Page No.
Date

2019140032
waw

Q1.

a 1) Infix to prefix:

(i) ((a+b) * (c/d * e))

*** + ab / c * de**

(ii) ((a+b-c) * (d/e))

***+ab-c/de**

~~* + a - bc / de~~

2) Application of queue:

→ Eg. CPU scheduling or Disk scheduling

where a resource is shared among multiple consumers.

→ Scheduling jobs can use priority queue.

→ breadth first tree traversal makes use of a queue.

→ Path finding algorithms.

→ When data is asynchronously transferred between two processes, eg I/O buffers, pipes, etc.

Page No.
Date

3) Algo:

2019140032

waw

- Take a count variable and initialize it to zero
- traverse the linked list and increment count variable
- When a node points to null, it means we reached the end and we return the value of count variable.

Code:

```
void no
int no_of_nodes(){

    struct node* temp = head;

    int count = 0;

    while ( temp != NULL)

    {

        temp = temp -> next;

        count ++;

    }

printf (" The number of nodes is %d", count);

return count;

}
```

Date

2019140032

waw

15

OR

1)- balance factor : The balance factor of a binary tree is the difference in heights of its two subtree ($h_R - h_L$).

the balance factor of a height balanced binary tree may take on of the value $-1, 0, +1$

If it is not among them then that node is unbalanced and rotation is needed.

2) to find balance factor :

Code :

```
void updatefactor (tree * root)
{
    if (root == NULL)
        return;
    root -> bf = height (root -> left) - height (root -> right);
}
```

Page No.
Date

2019140032
waw

```
// Height

int height (tree* root)

{
    if(root == NULL)
        return 0;

    return ( 1 + max (height (root -> left), height (root -> right));

}

// max

int max (int a, int b)

{
    if (a > b)
        return a;

    else
        return b;

}
```

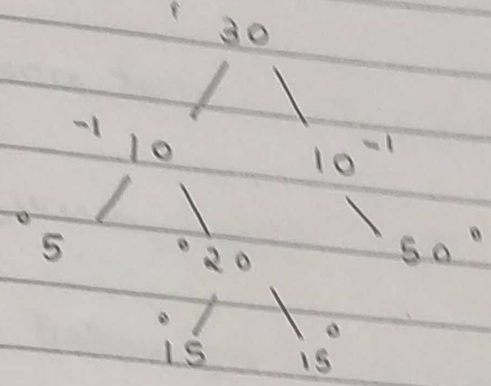
Q1

(B) OR

2017400032

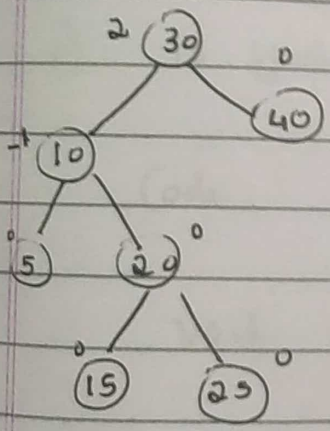
Waw

3)



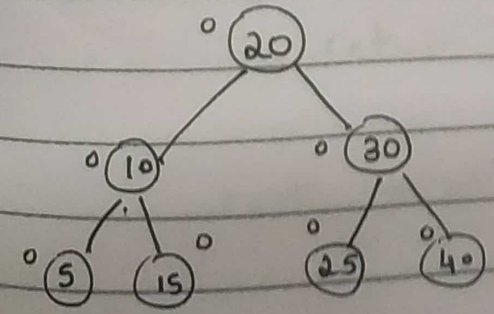
deleting 50:

As 50 has 0 children, we will directly delete 50 and then update the balace factor of the tree.



=> Here the balance factor of node 30 is 2, hence we perform rotation.

We perform LR rotation to get this final tree which is height balanced.



Final tree

Page No.
Date

2019140032
waw

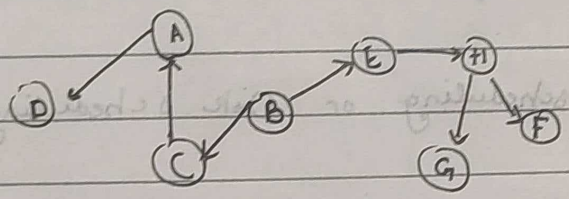
Q2.

