req^n
3) keeps key in sorted order for sequential traversing
4) Uses a hierarchical index to minimize the number of disks reads
5) uses partially full blocks to speed up insertions & deletion
6) keeps the index balanced with a recursive algorithm.
7) can handle an arbitrary number of insertions and deletions.

5a) Sort the following data using quick sort explaining every step.

21, 43, 51, 32, 20, 35, 8, 12.

Soln:

while data[P] <= data[pivot]
P++ → moves left to right
while data[Q] > data[pivot]
Q-- moves Right to left

(21) 43 51 32 20 35 8 12

Here, pivot = 21

P = 43

Q = 12.

~~P <= pivot & Q > Pivot~~

~~43 <~~ .

Since 43 > pivot. & 12 < pivot.

(21) 43 51 32 20 35 8 12

P Q

swap P & Q.

(21) 12 51 32 20 35 8 43

51 > 21

8 < 21

(21) 12 51 32 20 35 8 43

P Q

swap P & Q.

21 12 8 32 20 35 51 43

P Q

P > 21 Q < 21 swap.

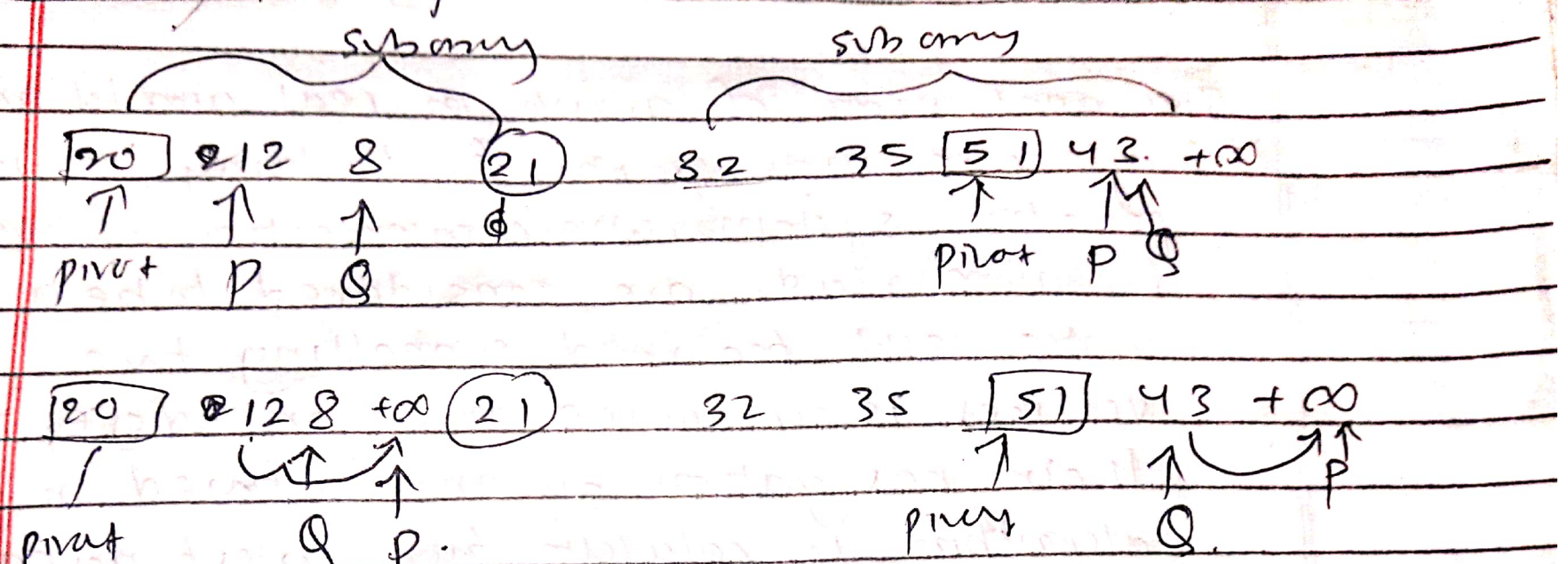
21 12 8 20 32 35 51 43

Q P

then is intersect betn movemnt of P and Q so swap pivot with Q.

20 12 8 21 32 35 51 43

~~pivot = 20.~~



8 12 20 21 35 35 43 51

Hence given data is sorted.

8 12 20 21 35 35 43 51

Page No.
Date

6a) What is a graph? Mention any two application of graph in real world. What are two methods used to represent graph? Explain with example.

Ans A graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph.

Formal defination: A graph G can be defined as a pair (V, E), where V is a set of vertices, and E is a set of edges between the vertices $E \subseteq \{(U, V) \mid U, V \in V\}$.

Two application of graph in real world are:

i) Google maps uses graphs for building transportation systems, where intersection of two (or more) roads are considered to be a vertex and the road connecting two vertices is considered to be an edge. Thus their navigation system is based on the algorithm to calculate the shortest path between two vertices.

ii) Computer Science: In computer science, graph is used to represent networks of communication, data organization, computational devices etc.

In Graph theory, graph representation is a technique to store graph into the memory of computer. They are:

i) Adjacency matrix
ii) Incidence matrix
} → static implementation

ii) Adjacency list → Dynamic implementation

1) Adjacency matrix

matrix formed with the help of vertices is called adjacency matrix.

let A be a matrix of order M*M, then each of the element a_{ij} can be represented as

$$a_{ij} = \begin{cases} 1, & \text{if there exist a direct path between } v_i \text{ to } v_j \\ 0, & \text{otherwise} \end{cases}$$

eg.

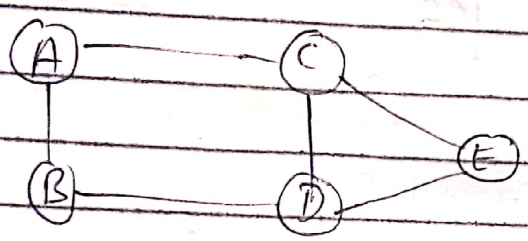


fig: undirected graph.

The adjacency matrix of above graph is

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

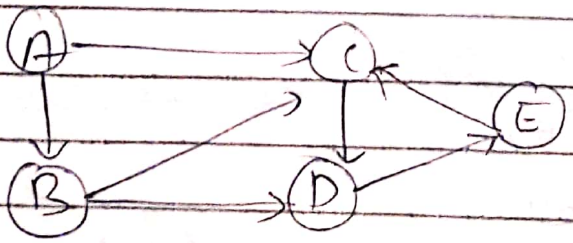


fig: directed graph.

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

2) Incidence matrix

A matrix formed with the help of vertex and edge is called incidence matrix. Let A be matrix of order M * N, then each of the element a_{ij} can represented as

$$a_{ij} = \begin{cases} 1 & \text{if there exist an edge } e_i \text{ incident on vertex } v_j \\ 0 & \text{otherwise.} \end{cases}$$

eg.

