*For all solutions cases are refered from Cormen "Introduction to Algorithm"

Q:-1

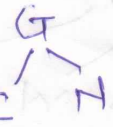
a) minimum degree $t=2$

no. of key $t-1$, $2t-1$
(1) (3)

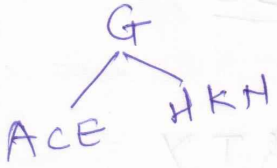
children (2) (4)

CNGAHEKQMFWLTZDPRXYS

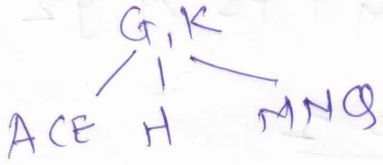
insert CGN

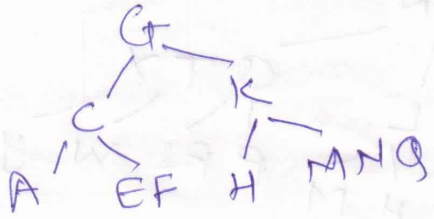


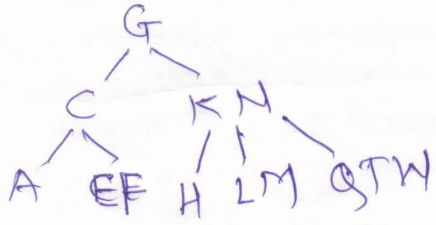
insert AHEKN

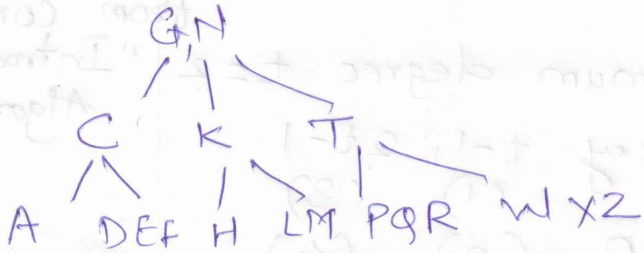


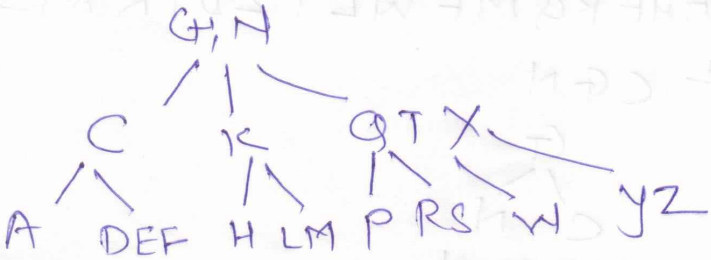
insert M



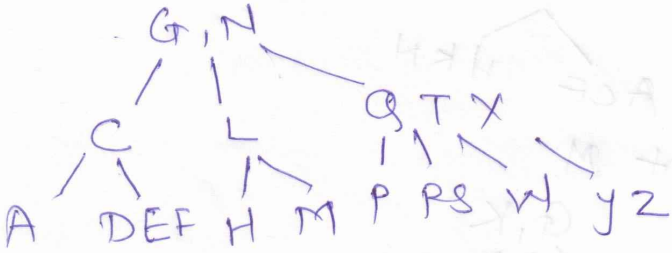




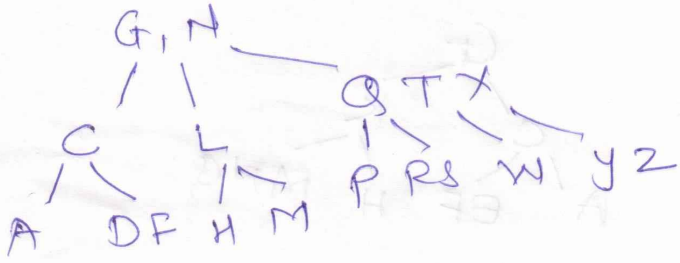




Delete K → <u>case 2b</u>



Delete E → <u>case 1</u>



Que. 1 b)

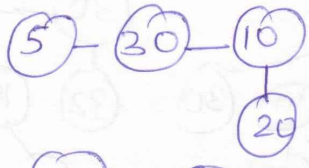
Binomial heap.