(A) We maintain two arrays visited and stack, we enter starting element and push the adjacents and mark it visited.

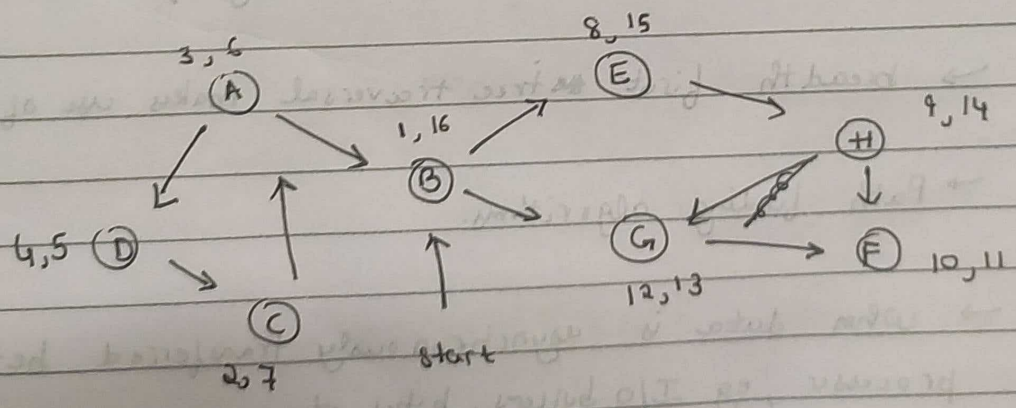
stack:			
~~D~~ 4/5	~~F~~ 10/11	~~G~~ 12/13
~~A~~ 3/6	H 9	~~H~~ 9/14
~~C~~ 2/7	E 8	~~E~~ 8/15
B 1	B 1	~~B~~ 1/16

If all the neighbours have been visited, then we backtrack and pop

DFS graph:



With start time, end time:



DFS order:

B → C → A → D → E → H → F → G

↑ start

↓ Finish.

Page No.
Date

20191A0032
UDAY

Q2

(B)

OR

Exctract Min is the most important operation on a fibonacci heap.

Here we remove the node with minimum value and the tree is re-adjusted.

Algo:

1. Delete the min node.

2. Set the min pointer to the next root in the root list.

3. Create an array of size equal to maximum degree of the trees in the heap before deletion.

4. Do the following steps (5 to 7) until there is no multiple roots with same degree.

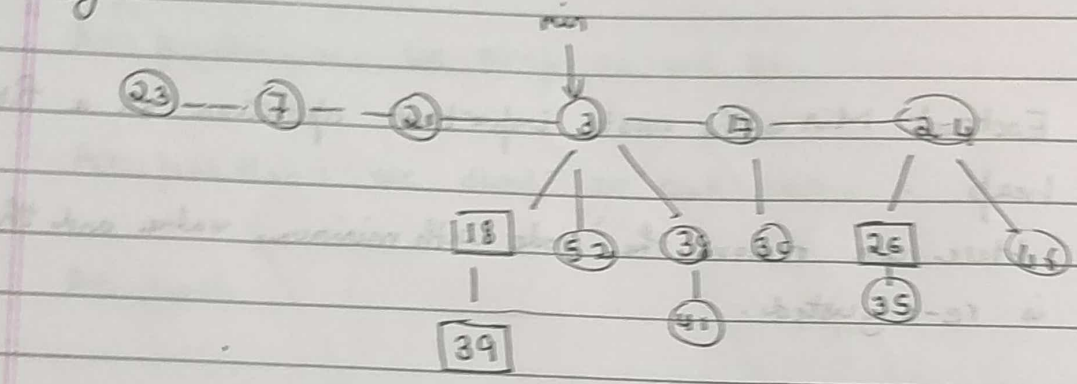
5. Map the degree of the current root (min-pointer) to the degree in array.

6. Map the degree of next root to the degree in array.

7. If there are more than two mappings for the same degree, then apply union operation to those roots such that the min heap property is maintained.