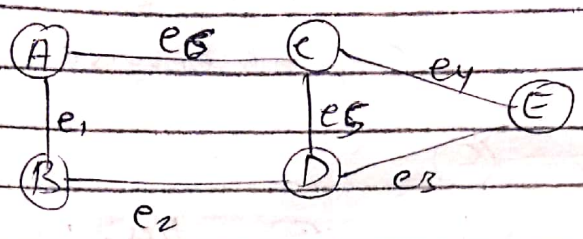
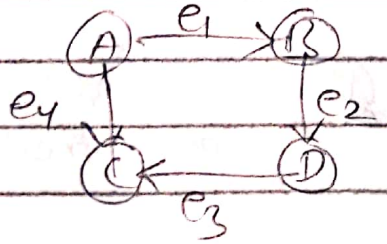


fig. Graph G.

The incidence matrix of G is

	e_1	e_2	e_3	e_4	e_5	e_6
A	1	0	0	0	0	1
B	1	1	0	0	0	0
C	0	0	0	1	1	1
D	0	1	1	0	1	0
E	0	0	1	1	0	0



	e_1	e_2	e_3	e_4
A	1	0	0	1
B	1	1	0	0
C	0	0	1	1
D	0	1	1	0

b) Define graph traversal. Differentiate between DFS and BFS with an example.

⇒ Graph traversing is the process of visiting each nodes in the graph in some systematic approach. There are two graph traversal methods.

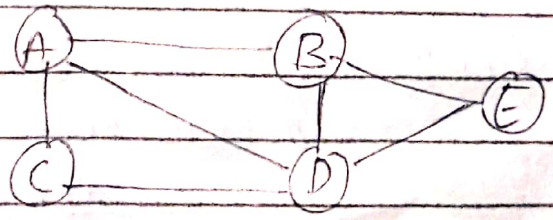
i) Breadth first Search (BFS)

ii) Depth first search (DFS)

BFS	DFS
i) uses queue data structure to store the nodes to be visited	i) used stack to store the nodes to be visited.
ii) Traverse a graph level wise	ii) Traverse a graph depth wise.
iii) Not suitable for decision making trees used in games or puzzle.	iii) More suitable for game or puzzle problems
iv) More suitable when the target node is closer to source	iv) more suitable when the target node is away from source node.
v) consumes more memory	v) consume less memory.

Example.

BFS traversal:



Step1: select vertex A as starting point (visit A)

- Insert A into Queue.

A					

step2: visit all adjacent vertices of A which are not visited.

- insert newly visited vertices & delete A.

	B	C	D		

b) Define graph traversal. Differentiate between DFS and BFS with an example.

=> Graph traversing is the process of visiting each nodes in the graph in some systematic approach. There are two graph traversal methods.

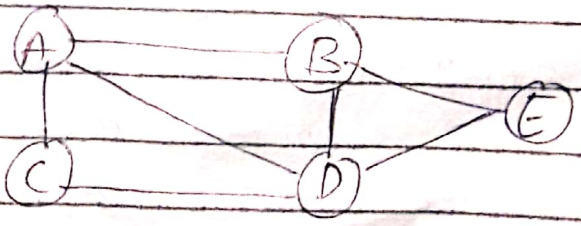
i) Breadth first traverSearch (BFS)

ii) Depth first search (DFS)

BFS	DFS
i) uses queue data structure to store the nodes to be visited	i) used stack to store the nodes to be visited.
ii) Traverse a graph level wise	ii) Traverse a graph depth wise.
iii) Not suitable for decision making trees used in games or puzzle	iii) more suitable for game or puzzle problems
iv) More suitable when the target node is closer to source	iv) more suitable when the target node is away from source node.
v) consumes more memory	v) consume less memory

Example.

BFS traversal:



Step1: select vertex A as starting point (visit A)

- Insert A into Queue.

A					

Step2: visit all adjacent vertices of A which are not visited.

- insert newly visited vertices & delete A.

	B	C	D		

Page No.
Date

Step3: visit all adjacent vertices of B which are not visited.
- insert newly visit vertices & delete B.

| | | C | D | E | | |

Step 4: for C similar. (no vertices to delete C)

| | | | D | E | | |

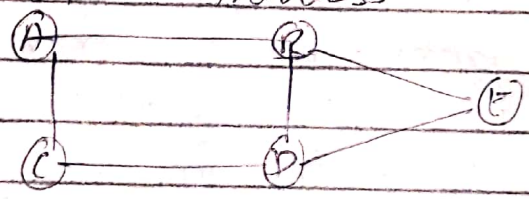
Step 5: for D no vertices remained, delete D.

| | | | | E | | |

Step 6: for E no vertices remained to visit delete E

| | | | | | | |

Example DFS Traversal.



Step 1: select one vertex A as starting point. (visit A)
→ Push A in to the stack.

| A | | | | | | |

Step 2: visit any adjacent vertex of A which is not visited. & push newly visited vertex into stack. repeat

| A | B | | | | | |

repeat same step until ~~finishing~~ visiting is not finished.

step3	A	B	E			
step4:	A	B	E	D		
step4:	A	B	E	D	C	

we halt traking an pop one by one from stack.

YEAR: 2022 Spring [Internal Exam]

3a) Construct an AVL tree from the given data.
35, 56, 64, 68, 65, 44, 31, 49, 45, 20, 25. Also, explain the methods of balancing AVL tree.

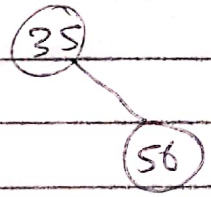
Ans Balance factor = [height of left subtree - height of right subtree]

BF = -1, 0, 1 to be balanced tree.

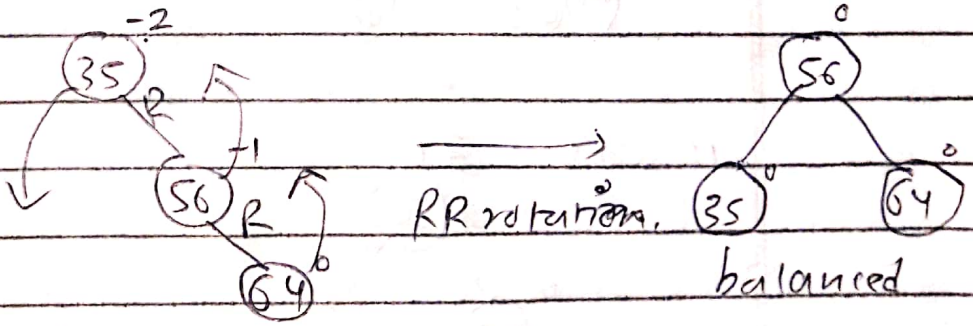
⇒ Given data 35, 56, 64, 68, 65, 44, 31, 49, 45, 20, 25

insert 35.

