* Algorithm : Algorithm is a step-by-step procedure for sloving a particular proble

* Types of data structure ⟸

(i) Linear
- arrays
- stack
- queue
- linked list

Ex ⇒

(ii) non-linear
- Trees
- graph

Ex ⇒

Q what are the operation perform on arra

ans ⇒ Traversing, Searching, Merging, Insertio deleetion, shorting.

* Data Structure :- Logical organization of data in a particular ~~mamner~~ manner.

or The logical organization of data to slove a particular problem in a particular ~~mamer~~ manner is called data structure.

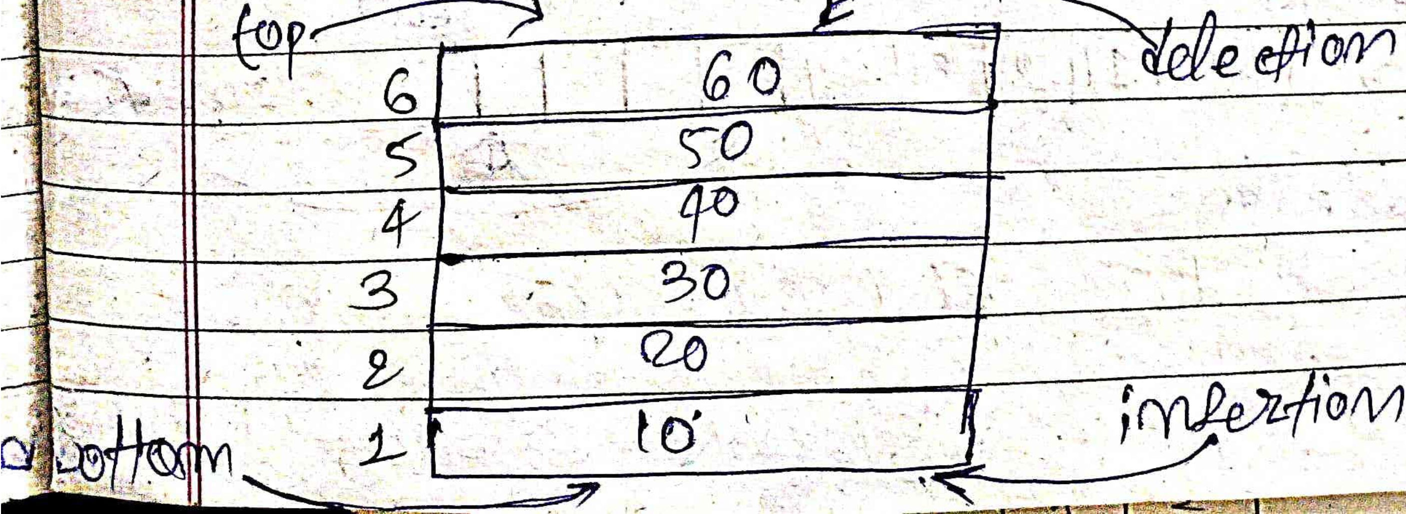
* What do you mean by time complexity ?

ans → Rate of growth of time taken by an algorithm to slove a particular problem with respect to input size.

* What do you mean by space complexity.

ans → space complexity refers to the amount of memory or storage space an algorithm uses to slove a problem with respect to input size.

* stack :- Stack is a linear data structure which operates in a LIFO (last in First out) or FILO (First in last out) pattern.



Standard stack operations :-

① push() → Insert element

② pop() → Delection element

③ isEmpty() → stack is empty

④ isfull() → Stack is full

⑤ Peek() → Access the item at the i position

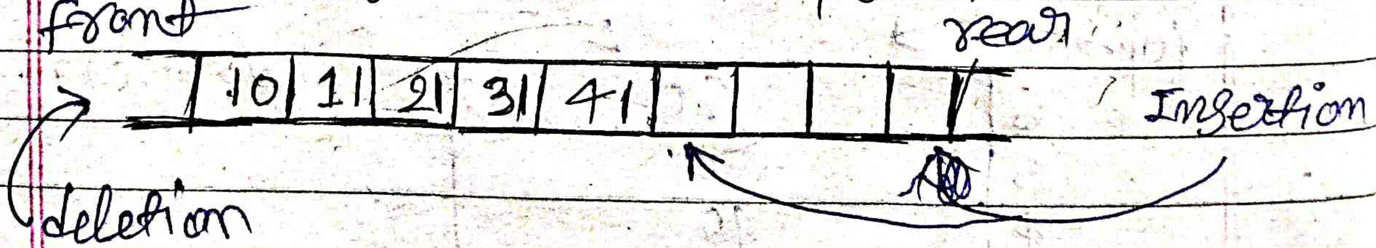
⑥ Count() → get the no of item in the stack.

⑦ change() → change the item at the i position

⑧ display() → Display all items in the stack.

* What is Queue Data structure?

ans ⇒ Queue is a linear data structure which Operates in a ~~First~~ (FIFO) First in first out or (LILO) last in last out pattern



Standard ~~of~~ Queue operations

① enqueue() ② dequeue() ③ isempty() ④ is full() ⑤ Peek() ⑥ count() ⑦ Change() ⑧ display()

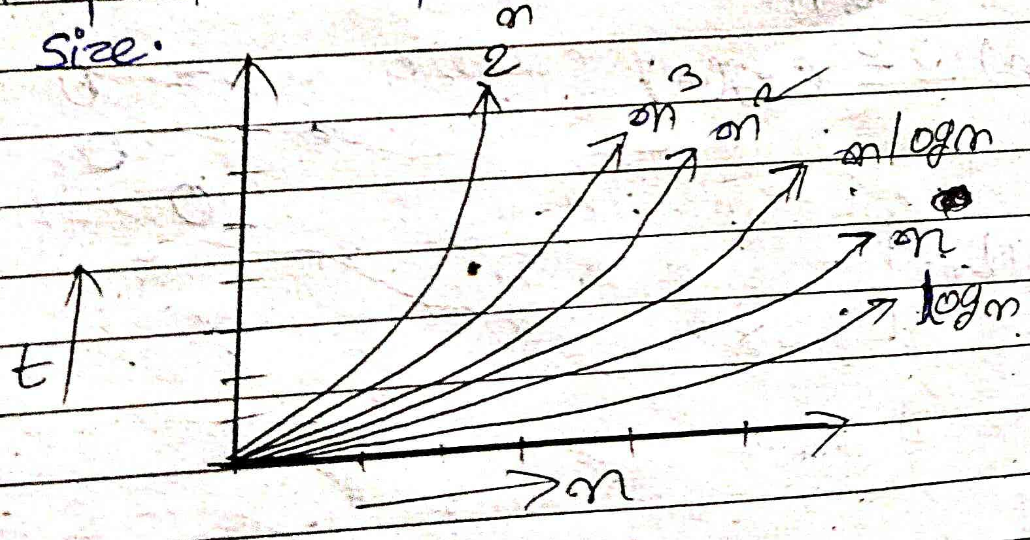
Types of data structure

Data structures

- **Linear data structures**
  - Array
  - Stack
  - Queue
  - Linked list
- **Non linear data structure**
  - Trees
  - graph
  - Tries

Complexity of algorithm

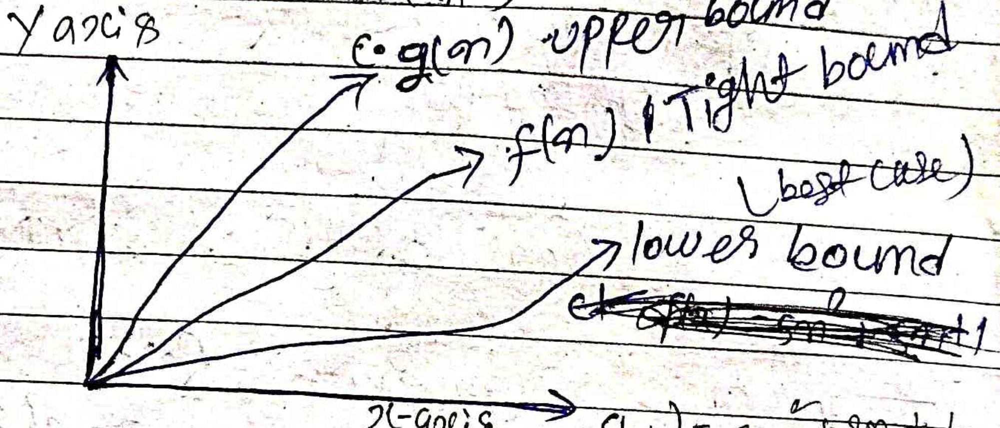
* Time complexity : Rate of growth of time taken by an algorithm to solve a particular problem with respect to input size.



Assymptotic Notation (O(1), θ, Ω)

1.* **Big O Notation:** The Big O notation is the mathematical way to express the upper bound or the longest amount of time an algorithm can possibly take to complete the program.

Constant time → $O(1)$
linear → $O(n)$
~~logrithem~~ logarithmic → $O(\log n)$
~~Quad~~ quadratic → $O(n^2)$
cubic → $O(n^3)$



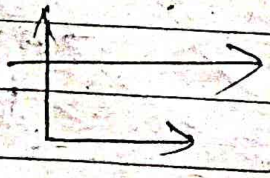
$f(n) = 5n^2 + 2n + 1$

$f(n) = O(g(n))$

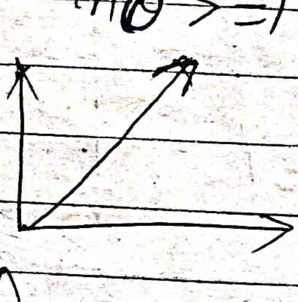
Iff

$f(n) <= c \cdot g(n)$ { where $n >= n_0$, $c > 0$, $n_0 >= 1$ }

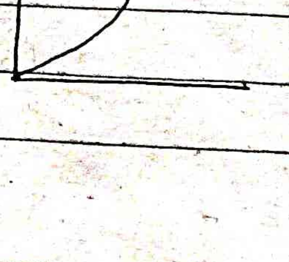
Ex → $O(1)$



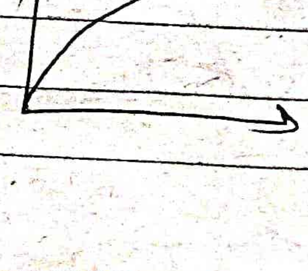
$O(n)$



$O(n^2)$



$O(\log n)$



Questions

$f(n) = 2n^2 + 3n \Rightarrow O(n^2)$

$f(n) = 4n^4 + 3n^3 \Rightarrow O(n^4)$

$f(n) = N^2 + \log n \Rightarrow O(n^2)$

$f(n) = 1200! \Rightarrow O(1)$

$f(n) = 3n^3 + 2n^2 + 5 \Rightarrow O(n^3)$

$f(n) = \frac{n^3}{300} \Rightarrow O(n^3)$

$f(n) = 5n^2 + \log n \Rightarrow O(n^2)$

$f(n) = \frac{n}{4} \Rightarrow O(n)$

$f(n) = \frac{n+4}{4} \Rightarrow O(n)$

Note → (i)

```
for (   )
{      }     → O(n)
for (   )
{      }     → O(m)
```

TC = $O(n+m)$

(ii)

```
for (0 — N)          → O(n)
{  for (0 — N)       → O(n)
     { }
                       O(n^2)
}
for (0 — N)          → O(m)
{
}
```

$O(n^2) + O(m) \Rightarrow$ $\boxed{TC: O(n^2+m)}$

* What is the time complexity of the following code

```
a = 0;
for (i=0; i<n; i++) {
{ for (j=N; j<i; j--)
  { a = a+i+j;
  }
}
```

```
for(0 — N)
{ for(0 — N) { }
}
```

$O(n^2)$

* what is the time complexity

```
int a=0 , b=0;
for (i=0; i<N; i++)
{   a=a+rand();
}
for (j=0; j<M; j++)
{ b=b+rand();
}
```

for (0 — N) O(n)

for (0 — M) O(M)

$\boxed{TC = O(n+m)}$

(3)

```
int a=0, b=0;
for (i=0; i<N; i++)
{ for (j=0; j<N; j++)
  { a=a+j;
  }
}
for (k=0; k<N; k++)
  { b=b+k;
  }
```

for(0 — N) O(n)
{ for (0 — N) O(n)
} TC = O(n²)

→ for(0 — N)
O(n)

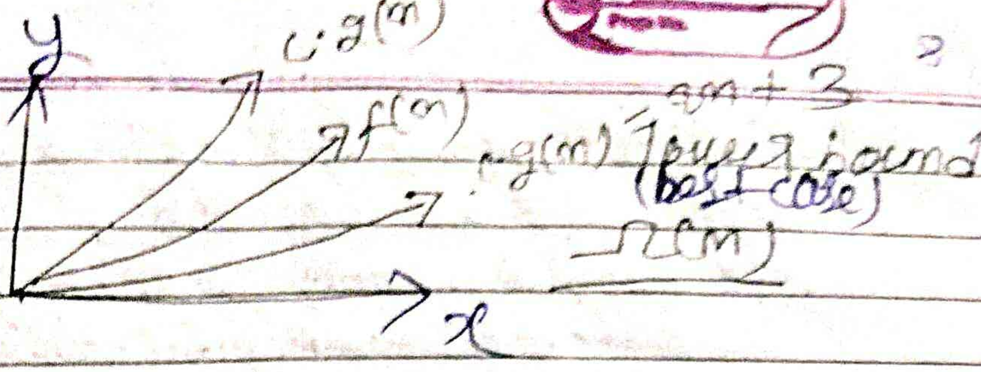
$O(n^2) + O(n) \Rightarrow O(n^2)$

(4) Big Omega (Ω) : Omega notation Specifically describes best case scenario. It represents the lower bound time complexity of an algorithm.

$$f(n) = \Omega(g(n))$$

Iff

$f(n) >= c \cdot g(n)$ { where $n > n_0$, $c > 0$, $n_0 >= 1$



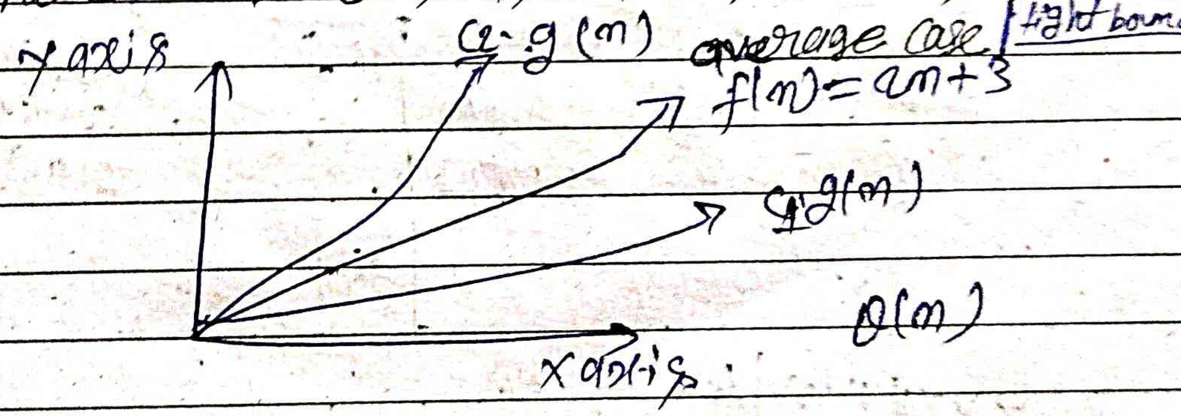
(3) Big theta (Θ) :- Big theta notation specifically describes average case scenario it represents the most realistic time complexity of an algorithm.

$$f(n) = \theta(g(n))$$

Iff

$$\boxed{\Theta(g(n)) = C1 \cdot g(n) <= f(n) <= C_2 \cdot g(n)}$$

(where $n > n0$, $C1, C2 > 0$, $n > n0$, $n0 >= 1$)



* Space complexity :- Space complexity refers to the amount of memory an algorithm uses to slove a problem with respect to input size.

Space complexity includes both Auxiliary space and space used by input.

