DSA Assignment / Tutorial

→ Bishal Poudel
→ 191318
→ BECE 3rd Day.

Date
Page

1. What is difference between data type and data structure?

Data types	Data structure
Data type is the form of a variable which is being used throughout the program. It defines that the particular variable will assign the values of the given data type only.	Data structure is the collection of different types of data. The entire data can be represented using an object and can be used throughout the entire program.
Implementation through data types is a form of abstract implementation.	Implementation through data structures is called concrete implementation.
It can hold values so it is dataless.	can hold different types of data within single object.
values can directly be assigned to the data type variables.	The data is assigned using some set of algorithms and operations like push, pop and so on.
No problem of time complexity	Time complexity come into play when working with data structures.
Example int, float, double	Example stacks, queue, list.

Date
Page

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

⇒ Abstract data type is a mathematical model of data structure that specifies the type of data stored, the operations supported by them and types of parameters of the operations.

ADT is the concept rather than real implementation in the fild of programming or development.
In software industry we need to store, retrieve, and operate on the data, based on these operations we define mathematical concept or logic and these logics are implemented using some programming language. where it is called data structures.

Before the implementation we need to define types of data, operations, efficiency of program or algorithm, which playes great role in software development.
Based on different types of software we need to build different algorithm or need to devlop different mechanism or model for that software where ADT is mainly used.
Hence in overall development of software, ADT is an important part.

Date
Page

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

=) stack is a data structure, which follow the rule of last in first out or first in last out.
Let define a stack of size max and TOS = -1
now we need to perform push and pop operation on stack [max].

(i) push()

1. If TOS = max - 1, then
   display "stak is full" and stop.
2. Read data
3. TOS ← TOS + 1
4. stack [TOS] ← data
5. stop.

(ii) pop()

1. If TOS = -1, then
   display "stack is empty" and stop.
2. item ← stack [TOS]
3. TOS ← TOS - 1
4. return item.

Date
Page

8. Convert the following infix expression to postfix using stack.

A+(B*C-(D/E^F)*G)*H

step	Scanned character	stack content	postfix expression
1.	A	—	A
2.	+	+	A
3.	(	+(	A
4.	B	+(	AB
5.	*	+(*	AB
6.	C	+(*	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*+

Postfix expression = ABC*DEF^/G*-H*+//

Date
Page

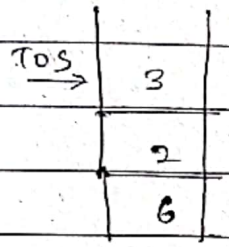
5. Evaluate the following ~~infix~~ postfix expression using stack.

6 2 3 + - 3 8 2 / + * 2 $ 3 +

⇒ Scan the characters from left to right.

1. ch = 6, 2, 3

push all in to stack.

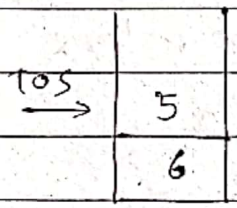


2. ch = +, pop two operands from stack.

op1 = 3, op2 = 2

r = op1 + op2 = 3 + 2 = 5

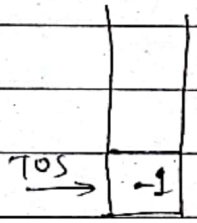
push r.



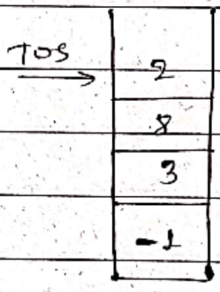
3. ch = -

op1 = 5, op2 = 6

r = 5 - 6 = -1



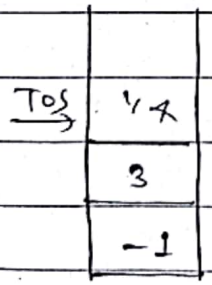
4. ch = 3, ch = 8, ch = 2



5. ch = /

op1 = 2, op2 = 8

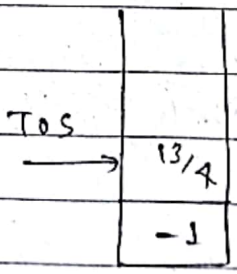
r = 2/8 = 1/4



6. ch = +

op1 = 1/4, op2 = 3

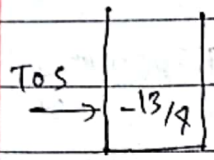
r = 1/4 + 3 = 13/4



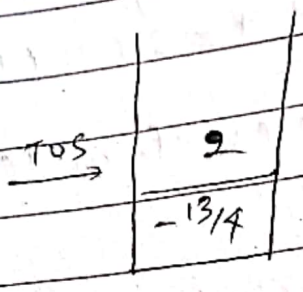
7. ch = *

op1 = 13/4 , op2 = -1

r = 13/4 * -1 = -13/4



8. ch = 2

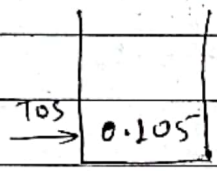


9. ch = $

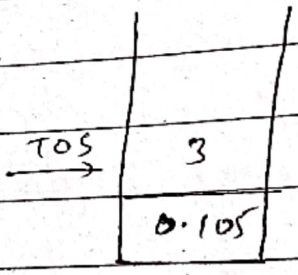
op1 = 2, op2 = -13/x

r = 2 $ -13/x

= 0.105



10. ch = 3

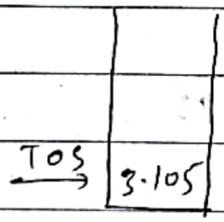


11. ch = +

op1 = 3 op2 = 0.105

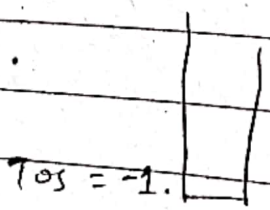
r = 3 + 0.105

= 3.105



12. ch = None.

Result = 3.105

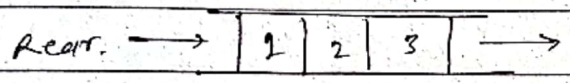


∴ Result = 3.105

Date
Page

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

⇒ Queue is the linear list which have two ends rear and front one for insertion and another for deletion. It follows first in first out or last in last out mechanism.



To add data we perform Enqueue() and To remove Dequeue().

(1) ~~Enque~~ Define a queue.

```
#define max 20
struct queue {
    int front;
    int rear;
    int items[max];
};
struct queue q;
```

(1) Enqueue()

2. If rear = max - 1 then,
   display "queue is full" and stop
3. Read data
4. If front = rear = -1
   front ← front + 1
   rear ← rear + 1
   q[rear] ← data
5. else
   rear ← rear + 1
   q[rear] ← data
6. stop.

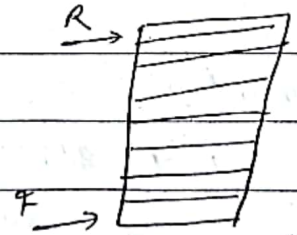
(ii) Dequeue()

1. If rear = front = -1 then,
   display "queue is empty" and stop.
2. If rear = front then,
   item ← q[front]
   rear ← front ← -1
   return item
3. else,
   item ← q[front]
   front ← front + 1
   return item
4. stop.

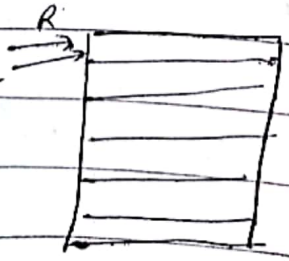
7. What are the limitation of linear queue. How this can be corrected using circular queue.

⇒ Main limitation of linear queue is, efficient space utilization. It shows queue is full when rear = front but in some case there will not be actually full queue.

Consider a queue.



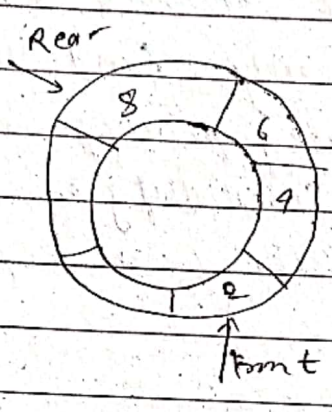
After deleting all data



Date
Page

Then when we perform enqueue() operation, we cannot perform it because it is full in case of linear queue. But we can solve this problem by using circular queue.

In circular queue, queue will be full when actually it is full.

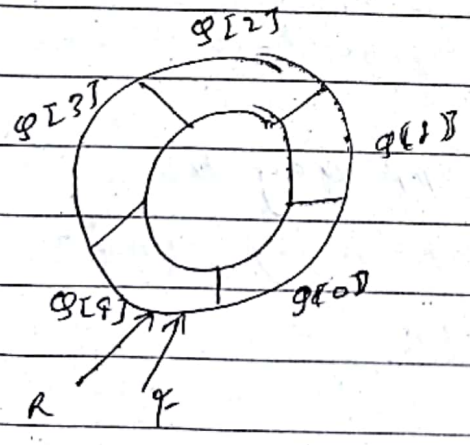


In circular queue, we change the index calculation algorithm of rear and front ends as

rear = (rear+1) % max-size

front = (front+1) % max-size

Consider same case where front and rear at last position and we need to insert some data.



Then, R = (R+1) % max-size

= (4+1) % 5

= 0

new value stored at q[0] position.

Hence in this way we resolve the limitation of linear queue using circular queue.

8. What is priority queue. explain its types.

⇒ priority queue is a type of queue in which elements can be inserted or deleted based on its priority.
In priority queue each element has been assigned a value called priority of element and element can inserted and deleted not only front or rear end but from any position using priority.
priority can be specified implicitly or explicitly such that elements are deleted and processed using following rules.

(a) Element of higher priority is processed before any element of lower priority.
(b) Element with the same priority are processed to the order in which they were inserted to the queue.

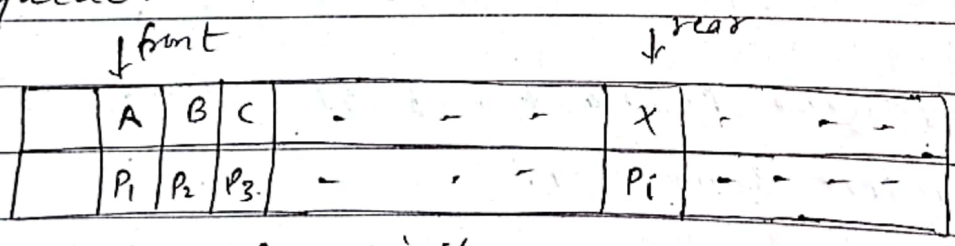


fig. priority queue

Here an element x having priority Pi may be deleted before an element at front having priority P1.

There are two types of priority queue:

1. Ascending priority queue
In this queue irrespective of arrival in the queue, element will be removed having smallest priority number associated with it rather then from front.

2. Descending priority queue.
In the descending queue, element associated with highest priority number will be removed first rather than the element at font.

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

⇒ Linked list is the linear data structure and is most important data structure. Unlike array they are not stored in contiguous location. In linked list nodes are linked using pointer. Each node contains:
i) Data → Data is stored at a particular address
ii) Reference → contains address of next node of the linked list.