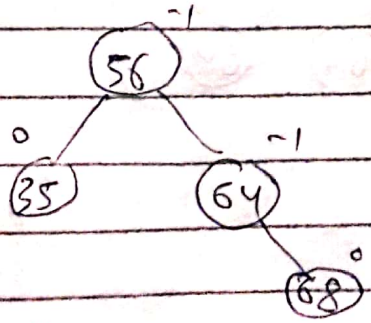
insert 56:



insert : 64

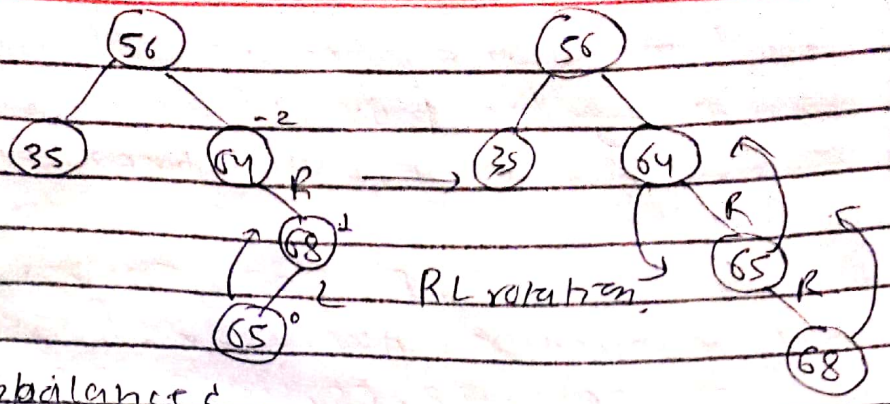


insert : 68.

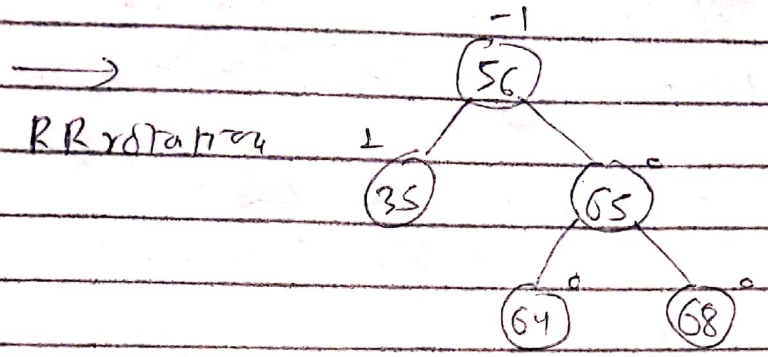


Date

Insert 65.

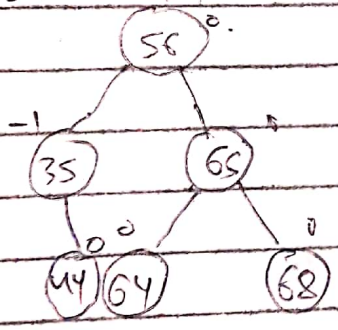


unbalanced

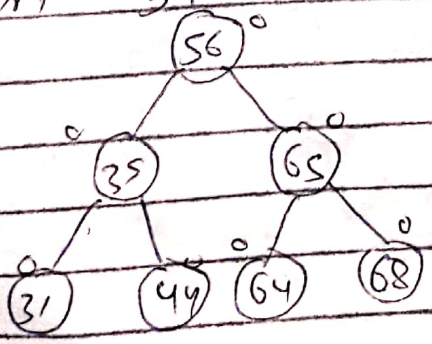


balanced.

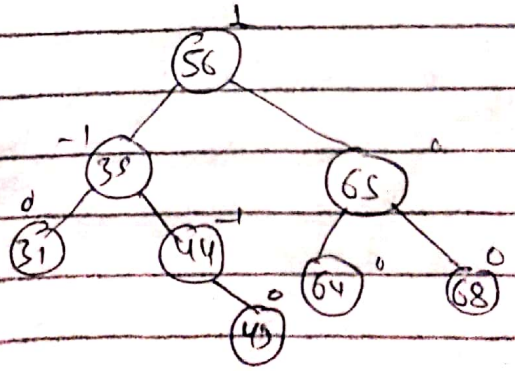
Insert 44!