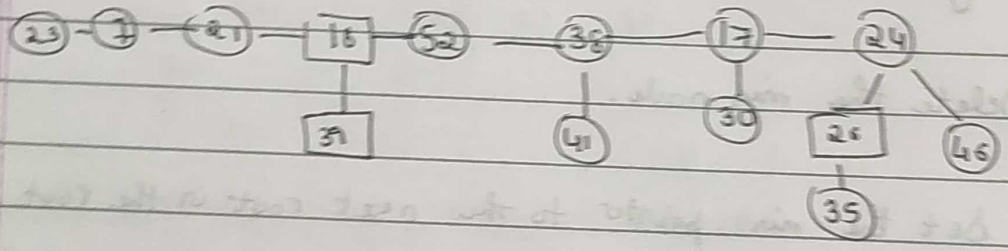
2019140038

Eg.

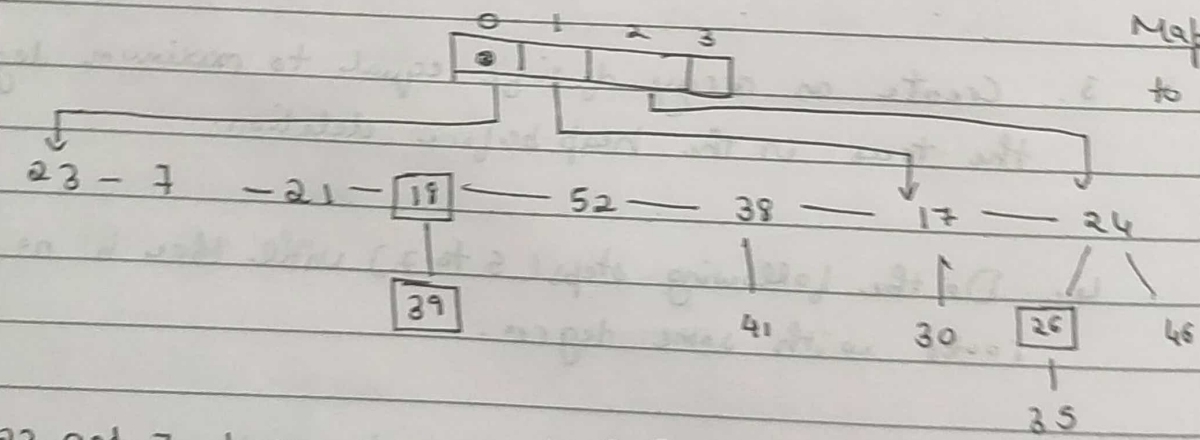


□ ⇒ marked

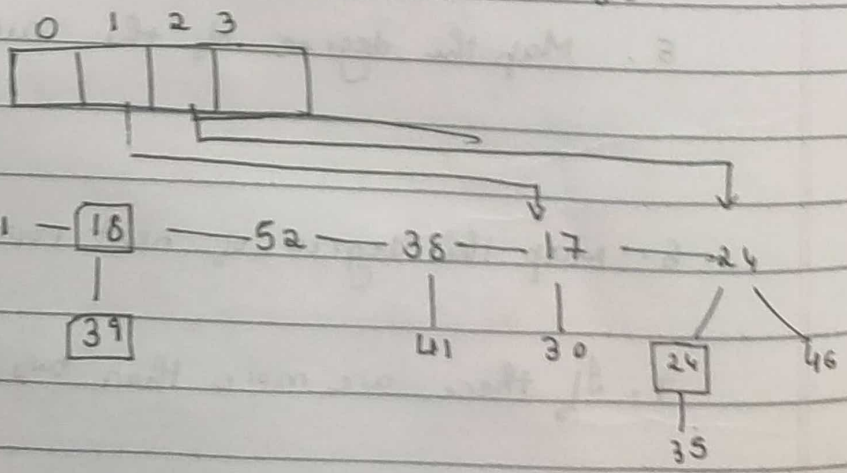
delete



Map degree to array



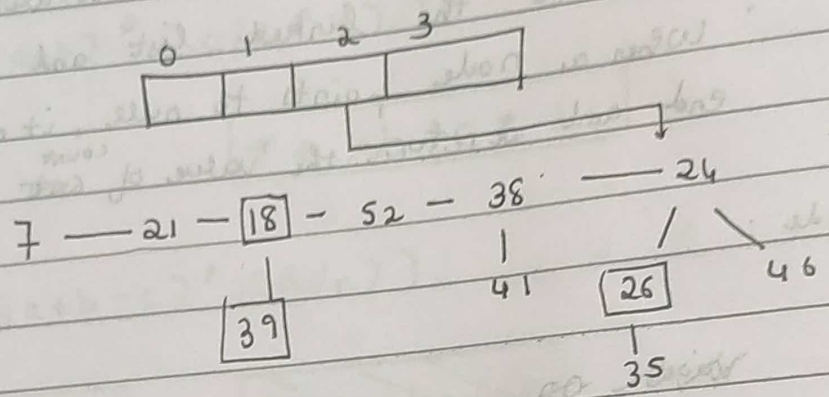
23 and 7 have same degree, ∴ Unite



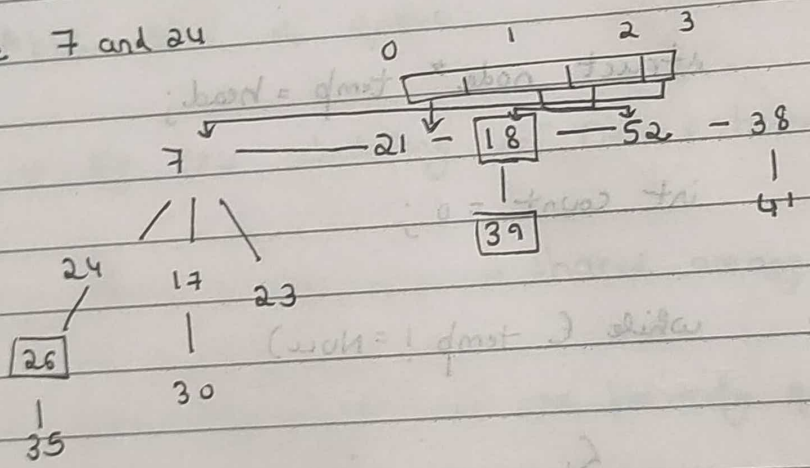
Min 7 ∴ root

2019140032
saw

7 and 17 have same degree ∴ unite

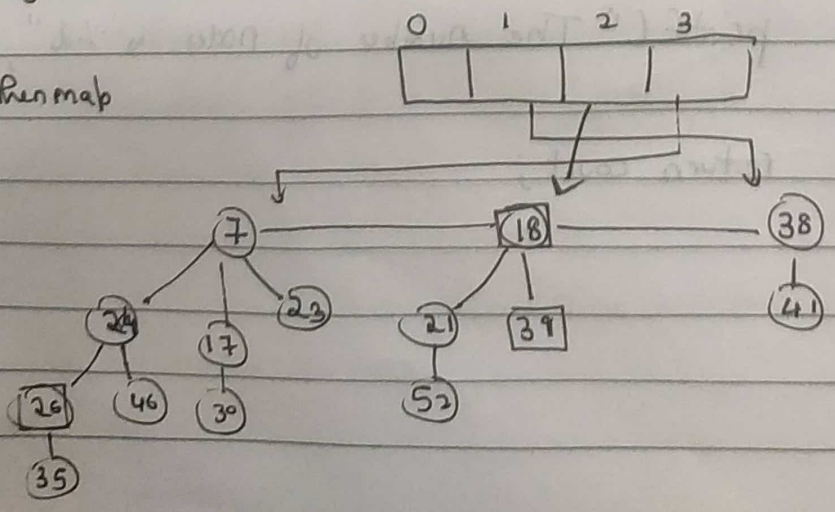


Unite 7 and 24



52 and 21 same degree unite then 21 and 18 same degree unite

then map



2019140032

Q3

(A). Properties of B-tree:

1. All leaves are at the same level.

2. A B-tree is defined by the term minimum degree 't'. The value of t depends upon disk block size.

3. Every node expect root must contain at least (ceiling) ([t-1] /2) keys. The root may contain minimum 1 key.

4. All nodes (including root) may contain at most t-1 keys.

5. No. of children of a node equals no. of keys in it plus one.

6. All keys of a node are sorted in increasing order. The child between two keys K1 and K2 contain all keys in the range from K1 and K2.