Space C = input size + Auxiliry space

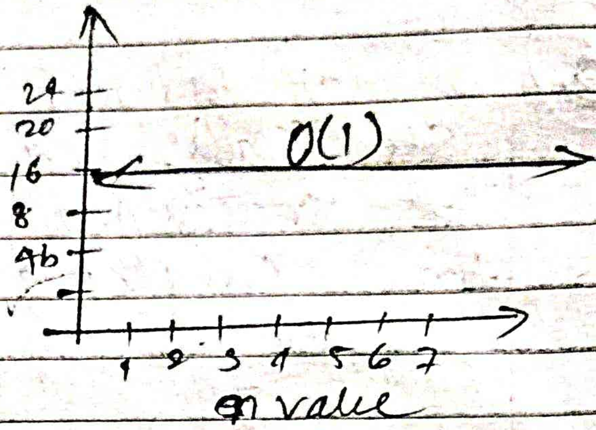
* Auxiliry space : Auxiliry space is the temporary space allocated by your algorithm to slove the problem, with respect to input size.

space C = input size + Auxiliry space

A-1 → fun add(n1, n2)
{ sum = n1+n2
return sum
}

size of bytes

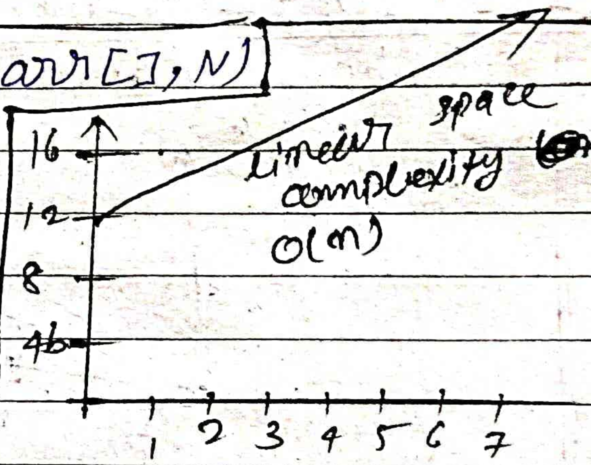
n_1 → 4 bytes
n2 → 4 bytes
sum = 4 bytes
Aux sp = 4 bytes
Total = 16 bytes



A2 → fun sumofNumbers(arr[], N)
{ sum = 0
~~{sum~~ for(i = 0 to N)
{ sum = sum + arr[i]
}
print(sum)
}

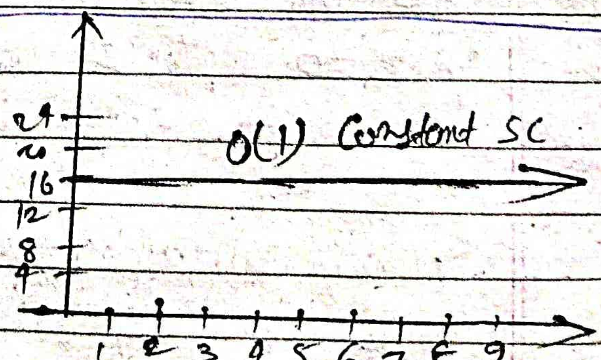
arr → N×4bytes
sum = 4 bytes
i = 4 bytes

Aux Sp = 4 bytes
tot = 4N + 12b
O(n)
4(n) + c

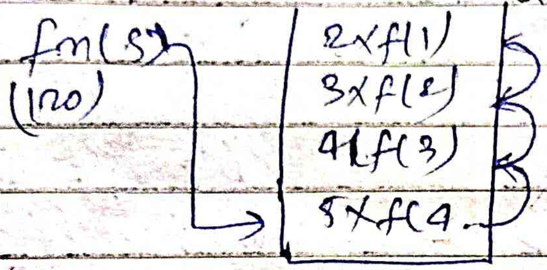


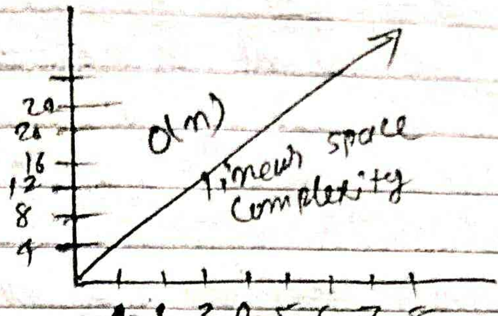
A→3 → int fact=1;
for (int i=1; i<=n; i++)
{ fact *= i; }
return fact;
fact = 4bytes
i = 4 bytes
n = 4 byte
Aux = 4 byte

TSp = 16 by



P = 4 → if (n <= 1)
{ return 1;
} else {
~~fact...~~
return (n * factorial2(n-1));
} 5 * f(4)
// n → 4 byte, AUSP = 4 bytes





fact ⇒ n × 4 bytes
S = 4 bytes + n bytes

Q what do you meant by time space Trade off

ans ⇒ Time space trade off is a way to slove a problem in less time by using more storage space or by slaving a problem in very little space by spending a long time.

Q1 difference b/w linear search and binary search.

	linear search	binary search
i	Also called sequential search	Also called half-interval search & logarithmic search
ii	less efficiency	High efficiency
iii	less complex than binary	More complex than linear
iv	time complexity O(n)	time complexity O(log n)
v	Best case to find the element in the first position	Best case to find the element in the middle position.
vi	No required of sorted array.	required of sorted array.

Imp Q. Efficiency of binary search or How binary search efficient to linear search.

ans → linear search

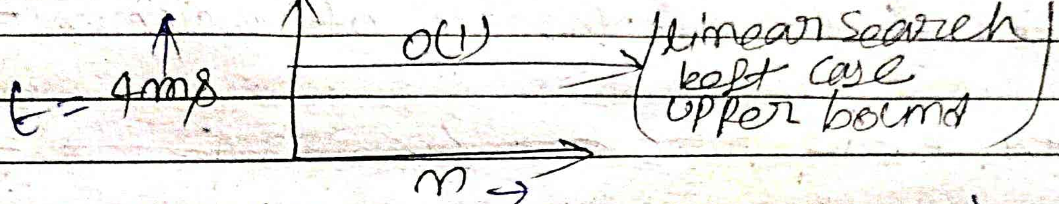
9	10	11	12	13
0	1	2	3	4

size = ~~4~~ 5

$x = 9$

at index = 0 best case $O(1)$

No of comparison = 1 (found)



(It becomes independent of n)

for worst case → (i) last
→ (ii) not found

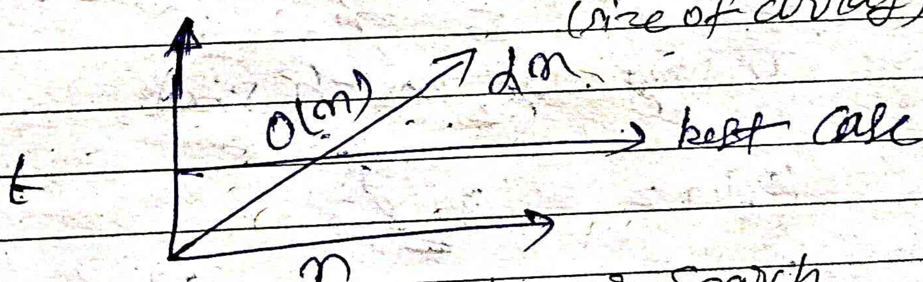
linear search

9	10	11	12
0	1	2	3

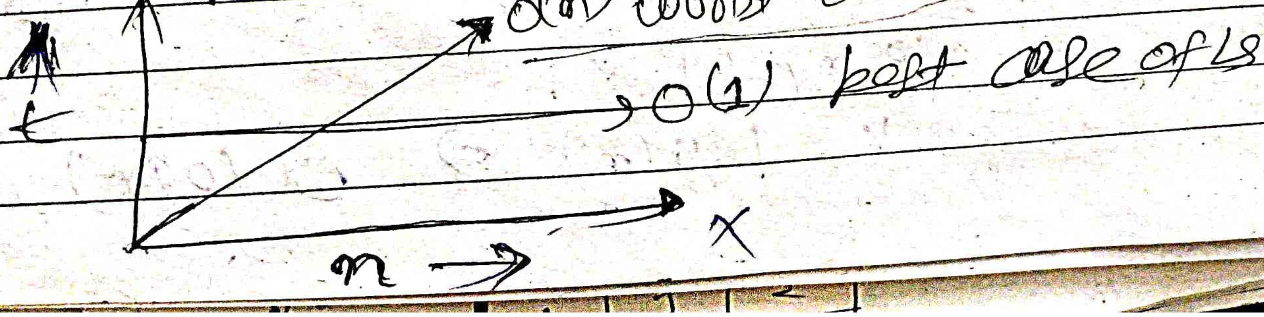
size = 4

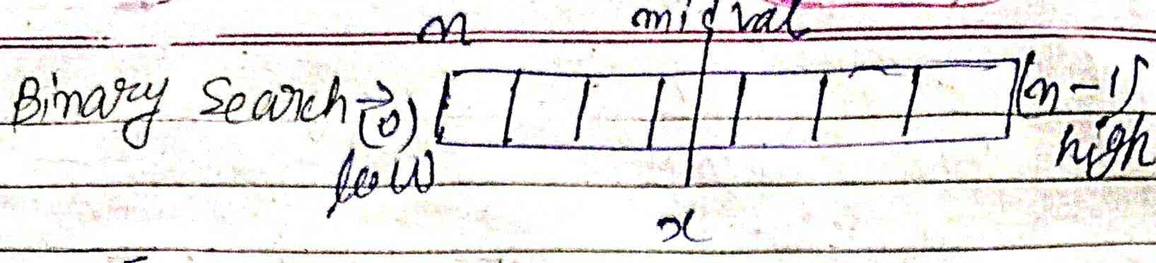
$x = 12$ No of comparison = 4 → found $O(n)$

$x = 17$ No of comparison = 4 → not found $O(n)$
(size of array)



Analysis of linear search





a[mid] = = x ? found

< >

Analysis of binary search

After first call the no of element $= \frac{n}{2} = \frac{n}{2^1}$

After second ——— $= \frac{n}{2} \times \frac{1}{2} = \frac{n}{2^2}$

" third ——— $= \frac{n}{2} \times \frac{1}{2} \times \frac{1}{2} = \frac{n}{2^3}$

" nth ——— $= \frac{n}{2^k}$

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k$

Analysis of worst case

nth call $= n = 2^k$

Taking $\log_2$ both side

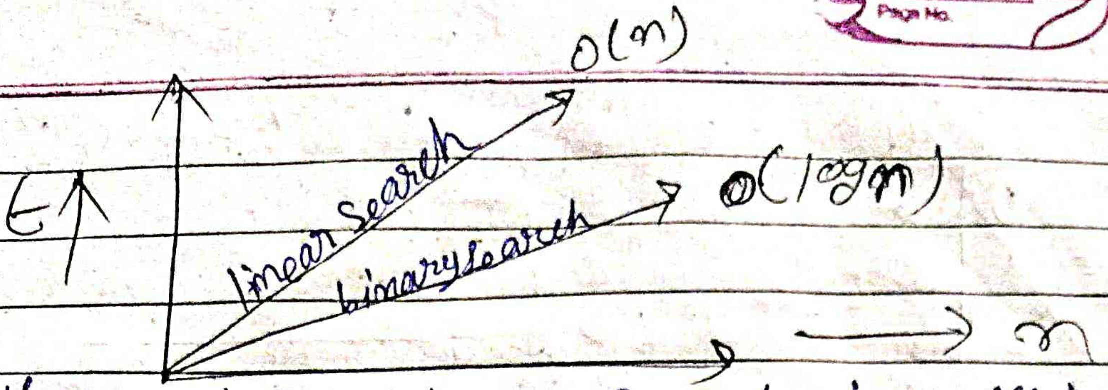
Rai

$\log n = \log_2 2^k$

$\log n = k \cdot \log_2 2 \quad \{\log_2 2 = 1\}$

$\log n = k \times 1$

$k = \log(n) \Rightarrow O(\log n)$



that's how binary search is efficient to linear search.

* How a multidimensional array is stored in a computer system & representation of column and row major order

ans⇒

A multidimensional array is stored in a computer system in a linear fashion even if It is multidimensional array because memory is a one dimensional thing.

There are two main ways to store a multidimensional array in memory

(i) Row-major order (ii) Column-major order

(i) Row-major order: In row-major order, the elements of the array are stored in rows, from left to right and top to bottom. The first row is stored first, followed by the second row, and so on.

left → right / Top → bottom

$$
\begin{array}{c|cccc|}
 & 0 & 1 & 2 & 3 \\
\hline
0 & 6 & 9 & 8 & 3 \\
1 & 0 & 7 & 9 & 4 \\
2 & 8 & 3 & 7 & 0 \\
3 & 5 & 1 & 6 & 1 \\
\hline
\end{array}
$$

ROW1 ROW2 ROW3 ROW4

6	9	8	3	0	7	9	4	8	3	7	0	5	1	6	1

→ Computer memory

(2) Column-major order: In Column-major order, the elements of the array are stored in columns, from top to bottom and left to right. The first column is stored first, followed by the second column, and so on.

$$
\begin{array}{c|cccc|}
 & 0 & 1 & 2 & 3 \\
\hline
0 & 6 & 9 & 8 & 3 \\
1 & 0 & 7 & 9 & 4 \\
2 & 8 & 3 & 7 & 0 \\
3 & 5 & 1 & 6 & 1 \\
\hline
\end{array}
$$

6	0	8	5	1	9	7	3	1	8	9	7	6	3	4	0

Col-1 Col-2 Col-3 Col-4

→ Computer memory

Schaum series

* Application of Stack

① Conversion of infix to prefix and postfix expression using stack.

* Infix Expression: <operand><operator><operand>
  (a) (+) (b)
  ↳ infix

* prefix expression: <operator><operand><operand>
  +ab → prefix

* postfix expression: <operand><operand><operator>
  ab+ → ~~postfix~~ postfix

^
*, /, %
+, -

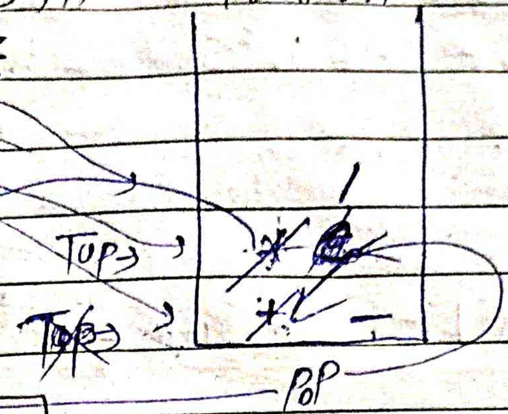
* Rules for convert infix to postfix.

1. Scan Expression from Left to Right.
2. Print OPERANDS as the the arrive.
3. If OPERATOR arrives & stack is empty, push this operator onto the stack
4. If incoming OPERATOR has Higher precedence than the Top of the stack, push it on stack.
5. If incoming Operator has lower precendence than TOP. ~~then stack last the~~ of the stack then POP and Print the TOP. Then ~~test~~ the incoming operator against the NEW TOP of stack.
6. If incoming operato has equal precendence with Top of stack, use Associativity rules.
7. For associativity of Left to Right - POP and print the Top of stack, then PUSH the incoming operator
8. For associativity of Right to Left - PUSH incoming operator on stack.
9. At the end of Expression, pop & print all Operators from the stack.
10. If incoming SYMBOL is '(' Push it onto stack.
11. If incoming SYMBOL is ')' POP the stack and Print Operators till '(' is found & discard it
12. If Top of stack is '(' Push operator on stack.