Insert 31

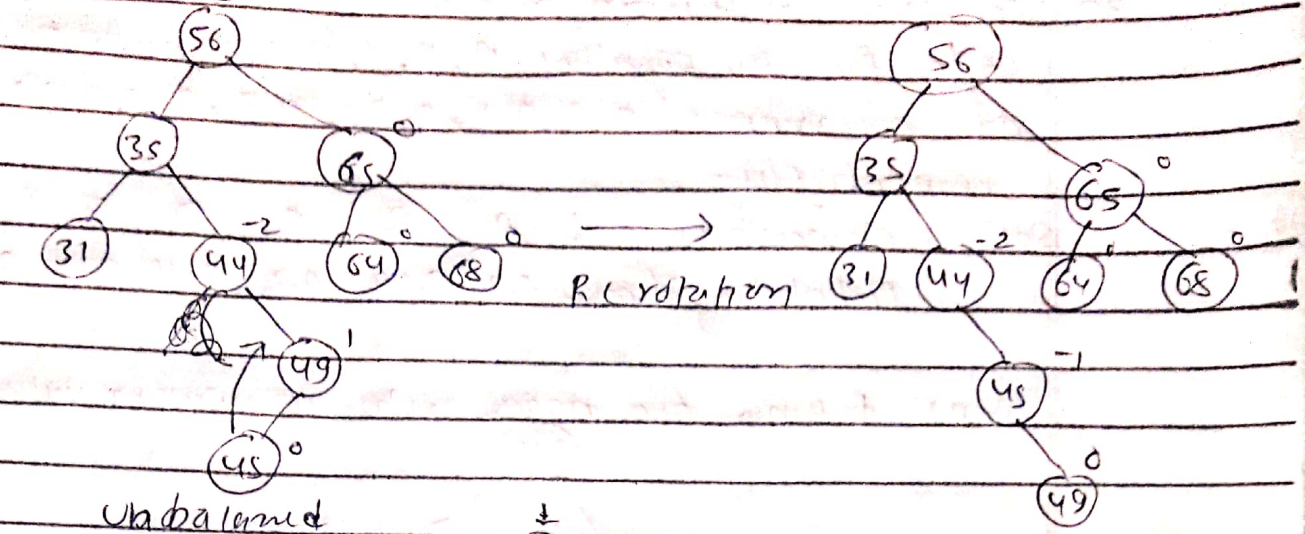


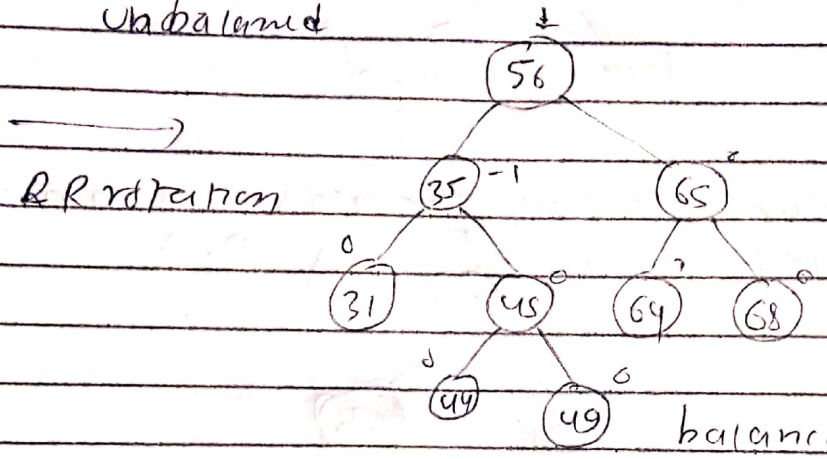
Insert 49!



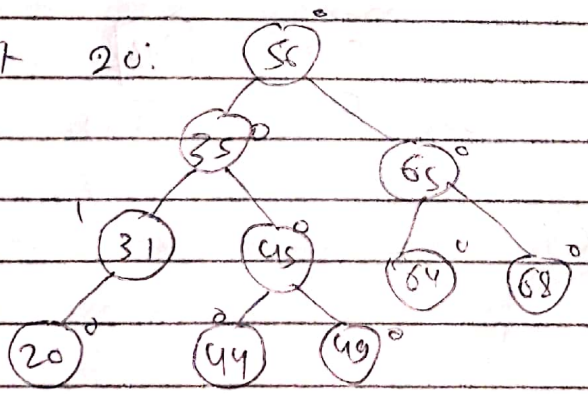
Page No.
Date

Insert 45.

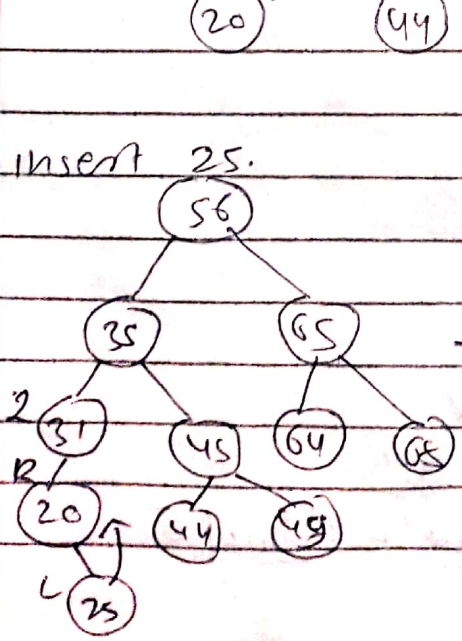


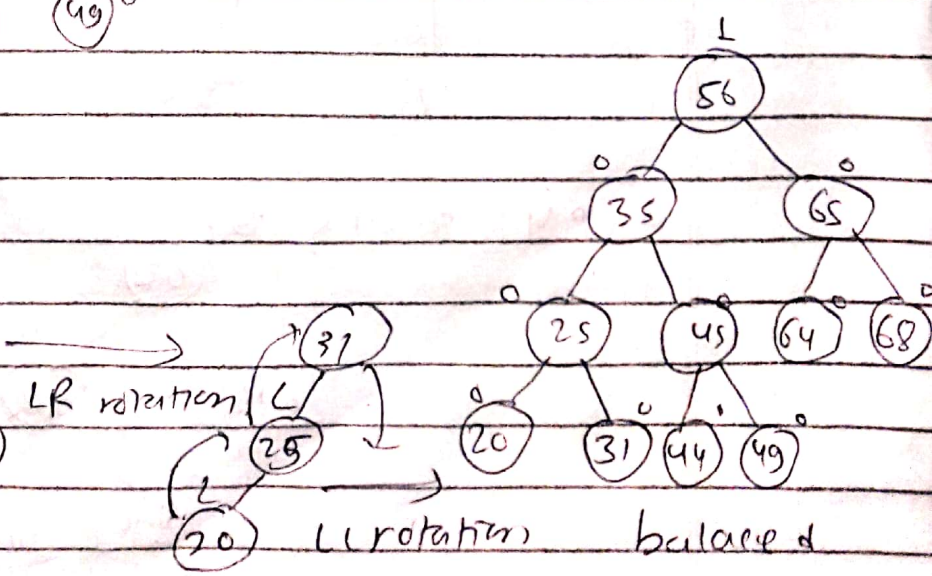


Insert 20.



Insert 25.





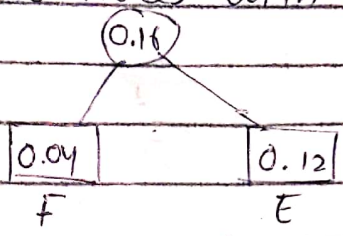
Page No.
Date

3b Construct Huffman tree and generate the Huffman code for the symbol A, B, C, D, E, F with the probability of occurrence are 0.2, 0.28, 0.2, 0.16, 0.12, 0.04 respectively.

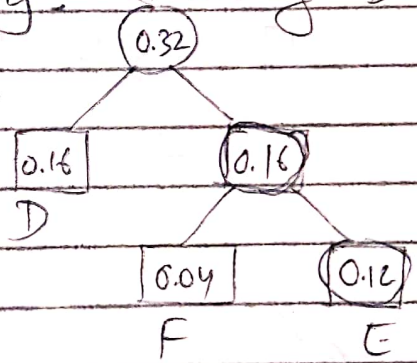
Soln: character : A B C D E F

probability of occ: 0.2 0.28 0.2 0.16 0.12 0.04.

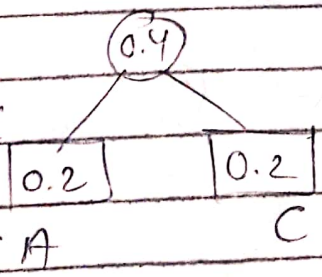
Step 1: taking two nodes with minimum value



step 2: Again taking two nods with minimum probability. (taking D.)