Due to dynamic implementation it has following advantages over arry:

① Dynamic data structure
They can shrink and grow at the runtime by deallocating or allocating memory. They do not need to specify size in very first like array.

(2) No memory wastage
- As the size of a linked list can grow or shrink at runtime, there is no chance of memory wastage. only required memory is allocated. But in array we declare memory at very first so there may error of memory underflow or overflow.

(3) Implementation.
other data structures such as queue and stack can be implement dynamically and easily using linked list than array.

(4) Insertion and deletion option.
Insertion and deletion is linked list is easy. not need to shift data after insertion or deletion. we can insert and delete at any position. But in case of array we need to perform shift operation when we delete and insert data.

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

⇒ To implement operation in doubly circular linked list first we need to define a structure as

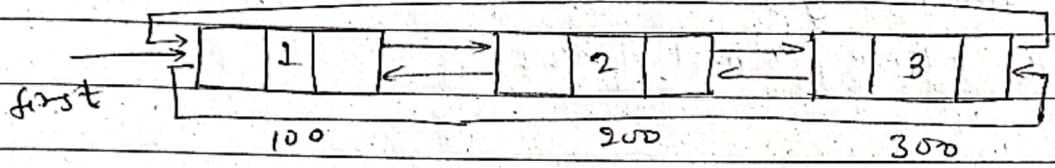
```
struct doubly_circular_list {
        int info;
        struct doubly_circular_list *prev, *next;
};
```

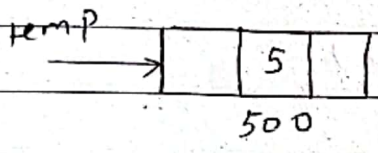
Date
Page

typedef struct doubly_circular_list DCLL;
DCLL *p, *first = NULL, *temp;

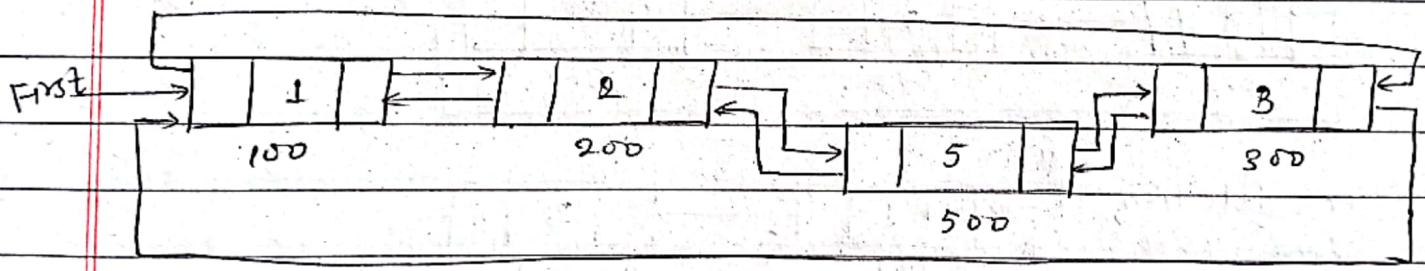
Now create a node for insertion.
temp = (DCLL*) malloc (size of (DCLL));
temp → info = data;

Consider we have following linked list before insertion.





After insertion at n = 3rd position.



1. p = first
2. while (n > 2)
   2.1 p = p → next
   2.2 n--
3. End while
4. temp → next = p → next
5. p → next → prev = temp
6. temp → prev = p
7. p → next = temp.

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

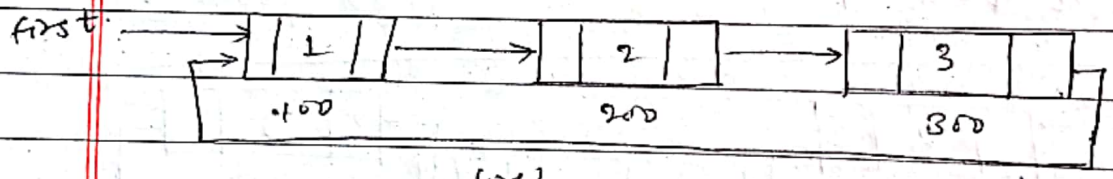
⇒ Lets define a node,

```
struct singly_circular_list {
        int info;
        struct singly_circular_list * next;
};
typedef struct singly_circular_list SCLL;
SCLL *p, *temp, *first = NULL;
```

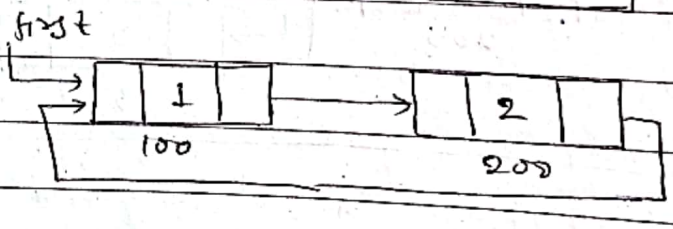
Create a node

```
temp = (SCLL *) malloc (sizeof(SCLL));
```

consider following linked list



After deletion,



1. p = first
2. while (p → next → next != NULL)
   2.1 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.

=) Linked list can be used to represent polynomial equation.

| coefficient | exponent | next |→

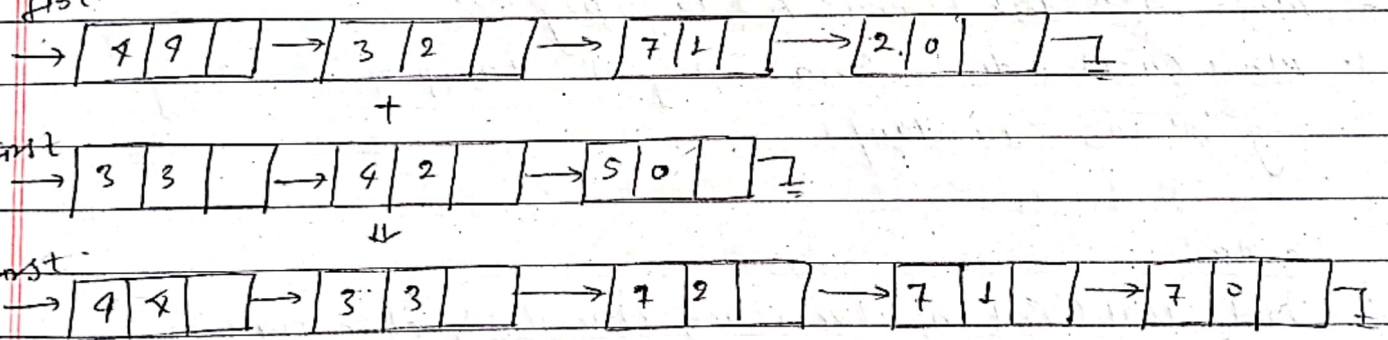
making node for polynomial

```
struct polynomial {
        int expi;
        int coeff
        struct polynomial *nexti;
}
```

Consider two polynomials, $A = 4x^4 + 3x^2 + 7x + 2$

$B = 3x^3 + 4x^2 + 5$

Then $C = 4x^4 + 3x^3 + 7x^2 + 7x + 7$

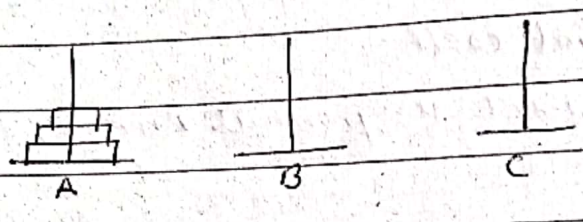


1. we have three linked list, l1, l2, l3
2. l1 = p1, l2 = p2,
3. while (p1 → next != NULL or p2 → next != NULL)
4. if p1 → exponent > p2 → expmont
5. l3 → push(p1)
6. else if p1 → expmont < p2 → expmont
7. l3 → push (p2)
8. else l3 → push (p1 + p2)
9. End while.

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

⇒ Tower of Hanoi is a game. There have 3 tower and we need to move all the disk from source tower to destination tower using next as auxilary. We can move only one disk at a time where bigger disk can't reside on top of smaller.

Algorithm:



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

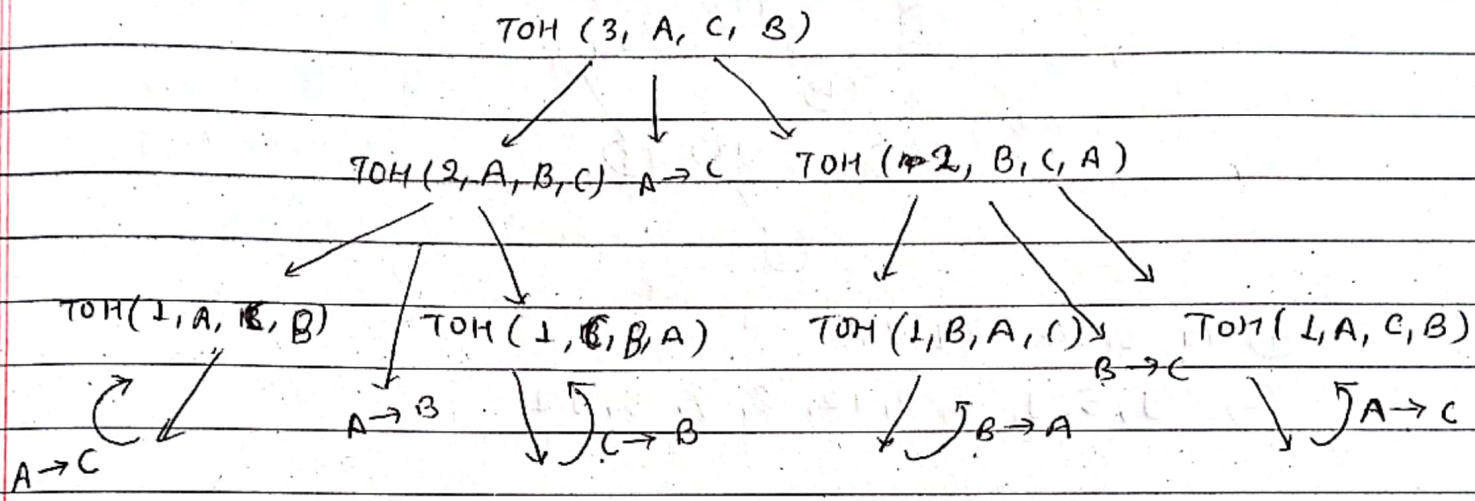
code:

```
void TOH (int n, char source, char dest, char aux) {
    if (n == 1) {
        cout << "Move disk " << n << " from " << source << " to " <<
            dest << endl;
        return;
    }
    TOH (n-1, source, aux, dest)
        cout << "Move disk " << n << " from " << source <<
            " to " << dest << endl;
    TOH ( n-1, aux, dest, source)
}
void main () {
    TOH (3, 'A', 'C', 'B');
}
```

14. Draw recursion tree for TOH when number of disk $n = 3$.

let, source = A, dest = C Auxiliary = B. and $n = 3$

Then,



Hence the execution is =>

A → C

A → B

C → B

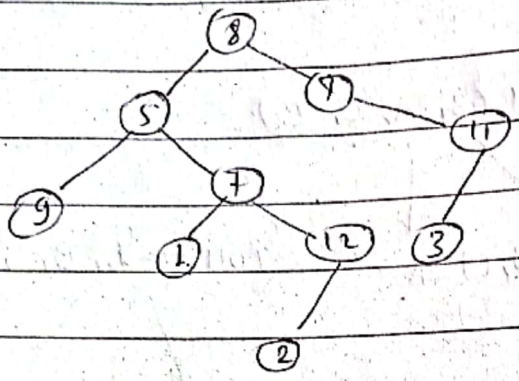
A → C

B → A

B → C

A → C

15. perform in-order, pre order and post order tree traversal.