1 Application

1 Infix to postfix expression

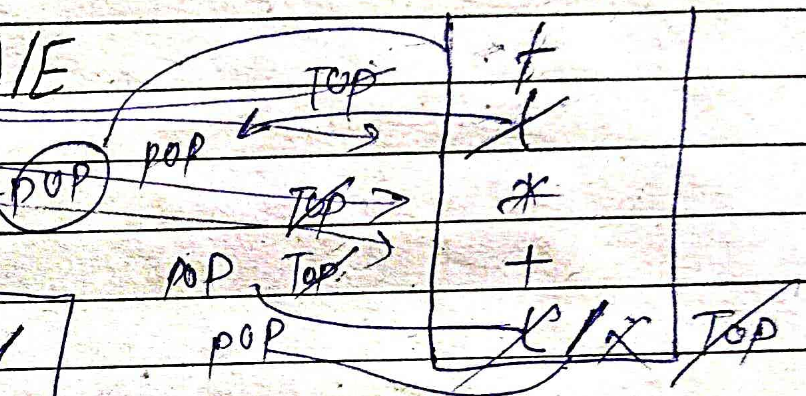
A+B*C-D/E



~~ABC*+DE/+~~

ABC*+DE/-

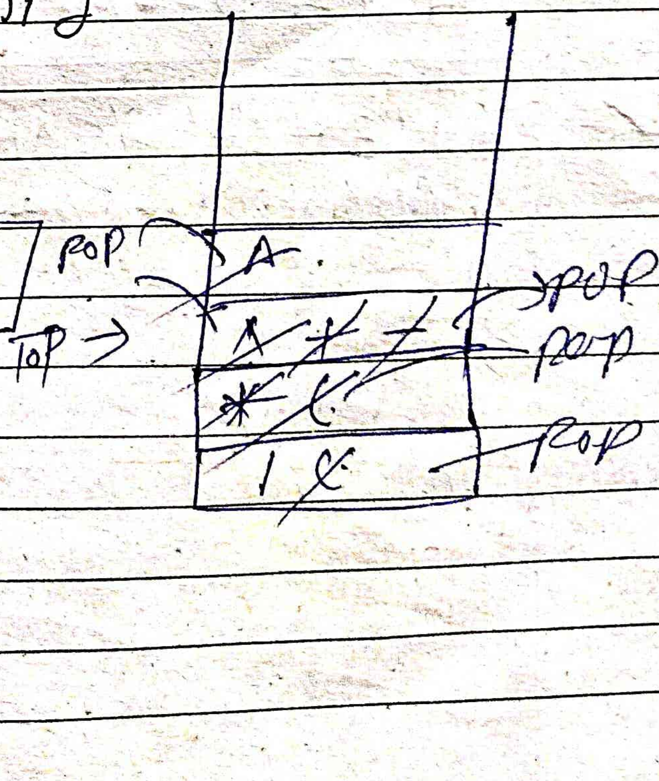
2 (A+B*(C-D))/E



ABCD-*+E/

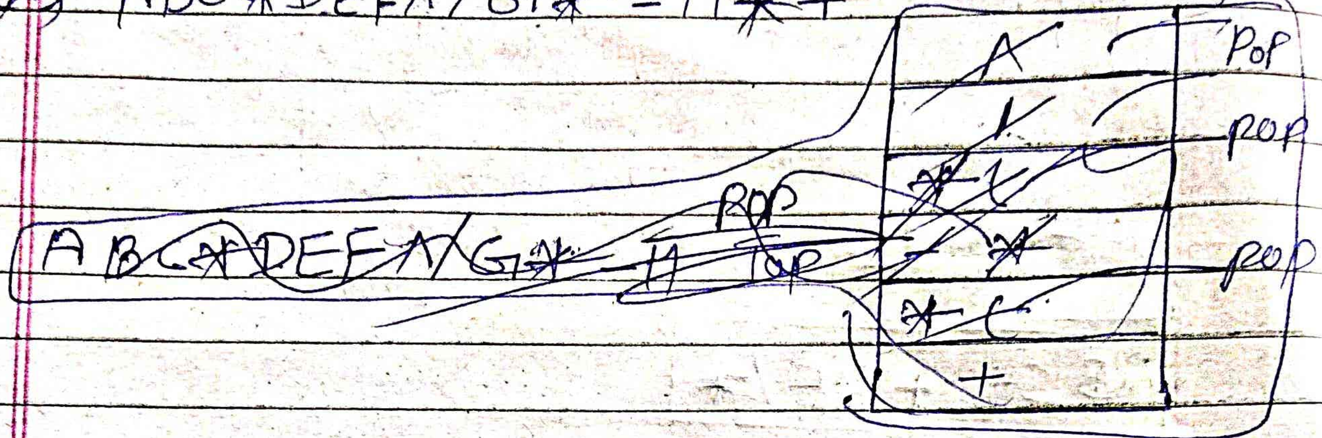
* ~~infix: A+(B*C-(D/E^F)*G)*H~~
* ((a+b-c)*d^e^f)/g

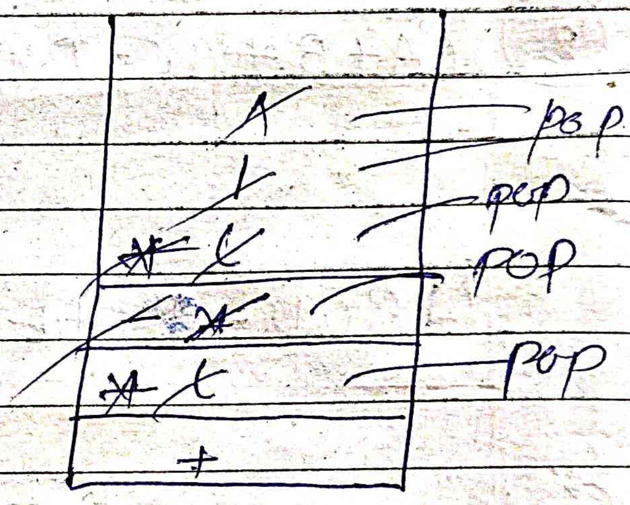
ab+c-def^^*g/



(M) $A+(B*C-(D/E \wedge F)*G)*H$

ans→ $ABC*DEF \wedge /G*-H*+$





$\boxed{ABC*DEF \wedge /G*-H*+}$

* $A+(B*C-(D/E \wedge F)*G)*H$

$ABC*DEF \wedge /G*-H*+$

(2) Applic

* Convert infix to prefix expression.

Infix expression

↓

Reverse & change '(' → ')' & ')' → '('

↓

infix to postfix

↓

output

↓

Reverse

↓

prefix Expression

1. Reverse infix expression & swap '(' to ')' & ')' to '('
2. Scan Expression from left to Right.
3. print Operands as the arrive.
4. If operands arrive & stack is empty, push to stack

~~(5) than the Top of the stack, push it on stack.~~

5. If incoming operator has higher precedence than the Top of the stack, push it on stack.

* 6. If incoming operator has ~~equal~~ equal precedence with Top of stack && incoming operator is '^' pop & print Top of stack. Then test the incoming operator against the New Top of stack.

* 7. If incoming operator has equal precedence with Top of stack, push it on stack.

8. and same as ~~post~~ infix to postfix

13. At the end Reverse output ~~string~~ string again.

② APP

* Convert infix to prefix

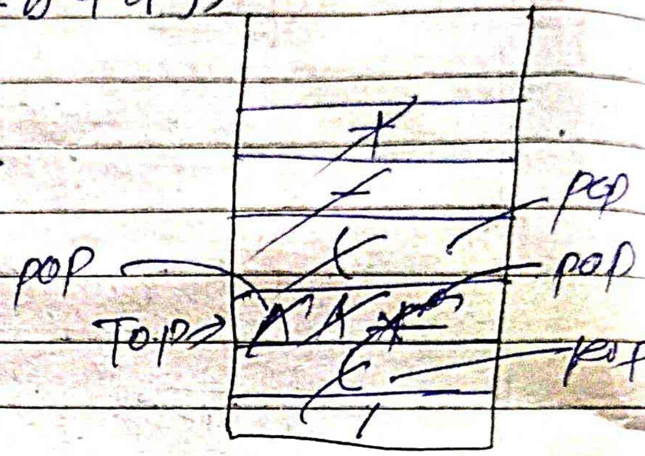
Q ((a+b-c)*d^e^f)/g

Reverse & swap (→) &) → (

g/(f^e^d*(c-b+a))

↓

C. into in to postfix



gfe^d^cba+-*/ — output

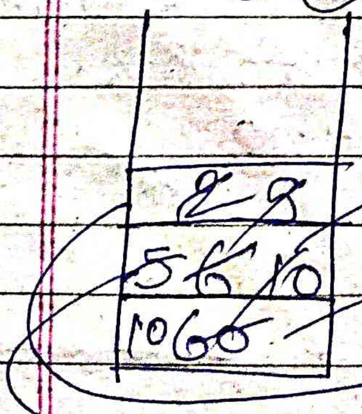
Reverse

/*-+abc^d^efg

③ APP

* Evaluation of postfix expression

Q 60, 6, /, 5, 2, *, 5, -, +



pop → A = 6, B / A = 60/6 = 10

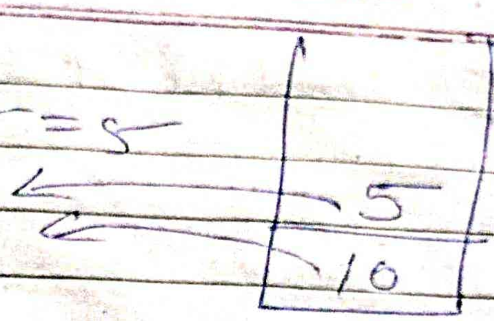
pop → B = 60

pop A = 2 B * A → 5×2 = 10

pop B = 5

$A = 5,\ B = 10$

$B - A = 10 - 5 = 5$



$A = 5$

$B = 10$

$B + A = 10 + 5 = 15$

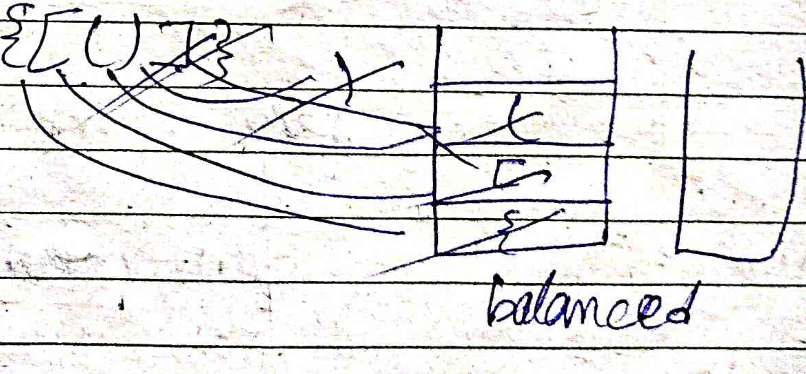
$\boxed{\text{Answer} = 15}$

→ stack में digit value को Left से लिखते जाना जब भी कोई भी operator मिलता है stack के ऊपर के दो element pop कर लो A और B में और दिए गए operator से value find कर लो और फिर उस value को stack में डाल दो फिर आगे जब कोई digit value हो उसे stack में फिर डालो आगे कोई operator हो तो ~~value~~ दो element pop कर value find करें ~~same~~ फिर store कर दो Stack में same process.

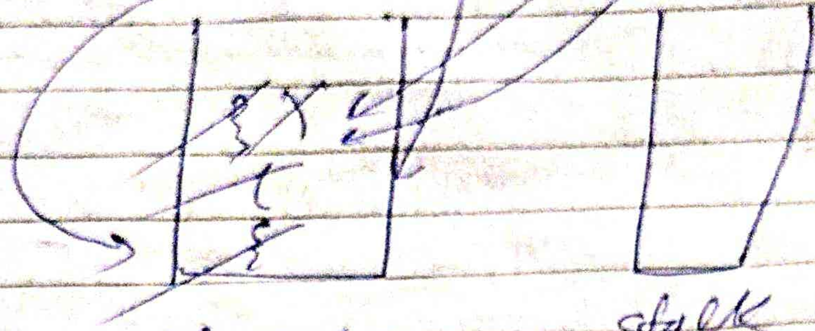
app

4 Balanced parenthesis checking

Q {[()]}



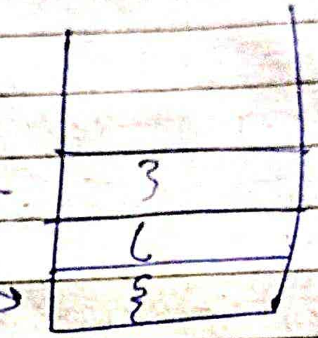
(ii) ~~{a+(b+c})~~ {a+(b+c)}



balanced

stack

(iii) { {a+(b+c})



(≠ }

stack

Not balanced

app

(5) factorial:

```
int factorial(n)
{ if (n==0)
   { return 1;
   }
else {
   return n * factorial(n-1);
}
```

```
int main()
{
r=factorial(4);
cout<<r;
}
```

n=4

f(4)=4*f(3)

f(3)=3*f(2)

f(2)=2*f(1)

f(1)=1*f(0)

f(0)=1

f(1)=1
f(2)=2*f(1)
f(3)=3*f(2)
f(4)=4*f(3)
main()

poped

Stack

* Why we need to convert infix to postfix expression?

ans ⇒ Infix expression are readable and slovable by humans. We can easily distinguish the order of operators and ~~also parenthesis to slove~~ we can use the parenthesis to slove that part first during mathematical expression. The computer cannot differentiate the operators and paranthesis easily. that's why we need to convert infix to postfix expression.

Q. write Algorithm that how you insert & delect element in array.

Solⁿ: Insert element in array.

1. start
2. Enter your position of array to insert element.
3. input: pos.
4. pos = pos - 1.
5. for loop → for(j=size-1; j>=pos; j--).
6. arr[j+1] = arr[j].
7. Enter element to be insert.
8. input element.
9. arr[pos] = ele;
10. print new array. 11. end

To delete element from array.

1. Start.
2. Enter position of array to be delect.
3. input pos.
4. pos = pos - 1.
5. for loop → for(i=pos; i<=size-1; i++)
6. arr[i] = arr[i+1].
7. print new array.
8. end.

Q. ~~Operation on stack, push, pop, overload underflow~~

Q. Enlist the various operations that can be performed on a data structure.

(1) Traversing : Visiting each element of a data structure one by one.

(2) Insertion : Adding a new element to a data structure

(3) Delection : Removing an element from data structure.

(4) Searching : Finding an element in a data structure.

(5) Sorting : Arranging the elements of a data structure.

(6) Merging : Combining two data structures into one.

(7) Update : Changing the value of an element in a data structure.

(8) Count : Counting the elemens which are present in a data structure.

(9) Reversal : Reversing the order of the elements in a data structure.

(10) Minimum/Maximum : Finding the minimum or maximum element in a data structure.

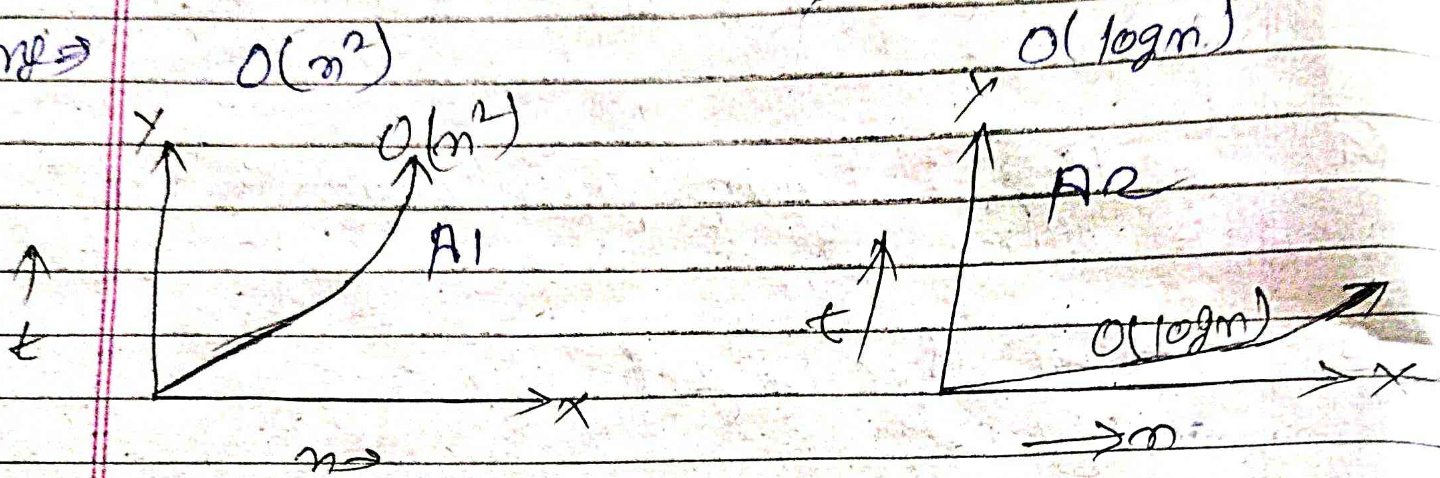
(11) Sum/Average : Finding the sum or average of the element in a data structure.