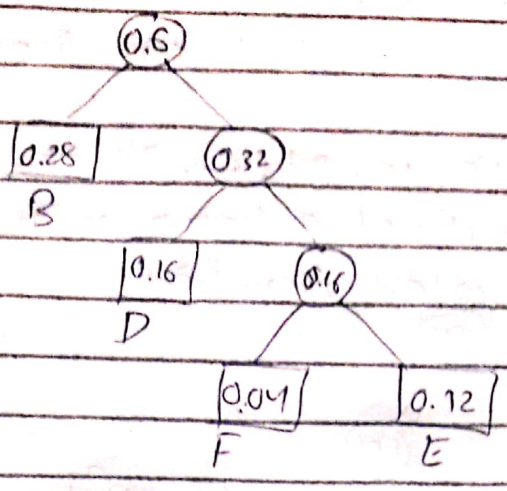
step 3: taking A & C



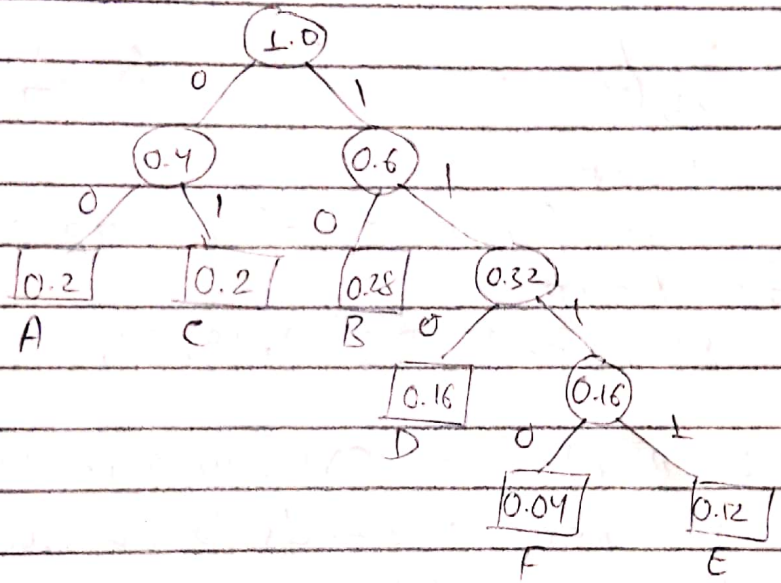
Date

taking B.

Step 4:



Now from step 3 and 4,



Each character occupies 1 byte = 8 bits.

character	probability	code	size. ∴
A	0.2	0 0	= 0.2 × 2 = 0.4
B	0.28	1 0	= 0.28 × 2 = 0.56
C	0.2	0 1	= 0.2 × 2 = 0.4
D	0.16	1 1 0	= 0.16 × 3 = 0.48
E	0.12	1 1 1	= 0.12 × 4 = 0.48
F	0.04	1 1 1 0	= 0.04 × 4 = 0.16
6 × 8 = 48 bits	1 × 8 = 8 bits		= 2.48 bits

without encoding the total size of string was 5 bits

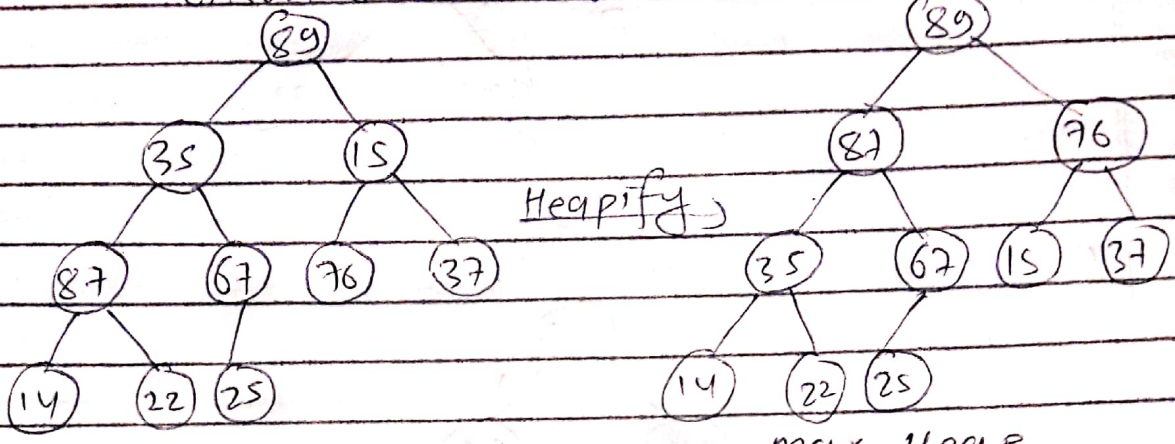
After encoding the size reduced to ~~48 + 1 + 2.48~~ ∴ 2.48 bits

Date

4b) Create the heap structure form the following sequence data and sort them using heap sort.

Sol) 89, 35, 15, 87, 67, 76, 37, 14, 22, 25.

Step 1: construct a heap with given list of unsorted number & convert of max heap.



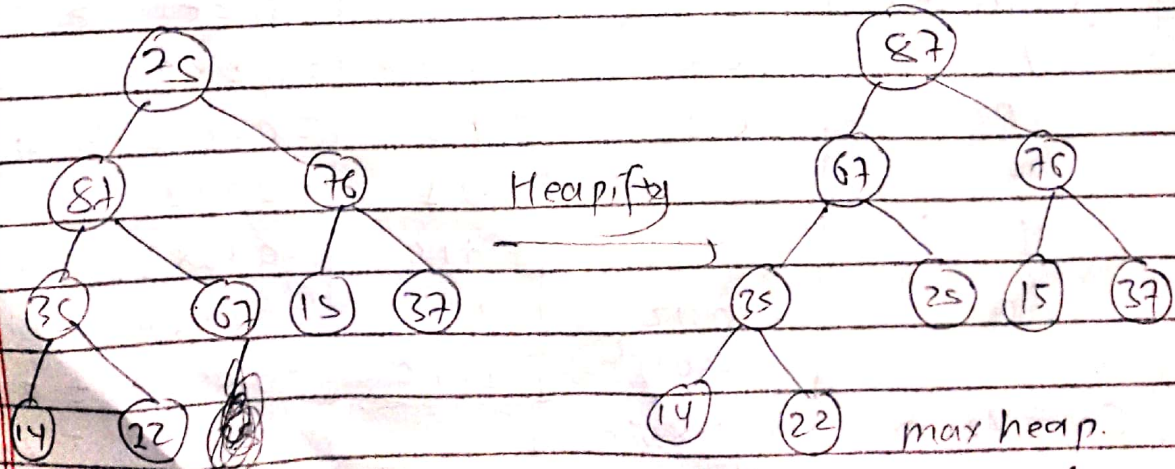
Heap

max Heap

list of numbers after heap converted to max Heap.

89, 87, 76, 35, 67, 15, 37, 14, 22, 25

Step 2: Delete root (89) from max heap. To delete root node, it needs to be swapped with last node (25). After delete, tree needs to be heapify to make it max Heap.