L = 20 10 5 30 24 48 14 43 29 31 32

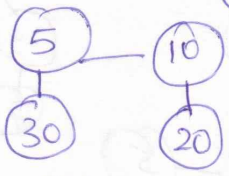
1) Insert 10, 20 2010

case4

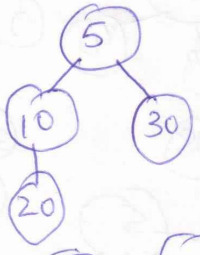
2) Insert 5
30



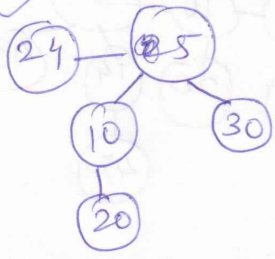
Case 3



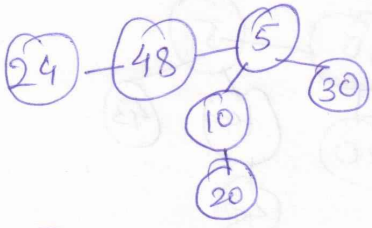
Case 3



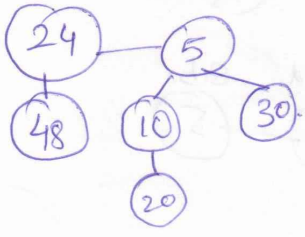
3) Insert 24



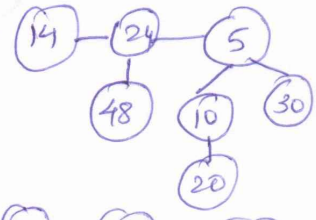
4) Insert 48



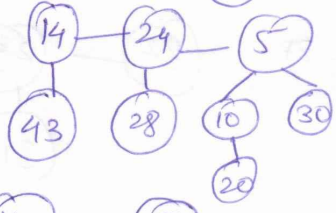
Case 3



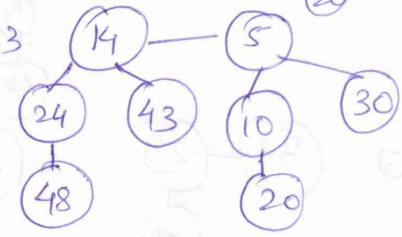
5) Insert 14



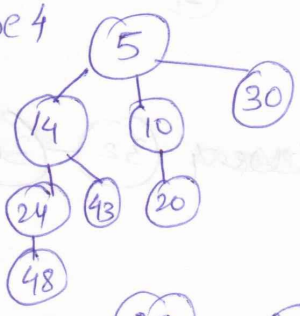
6) Insert 43
Case 3



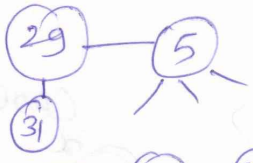
Case3



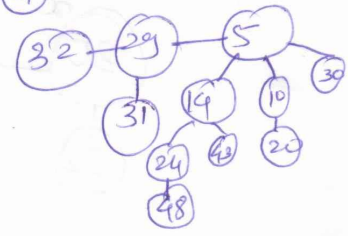
Case 4



7) Insert 29
31
Case3

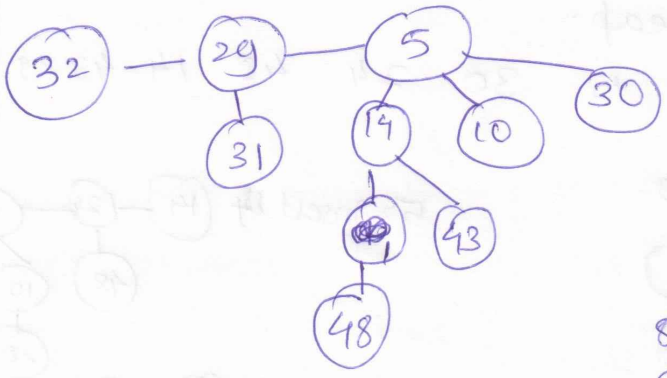


8) Insert 32

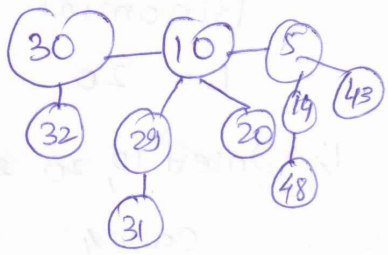


Delete 24.