Q For the same problem P, if you have two different algorithm A_1 and A_2 with time complexities as follows:

· Algorithm $A_1 \Rightarrow O(n^2)$

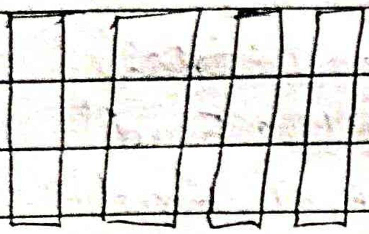
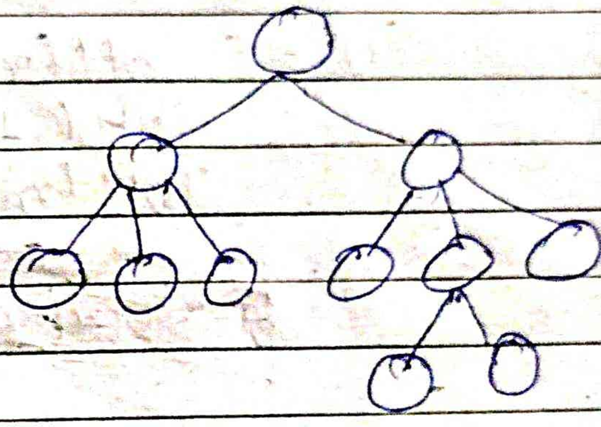
Algorithm $A_2 \Rightarrow O(\log n)$ which algorithm you will choose?

Ans ⇒



As we can see A_2 has lower bound and A_1 has tight bound that's why I choose A_2 Algorithm.

Q What is difference between linear and a non linear data structure with example

	linear	non-linear
(i)	examples: Arrays, stack queue, linklist.	Examples: Trees, graph Sets, Tables
(ii)	Linear data structure 	Non Linear data structure 
(iii)	It can be traversed in single run.	It cannot be traversed in a single run.
(iv)	Not very memory efficient	More memory efficient
(v)	used in simple application	used in complex appliat
(vi)	data elements are arranged sequentially in a single line.	data elements are arranged hierarchically in multiple levels.

* Demonstrate the use of stack data structure is used to slove toh problem.

* Write a Recursive Algorithm to slove Tower of Hanoi problem.

Sol^a ->

```
ToH(N, Beg, AUX, end)
{    if (n==1)
     { Beg -> End;
       return;
     }
  else { ToH(n-1, Beg, End, AUX)
         ToH(1, Beg, AUX, End)
         ToH(n-1, AUX, Beg, End)
}
```

~~TOH(2, A, C, B) -> TOH(1, A, B, C) -> TOH(1, A, C, B) -> TOH(1, C, A, B)~~

~~TOH(3, A, B, C) -> TOH(1, A, B, C)~~

disk 1 moved from	A -> C
disk 2 moved from	A -> B
disk 1 moved from	C -> B
" 3 " "	A -> C
" 1 " "	B -> A
" 2 " "	B -> C
" 1 "	A -> C

2

$\to$ TOH(1, A, B, C) $\to$ TOH(1, A, B, C)

1 Beg end Aux

$\to$ TOH(2, A, C, B) $\to$ TOH(1, A, C, B) $\to$ 3 TOH(1, A, C, B)

$\to$ TOH(1, C, A, B) $\to$ 4 TOH(1, C, A, B)

Beg Aux End

TOH(3, A, B, C)

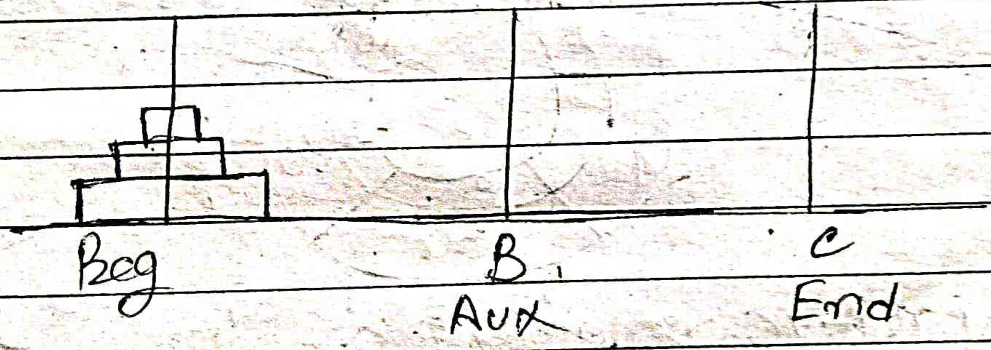
5 $\to$ TOH(1, A, B, C) $\to$ TOH(1, A, B, C)

Beg Aux end

7 $\to$ TOH(1, B, C, A) $\to$ TOH(1, B, C, A)

6 $\to$ TOH(2, B, A, C) $\to$ 8 TOH(1, B, A, C) $\to$ TOH(1, B, A, C)

9 $\to$ TOH(1, A, B, C) $\to$ TOH(1, A, B, C)



* What is TOH ~~graph~~.

ans $\Rightarrow$ Tower of Hanoi is a classic puzzle or mathematical problem that involves moving a stack of disks from one rod to another while adhering to certain rules.

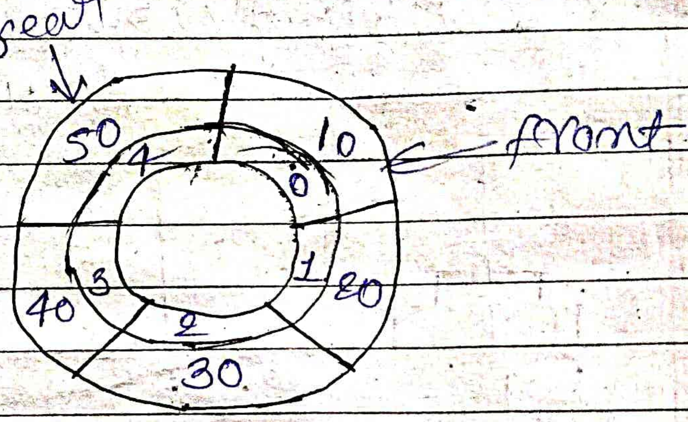
* What is circular queue?

ans ⇒ A circular queue, also known as a circular buffer or ring buffer that serves as a linear collection of elements with a fixed size unlike a regular queue. It is a linear data structure in which the last position is connected back to the first position to make a circle.

linear queue ⇒

10	20	30	40	50
0	1	2	3	4

circular queue.



pyq * Comparitive analysis of linear ~~queue~~ queue and circular queue

ans ⇒ f 0 | 1 | 2 | 3 | 4 r

initialization - int arr[5];
int rear = -1;
int front = -1;
int size = 5-1;

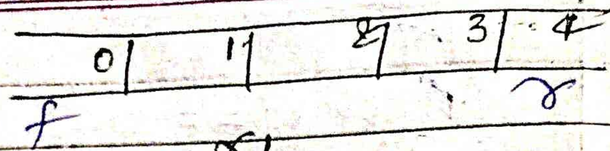
```
is empty()
{ if (front == -1 && rear == -1)
    { return true
    }
else {
    return false
    }
```

```
isfull()
{ if (rear == size)
  { return true;
  }
else {
       return false;
     }
```

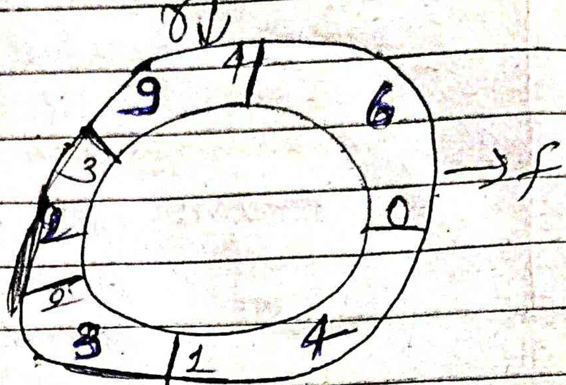
```
enque(value)
{ if(isfull())
  { return; }
else if(isempty())
{ rear = front = 0;
   arr[rear] = value;
}
else { rear++;
       arr[rear] = value;
     }
```

```
dequeue()
{ int x=0;
  if(isempty())
  { return; }
  else if (front == rear)
  { x = arr[front]
  front = rear = -1;
  }
  else
  { x = arr[front]
    front++;
  }
  return x;
}
```

Initialization-




```
int size=5; int arr[5];
N=size;  int rear=-1;
         int front=-1;
```




```
is empty()
{ if(front==-1 && rear==-1)
      return true
  else
      return false
}
```

```
is full()
{ if((rear+1)%N == front)
      return true;
  else
      return false;
}
```

```
enque(value)
{ if(isfull())
      return;
  else if(isempty())
  { rear=front=0;
    arr[rear]=value;
  }
  else { rear=(rear+1)%N;
         arr[rear]=value;
}
```

```
dequeue()
{ ~~front=(front+1)%N;~~
      int x=0;
  if(isempty())
      return;
  else if(front==rear)
  { x=arr[front];
    front=rear=-1;
  } else { x=arr[front];
           front++;
         } return x;
}
```

ME- RAUSHAN KUMAR
: 2221139
2203751

Assignment -1

42 ①

Q) What are sparse matrices? Discuss the type of sparse matrices. Mention the efficient method to store sparse matrices in a computer system.

Ans ⇒ A sparse matrix is a two-dimensional data object made of m rows and n columns, therefore having total m×n values. If most of the elements of the matrix have ~~and~~ 0 value, then it is called a sparse matrix.

Ex ⇒

$$\begin{bmatrix} 0 & 0 & 3 & 4 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 \end{bmatrix}_{m \times n}$$

Types of sparse matrices

1. Lower triangular sparse matrix
2. Upper triangular sparse matrix
3. Tri- diagonal matrix

1. Lower Triangular Matrix : In a lower triangular sparse matrix, all elements above the main diagonal have a zero value. This type of sparse matrix is also known as a lower triangular matrix. If you see its pictorial representation, then you find that all the elements having non-zero value are appear below the diagonal.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 & 0 \\ 1 & 4 & 3 & 0 & 0 \\ 9 & 8 & 7 & 1 & 0 \\ 1 & 2 & 7 & 8 & 9 \end{bmatrix}_{5 \times 5}$$

② Upper Triangular Matrix: In the upper triangular sparse matrix, all elements below the main diagonal have a zero value. This type of sparse matrix is also known as an upper triangular matrix.

$$\begin{bmatrix} 1 & 1 & 2 & 5 & 8 \\ 0 & 2 & 8 & 9 & 7 \\ 0 & 0 & 3 & 7 & 2 \\ 0 & 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 0 & 9 \end{bmatrix} \quad 5\times5$$

③ Tri-diagonal matrix: Tri-diagonal matrix is also another type of a sparse matrix, where elements with a non-zero value appear only on the diagonal or immediately below or above the diagonal.

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 5 & 2 & 8 & 0 & 0 \\ 0 & 8 & 3 & 2 & 0 \\ 0 & 0 & 4 & 1 & 5 \\ 0 & 0 & 0 & 7 & 9 \end{bmatrix} \quad 5\times5$$

The most efficient method to store sparse matrices in a computer system is to use a compressed storage formate. Compressed storage formats store only the non-zero elements of the matrix, along with additional information to allow for efficient access to these elements.

In CSR formate the matrix is stored as three arrays:

- Values Array: Store the non-zero values of the matrix row-wise.

- Column Indices Array: Store the column indices corresponding to each non-zero value in the same order as the values array.

• Row pointers Array: Store the index positions in the values and column indices arrays where each row starts.

Example:

$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 1 & 0 & 0 & 0 & 2 \\ 1 & 0 & 3 & 4 & 0 & 0 \\ 2 & 5 & 0 & 0 & 6 & 0 \\ 3 & 0 & 0 & 7 & 0 & 8 \end{array}$$

• Values array (A): [1, 2, 3, 4, 5, 6, 7, 8]

• Column Indices array: [0, 4, 1, 2, 0, 3, 2, 4]
0 0 1 1 2 2 3 3

• Row pointers array: ~~[0, 2, 4, 7, 8]~~ [0 0 1 1 2 2 3 3]

Q2. Describe the implementation of priority queue.

ans ⇒ There are two way to implement priority queue.

(i) one-way list implementation

(ii) 2D-array implementation

A priority queue is a collection of elements such that each element has been assigned a priority and such that the order in which elements are deleted and processed comes from the following rules.

(i) An element of higher priority is processed before any element of lower priority.

(ii) Two elements with the same priority are processed according to the order in which they were added to the queue.

One-way List representation of a priority queue
* space-efficient
* not-time efficient

(a) Each node in the list will contain three items of information: an information field INFO, a priority number PRN and a link number LINK.

(b) A node x precedes a node y in the list (1) when x has higher priority than y or (2) when both have the same priority but x was added to the list before y.

This means that the order in the one-way list corresponds to the order of the priority queue.

START → AAA | 1 → BBB | 2 → CCC | 2 → DDD | 4 → EEE | 4 → FFF | 4 → GGG | 5 | X

2D-Array implementation of a priority queue.

Another way to maintain a priority queue in memory is to use a separate queue for each level of priority. Each such queue will appear in its own circular array and must have its own pair of pointers, FRONT and REAR. In fact, if each queue is allocated the same amount of space a two dimensional array QUEUE can be used instead of the linear arrays.

	FRONT	REAR
1	2	2
2	1	3
3	0	0
4	5	1
5	4	4

	1	2	3	4	5	6
1		AAA				
2	BBB	CCC	XXX			
3						
4	FFF					
5				GGG	DDD	EEE

Comparision

One way list representation	array presentation of priority queue
(i) less time efficient	(i) more time efficient
(ii) more space-efficient than array	(ii) less space-efficient than one way
(iii) overflow occurs only when the total number of elements exceeds the total capacity.	(iii) overflow occurs when the number of elements in any single priority level exceeds the capacity for that level.

37

NAME : RAUSHAN KUMAR

CRN : 2221139

URN : 2203751

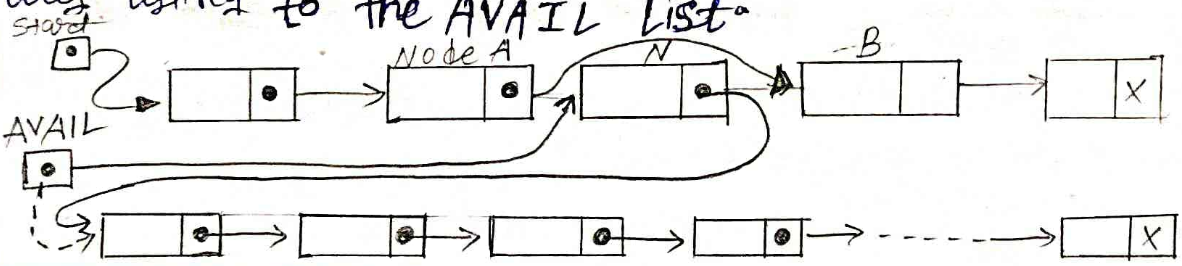
Branch : IT

Section : B

DSA Assignment - 2

(1Q) what is AVAIL list? Describe the method of GARBAGE Collection which is performed by operating system for managing free-storage list.

ans: AVAIL list is a free storage list maintained by the operating system to keep track of the available memory. It is a linked list of memory blocks that are not currently in use. When a process requests memory, the operating system allocates a block of memory from the AVAIL List. When a process terminates, the operating system returns the memory block it was using to the AVAIL List.



(GC) Two steps process:

Step: 1 GC will sequentially visit all nodes in memory, and mark all nodes which are being used in program.

Step 2: It will collect all unmarked nodes and place them in free storage area.