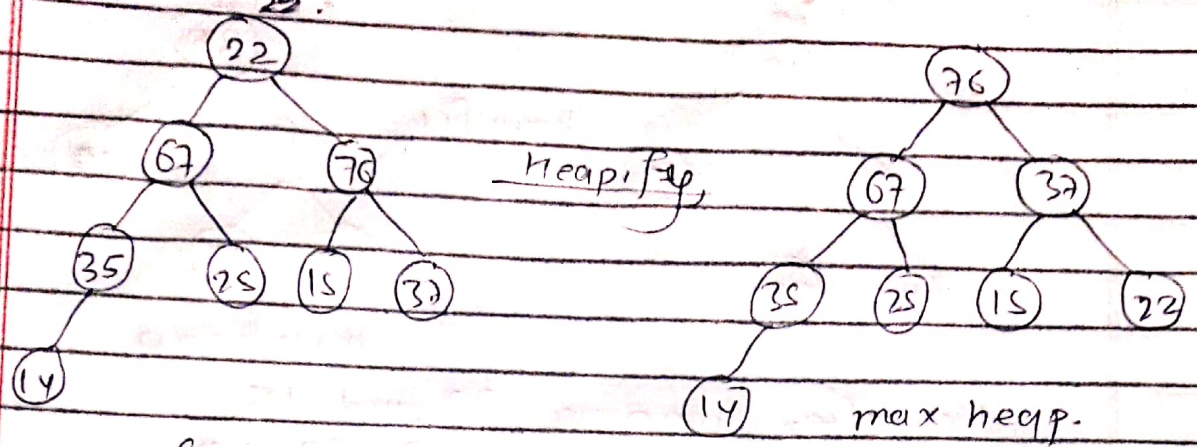
max heap.

list of number after swaping 89 & 25

25, 87, 76, 35, 67, 15, 37, 14, 22, 89

Page No.
Date

Step 3: Delete root node (87) & swap with last node(22)

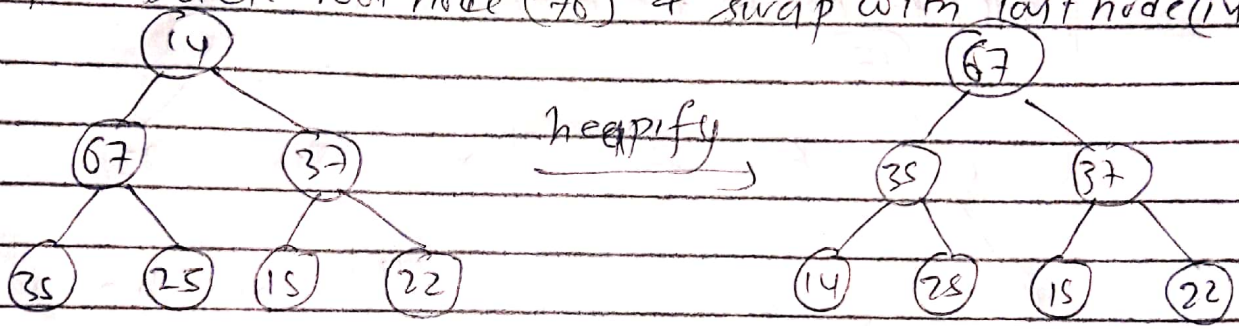


max heap.

After swaping 87 & 22

List: 25, 22, 76, 35, 67, 15, 37, 14, 87, 89.

Step 4: Delete root node (76) & swap with last node(14)

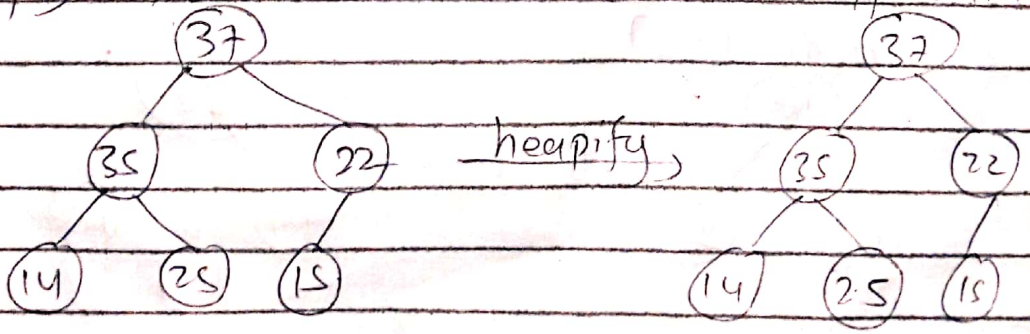


max heap.

List: after swaping 76 & 14.

25, 22, 14, 35, 67, 15, 37, 76, 87, 89

Step 5: Delete root node (67) & swap with last node 37



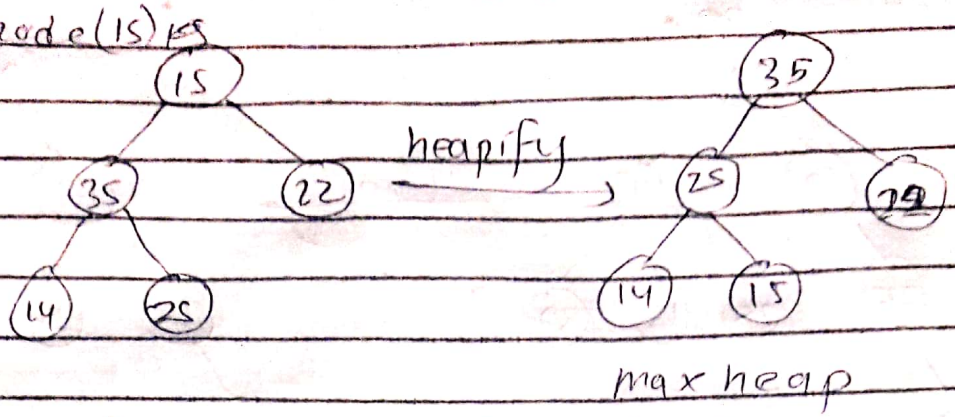
max heap.

list after swaping 67 & 37

25, 22, 14, 35, 37, 15, 67, 76, 87, 89

Date

Step 6: Delete root node (37) and swap with last node (15) ✓



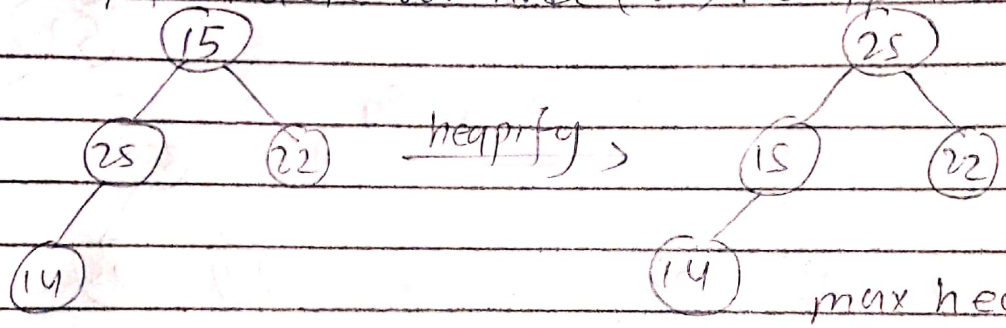
max heap

list after swaping 37 and 15

25, 22, 14, 35, 15, 37, 67, 76, 87, 89

↑ not sorted.