(1) In order (LVR)

⇒ 9, 5, 1, 7, 2, 12, 8, 4, 3, 11

(2) pre order (VLR)

⇒ 8, 5, 9, 7, 1, 12, 2, 4, 11, 3

(3) post order (LRV)

⇒ 9, 1, 2, 12, 7, 5, 3, 11, 4, 8.

16. Construct the tree from given values.

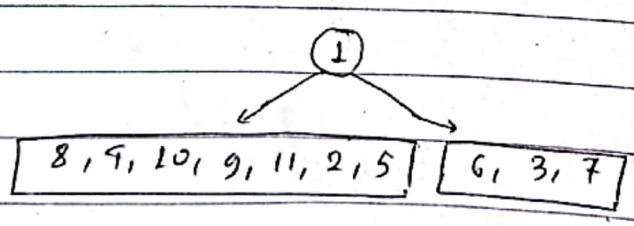
preorder Traversal: 1, 2, 4, 8, 9, 10, 11, 5, 3, 6, 7

(VLR) ↓

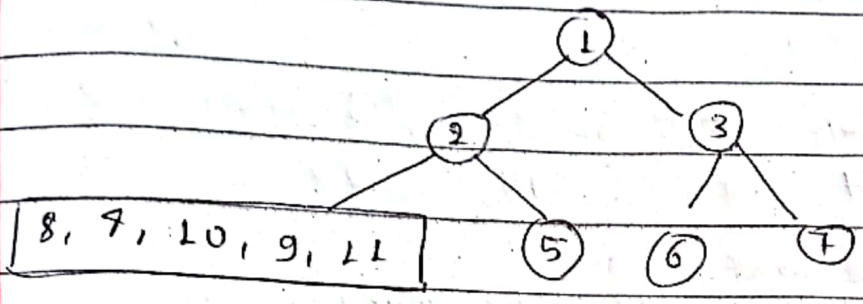
Inorder Traversal: 8, 4, 10, 9, 11, 2, 5, 1, 6, 3, 7

↳ (LVR)

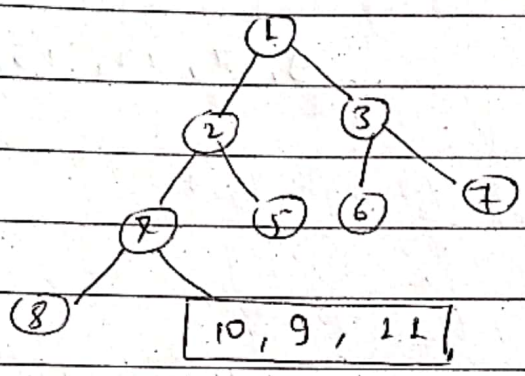
(1)



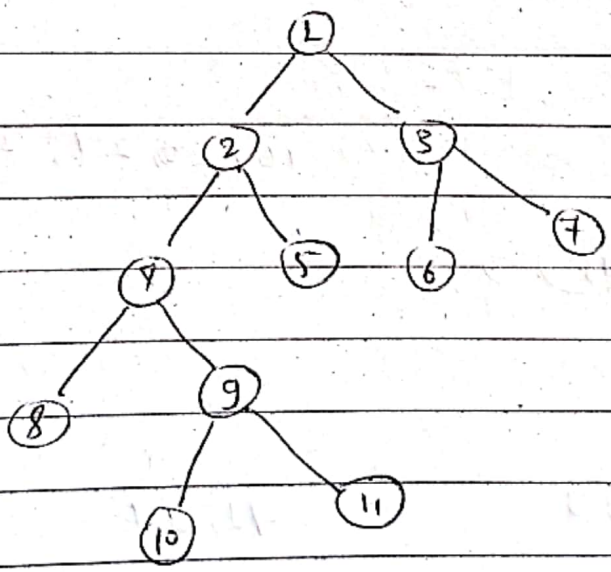
(2) root node = 2, 3.



3. root node = ~~4~~



4. root node = 9



A)

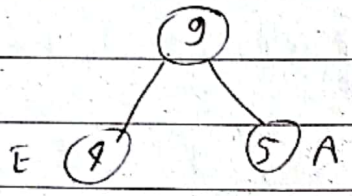
Date
Page

17. Construct Huffman Tree for following values.

value	E	A	C	F	D	B
frequency	4	5	7	12	15	25

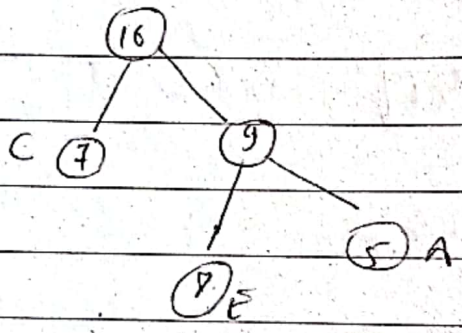
Taking two nodes with less weight & frequency.

(1)



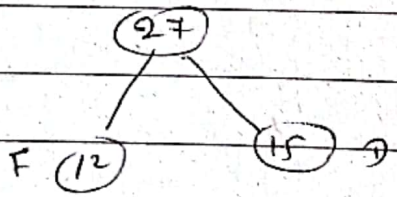
9, 7, 12, 15, 25

(2)



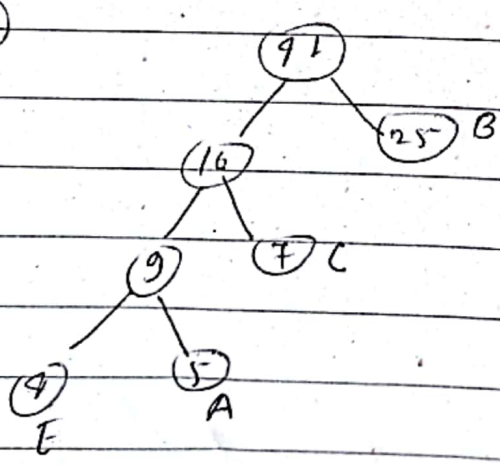
16, 12, 15, 25

(3)

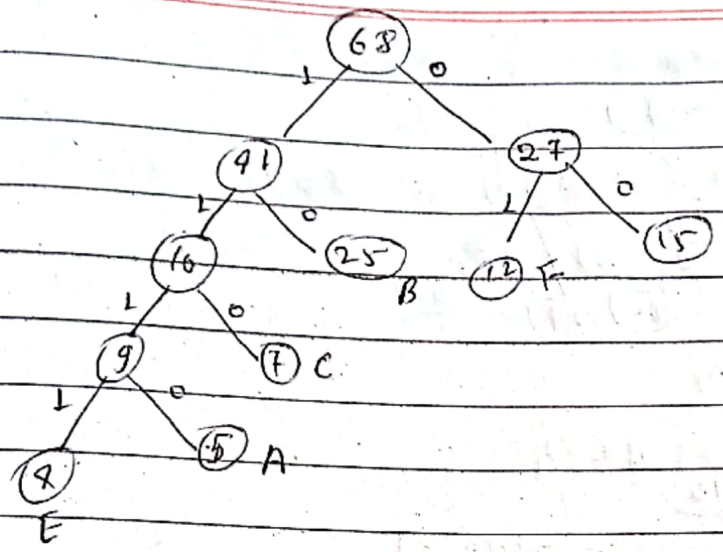


16, 27, 25

(4)



41, 27



E → 1 1 1 1

A → 0 1 1 1

C → 0 1 1

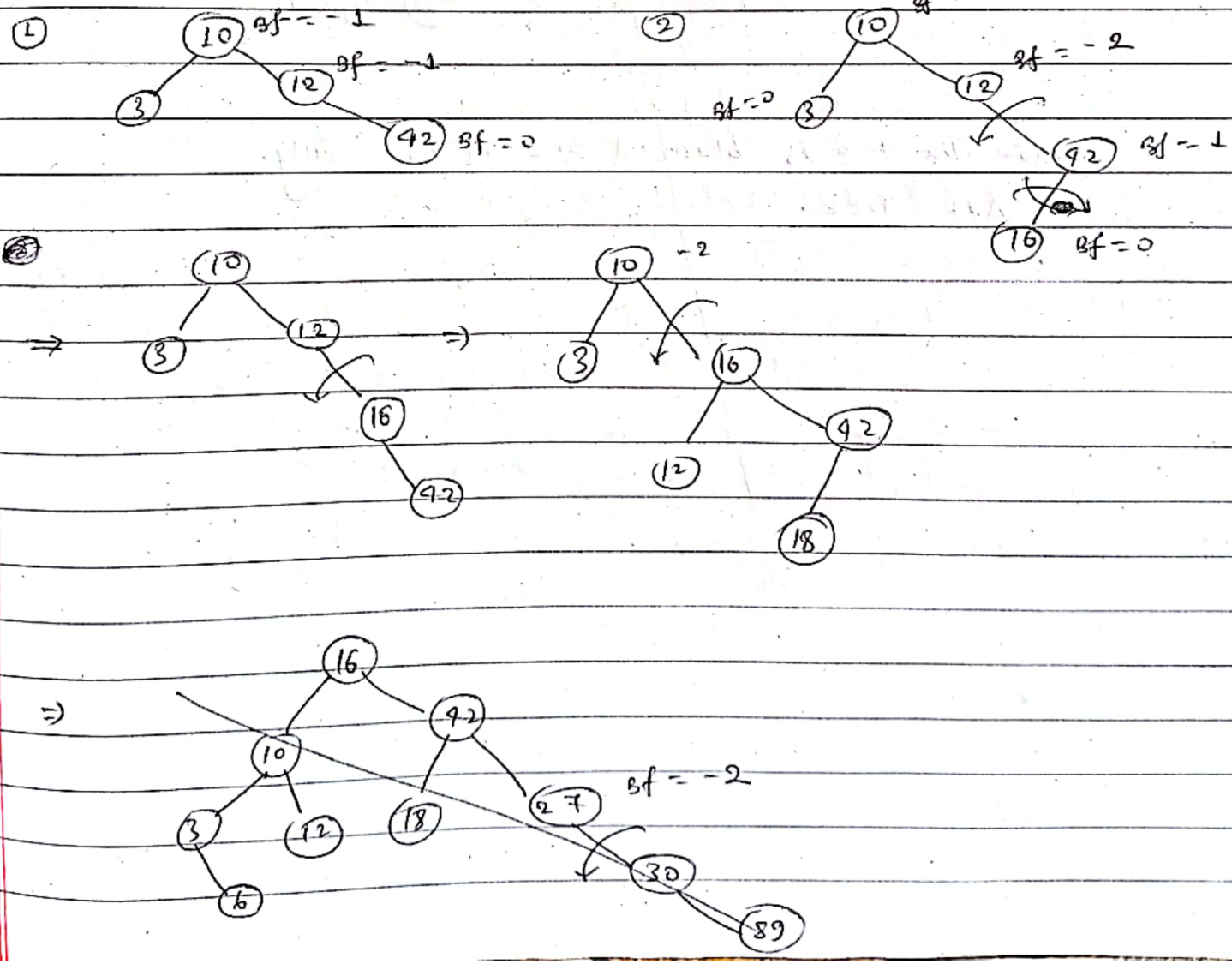
B → 0 1

F → 1 0

D → 0 0

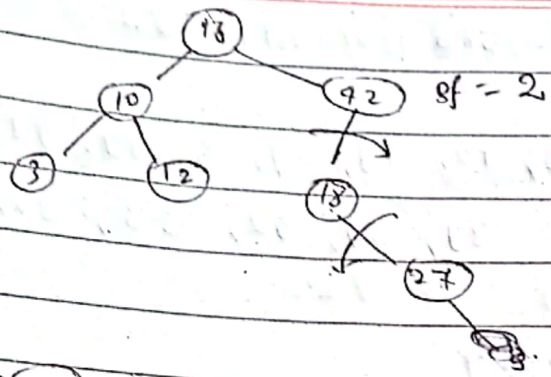
18. Construct an AVL tree for following values.

