Name : Ananya Prasad

Reg No: 20BCE10093

Date : 2 sept, 2021

Code/slot : CSE 2002 / C11+DC1

Faculty : Meenakshi Ma'am

Exam : Interim 2021-22 / TEE

1) Insertion sort is one of the sorting algorithms which assumes that a single element is always sorted. ①

The array gets divided into two sub arrays - sorted and unsorted array.

Principle ⇒

Compare the element with the adjacent element.

On comparison ; find the position in sorted array where the element can be inserted.

Shift the elements to the right and insert the 'value'.

Repeat till the unsorted array becomes empty and all the elements are sorted

~~Example~~.

It is not the best sorting technique, but is far better than bubble sort and selection sort.

Time complexities ⇒ Worst case → $O(n^2)$

Best case → $O(n \log n)$

Average case → $O(n \log n)$

Example ⇒ 7 2 4 1 5 3

7	2	4	1	5	3
0	1	2	3	4	5

⇒ pick up value = 2, take it out and store in value to create a hole. Shift cells to right which are greater than 2., shift 7 towards hole here

2	7	4	1	5	3
0	1	2	3	4	5

→ 4 = value ⇒

2	4	7	1	5	3
0	1	2	3	4	5

⇒ 1 = value

1	2	4	7	5	3
0	1	2	3	4	5

⇒ 5 = value ⇒

1	2	4	5	7	3
0	1	2	3	4	5

⇒ 3 = value.

1	2	3	4	5	7

⇒ 7 = value = sorted.

Pseudocode →

Insert (A, n)

```
{       for  i ← 1 to n-1
        {       value ← A[i]                              } C1 (n-1) times
                hole ← i

                while ( hole > 0 && A [hole -1 > value) →
                A [hole] ← A [hole -1]
                hole ← hole -1                             } C2
        }
        A[hole] ← value.                                   → C3 (n-1) times
}
```

$T(n) = (C_1 + C_3)(n-1)$

$= an + b = O(n) = a^n + bn + c = O(n^2)$

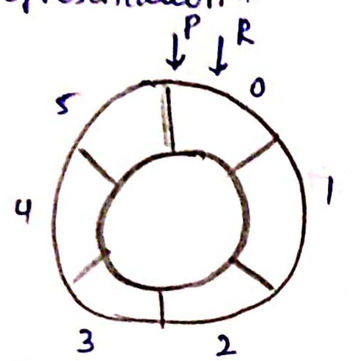
$+$

$\frac{n(n-1)}{2} C_2$

(2) Circular Queue

* Linear data structure.
* Follows FIFO principle → First in First out . for a node.
* In circular queue last node is connected back to the first node to make it a circle.
* Elements are added at the rear end and elements are deleted at front of the queue.
* Front and rear are both pointing to the beginning of the array.
* Time taken for insertion and deletion O(1).

Representation →



Queue empty
max = 6
Front = rear = 0
(Insertion & deletion from same point)

Algorithm for insertion and deletion ~~from~~ in circular queue.

Insertion →

Step 1 :- If FRONT = 0 and REAR = MAX = -1 OR REAR = FRONT - 1 then
　　　　Write "Overflow"
　　　　GOTO step 4
　　End If

Step 2 : If FRONT = -1 and REAR = -1 , then
　　　　SET FRONT = REAR = 0
　　ELSE IF REAR = MAX - 1 and FRONT != 0
　　　　SET REAR = 0
　　ELSE
　　　　SET REAR = REAR + 1.
　　End if.

Step 3 : set Queue [REAR] = Val

Step 4 : Exit.

Deletion →

Step 1 : IF Front = -1 then
    write " Underflow "
    GOTO step 4
  End IF .

Step 2 : SET val = Queue [Front]

Step 3 : IF Front = Rear
    SET Front = Rear = -1
  ELSE
    IF Front = MAX - 1
      SET Front = 0
    ELSE
      SET Front = Front + 1
    End IF
  END IF

Step 4 : Exit .