Step 7: Delete root node (35) & swap last node (15)

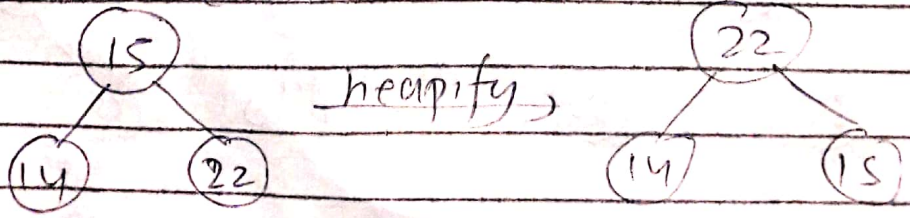


max heap.

list after swaping 35 & 15.

25, 22, 14, 15, 35, 37, 67, 76, 87, 89.

Step 8: Delete root node (25) & swap by last node (15)



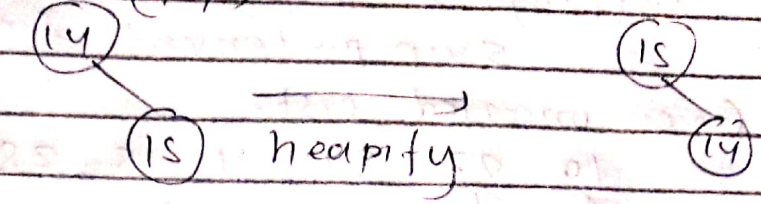
list after swaping 25 & 15

15, 22, 14, 25, 35, 37, 67, 76, 87, 89

↑ not sorted.

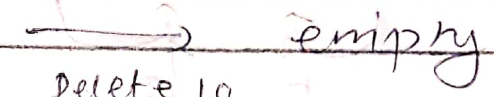
Page No.
Date

Swap step 9: Delete root node(22) and swap with last node (14).



After swaping 22 & 14
list: 15, 14, 22, 25, 35, 33, 67, 76, 87, 89
↑ not sorted.

Step 10: Delete root node(15) & swap with last node (14)



After swaping 15 & 14.
list is sorted as:
14, 15, 22, 25, 35, 37, 67, 76, 87, 89.

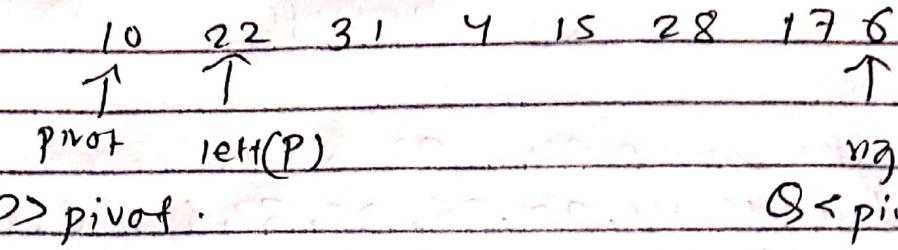
5a) Trace quick sort algorithm for following data
10, 22, 31, 4, 15, 28, 17, 6.

Sol) Steps: 1) Define pivot element, left(P) & right(Q)
Step 2: i) compare left list with pivot.
↳ if list[left] is greater than pivot stop left(P) otherwise move left to the next.
(ii) compare right list with pivot
↳ if list[right] is smaller than pivot stop right(Q) otherwise move right to the previous.
Step 3: Repeat the same until left(P) >= right(Q)
i) if both left(P) & right(Q) are stopped but left < right then swap list[left] with list[right] & continue the process

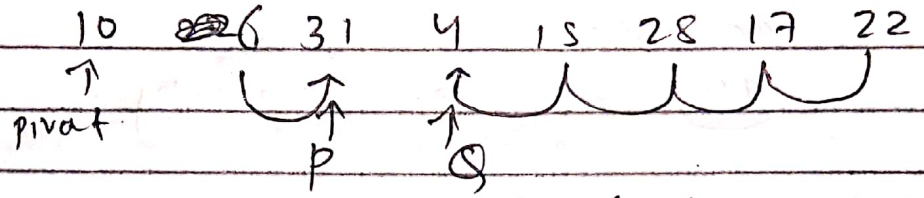
ii) if left >= right then swap list[pivot] with list[right]

swap pivot element & Q.

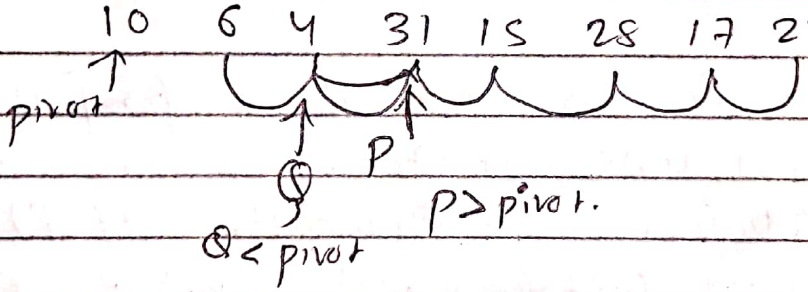
Given unsorted list.



swap P & Q. Since ~~P > Q~~, index of P < index of Q.



swap P & Q [Since ~~P > Q~~, index of P < index of Q]



Swap pivot and Q. [~~Since P.~~

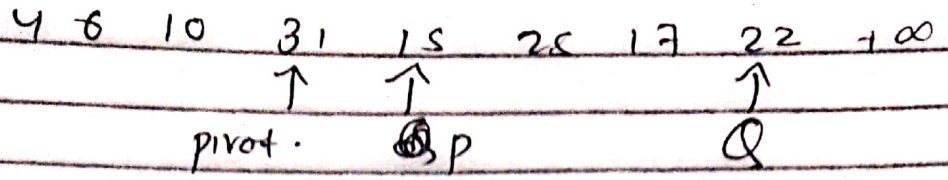
Since index of P > index of Q.