2

5

7

1

Garbage collection: Garbage collection is an automat[ic] dynamic memory management that identifies dead mem[ory] blocks and reallocate storage for reuse.

other words:

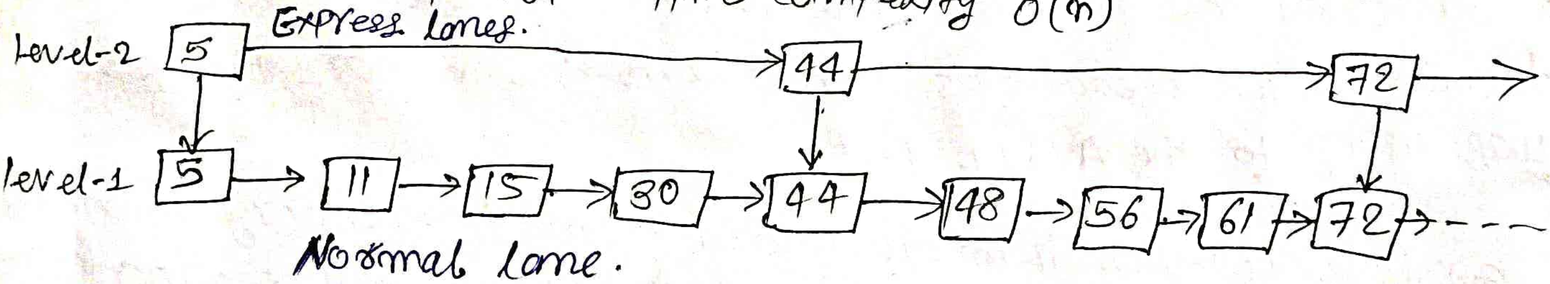
The operating system may periodically collect all the deleted space onto the free-storage list. This machanism is called Garbage collection.

Q2. Explain the implementation of skip list.

ans: Skip list is a data structure that is used for storing a sorted list of items with the help of hierachy of linked lists.

* what do you mean by skip list.

The Skip list data structure skips over many of the items of the full list in one step, that's why it is known as skip list. Time complexity O(n)



- It can consists many levels
- All keys appear in level
- If key appear in level i then it also appears in all levels below i.

A SkipNode has: → val

→ next

⇒ down

val → next

↓

down

Algorithm : Searching in skip list

```
SkipNode* search (int key)
    curr = head
    while curr != NULL
        while curr->right != NULL && curr->right->val <= key
            curr = curr->right
        if curr->val == key
            break;
        curr = curr->down
    return curr
```

Advantage of skip list

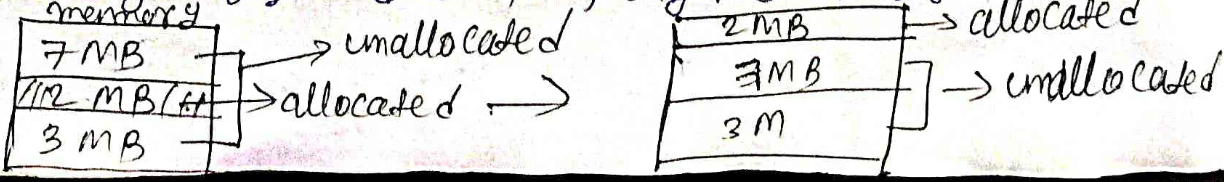
- The time complexity of search can become $O(\log n)$ in average case with $O(n)$ extra space.

thus . It improve the search.

Disadvantage of skip list

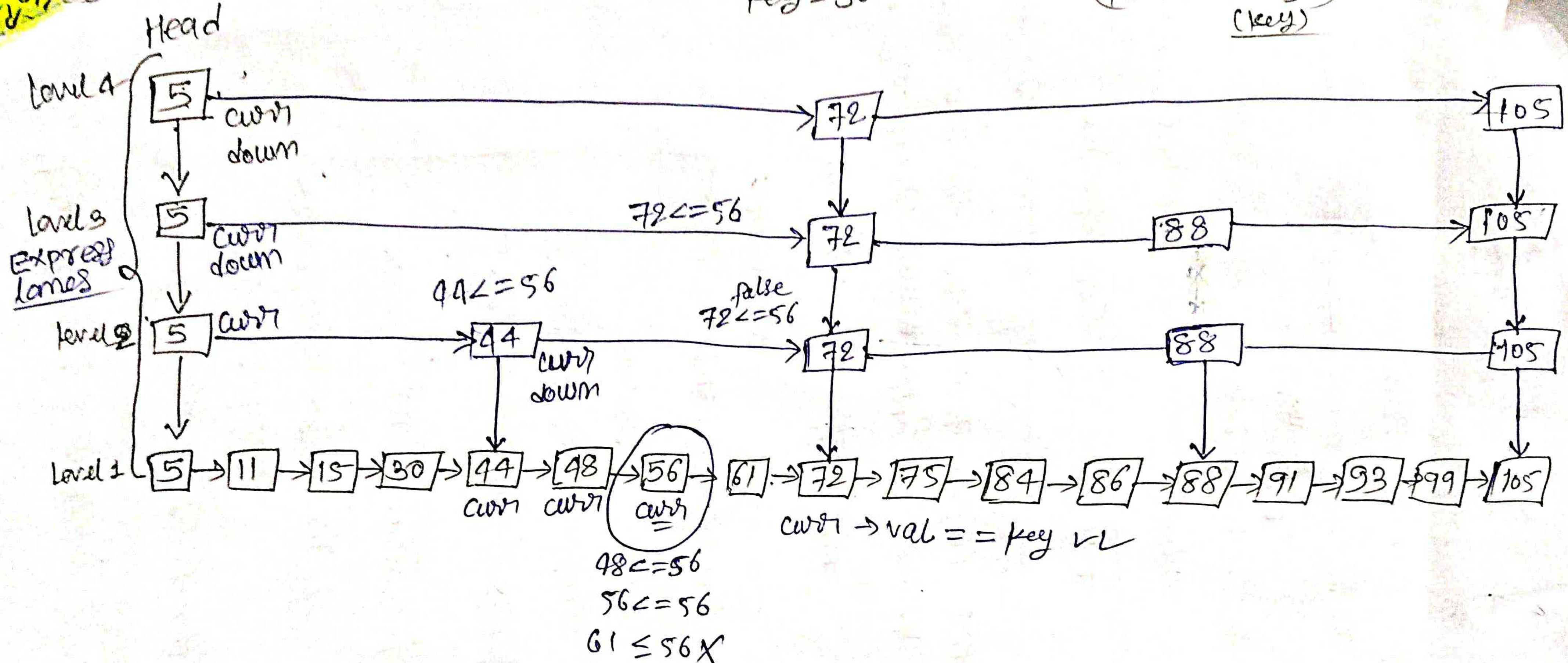
- It require more memory.

→ Garbage compaction: Garbage compaction is the process of sepration of the allocated memory and ~~data~~ ~~deallocated~~ unallocated memory. This process is often with garbage collection, It helps to recover memory lost due to fragmentations.



Key = 56

(To search 56)
(key)



(pyq) illustrate the efficient way of storing sparse matrices in memory.

ans →

$$
\begin{array}{c|cccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 \\
1 & 0 & 0 & 4 & 0 & 0 & 0 & 6 & 0 \\
2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 8 & 0 & 0 & 0 & 3 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\
6 & 0 & 0 & 10 & 0 & 7 & 0 & 0 & 0
\end{array}
\quad 7\times 8
$$

Two efficient way of storing sparse matrices

(i) Array representation

(ii) linked representation.

(1)

	0	1	2	3	4	5	6	7	8
row	0	1	1	2	4	4	5	6	6
col	4	2	6	0	1	5	7	2	4
value	5	4	6	1	8	3	2	10	7

(2) Link list rep:

row	col	value	next node address

[0 | 4 | 5 | •] → [1 | 2 | 4 | •] → [1 | 6 | 6 | •] → [2 | 0 | 1 | •] → [4 | 1 | 8 | •] → [4 | 5 | 3 | •] → [5 | 7 | 2 | •] → [6 | 2 | 10 | •] → [6 | 4 | 7 | X]

Assignment - 3

①

34

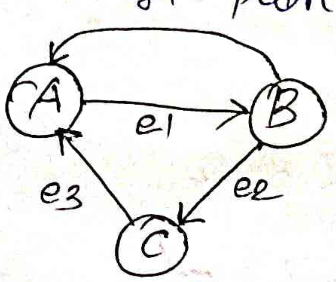
NAME - RAUSHAN KUMAR
CRN : 2221139
URN : 2203751
Branch : IT - B2

Raushan Kumar

Q.1 (a) what is a path matrix (or reachability matrix)?

ans: The matrix contains information about the path exist or not between every pair of vertices or A path matrix is a matrix that is used to determine the shortest path between two nodes in graph.

Ex:-



Adjacency matrix

$$Q = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \end{array}$$

$$Q^2 = Q \times Q \quad \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \end{array} \times \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \end{array} = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$Q^3 = Q \times Q \times Q \quad \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{array} \times \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

$$B = Q + Q^2 + Q^3 \quad \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 2 & 2 & 1 \\ 3 & 2 & 2 \\ 2 & 1 & 1 \end{bmatrix} \end{array}$$

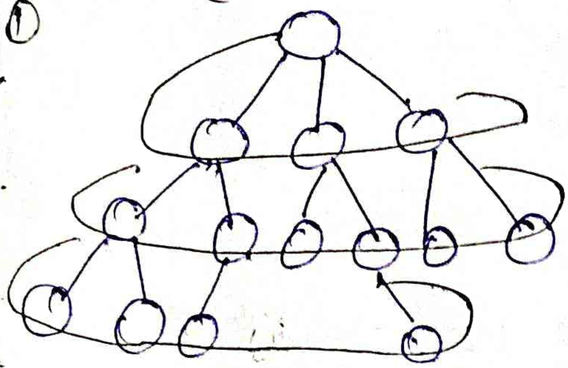
Now we can find path matrix using matrix B is:

$$P = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & \end{array} \\ \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \end{array}$$

(b) Differentiate between the graph traversal technique of BFS (Breath first search) & DFS (Depth first search)

(BFS)

(i)



(ii) Need queue data structure.

(iii) Need more space.

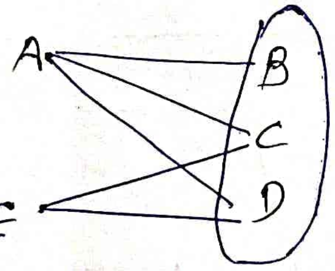
(iv) It focous all the neighbours of a ~~node~~ Node at a time

(v) Time complexity is O(V+E)

(vi) Applications: Shortest path finding problem

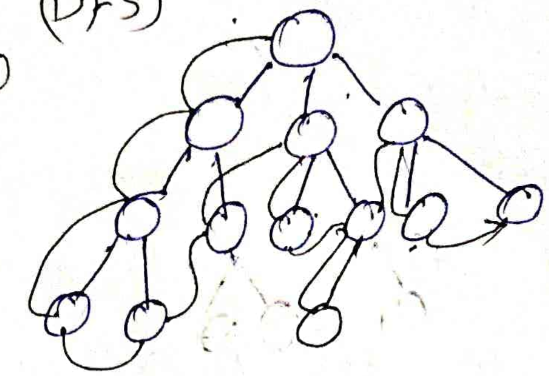
(b) To check wheather the graph is Bipartide or not.

(vii)



(DFS)

(i)



(ii) Need stack data structure.

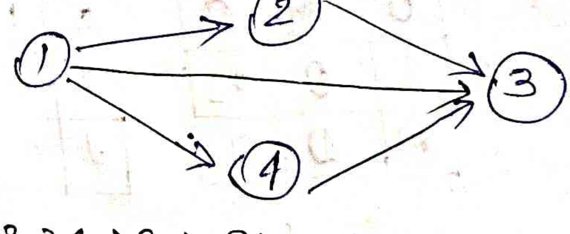
(iii) Need less space than BFS.

(iv) ~~we~~ It focous on childs of one Node.

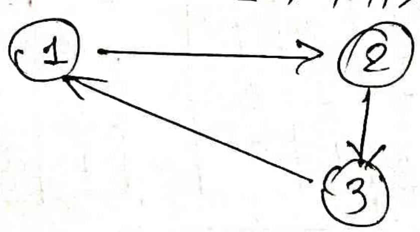
(v) Time complexity is O(V+E)

(vi) DFS is used ~~in~~ topological sorting process, scheduling, cycle detection.

(vii)



1→2→4→3→ [1, 2, 4, 3]



Q2. Compare the techniques and complexities of various sorting algorithms. → buble ~~sort~~ sort shell sort.
→ insertion sort.
→ Selection sort
→ quick sort
→ merge sort
→ heap sort

Comparison of sorting ÷ Time complexity

Sorting Techniques	Best case	Average case	Worst case
1. Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
2. Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
3. Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
4. Quick sort	$O(n\log n)$	$O(n\log n)$	$O(n^2)$
5. Merge sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
6. Shell sort	$O(n)$	$O(n(\log n)^2)$	$O(n(\log n)^2)$
7. Heap sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$

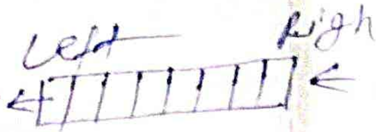
Sorting techniques	Space complexity	Stability	Auxiliary space	Locality of reference
1. Bubble sort	$O(1)$	Yes	Not required	Poor
2. Insertion sort	$O(1)$	Yes	Not required	Good
3. Selection sort	$O(1)$	No	Not required	Fair
4. Quick sort	$O(\log n)$	No	Not required	Good
5. Merge sort	$O(n)$	Yes	Required	Poor
6. Heap sort	$O(1)$	No	Not required	Fair
7. Shell sort	$O(1)$	No	Not required	Good.

Doublended Queue (Dequeue)

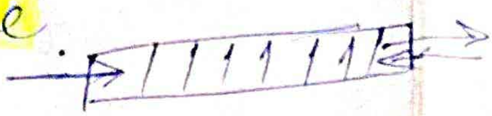
int a[5];

(input only in one side add both side)

input → Restricted Dequeue



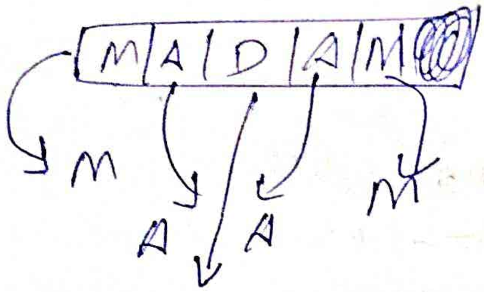
output → restricted Dequeue



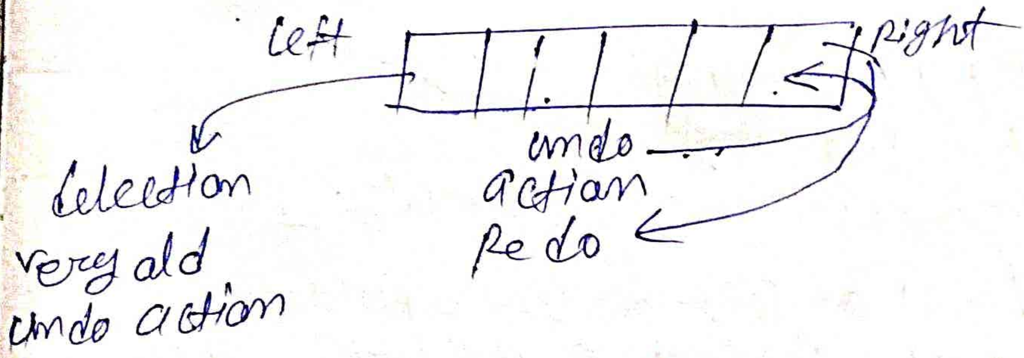
(output only in one side dele both side)

→ Example of input Restricted Dequeue

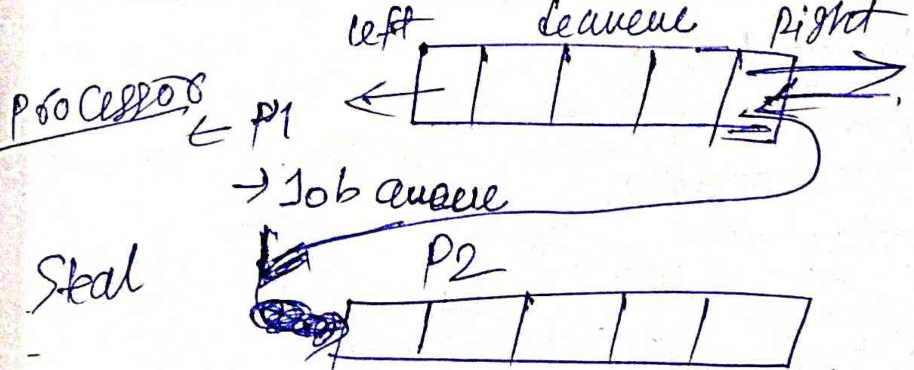
(a) Palindrome checking



(b) undo and Redo operations -



(c) A - Steal job scheduling algorithm



Explain output restricted, draw the diagram input restricted.