7. B-tree grows and shrinks from root which is unlike binary search tree. BST grow downward and also shrink from downwards.

8. Like other balanced BSTs, time complexity to search, insert and delete is O(log n).

Page No.
Date

2019140032
wow

9. It is optimized for systems that read and write large blocks of data. It is common in database and file systems.

10. It is used for insertion, deletion and seaching operations.

Insert operation:

Case (a): ~~Odd degree~~ No splitting: If ~~after~~ the number of keys are less than or equal to m-2 then we directly add it in that node.

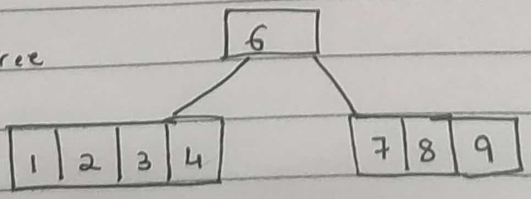
Case (b) If odd degree and more than m-1 keys in that node:

We divide that node into two nodes (with min no. of keys) and the middle key is promoted to parent.

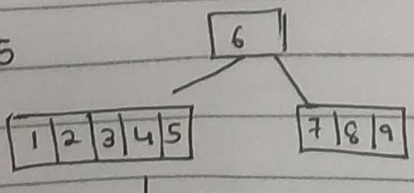
If the parent overflows then repeat [recursion]

If root overflows then make a new node and make that the root.

Eg. order 5 b tree



Insert 5

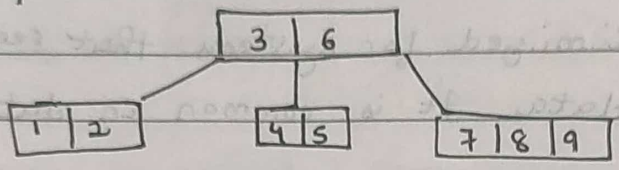


↳ this node overflows

Page No.
Date

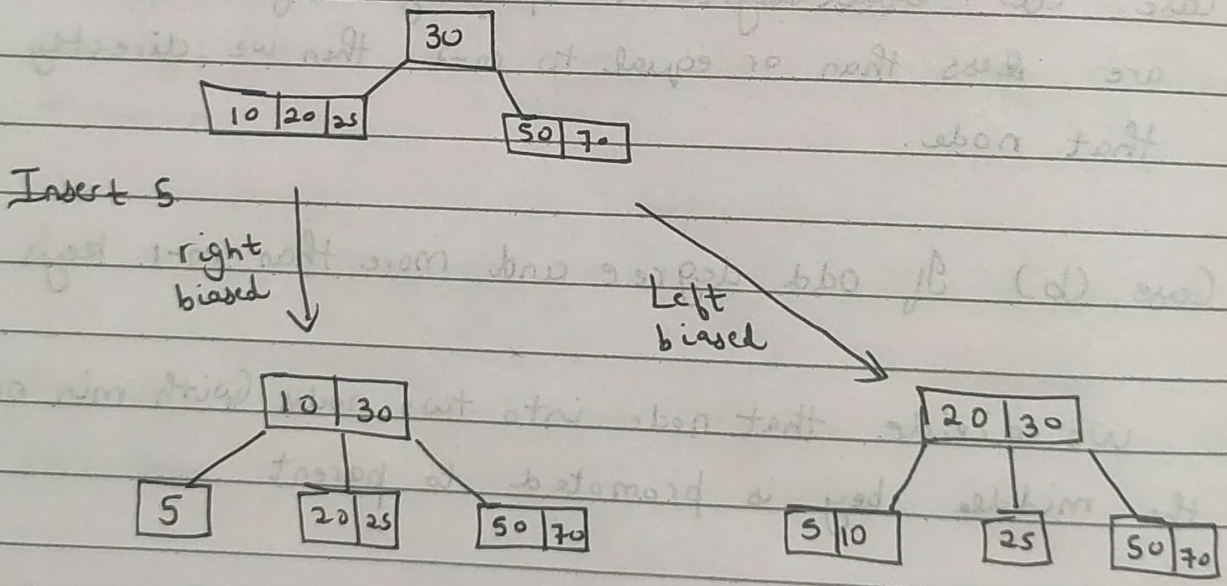
2019140032
UCW

∴ Split



Case (c) : If order is even then according to the question we do either right or left biased splitting.