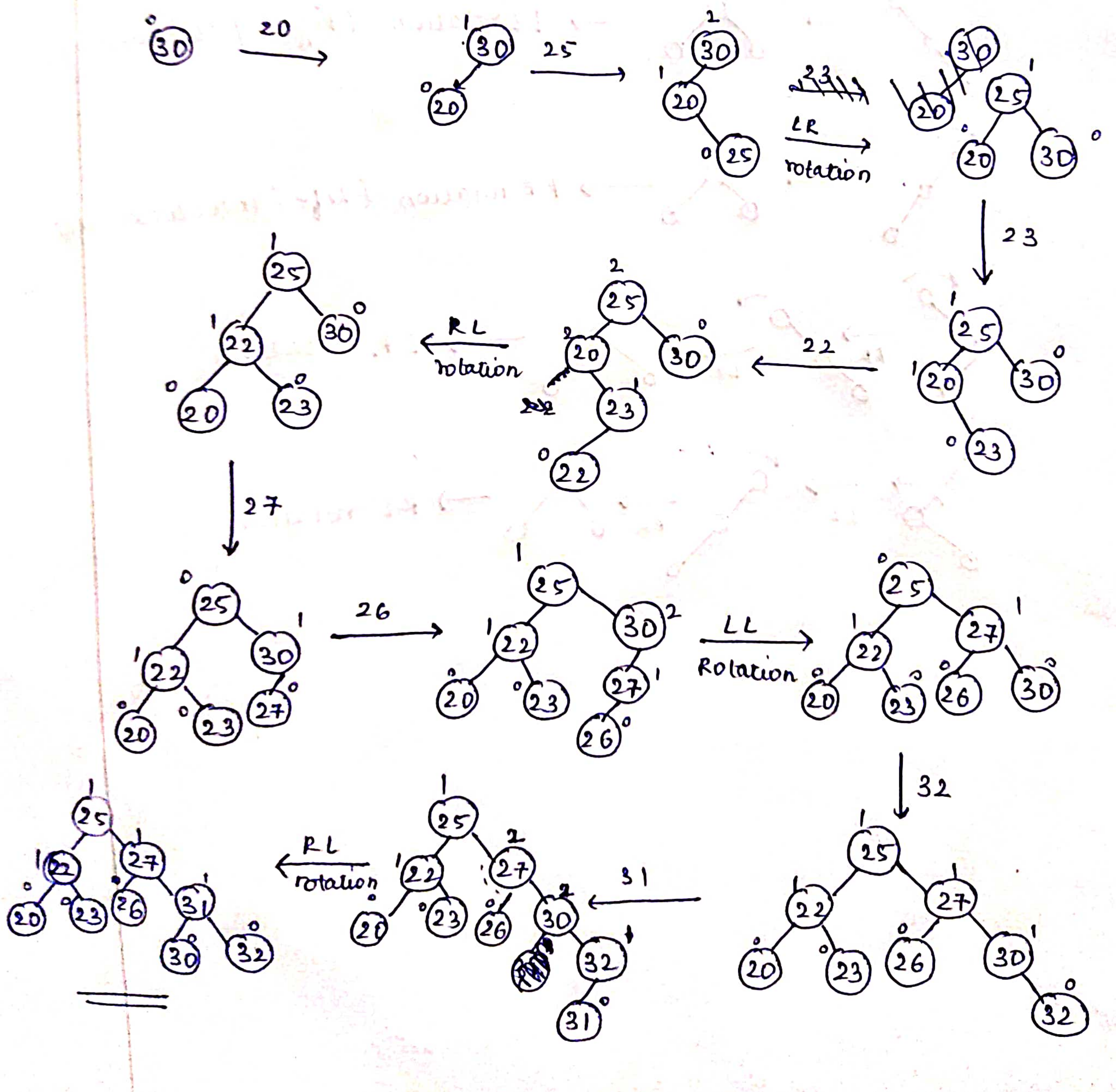
(3) **AVL TREES**

* Self balancing trees binary search trees : Principle.
* The difference between the heights of left and right subtrees cannot be more than one for all nodes.

~~Nodes~~

* Difference in nodes can be either -1 or 0 or 1.
* Balance factor = height of left - height of right.
* The balance factor varies from -1, 0 and 1.

→ 30, 20, 25, 23, 22, 27, 26, 32, 31



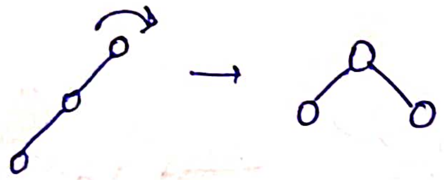
AVL trees are height balancing trees, insertion and deletion have low time complexities.

~~Properties~~
Properties →

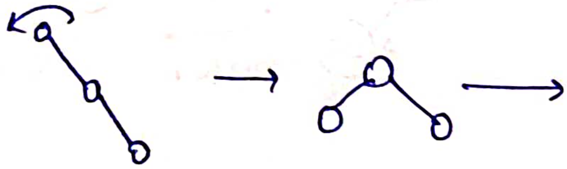
AVL trees properties are identical to binary search trees except the fact that for every node in AVL, height of left and right subtrees can differ by atmost 1.