Implementation of dequeue

insertion at left where N = size of array

left = (left − 1 + N) % N

Dequeue

Delection from left

left = (left + 1 + N) % N

Insertion at Right

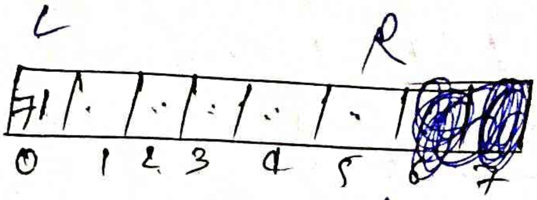
right = (righ + 1) % N

· Delection from right

right = (right − 1 + N) % N

Implementing of dequeue : using a circular array

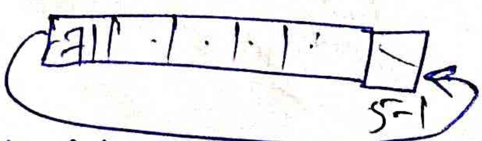
N = 6



int left = −1, right = −1; [empty]

① Insert −71 at left ⇒ left = Right = 0
  or 71 at right a[left] = 71

② ~~insert 80 at right~~
  insert 69 at left.



left = 0

= (0 − 1 + 6) % 6 = 5

= a[left] = 69

right = 0

(0+1) % 6 =

right = 1

a[right] = 80

(4) insert 66 at left

71	69	87		66	

(2-1+6) % 6.

10 % 6 = 4

a[left] = 66

(5) insert 87 at right side

((1+1)) % 6 = 2 % 6 = 2

if(2 == left)

a[right] = 87

(6) delect at right

(old right - 1 + N) % N

(2-1+6) % 6 = 7 % 6 = 1

(7) delection at left

old left = 4

((left+1+N) % N

(4+1+6) % 6 = 11 % 6 = 5

~~left = 5~~

Q why is circular queue better than linear queue?

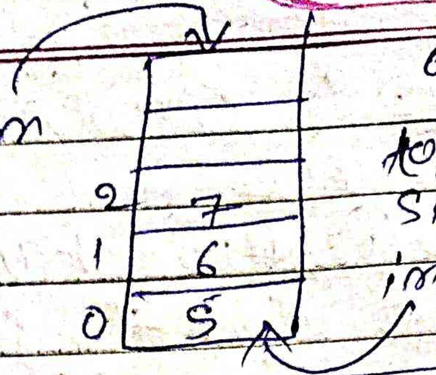
ans ⇒ (i) Efficient use of memory: In a circular queue, when the rear pointer reaches the end of the queue. It wraps around to the begining, which allows for efficient use of memory compared to a linear queue.

(ii) Easier for insertion-deletion: In the circular queue, if the queue is not fully occupied then the elements can be inserted easily in the vacant locations. but in linear queue insertion not possible. once the rear reaches the last index.

(iii) Better performance: circular queue offers better performance in situations where data is frequently added and removed from the queue as compared to a linear queue.

(iv) Reduced overflow risk: Circular queues are less likely to overflow than ~~linear~~ linear queues. This is because circular queues can use all of the available space even if the queues in not full.

Stack algorithm:



arr[5];
top = -1
size = 4;

```
is_empty()
{ if(top==-1)
{ return true;
} else
{
 return false;
}
}
```

```
is-full()
{ if(top==size)
 { return true;
 }
else {
 return false;
 }
}
```

```
push( int value)
{ if(is-full())
{ cout<<" overload";
}
else { top++;
 arr[top]= value;
 }
}
```

```
pop( )
{ if(is-empty())
{ cout<<" underflow";
}
else { int popV= arr[top];
 arr[top]= 0;
 top--;
 return popV;
 }
}
```

To create a linear linked list

Node

| info | → | 4 | x |

| 3 | → | 4 | x |

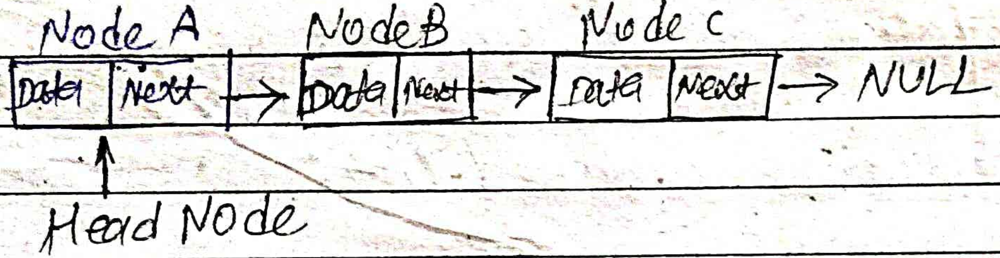
```
#include<iostream>
# include< stdlib.h>
using namespace std;
struct Node
{ int data;
   Node *next;
};
 Node * head = NULL;


void insertinLL( int val )
 { Node * newnode =(Node * )malloc(sizeof(Node))
newnod -> data = val;
newnode-> next = NULL;
Node * last = head;
 if ( last == NULL)
 {
   ~~newnode~~ head = newnode;
        return;
 }
```

```
else { while ( last -> next != NULL )
    { last = last -> Next;
    }
    last -> Next = newnode;
      return;
    }
}
struct node
{  int data;
   Node * next;
};
 Node * head = NULL;
void insertionLL(int val)
{ Node * newnode = (Node*) mallo
```

* What is Linked List: A Linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.
. The elements in a linked list are linked using pointers (entity that point to the next element)

other words
A linked list consists of nodes where each node contains a data field and a reference (link) to the next node in the list.



Linked list consists of nodes where each node contains a data field and a reference (link) to the next node in the list.

(4) Linked List vs Arrays

Advantages of Linked List over arrays:

(i) Dynamic size

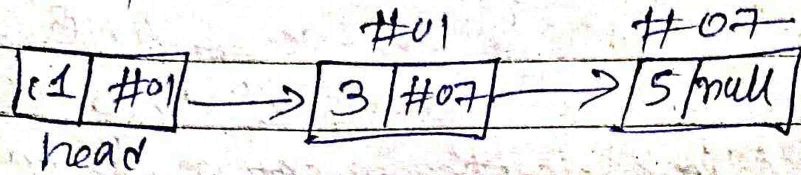
(ii) Ease of insertion / deletion

Disadvantages of Linked list over Array:

(i) Random access is not allowed. we have to access elements sequentially starting from the first node.

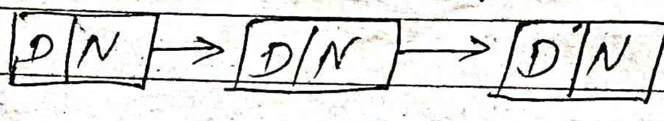
② Extra memory space for a pointer is required with each element of the list.

③ Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.



Operations of linked list :-

1. Traversing a linked list:



2. Append a new node (to the end) of a list.

3. Prepend a new node (to start)

④ Inserting a new node to a specific position in the list.

⑤ Deleting a node from the list.

⑥ Updating a node in the list.

Types of linked list

D = Data ⇒ P = Previous N = Next

1. singly linked list → head [D|N] → [D|N] → [D|N]
2. Doubly linked list → head [P|D|N] ⇄ [P|D|N] ⇄ [P|D|N]
3. Circular linked list → head [D|N] → [D|N] → [D|N]

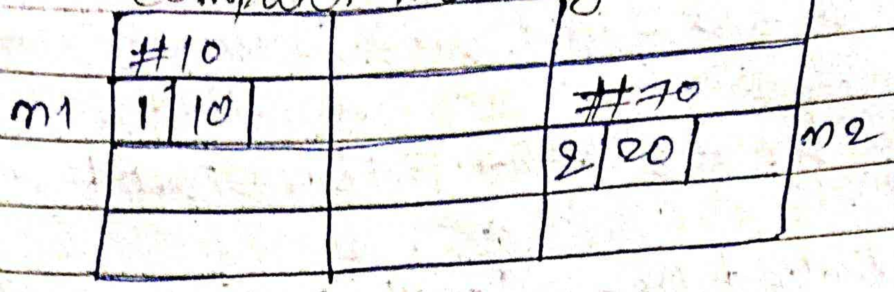
Some application of linked List

- Linked lists can be used to implement stack, queue,
- Linked Lists can also be used to implement Graphs,
- Implemaenting Hast Tables
- undo functionality in photoshop or word

Creating a node

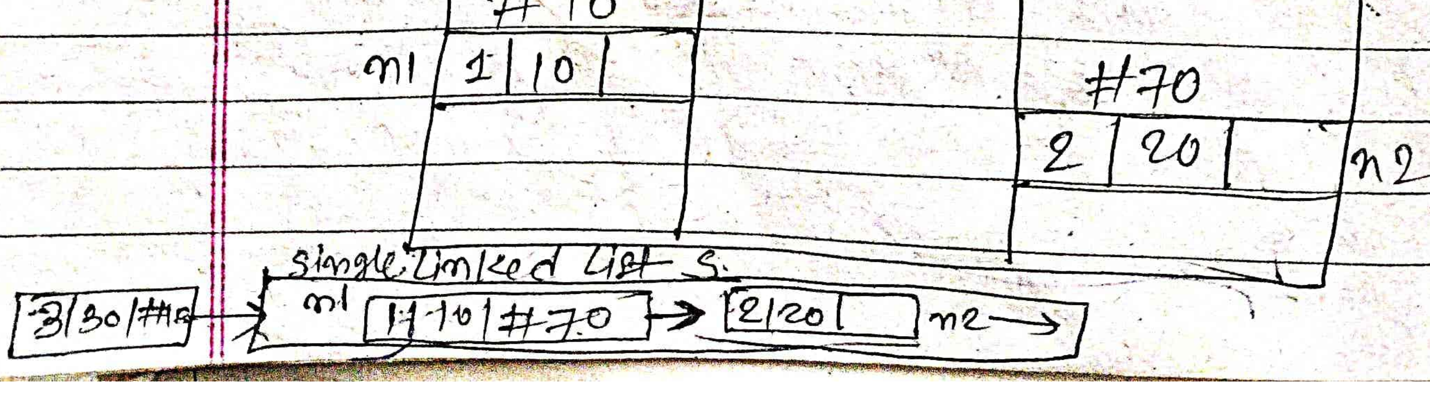
```
class Node {
public: int key;
        int data;
        Node* next;
        Node()
        { key=0;
          data=0;
          next = NULL;
        }
Node (int k, int d)
{ key = k;
  data = d;
  next = NULL;
}
};
int main()
{ Node n1(1,10);
  Node n2(2,20);
}
```

Computer Memory




```
Class SinglyLinkedList
{ public:
    Node * head;
SinglyLinkedList (Node * h)
void prependNode (Node * h)
void appendNode (key)
void insertNode (key)
Void deleteNode (key)
Void updateNode (key)
}
    int main ()
  { Node n1 (1,10);
    Node n2 (2, 20);
    Node n3 (3, 30);
  SinglyLinkedList S(&n1);
  S. appendNote (&n2);
  S. prependNote (&n3);
}
```

Computer Memory



```
#include<iostream>
#include<stdlib.h>
using namespace std;
 struct Node
1 {
  int data;
  Node *next;
 };
Node * head=NULL;
Void insertinLL (int val)
{
 Node *newnode =(Node *)malloc(sizeof(Node));
 newnode -> data =val;
newnode -> next =NULL;
 Node * last = head;
  if (last == NULL)
  { head =newnode;
   return;
2 }
 else
 { while (last ->next != NULL)
  {
    last = last ->next;
  }
  last ->next =newnode;
  return;
 }
}
```

```
void display ()
{ Node * ptr = head;
  if (ptr == NULL)
{ cout<<"No elements are there in
  Linked List";
}
 else
3 { cout<<"Elements in linked list are:\n";
  while (ptr != NULL)
  {
   cout<< ptr->data;
   ptr = ptr->next;
  }
 }
}
void deletedata (int todel)
{
 Node * temp = head;
 Node * prev;
while (temp != NULL && temp->data != todel)
{
  prev = temp;
4 temp = temp->next;
}
 if (temp == NULL)
{
cout<<"\n"<< todel <<"not found \n";
return;
}
```

```
else {
cout<<"\n deleting"<<todel<<" from the linked
  list \n";
  prev->next = temp->next;
  free(temp);
}
int main()
{ cout<<" inserting 3 in the linked list \n";
  insertinLL(3);
 cout<<" Inserting 4 in the linked list \n";
  insertinLL(4);
 cout<<" Inserting 55 in the linked list \n";
  insertinLL(55);
 cout<<" Inserting 184 in the linked list \n";
  insertinLL(184);
 cout<<" Inserting -120 in the linked list \n";
  insertinLL(-20);
 cout<<"\n currently the elements present in the
  linked list are:\n";
   display();
 cout<<"...................................\n";
  cout<<" delete 30";
  deletedata(30);
   cout<<" delete 55";
  deletedata(55);
   cout<<" delete -20";
  deletedata(-20);
   cout<<" delete 7";
    deletedata(7);
```

```
cout<<"..........................\n";
cout<<"\n Now, the elements present in the
    linked list are:\n";
        display();
    return 0;
}
```

Output

Inserting 3 in the linked list
" 4 " " "
" 55 " " "
" 184 ———
" -20 ———

currently the elements the elements present in the linked list are:
Elements in linked list are;
3
4
55
184
-20

....

delete 30
30 not found
delete 55
deleting 55 from the linked list
deleting -20
deleting -20 from the linked list
delete 7
7 not found

....

Now, the elements present in the linked list are:

Elements in linked list are:

3

4

184

~~[illegible]~~

Q Demonstrate with example an efficient method of storing a tridiagonal matrix in computer memory.

ans ⇒

$$A = \begin{pmatrix} a_{11} & & & & \\ a_{21} \rightarrow & a_{22} & & & \\ a_{31} \rightarrow & a_{32} \rightarrow & a_{33} & & \\ \cdots & \cdots & \cdots & & \\ \cdots & \cdots & \cdots & \cdots & \\ \vdots & & & & \\ a_{n1} \rightarrow & a_{n2} \rightarrow & a_{n3} \rightarrow & \cdots \rightarrow & a_{nn} \end{pmatrix}$$

$B[1] = a_{11}$, $B[2] = 21$, $B[3] = 22$, $B[4] = a_{31}$, ---

observe first B.

$$1+2+3+4+ \text{---} n = \boxed{\frac{n(n-1)}{2}}$$

$$\boxed{B[L] = a_{JK}}$$

L represent the no of elements in the list up to the a_{JK}.

$$1+2+3+ \text{---} +(J-1) = \frac{J(J-1)}{2}$$

element in the rows above

a_{jK} and K are elements in row J.

$$\boxed{L = \frac{J(J-1)}{2} + K}$$

* Analyze and compare the mechanisms to implement priority queue.

ans ⇒

Compare

	one way list impleme	2D array implementation
1	less time efficient	more time efficient
2	more space-efficient than array	less space efficient than one way.
3	overflow occurs only when the total number of elements exceeds the total capacity.	overflow occurs when the total number of element in any single priority level exceeds the capacity of that level.

one way → START [•] → [A \| 1 \| •] → [B \| 2 \| •] → [C \| 2 \| •] → [D \| 4 \| •] → [E \| 4 \| •] → [F \| 4 \| •] → [G \| 5 \| X]

array →

	front	Rear
1	2	2
2	1	2
3	0	0
4	5	1
5	4	4

	1	2	3	4	5	6
1		A				
2	B	C				
3						
4	F				D	E
5				G		

5X6

MST - 2

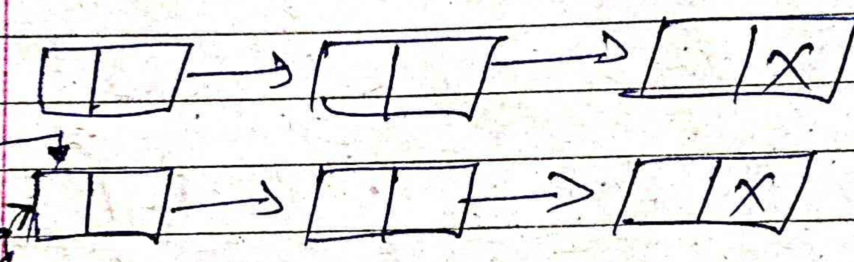
* Seauential representation and linked representation.