Eg. order 4



ADS

2019140032

saw

Q3.

(B).

Desired properties of hash function:

→ The hash function should uniformly distribute the data across the entire set of possible hash values. This reduces the the number of collisions and thus increasing efficiency.

→ ~~The hash~~ Each table position equally likely for a key.

→ The hash function should generate very different hash values for similar strings.
this allows distribution of data over the hash table.

→ The hash function should be computed with efficiency.

→ The hash value is fully determined by the data being hashed.

→ The hash function uses all the input data.

Page No.
Date

2019140082
Waw

(i) Linear probing:

m = 13.

formula: (k + i) mod m = h(k; i)

→ • 26 %. 13 = 0 ✓ h(26; 0)

∴ T[0] = 26

• h(52; 0) = (52+0) %. 13 = 0 ×

collision as filled ∴ increment i

1st probe:

h(52; 1) = (52+1) %. 13 = 1 ✓

T[1] = 52.

• h(12; 0) = (12+0) %. 13 = 12 ✓

T[12] = 12

• h(24; 0) = (24+0) %. 13 = 11 ✓

T[11] = 24

• h(77; 0) = (77+0) %. 13 = 12 ×

Collision

1st probe:

h(77; 1) = 78 %. 13 = 0 ×

Collision

2nd probe:

h(77; 2) = 79 %. 13 = 1 ×

3rd probe :

$h(77;3) = 80\% \cdot 13 = 2$ ✓

$\therefore T[2] = 77.$

- $h(80;0) = (80+0)\% \cdot 13 = 2$ ×

Collision

1st probe

$h(80;1) = (80+1)\% \cdot 13 = 3$ ✓

$\therefore T[3] = 80$

- $h(75;0) = (75+0)\% \cdot 13 = 10$ ✓

$T[10] = 75$

- $h(30;0) = (30+0)\% \cdot 13 = 4$ ✓

$T[4] = 30$

∴ Final table

T ⇒

0	26
1	52
2	77
3	80
4	30
5	-
6	-
7	-
8	-
9	-
10	75
11	24
12	12

Page No.
Date

2019140032
wow

(ii) Quadratic:

formula:

$$h(k, i) = [h_1(k) + c_1 i + c_2 i^2] \bmod m$$

where $c_1 = 0$, and $h_1(k) = k$; $c_2 = 1$

$$= (k + 0 \cdot i^2) \bmod m$$

- $h(26; 0) = 26 \,\%\, 13 = 0$ ✓

  $T[0] = 26$

- $h(52; 0) = 52 \,\%\, 13 = 0$ ×

  $h(52; 1) = (52 + 1^2) \,\%\, 13$

  $= 1$ ✓

  $T[1] = 52$

- $h(12; 0) = 12 \,\%\, 13 = 12$ ✓

  $T[12] = 12$

- $h(24; 0) = 24 \,\%\, 13 = 11$ ✓

  $T[11] = 24$

- $h(77; 0) = 77 \,\%\, 13 = 12$ ×

  $h(77; 1) = 78 \,\%\, 13 = 0$ ×

  $h(77; 4) = 81 \,\%\, 13 = 3$ ✓

  $T[3] = 77$.

Page No.
Date

2019140082
waw

- $h(80, 0) = 80 \% 13$
  $= 2$ ✓
  $T[2] = 80$

- $h(75, 0) = 75 \% 13$
  $= 10$ ✓
  $T[10] = 75$

- $h(30, 0) = 30 \% 13$
  $= 4$ ✓
  $T[4] = 30$

Final table:

T =>	
0	26
1	52
2	80
3	77
4	30
5	-
6	-
7	-
8	-
9	-
10	75
11	24
12	12