10, 12, 3, 92, 16, 18, 27, 30, 6, 89, 22, 7, 33

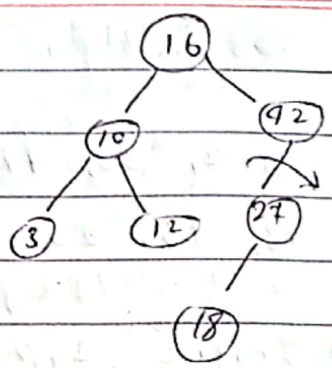


Date
Page

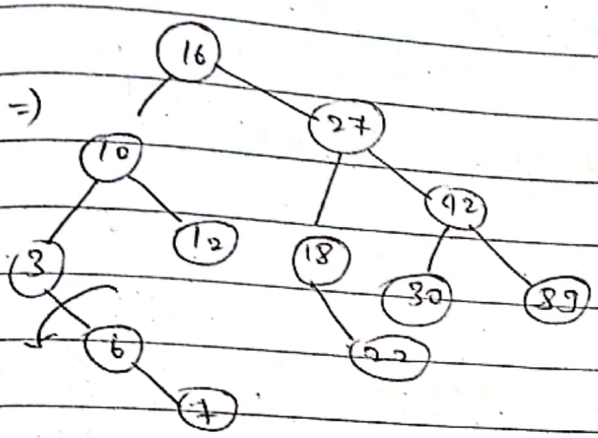
=>



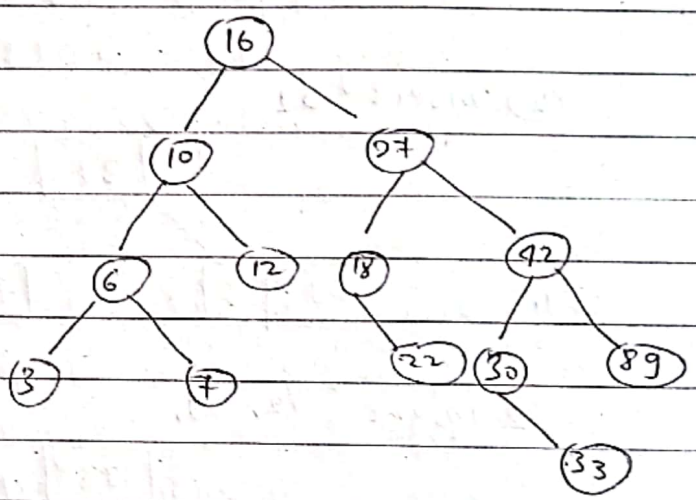
=>



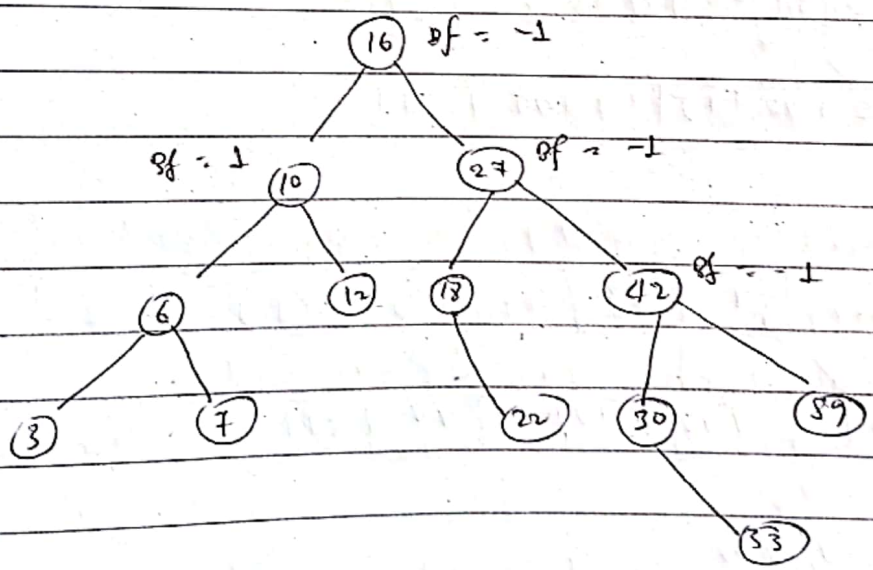
=>



=>



=>

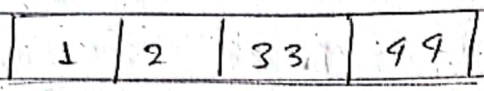


This is balanced and required AVL tree.

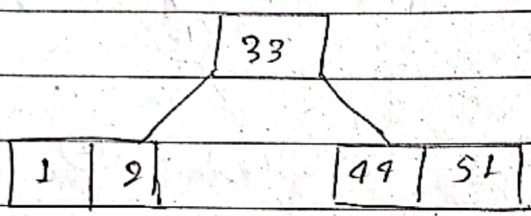
19. Construct a B-tree of order 5.

1, 2, 33, 44, 51, 12, 19, 7, 6, 89, 91, 45, 62, 11, 13, 10, 20, 29, 39, 59, 62, 83, 35, 7