Rotations in AVL trees →

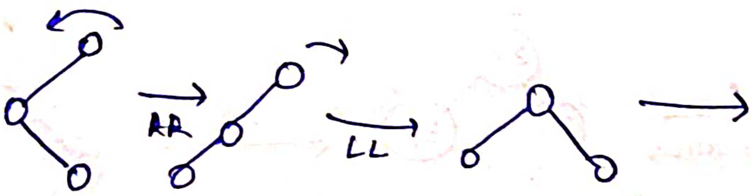
single rotations (for degenerate parts)



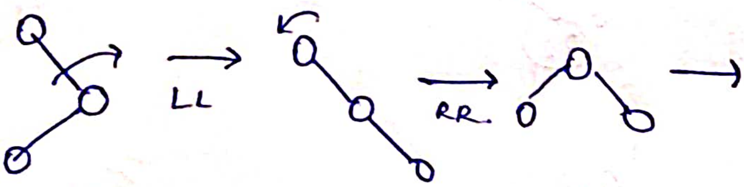
→ LL rotation (Right / clockwise)



RR rotation (left/anticlockwise)



→ LR rotation



→ RL rotation

4

QUICK SORT

Divide and conquer algorithm

First divides a large list into two smaller sublists, low and high.

Steps →

Pick an element (pivot)

Reorder the list so that all elements with values less than pivot come before pivot and greater after. (Partition)

Recursively apply the above steps to the sublist of elements with smaller values and separately for greater.

eg

17	11	24	19	13	27	16	29	23

pivot = 3

move to the end.

17	11	24	19	23	27	16	19	13

Partition

Move the left to the right until it reads a value greater than or equal to the pivot.

17 → move the right bound to the left and till it reaches a value greater than or equal to 17.

11	17	24	19	23	27	16	19	13

13 < 17, swap

11	13	24	19	23	27	16	19	17

Left sublist, sorted, calling quicksort.

pivot 27, swap 27 & 17

11	13	24	19	23	17	16	19	27

again partition

exchange 29 and 27

11	13	24	19	23	17	16	27	29

Call quicksort on left sublist of 27

pivot 23, swap 23 & 16

us   S

11	13	24	19	16	17	23	27	29

Repeat.

Swap 24 & 17

~~11~~	13	17	19	16	24	23	27	29

swap 24 and 23

11	13	17	19	16	23	24	27	29

19 → 16

17 → 16 ⇒

11	13	10	17	19	23	24	27	29

⇒ sorted

Time complexities ⇒ Worst case = $O(n^2)$

Best case = $O(n \log n)$

Average case = $O(n \log n)$

— x —————— x —————— x ——————

— 2 3

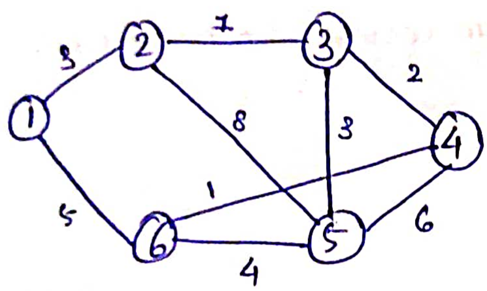
(5) Minimum spanning tree of a weighted, connected and unidirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree.

The sum of weights of each edge gives the weight of the spanning tree.

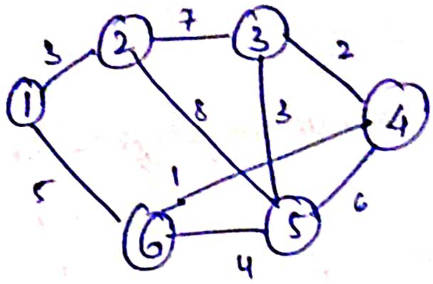
Properties of a spanning tree

* There are many minimum spanning trees possible, but the minimum weight is only one.
* Spanning trees are non-cyclic.
* It has n vertices and (n-1) edges.

(ii)



PRIM ALGORITHM



Step 1 = Start

Step 2 : Start from 1 . compare 3 and 5 , 3 is minimum .

1 → 2

Step 3 : At 2 , compare 3 , 7 , 8 . 3 already visited, ignore .
7 is the minimal sost .

1 → 2 → 3

Step 4 : From 3 , compare 7 , 3 , 2 .
7 already visited , ignore .
2 is the mimal cost .