stack, queue

Implement stack and queue using:
(i) senauential representation (array) → stack, → queue
(ii) linked representation → stack, → queue

* linked representation of stack

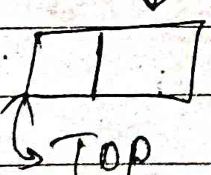
Structure Node
{ int data;
Node * Top;
};



top

Node * top;

insertion & deletion with only (TOP)



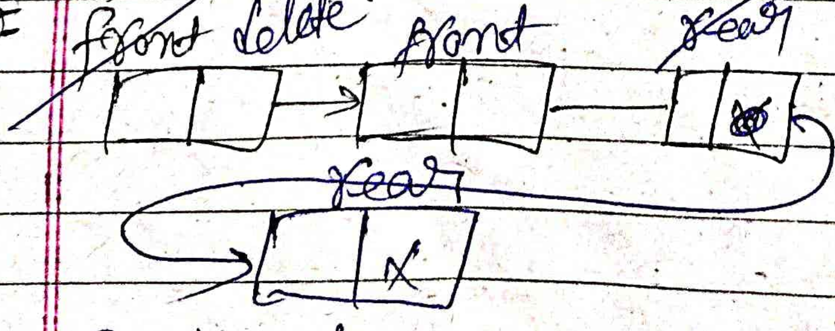
TOP

for delete → ;; Top → next ;
TOP

* for queue

struct Node
{ int data;
Node * front;
Node * rear;
};

~~front~~ delete

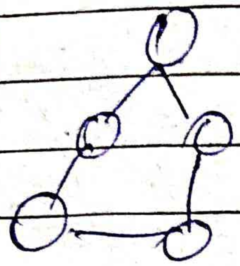


front ~~rear~~ ~~rear~~

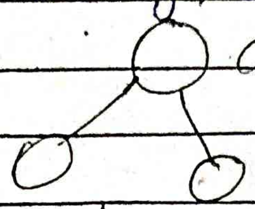
for insertion

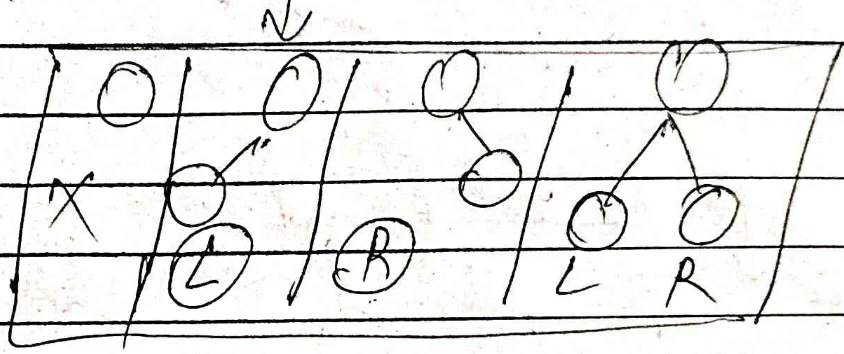
Trees (non-linear data structure)

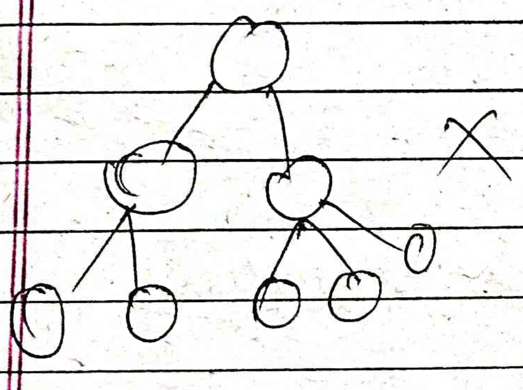
Hierarchial relationship

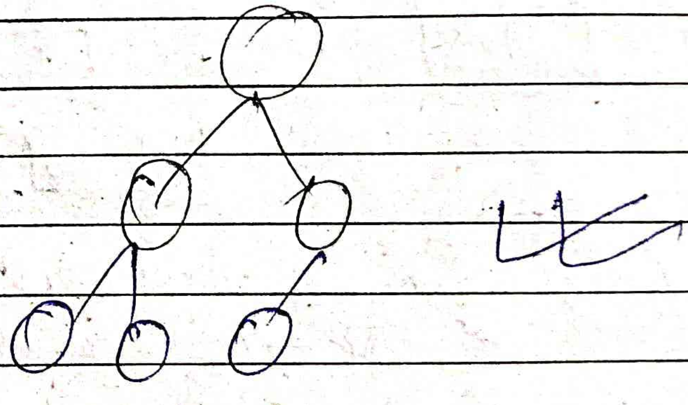


Binary Tree









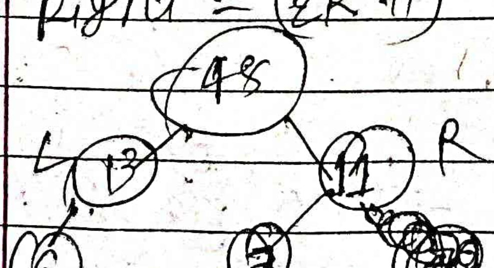
* Representation of Trees (Array)

Array

Node → K^{th}

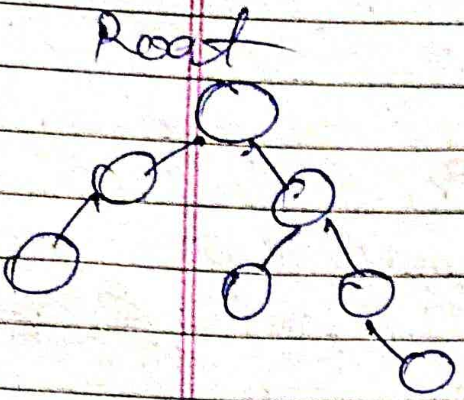
left = $2K^{th}$

Right = $(2K+1)^{th}$

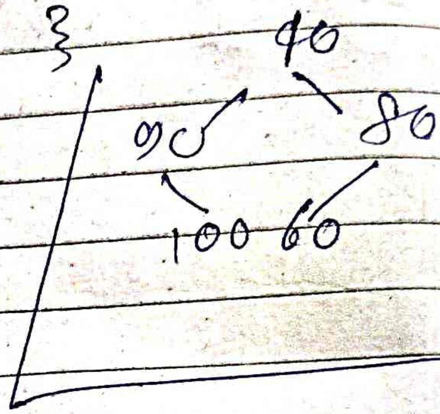


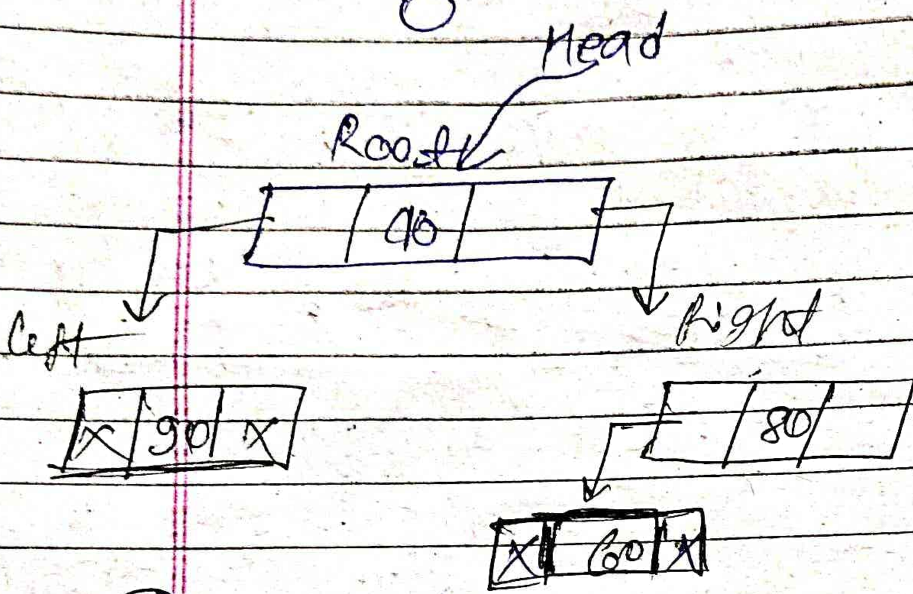
1	48	K = 1
2	13	2K = 2*1
3	11	2K+1 = 2*1+1
4	6	2K =
5		2K+1
6	7	
7		

* Representation of Trees using linked list



struct Node
{ int data;
Node * left;
Node * right;
}

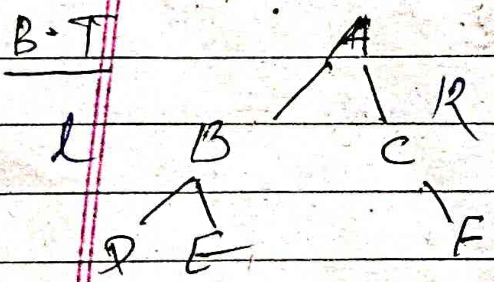




* Tree traversal technique.

* pre-order
* In - order
* post-order

B.T



* pre-order → Root → left → Right
* In - " → left Root Right

Recursive in nature.

* post - " → left right Root

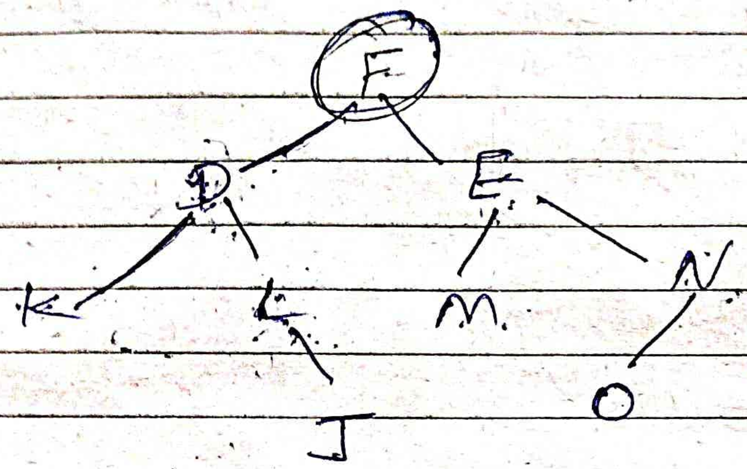
2 Mark → B Tree Apply this pre-In post order and traverse

pre order: A | BDE | CF = ABDECF

In-order: DBE | A | CF = DBEACF

post order: DEB | FC | A = DEBFCA

Q



Pre: F | DKLJ | EMNO = FDKLJEMNO

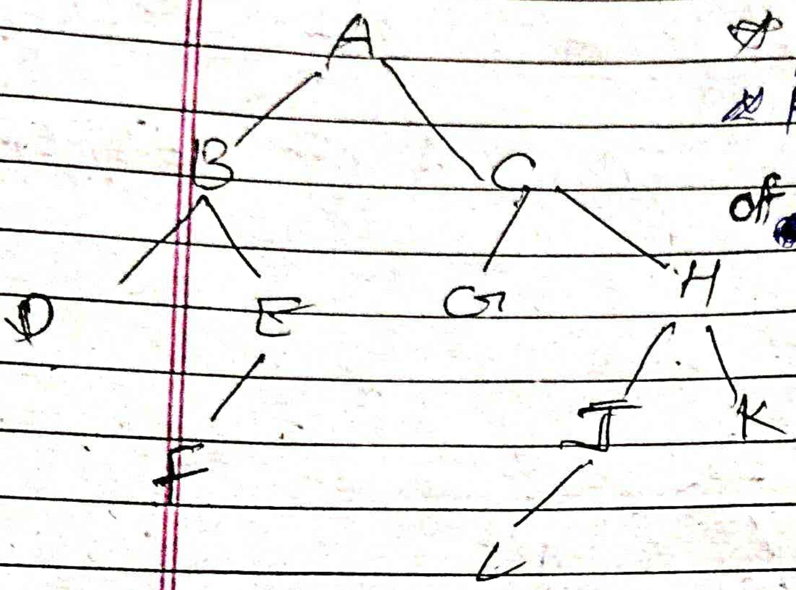
Inorder: KDLJ | F | MEON = KDLJFMEON

Post: Left	Right	Root
KJLD	MONE	F

= KJLDMONEF

by default in order

Threaded Binary tree



- Avoid using NULL pointer
- Rather than using Null ptr. store the address of the next node in the traversal.

Threaded

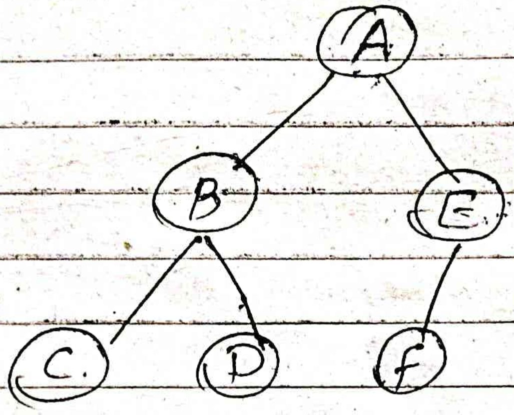
- one way threaded (only previous store का address)
- two way threaded (previous & next both का address store करते)

PM

PYQ's Apply two-way threading with header node in a tree.

ans→ Left pointer of the first node and right pointer of last node will contain NULL value.

→ two way threading stores previous as well as next address. It avoid using Null ptr Rather than using Null pointer stores the address of the next Node in the traversal.