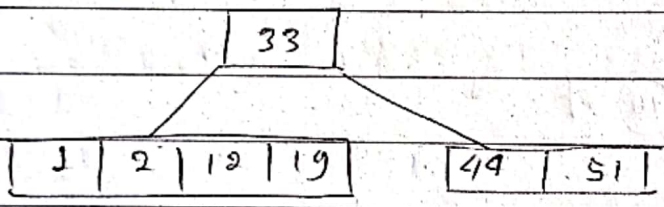
(1) Insert 1, 2, 33, 44



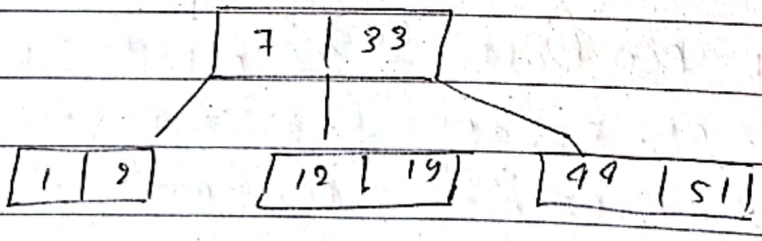
(2) Insert 51



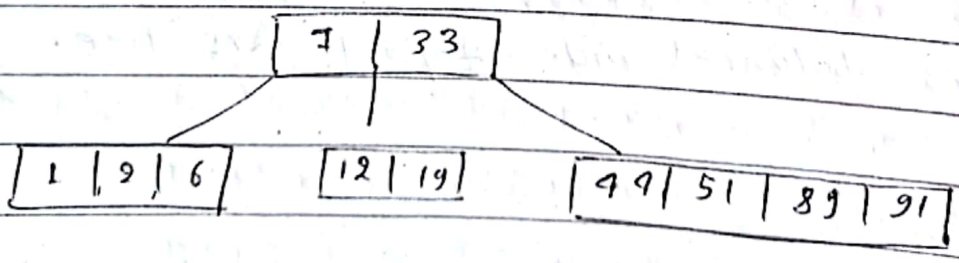
(3) Insert 12, 19,



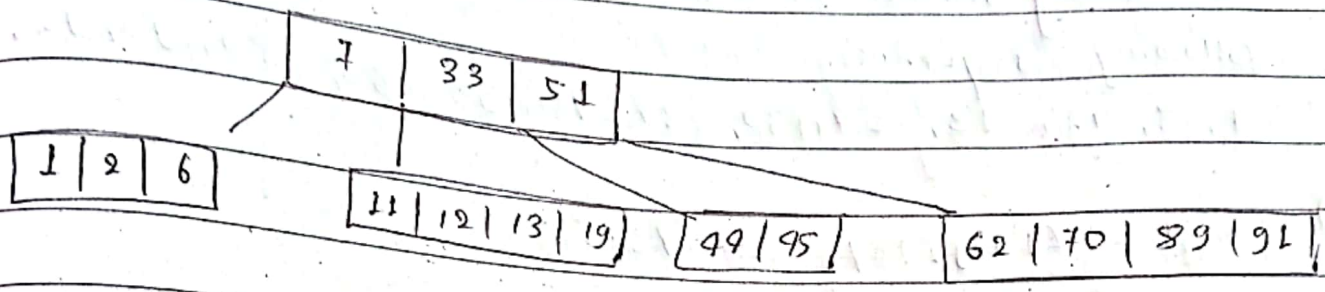
(4) Insert 7,



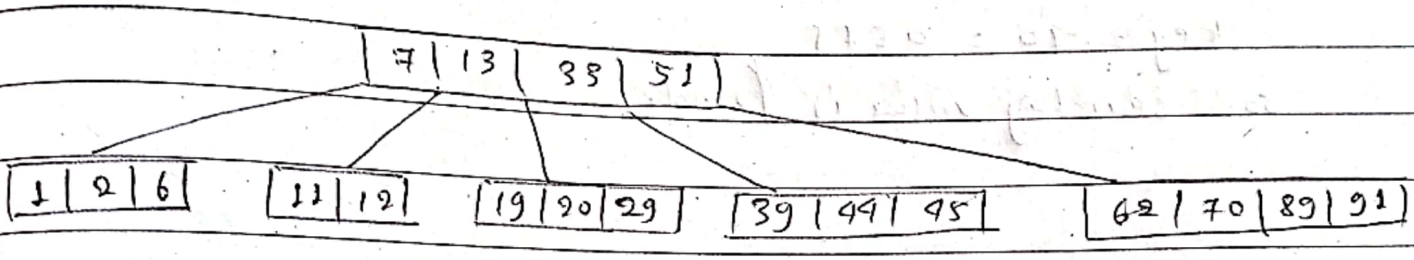
(5) Insert 6, 89, 91,



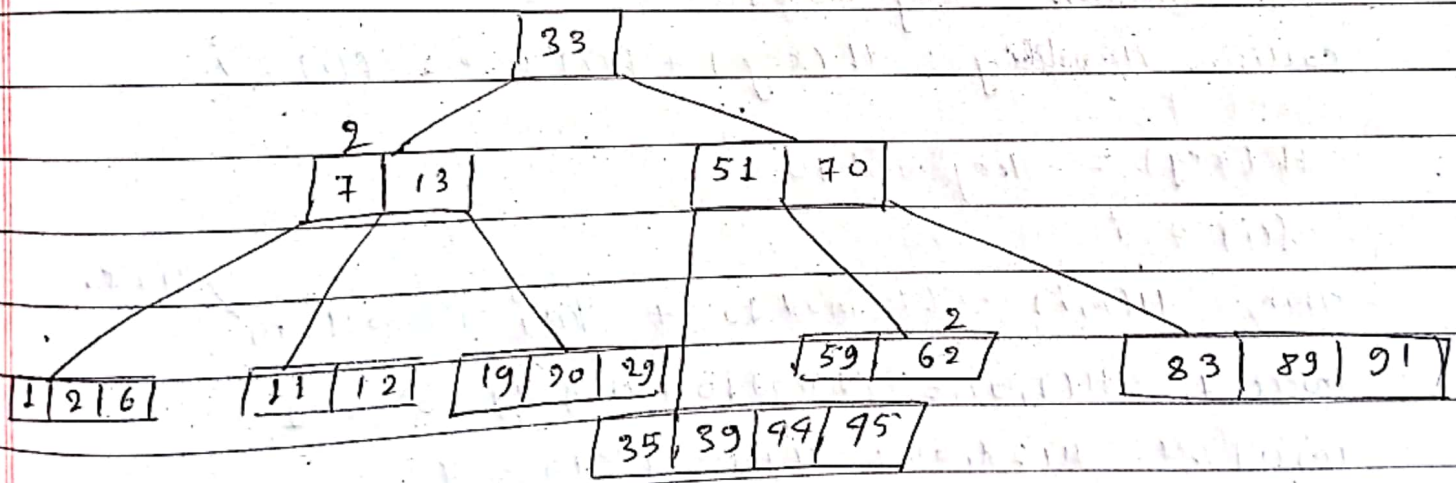
(6) insert 45, 62, 11, 13, 70,



(7) insert. 20, 29, 39,



(8) insert 59., 62, 83, 35, 7



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

1, 8, 12, 13, 29, 30, 37, 40, 50, 60, 80, 90, 92.

=)

~~mid = (1+92)/2 => 46.5~~

mid = 1 + 13 = 7

now a[mid] = a[7] = 40

key = 40 = a[7]

so searchy value 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

H(key) = key mod 10

f(i) = i

Then, H(k,i) = (k mod 10 + i) mod m (size).

insert 1, H(1,0) = (1 mod 10 + 0) % 10 = 1

insert 27, H(27,0) = 7+0 = 7 % 10 = 7

insert 6, H(6,0) = 6 % 10 + 0 = 6 % 10 = 6

insert 87, H(87,0) = 87 % 10 + 0 = 7 % 10 Hash collision

H(87,1) = 87 % 10 + 1 = 8 % 10 = 8

insert 47, H(47,0) = 47 % 10 + 0 = 7 % 10 hash collision

H(47,2) = 47 % 10 + 1 = 8 % 10 hash collision

H(47,3) = 47 % 10 + 2 = 9 % 10 = 9

Date
Page

insert 7, H(7,0) = 7%10+0 = 7%10 hash collision

H(7,1) = 7%10+1 = 8%10 hash collision

H(7,2) = 7%10+2 = 9%10 hash collision

H(7,3) = 7%10+3 = 10%10 = 0

insert 8, H(8,0) = (8%10+0)%10 = 8 hash collision

H(8,1) = (8%10+1)%10 = 9

H(8,2) = (8%10+2)%10 = 0

H(8,4) = (8%10+4)%10 = 2

insert 17, H(17,0) = (17%10+0)%10 = 7 hash collision

H(17,6) = (17%10+6)%10 = 3

insert 37, H(37,0) = (37%10+0)%10 = 7 hash collision

H(37,7) = (37%10+7)%10 = 4

insert 67, H(67,0) = (67%10+6)%10 = 7 hash collision

H(67,8) = (67%10+8)%10 = 5

Now hash table is,

0	1	2	3	4	5	6	7	8	9
7	18	28	17	37	67	6	27	87	47

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

=) H(key, i) = (key mod 10 + i*i) mod 10

insert 1,

H(1, 0) = (1 % 10 + 0) % 10 = 1

insert 27,

H(27, 0) = (27 % 10 + 0) % 10 = 7

insert 6,

H(6, 0) = (6 % 10 + 0) % 10 = 6

Insert 87

H(87, 0) = (87 % 10 + 0) % 10 = 7 hash collision

H(87, 1) = (87 % 10 + 1) % 10 = 8

insert 47,

H(47, 0) = 7 hash collision

H(47, 1) = 8 hash collision

H(47, 2) = (87 % 10 + 4) % 10 = 1 hash collision

H(47, 3) = (87 % 10 + 9) % 10 = 6 hash collision

H(47, 4) = (87 % 10 + 16) % 10 = 3

insert 7, H(7, 0) = 7

H(7, 1) = 8

H(7, 2) = 1

H(7, 3) = 6

H(7, 4) = 3

} hash collision

H(7, 5) = (7 % 10 + 25) % 10 = 2

DSA Tutorial

- Bishal Pandit
- 191318

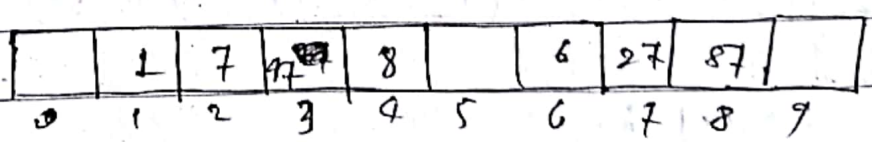
Date
Page

insert 8, H(8,0) = (8%10+0)%10 = 8
H(8,1) = (8%10+1)%10 = 9
H(8,2) = (8%10+9)%10 = 2
H(8,3) = (8%10+9)%10 = 7
H(8,4) = (8%10+16)%10 = 8

} Hash collision.

insert 17, H(17,0) = 7, H(17,1) = 8, H(17,2) = 1
H(17,3) = 6, H(17,4) = 3, H(17,5) = 2
H(17,6) = (17%10+36)%10 = 3
H(17,7) = (17%10+49)%10 = 6
H(17,8) = (17%10+64)%10 = 1
H(17,9) = (17%10+81)%10 = 8
H(17,10) = (17%10+100)%10 = 7
H(17,11) = (17%10+121)%10 = 8
H(17,12) = (17%10+144)%10 = 1
H(17,13) = (17%10+169)%10 = 6
H(17,19) = 3
H(17,15) = 2
H(17,16) = 3
H(17,17) = 6
H(17,18) = 1
H(17,19) = 8
H(17,20) = 7
H(17,21) = 8
H(17,22) = 1
H(17,23) = 6
H(17,24) = 3
H(17,25) = 2

All hash collision.



Input sequence: 1, 27, 6, 87, 97, 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)

$H(key, i) = ((key \bmod 10) + i * (7 - key \bmod 7)) \% 10$

Insert 1,

$H(1,0) = (1 \% 10) + 0 * (7 - 1 \% 7) = 1 \% 10 = 1$

Insert 27.

$H(27,0) = (27 \% 10) + 0 (7 - 27 \% 7) = 7 \% 10 = 7$

Insert 6,

$H(6,0) = (6 \% 10) + 0 (7 - 6 \% 7) = 6 \% 10 = 6$

Insert 87

$H(87,0) = 7$ hash collision.

$H(87,1) = 87 \% 10 + 1 (7 - 87 \% 7) = 11 \% 10 = 1$

$H(87,2) = [7 + 2 * (7-3)] \% 10 = 5$

Insert 97,

$H(97,0) = 7$, ~~$H(97,1) = 1$, $H(97,2) = 5$~~

$H(97,2) = (7 + 2 * (7-5)) \% 10 = 9$

insert 7.

$H(7,0) = 7$

$H(7,1) = (7 + 1 * (7 - 0)) \% 10 = 4$

Insert 8.

$H(8,0) = (8 \% 10 + 0 * (7 - 8 \% 7)) \% 10 = 8$

Date
Page

insert 17.

$H(17,0) = 7$

$H(17,1) = (7 + 1*(7-3)) \% 10 = 1$ } Hash collision

$H(17,2) = (7 + 2*4) \% 10 = 5$

$H(17,3) = (7 + 3*4) \% 10 = 9$

$H(17,4) = (7 + 9*4) \% 10 = 3$

insert 37.

$H(37,0) = 7$

$H(37,1) = (7 + 1*(7-2)) \% 10 = 2$

insert 67.

$H(67,0) = 7$

$H(67,1) = (7 + 1*(7-7)) \% 10 = 0$

Now the hash table is

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

Date
Page

22. Sort the following sequence using quick sort and heap sort.

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

=) quick sort:

until down > pivot increase down

until up <= pivot decrease up.

if down < up (index)

swap (down ↔ up) values

else swap (up ↔ pivot) values.

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

pivot = 24

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

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

pivot = 13, pivot = 45

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

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

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

Date
Page

pivot = 12, 17, 44, 61.