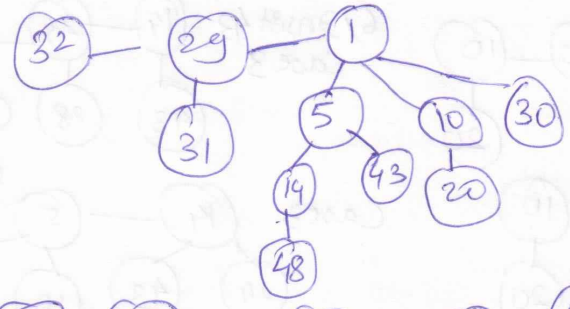
1)



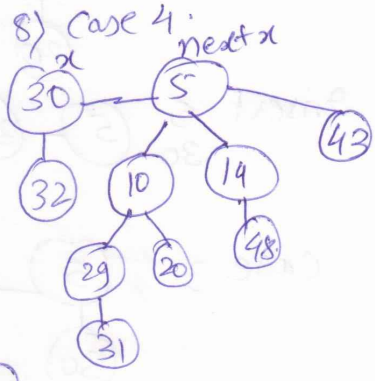
7) case 4



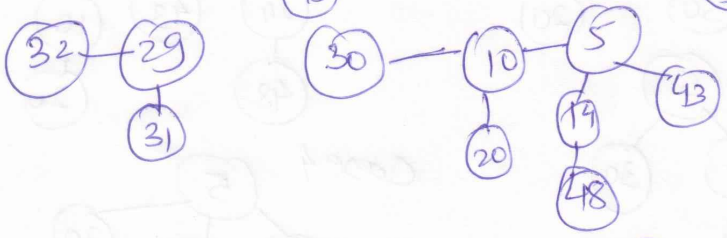
2)



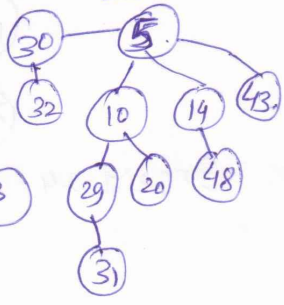
8) case 4.



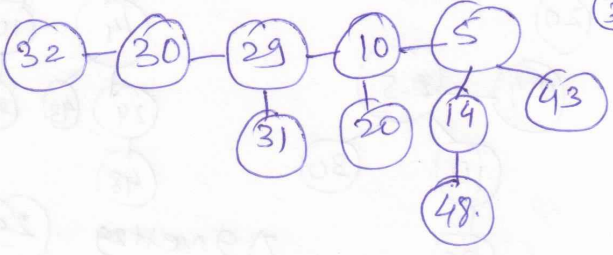
3)



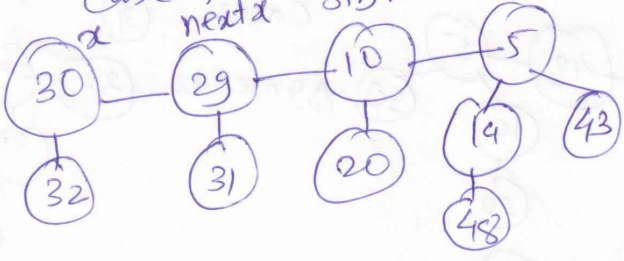
9) case 1



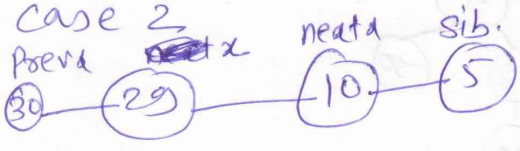
4) ~~case 4~~.



5) case 4



6) case 2



Que. 2a

Solution.

1) Bellman ford algorithm is a single-source shortest path algo, which allows for negative edge weight and can detect negative cycles in a graph.

Dijkstra algo is also single-source shortest path algo. However, the weight of all edges must be non-negative

2) Bellman ford algo has $O(|V||E|)$ complexity

Dijkstra algo has $\Theta((|E|+|V|)\log(|V|))$ complexity.

3) The Bellman ford algo's nodes contain only the information that are related to. This info allows that node just to know about which neighbor nodes can it connect and the node that the relation come from, mutually.