1 → 2 → 3 → 4

Step 5 : At 4 , compare 2 , 1 , 6 .
2 already visited , ignore .
1 is the minimal cost .

1 → 2 → 3 → 4 → 6

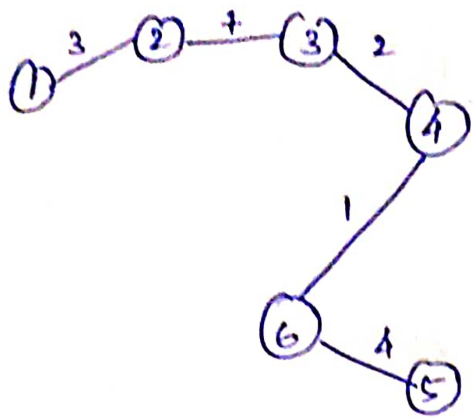
Step 6 : From 6 , compare 1 and 4 .
1 already visited , ignore .
Choosing 4 .

1 → 2 → 3 → 4 → 6 → 5

Step 7 : Stop .

∴ V = 6 , E = 5 .

Minimum spanning tree ⇒



Minimum weight = 3+7+2+1+4
= 17

KRUSKAL'S ALGORITHM

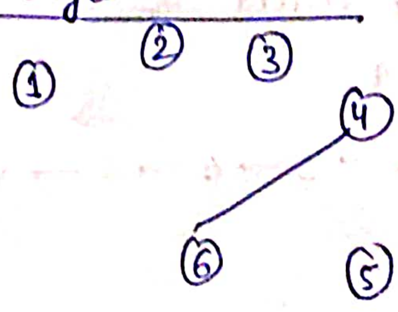
Step 1 : Start

Step 2 : Arrange all the weight and edges in the increasing order of weights -

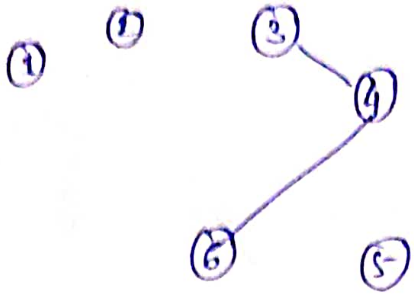
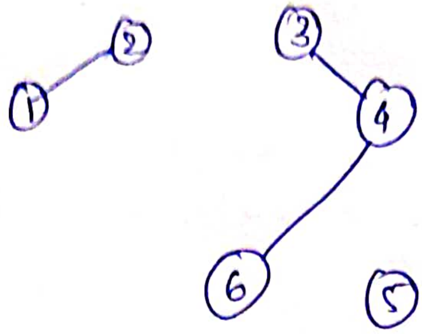
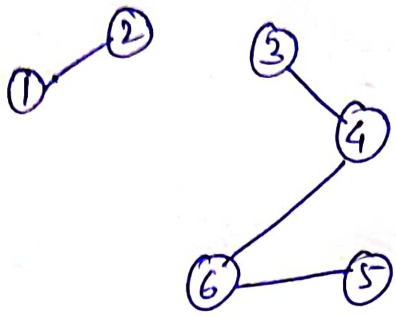
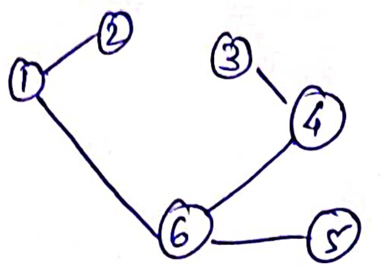
weights	edge
1	6 → 4
2	3 → 4
3	1 → 2
4	6 → 5
5	1 → 6
6	5 → 4
7	2 → 3
8	3 → 5

Step 3 : The algorithm ⇒

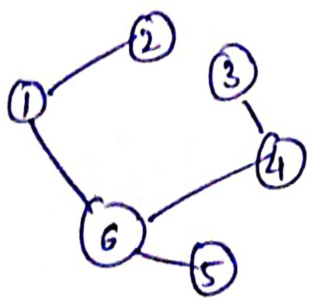
	Edge	weight	stages
i	6 → 4	1	



No angle, so included in the tree

	Edge	Weight	Stages	
(ii)	3 → 4	2		No cycle, include
(iii)	1 → 2	3		No cycle, include the edge
(iv)	6 → 5	4		No cycle, include the edge.
(v)	1 → 6	5		No cycle, include the edge

Now, V = 6 , E = 5 ∴ we the reached the spanning tree.



Minimum weight = 1 + 2 + 3 + 4 + 5 = 15