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

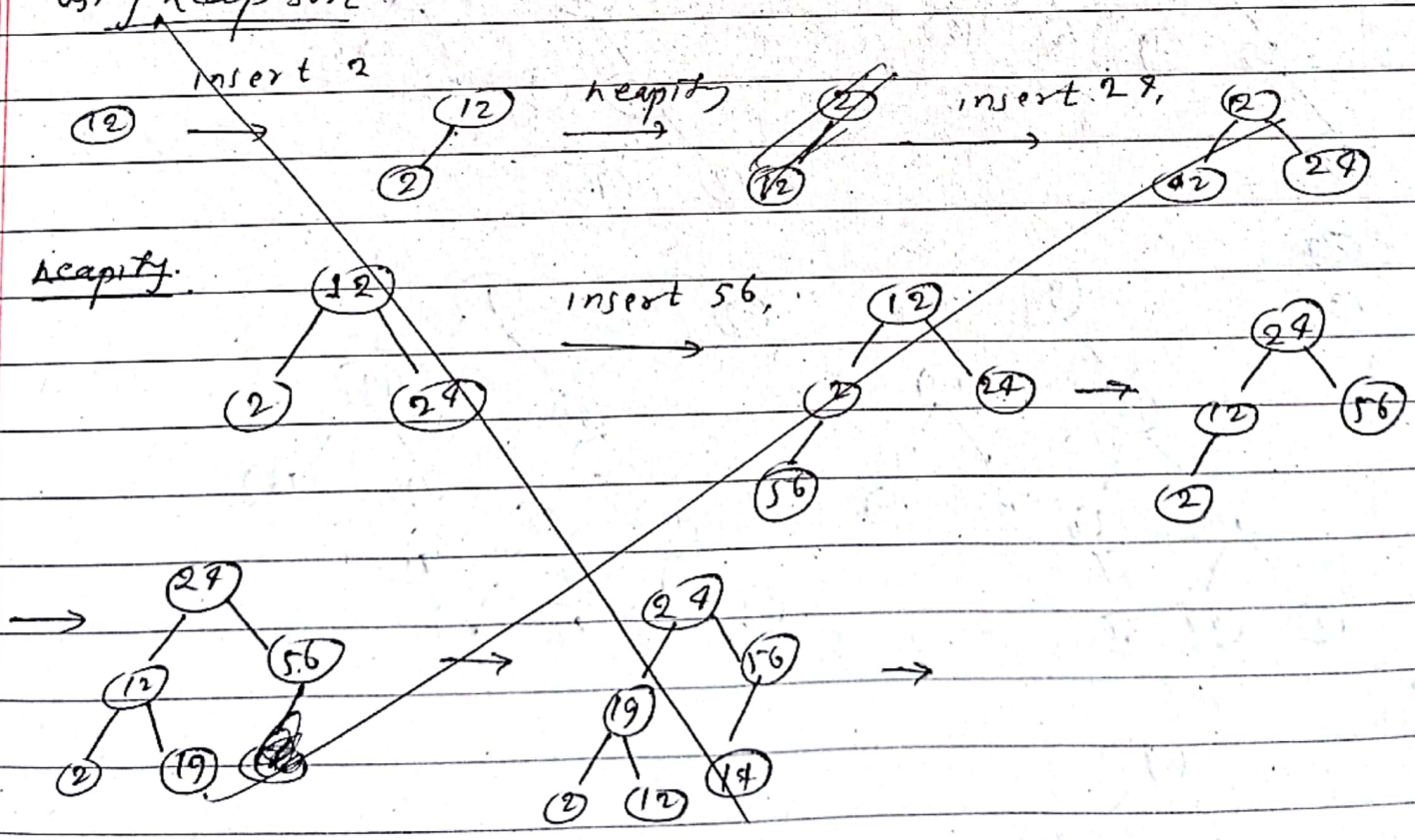
2, 12, 13, 17, 19, 24, 27, 44, 45, 61, 56, 90

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

∴ sorted sequence,

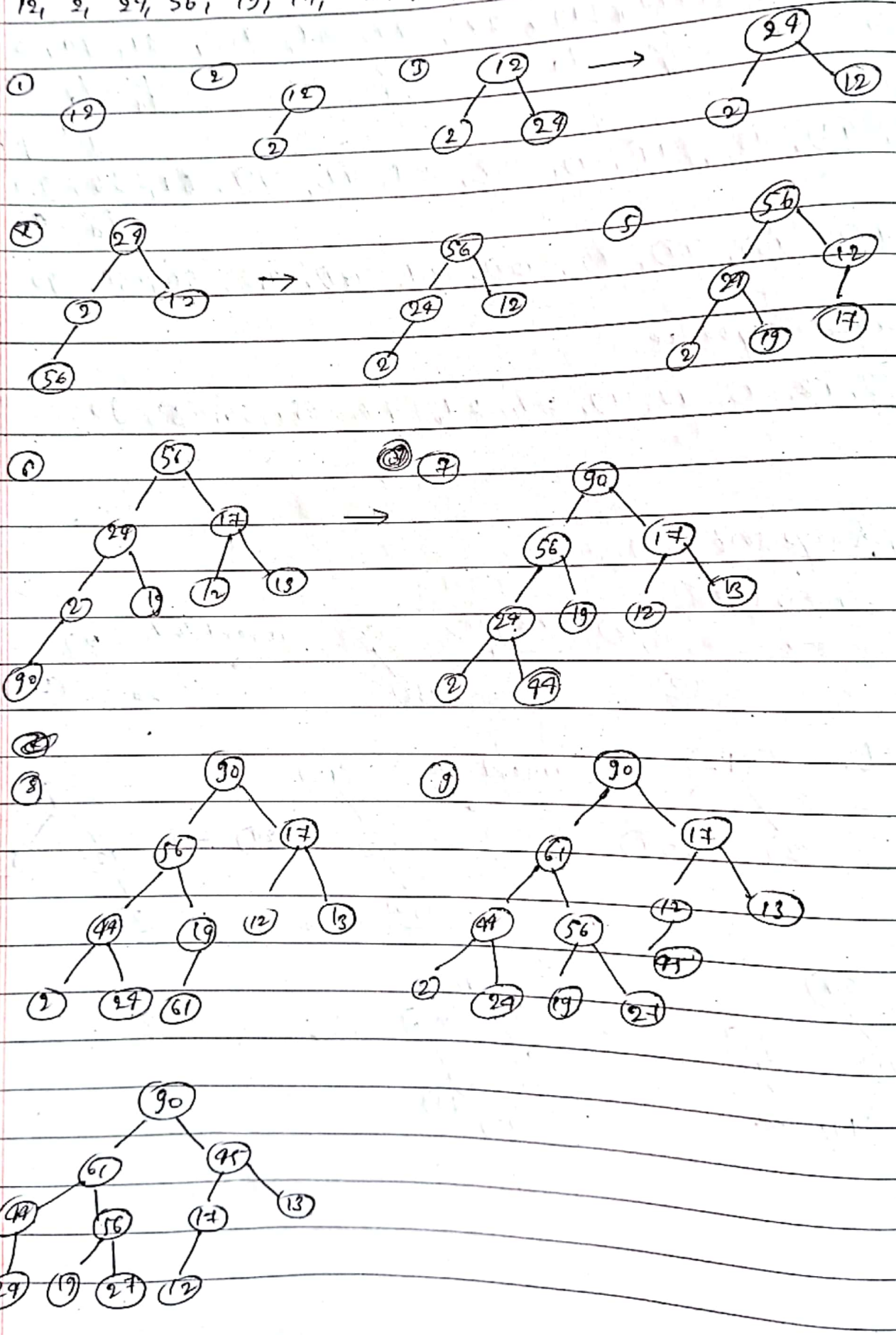
2, 12, 13, 17, 19, 24, 27, 44, 45, 56, 61, 90.

using heap sort:



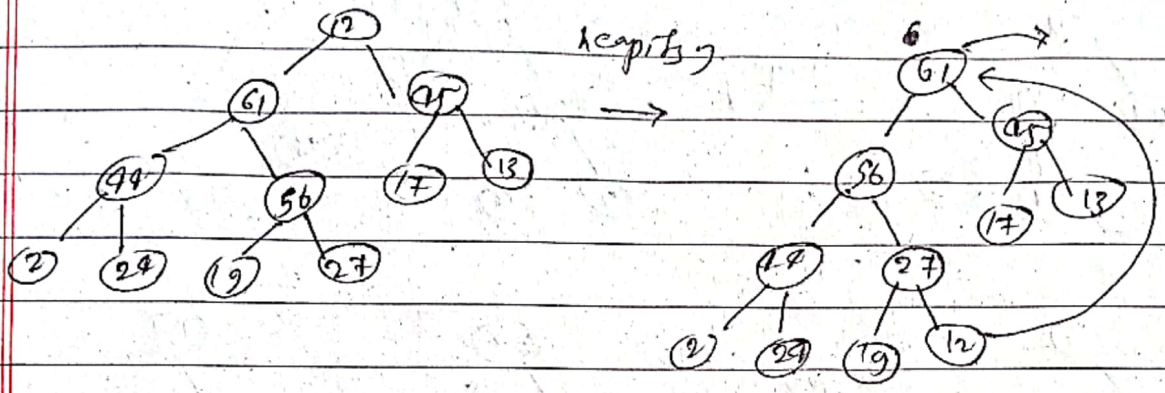
Using heap sort:

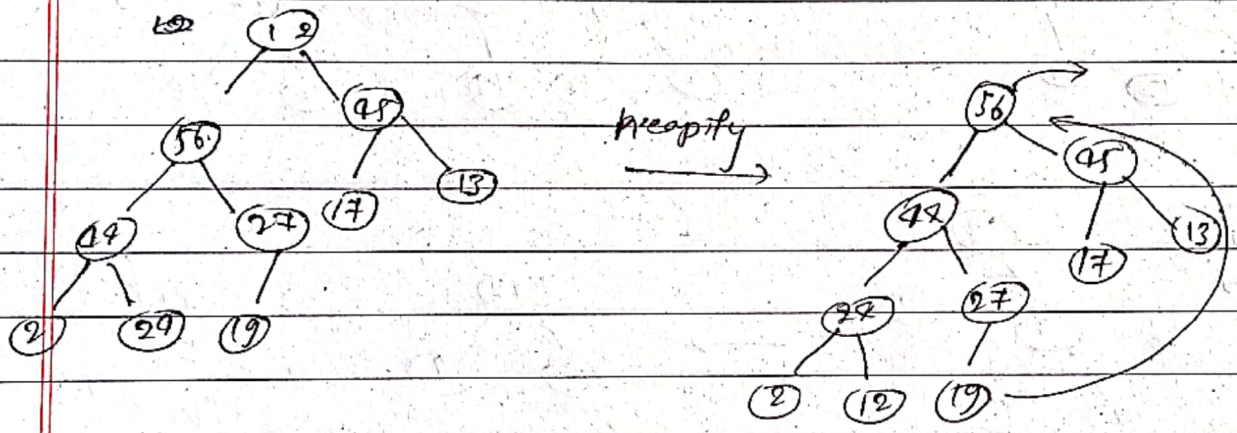
12, 2, 24, 56, 19, 17, 13, 90, 44, 61, 27, 45.