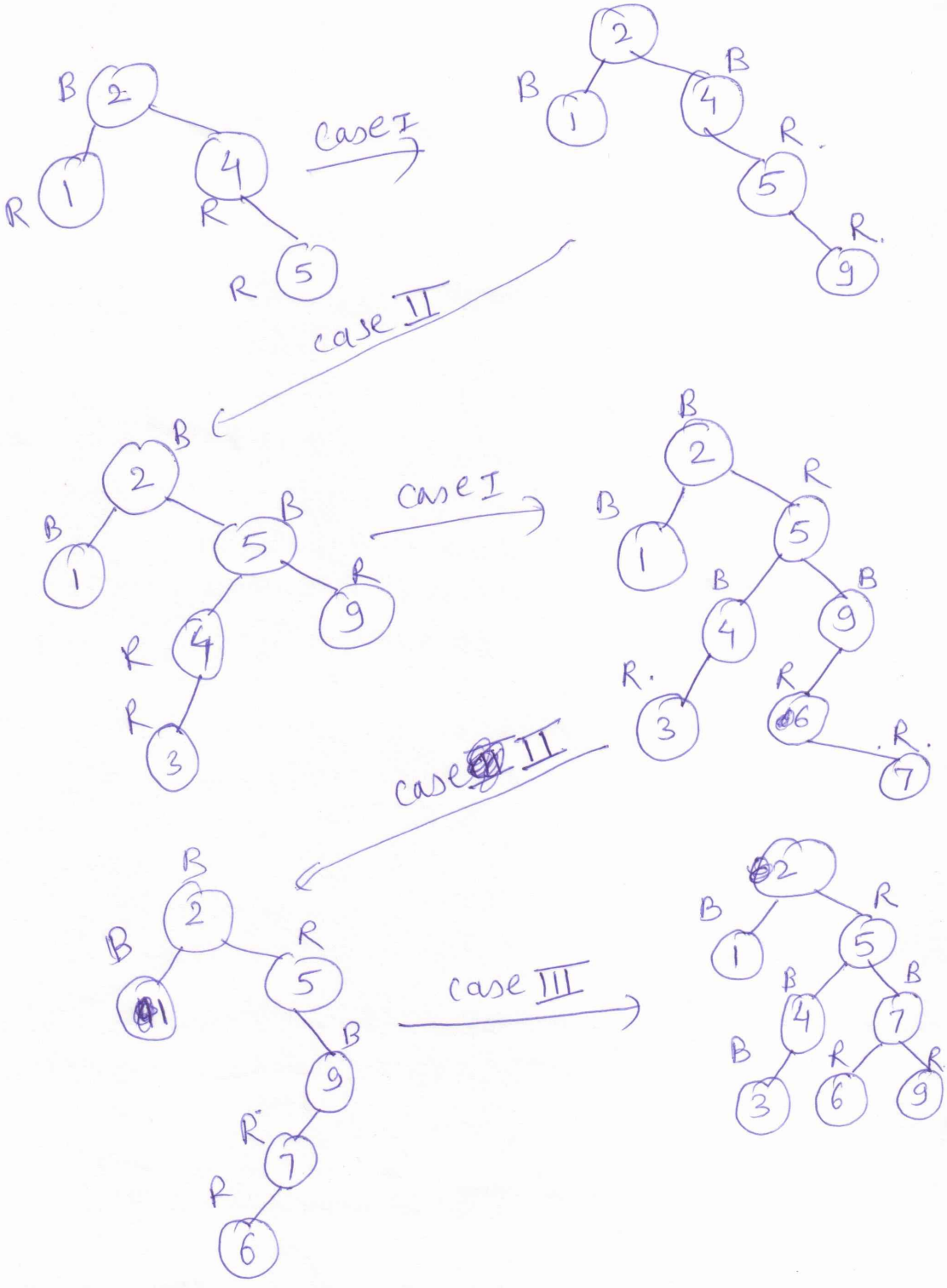
Dijkstra's algo contains whole info of a network.

4) Dijkstra's algo is faster than Bellman-Ford's algo. But Bellman-Ford algo can be more useful to solve some problems, such as negative weights of paths.

Que. 2b.

Solution.

2 1 4 5 9 3 6 7



Q2c) Suppose that the graph G = (V, E) is represented as an adjacency matrix. Give a simple implementation of Prim's algorithm for this case that runs in $O(V^2)$ time.

Solution:

We would need to sort through V twice. On line 8, rather than the adjacency list, we would loop from 1 to V.

Here's what it would look like:

MST-PRIM(G, r) /* G = (V, E) */

```
1.For i=1 to V
2.Dist[i]=∞
3 Pred[i]=∞
4.Dist[r]=0
5 CREATE minimum priority queue, Q, for indexes of vertices
6 If Q not empty
7. i=EXTRACT-MIN(Q)
8. For j=1 to V
9. If m[i,j]=1
10 If j is in Q and weight(i, j) < dist[j]
11. Pred[j]=weight (i,j)
```

Q2.d).

d.) Explain the structure of fibonacci heaps.