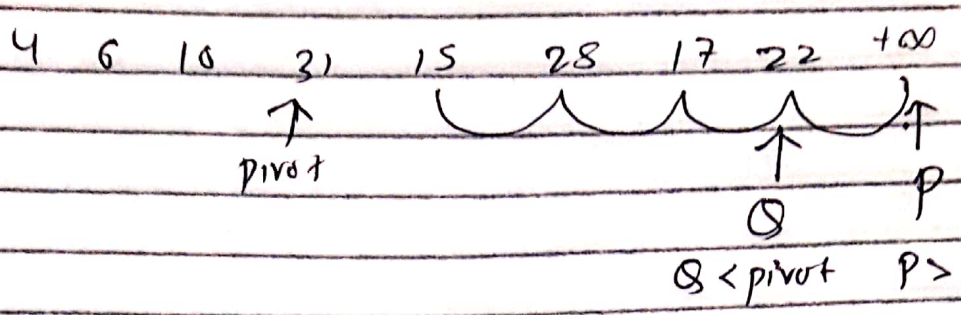
4 6 [10] 31 15 28 17 22.

Here we observed that list of numbers lie to the left side of 10 are small and to the right side are greater. This means 10 is placed at the correct position.

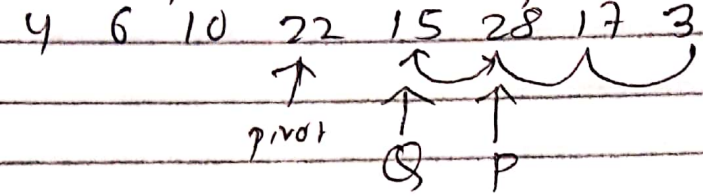
Also, list of left are sorted whereas right's list aren't. So ~~reap~~ repeat the process of for the numbers to the right side of 10.

Date





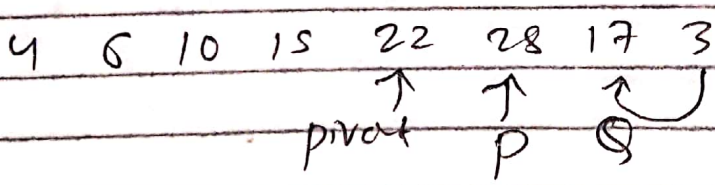
Since position or index of P is greater than index of Q. or the movement is intersected. So swap Q with pivot element.



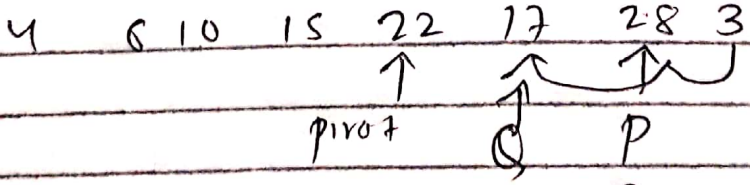
swap Q & pivot

4 6 10 15 22 28 17 31

list until 15 is already sorted.



no intersection of movement so swap P & Q.

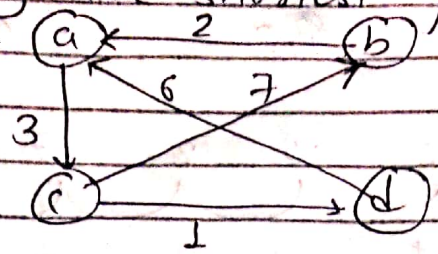


there is intersection of movement so swap Q by pivot element

4 6 10 15 17 22 28 31

The given unsorted list is sorted.

6a) Applying Floyd-Warshall algorithm for constructing the shortest path.



Solution: initial distance matrix.

Step 1:

D_0 =

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

Step 2: create a matrix D_a using D_0

element of 1st row & column are left same where other elements are ~~calculate~~ calculation of shortest path between some to destination through intermediate vertex 'a'.

D_a =

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	∞	7	0	1
d	6	∞	9	0

Step 3: Create a matrix D_b using D_a.

2nd rows & column's elements are same.

Calculate path ~~with help of intermed~~ having through intermediate vertex 'b', a,

(for eg: ~~b→d ⇒ a→b⇒c.~~ c→a ⇒ c→b→a)

Page No.	
Date	

$D_b =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	9	7	0	1
d	6	∞	9	0

Step 4: create matrix D_c using D_b where intermediate vertex is c, a, b. element of 3rd row & columns are left same.

$D_c =$

	a	b	c	d
a	0	10	3	4
b	2	0	5	∞
c	9	7	0	1
d	6	∞	9	0

~~D_c~~

$D_c =$

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	9	7	0	1
d	6	16	9	0

Note: can travel from any vertex with help of a, b, c

Step 4: Create matrix D_d using D_c where intermediate vertex is d, a, b, c.

$D_d =$

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	7	7	0	1
d	6	16	9	0

This last matrix D_4 represents the shortest path between every pair of vertex.

6b) Define graph. What are the difference between traversing in graph and traversing in tree? Explain with suitable example.

Ans: A Graph G is a collection of two sets V and E where V is the collection of vertices $V_0, V_1 \cdots V_{n-1}$ and called nodes & where E is collection of edges $e_1, e_2, \cdots e_n$. where an edges in an arc which connects two nodes. This can be represented as.

$G = (V, E)$

$V(G) = (V_0, V_1 \cdots V_{n-1})$ or set of vertice

$E(G) = (e_1, e_2 \cdots e_{n-1})$ or set of edges.

Difference between traversal in graph and traversal in tree.

1. There is no 1st node or root node in graph. Hence traversal can start from any node. In tree or list when we start traversing from 1st node, all nodes are traversed which are reachable from the starting node. If we want to traverse all the reachable nodes we again have to select another starting node for traversing the remaining nodes.

2. In tree while traversing we never encounter a node more then once but while traversing graph, there may be a possibility that we reach a node more than once.

3. In tree traversal, there is only one sequence

in which nodes are visited but in graph for the same techique of traversal there can be different sequences in which node can be visited.

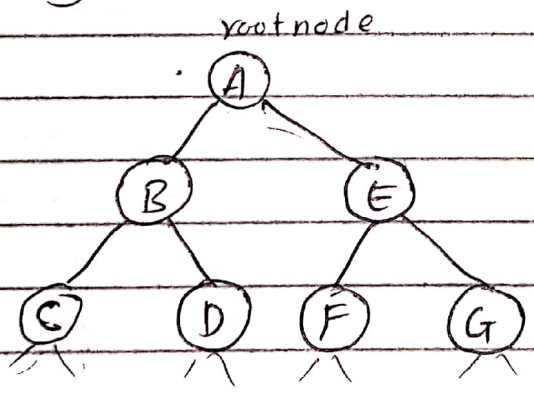
Tree traversal

1) Preorder traversal
   → visit root node
   → visit all nodes in left subtree
   → visit all nodes in right subtree.

2) Inorder traversal
   → First, visit all the nodes in left subtree.
   → then the root node.
   → visit all nodes in the right subtree.

3) Postorder traversal.
   → First, visit the nodes in left subtree
   → then nodes in right subtree
   → lastly visit the root node.

Eg:



Preorder traversal: A B C D E F G
Inorder traversal: C B D A F E G
postorder traversal: C D B F G E A

Graph traversal: same as: [2021 spring 6b)]