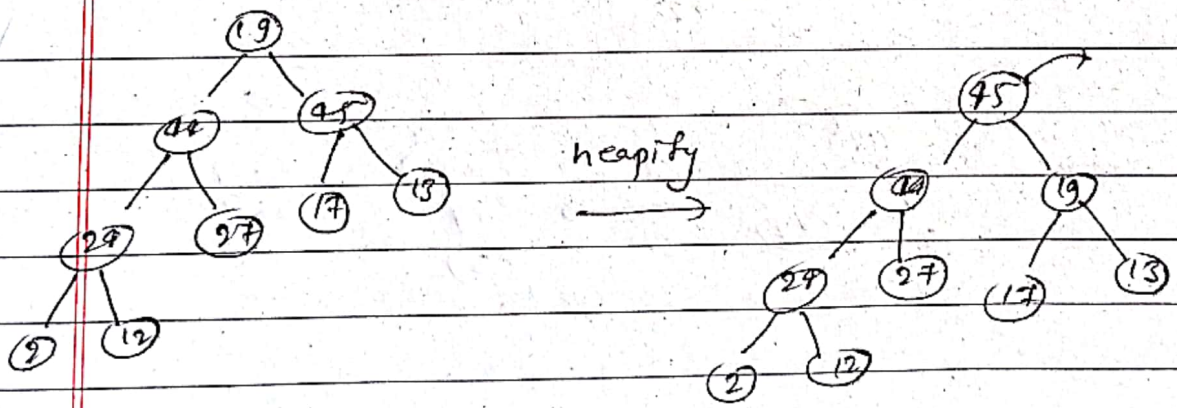


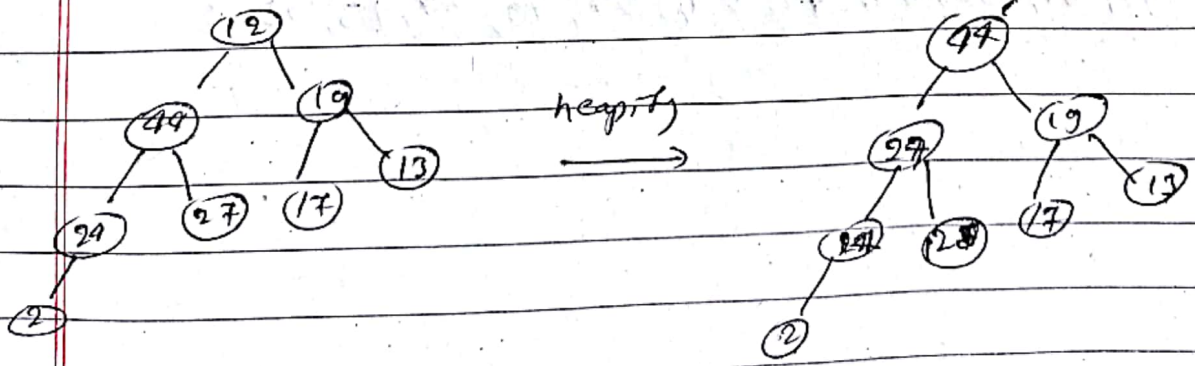
Now heap is max heap, start sorting..

op list = 90, 61, 56, 45, 44

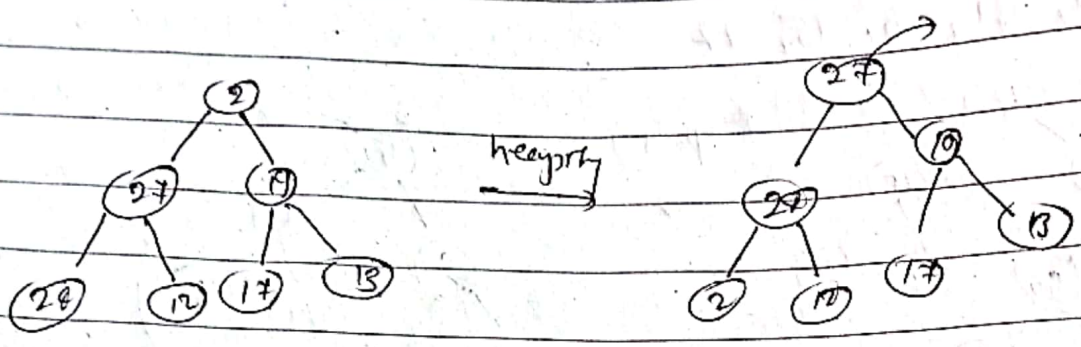


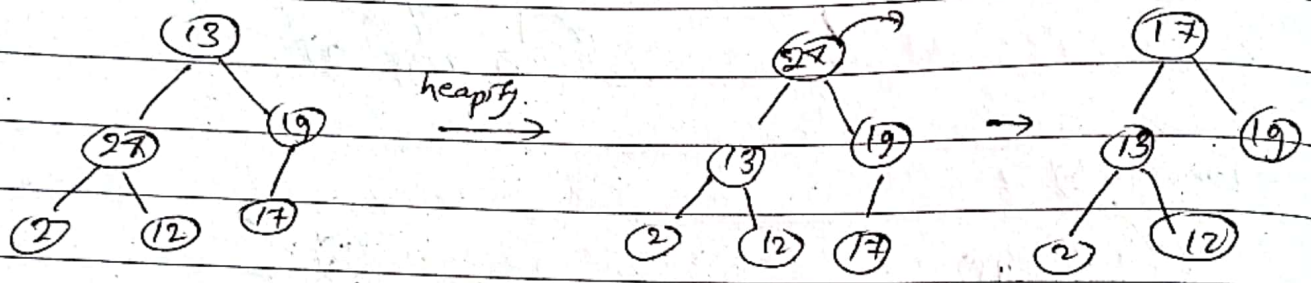


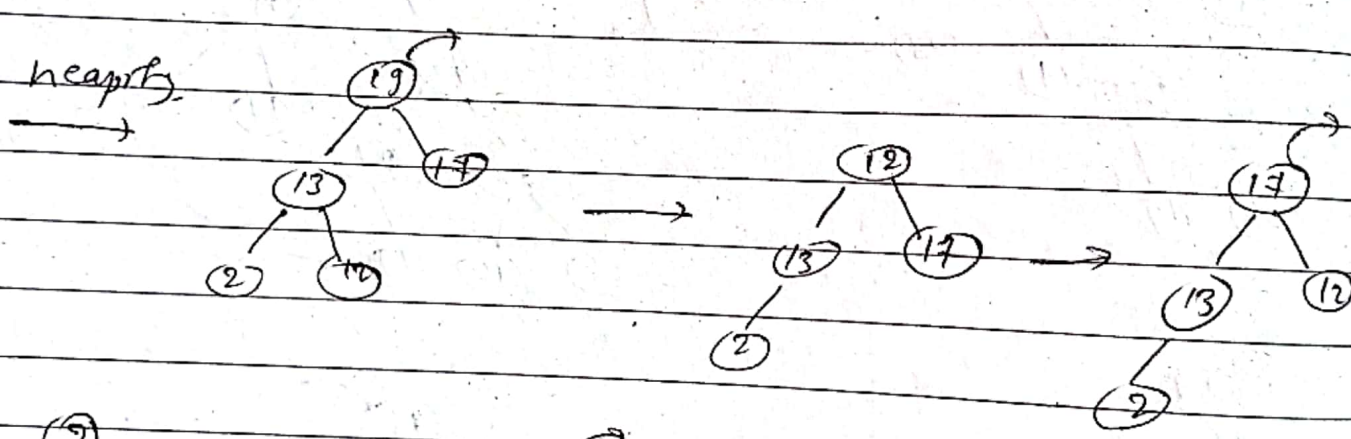


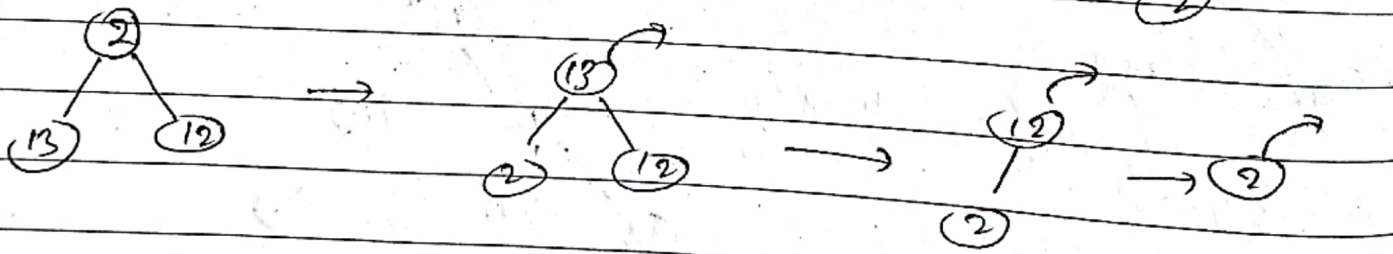


o/p list = 90, 61, 56, 45, 44, 27, 24, 19, 17, 13, 12, 2









sorted list

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

Date
Page

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

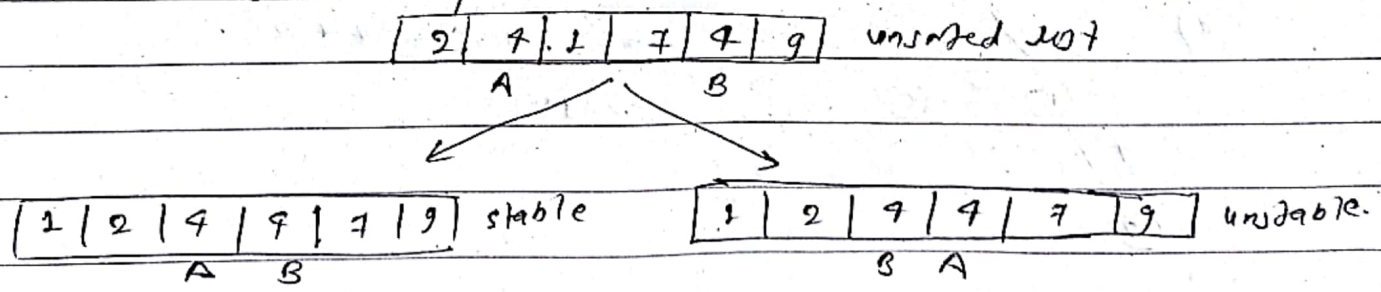
⇒ Using any technique, arrangment of data in ascending or descending order is called sorting.

(a)

Internal sorting	External sorting
- It is any data sorting process that takes place entirely within the main memory of computer.	- It is data sorting process where values to be sorted are normally held on larger hard disk.
- Suitable for small data sorting	- Used for large data sorting.
- It is faster	- It is slower.
- It uses RAM.	- It uses hard disk.

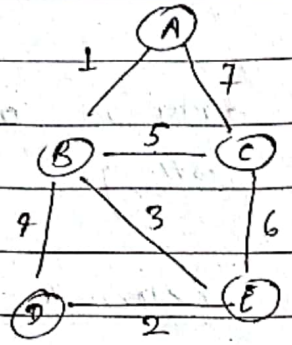
(b)

Stable sort	Unstable sort
- A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in the sorted output as they appear in unsorted input.	- A sorting algorithm is said to be unstable if there are two or more objects with equal keys which don't appear in same order before and after sorting.



eg. bubble, merge, insertion sort. eg. selection, quick sort.

(27) Find the MST for following graph using Prims and kruskal Algorithm.



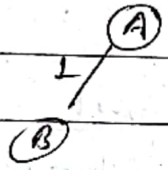
=) (1) using prims Algorithm.

starting from A.

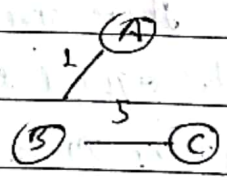
Active node list = A, B, C, D, E

a)

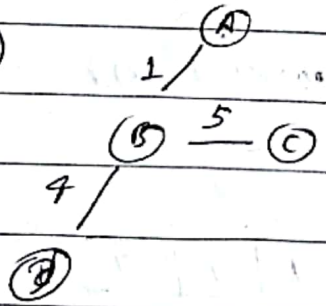
b)



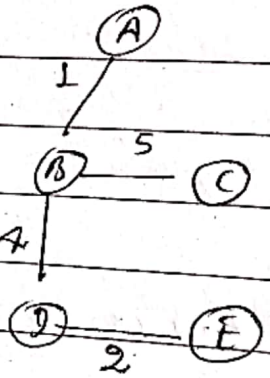
c)



d)



e)



edges belongs to MST are

(A, B), (B, C), (B, D), (D, E)

weight of MST

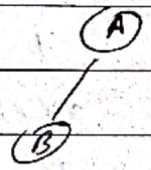
=) 1 + 5 + 4 + 2

→ 12.