Ans) Structure of Fibonacci heaps

Like a binomial heap, a Fibonacci heap is a collection of min-heap-ordered trees. The trees in a Fibonacci heap are not constrained to be binomial trees, however. Figure 20.1(a) shows an example of a Fibonacci heap.

Unlike trees within binomial heaps, which are ordered, trees within Fibonacci heaps are rooted but unordered. As Figure 20.1(b) shows, each node x contains a pointer p[x] to its parent and a pointer child[x] to any one of its children. The children of x are linked together in a circular, doubly linked list, which we call the child list of x. Each child y in a child list has pointers left[y] and right[y] that point to y's left and right siblings, respectively. If node y is an only child, then left[y] = right[y] = y. The order in which siblings appear in a child list is arbitrary.

Circular, doubly linked lists (see Section 10.2) have two advantages for use in Fibonacci heaps. First, we can remove a node from a circular, doubly linked list in O(1) time. Second, given two such lists, we can concatenate them (or "splice" them together) into one circular, doubly linked list in O(1) time. In the descriptions of Fibonacci heap operations, we shall refer to these operations informally, letting the reader fill in the details of their implementations.

min[H]

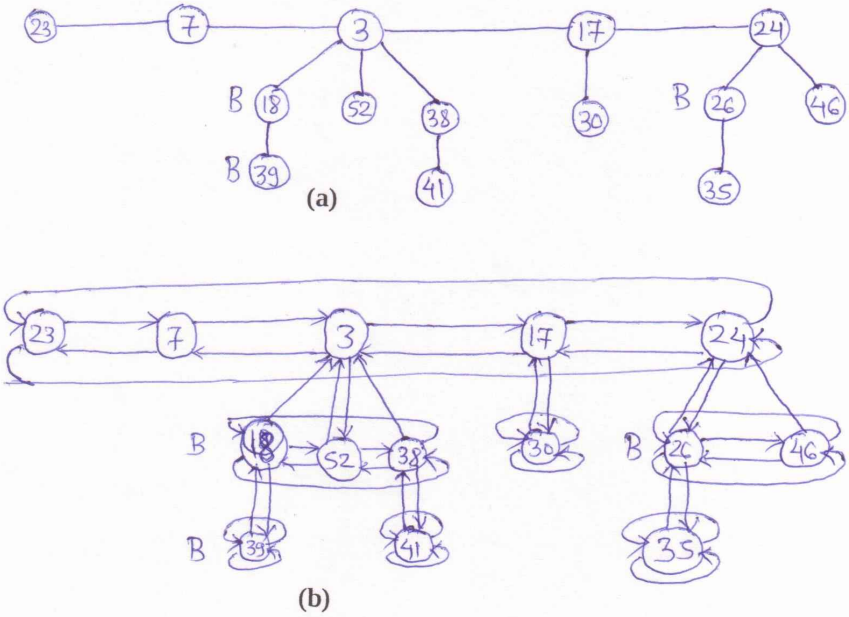


Figure (a) A Fibonacci heap consisting of five min-heap-ordered trees and 14 nodes. The dashed line indicates the root list. The minimum node of the heap is the node containing the key 3.The three marked nodes are blackened. The potential of this particular Fibonacci heap is 5+2·3 = 11.

(b) A more complete representation showing pointers p (up arrows), child (down arrows), and left and right (sideways arrows). These details are omitted .

all the information shown here can be determined from what appears in part (a).

Two other fields in each node will be of use. The number of children in the child list of node x is stored in degree[x]. The boolean-valued field mark[x] indicates whether node x has lost a child since the last time x was made the child of another node. Newly created nodes are unmarked, and a node x becomes unmarked whenever it is made the child of another node. Until we look at the D ECREASE -K EY operation , we will just set all mark fields to FALSE .

A given Fibonacci heap H is accessed by a pointer min[H] to the root of a tree containing a minimum key; this node is called the minimum node of the Fibonacci heap. If a Fibonacci heap H is empty, then min[H] = NIL .

The roots of all the trees in a Fibonacci heap are linked together using their left and right pointers into a circular, doubly linked list called the root list of the Fibonacci heap. The pointer min[H] thus points to the node in the root list whose key is minimum. The order of the trees within a root list is arbitrary.

We rely on one other attribute for a Fibonacci heap H : the number of nodes currently in H is kept in n[H].