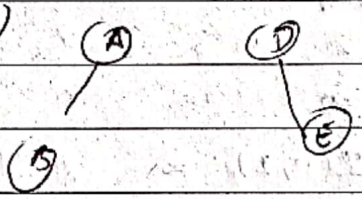
(ii) using Kruskal Algorithm:

edges		priority queue	
(A, B)	(A, B)	1	1
(A, C)	(D, E)	2	2
(B, C)	(B, E)	5	3
(B, D)	(B, D)	7	4
(B, E)	(B, C)	9	5
(C, E)	(C, E)		6
(D, E)	(A, C)		7

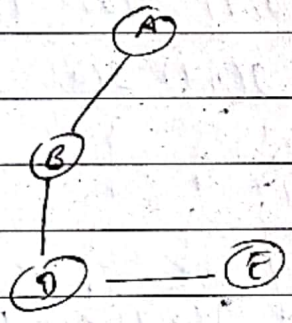
a)



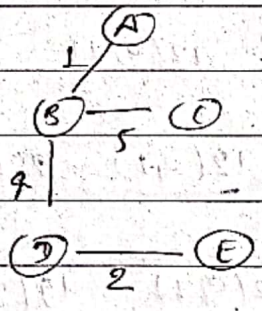
b)



c)



d)

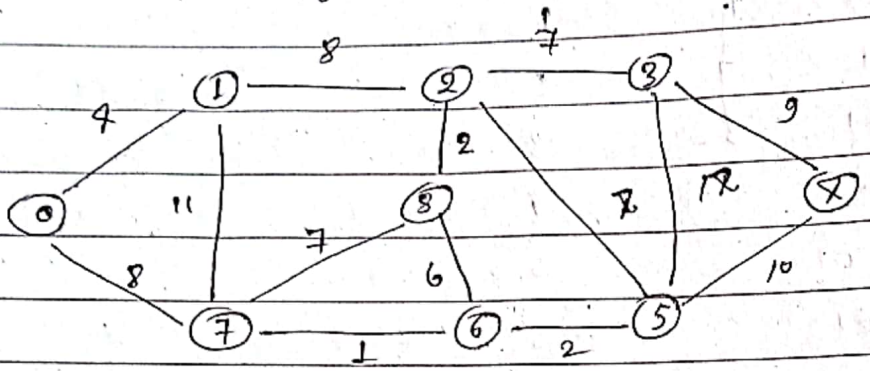


Edges belong to MST ⇒ (A, B), (B, C), (B, D), (D, E)

Weight of MST ⇒ 1 + 5 + 4 + 2

⇒ 12.

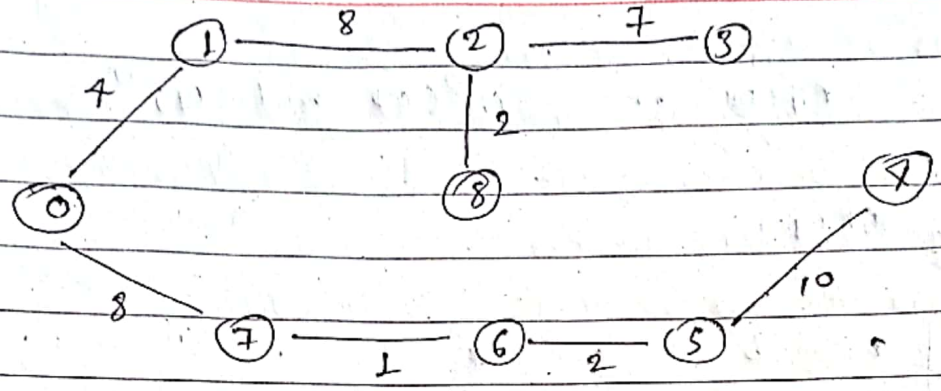
25. Find the shortest path in the following graph using Dijkstra Algorithm.



starting from node 0.

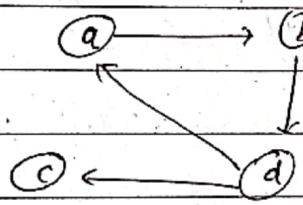
selection.	0	1	2	3	4	5	6	7	8
0	0	4(0)	∞	∞	∞	∞	∞	8(0)	∞
0,1,	0	4(0)	12(0,1)	∞	∞	∞	∞	8(0)	∞
0,1,7	0	4(0)	12(0,1)	∞	∞	∞	9(0,7)	8(0)	15(0,7)
0,1,7,6	0	4(0)	12(0,1)	∞	∞	11(0,7,6)	9(0,7)	8(0)	15(0,7)
5	0	4(0)	12(0,1)	25(0,7,6,5)	21(0,7,6,5)				
2	0	4(0)	12(0,1)	19(0,1,2)	21(0,7,6,5)		9(0,7)		19(0,1,2)
8	0			19(0,1,2)	21(0,7,6,5)	11(0,7,6)	9(0,7)	8(0)	19(0,1,2)
3	0	4(0)	12(0,1)		21(0,7,6,5)		9(0,7)	8(0)	19(0,1,2)
4	0								

∴ selected nodes ⇒ 0, 1, 7, 6, 5, 2, 8, 3, 4



It gives the shortest path.

96. Find transitive closure of following graph.



$r_{ij}(k) = r_{ij}\wedge(k-1)$ or $r_{ik}\wedge(k-1)$ and $r_{kj}\wedge(k-1)$

$k = 1, 2, 3, 4, \ldots n$, n is the number of node in the graph.

Table for $r(0)$

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

$k = 1,$

$r_{ij}(1) = r_{ij}^{0}$ or r_{i1}^{0} and r_{1j}^{0}

table for r(1)

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

$k = 2.$

$r_{ij}^{2} = r_{ij}^{(1)}$ or $r_{i2}^{(1)}$ and $r_{2j}^{(1)}$

table for r(2)

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

$k = 3.$

$r_{ij}^{3} = r_{ij}^{2}$ or r_{i3}^{2} and r_{3j}^{2}

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

Date
Page

$k = 4$

$r_{ij}^{4} = r_{ij}^{3}$ or $(r_{i4}^{3}$ and $r_{4j}^{3})$

table for $r(4)$

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

which is required matrix ~~using~~ for ~~find~~ transitive closure.

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

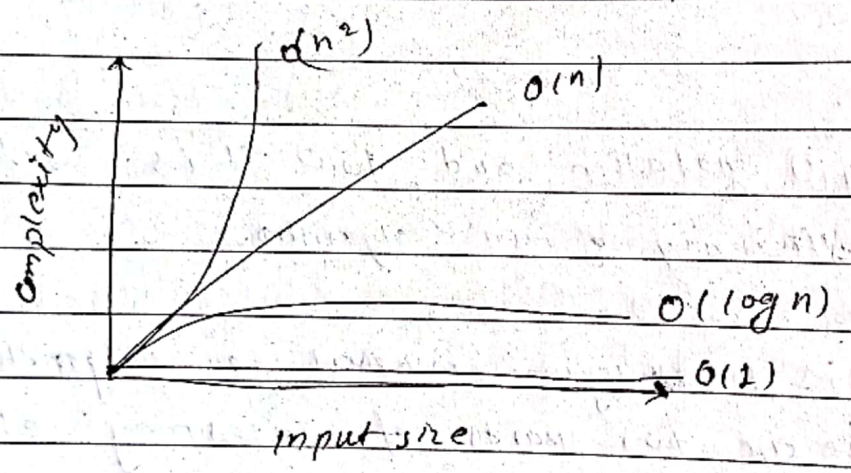
⇒ Big O notation is a way to measure an algorithm's efficiency. There are two parts of measuring efficiency time complexity and space complexity. Time complexity belongs to how long the function takes to run in terms of its computational step and space is amount of memory used by the function. Big O notation measures the time it takes to run ~~~~ function as the input grows.

Date
Page

Big O is sometimes referred to as the algorithm's upper bound, meaning that it deals with the worst-case scenario. Best-case scenario doesn't really tell us anything. We use worst-case to remove uncertainty The algorithm never perform worse than we ~~perfor~~ except.

It consider different runtime complexity.

- O(1) → constant runtime complexity
- O(n) → linear
- O(n²) → quadratic
- O(log n) → logarithmic



O(1) notation has the best scalibility as the input size grows, while the algorithm with O(n²) has the worst scalibility among all.

lets consider two cases. Binary and linear search.

- Best case for linear search would be if we want to search value 1, which is the first element of the array. In this case we have O(1) complexity.

- Worst case for linear search, if we want to search 9 which is the last element of the array. we need to traverse each element so, we have O(n).

- In case of binary search, we want to search a value which is at the middle of array. we have O(1).
- For worst case we want to search first or last element. In this case we have O(log n).

In this case Big O notation help us to determine the efficiency of an algorithm.

28. Explain divide and conquer strategy algorithm:

=) Divide and conquer strategy involves three parts:

1. Divide
   - Dividing the problem into smaller sub problems.

2. Conquer
   - solve sub problems by calling recursively until solved

3. Combine
   - combine the sub problems to get the final solution of the whole problem.

There are some algorithm which follows divide and conquer strategy:

1. Quick sort

Algorithm picks a pivot element and rearrange the array elements so that all smaller element than pivot are move to left while larger are moved to right side. Finally the algorithm recursively sorts the subarrays on the left and right of the pivot element.

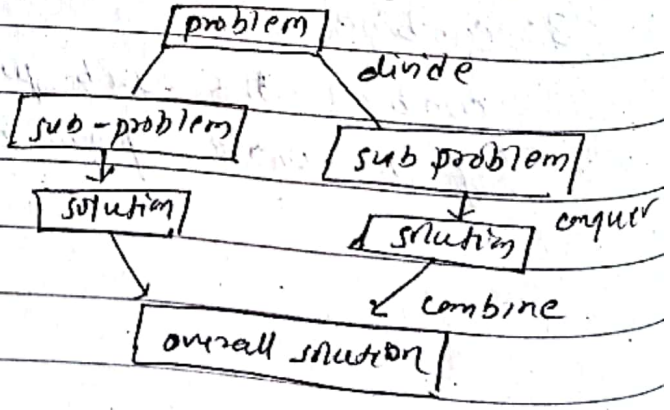
2. merge sort

It divide the array into two halves, sort them and finally merges two sorted halves.

3. ~~closest path~~ at

In this way divide and conquer strategy are used in various algorithm. which make efficient algorithm. problems are solved recursively so the code becomes easy.

There are more other algorithms such as, closest pair of points, karatsuba algorithm for fast multiplication, etc. Cosley - Tukey fast Fourier Transform algorithm (FFT) etc.



THE END