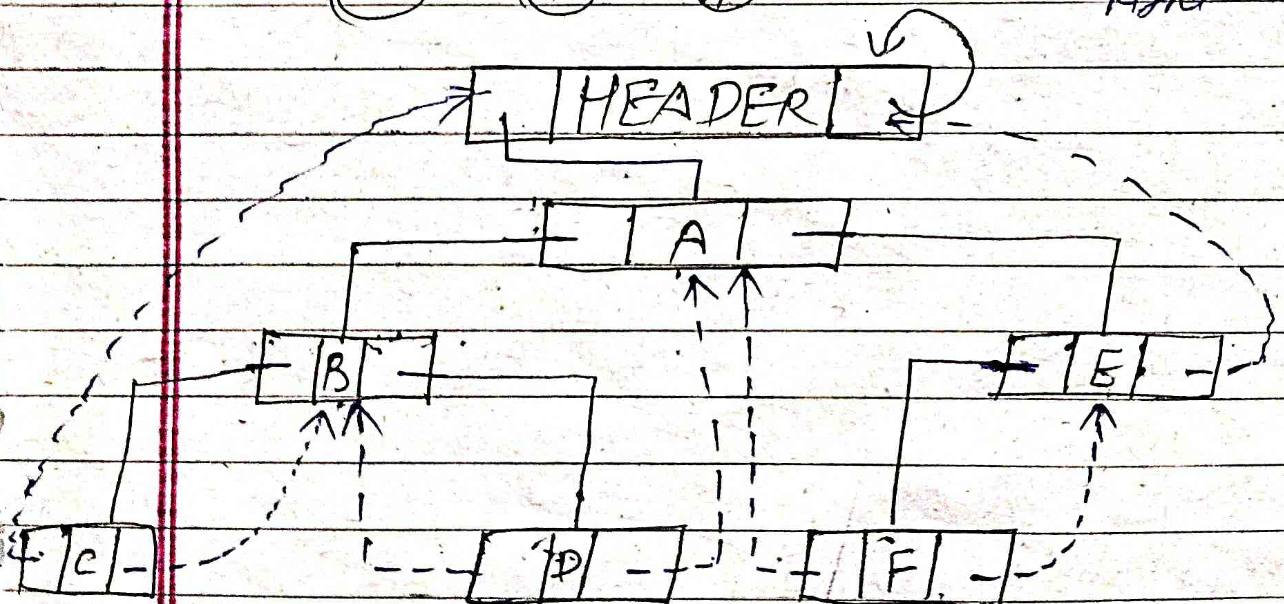
Inorder

Left	Root	Right
CBD	A	FE

left

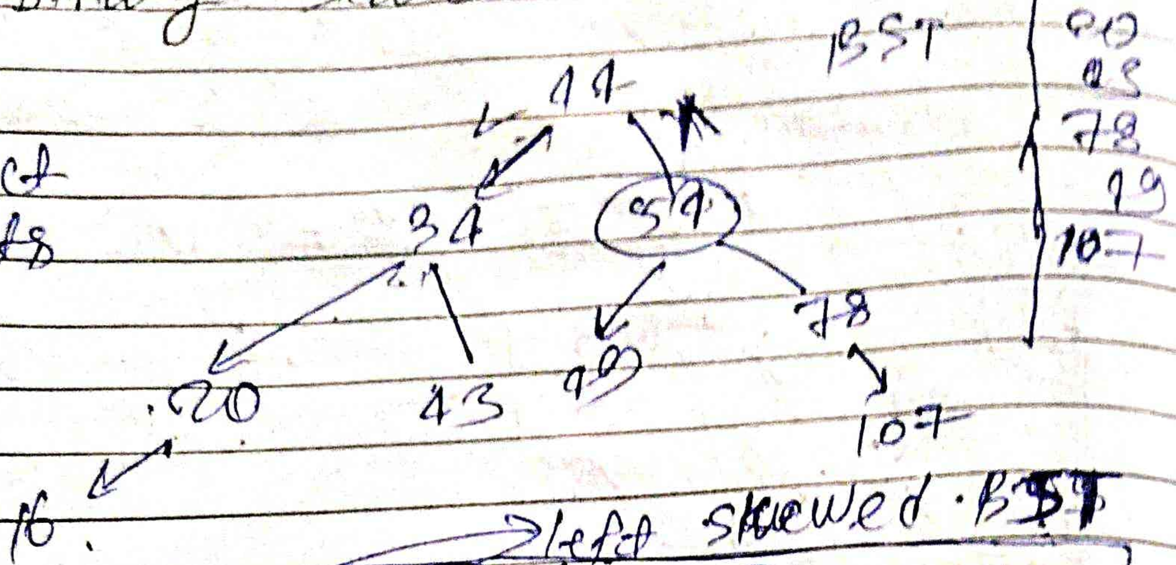
≈ CBDAFE

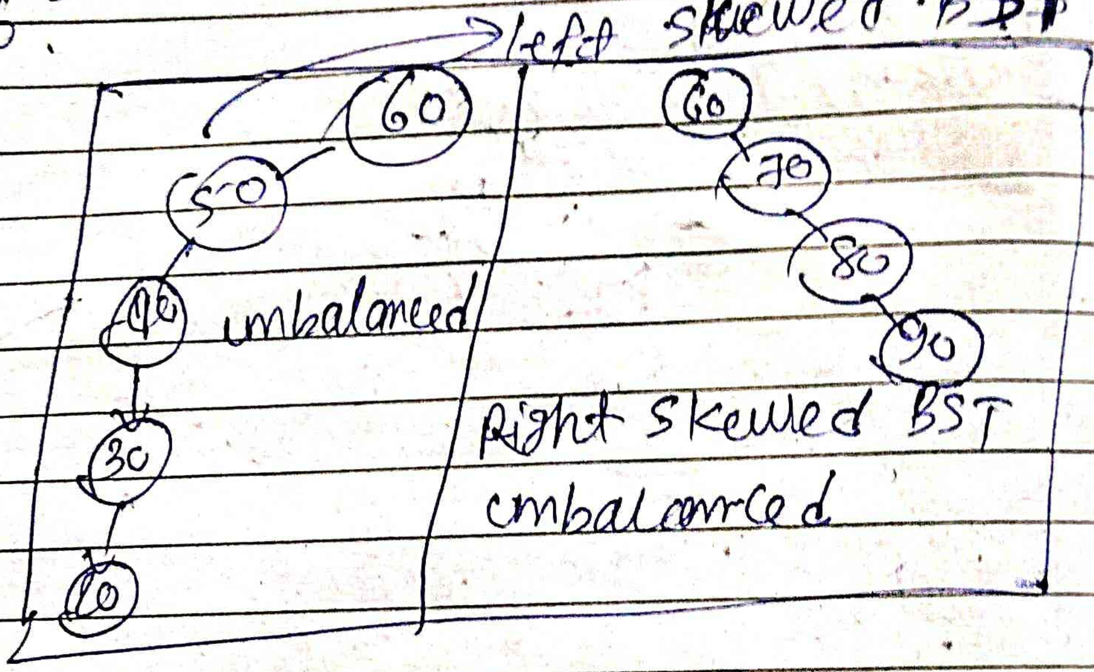
Right



* Binary Search Tree (BST)

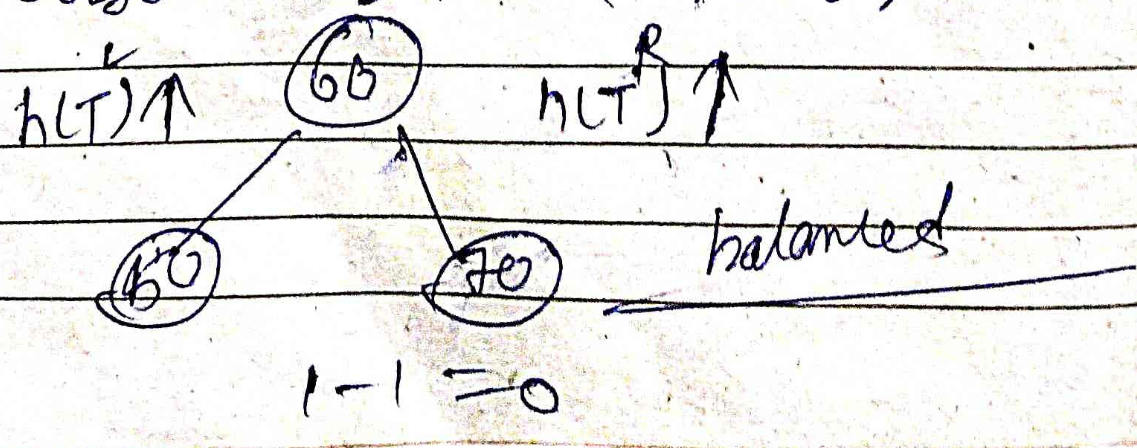
distinct elements

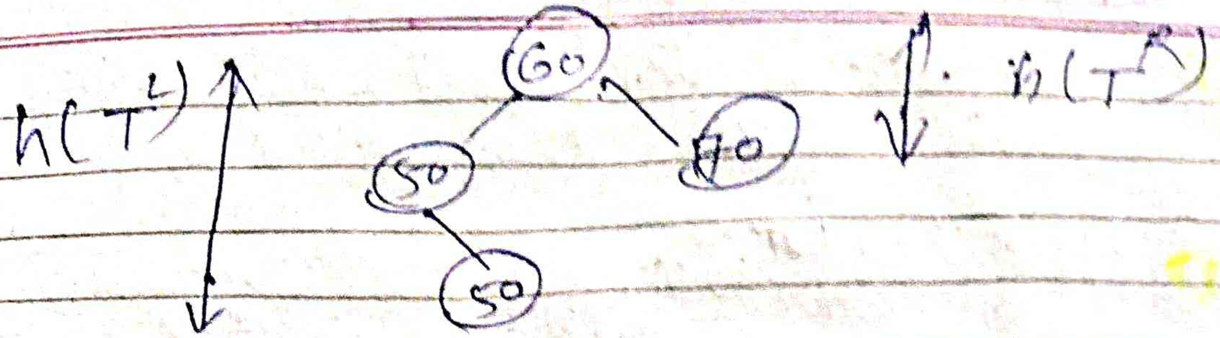




* Balancing of BST.

AVL Tree: self-balancing BST.
Adelson Velskii & Landis.

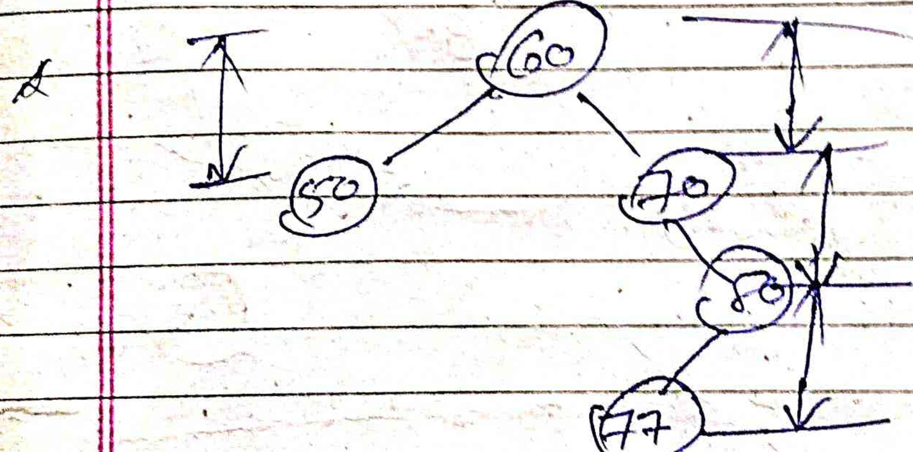




$h(T^L) - h(T^R) =$
$2 - 1 = 1$

AVL Tree

$\boxed{h(T^L) - h(T^R) \le = 1}$ | left subtree



$h(T^L) - h(T^R) \le = 0$
$1 - 3 = |-2| = 2$
~~not~~ balanced

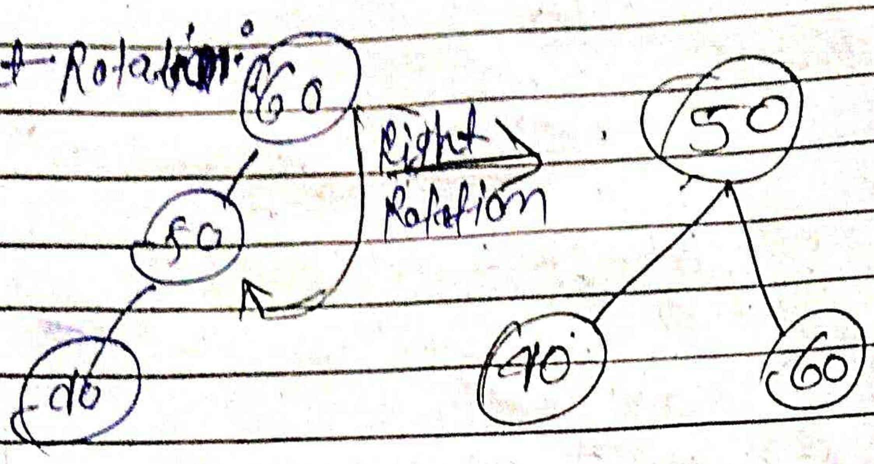
How can you say that a skewed does not give requirement AVL Tree

Q.1) Explain the way by which an unbalanced BTree on unbalanced of an AVL Tree. (Imp)

Rotations in BST to make it an AVL Tree ie (a balanced tree)

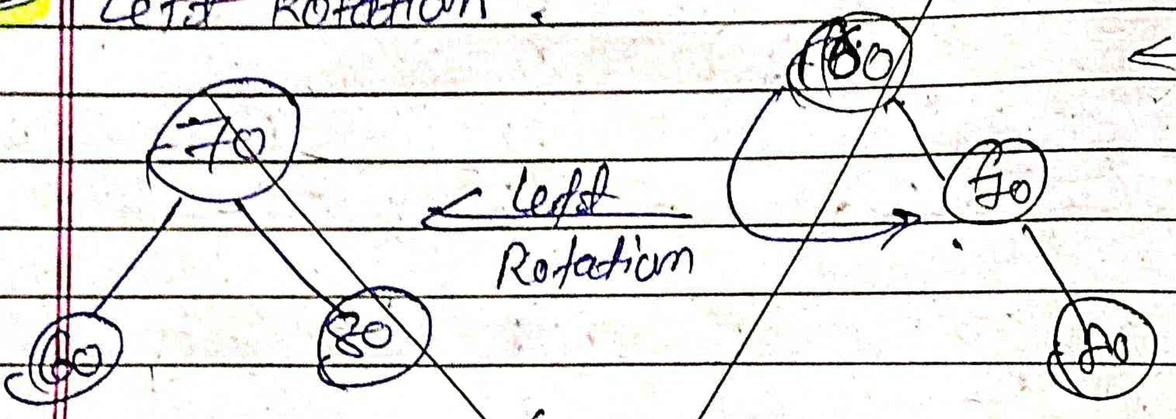
ans →

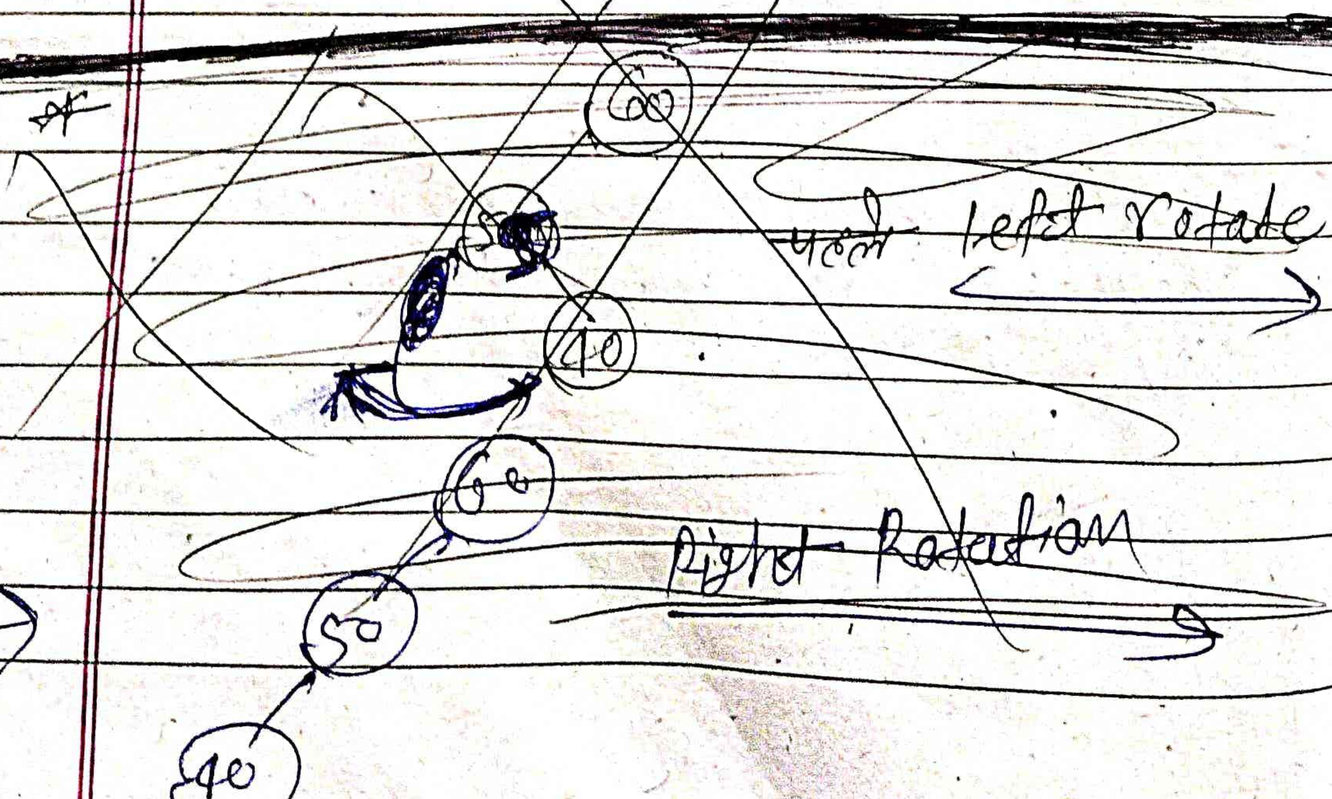
① Right Rotation:

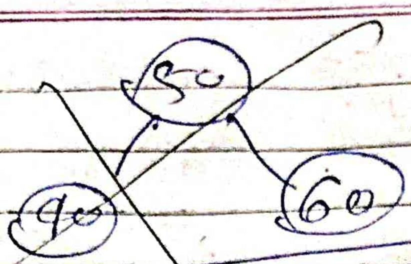


ये सही है।

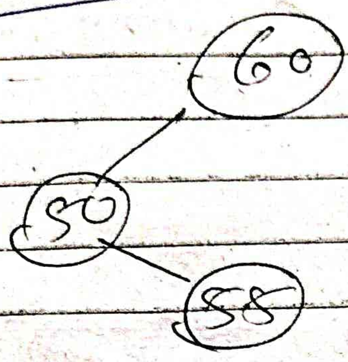
② Left Rotation:



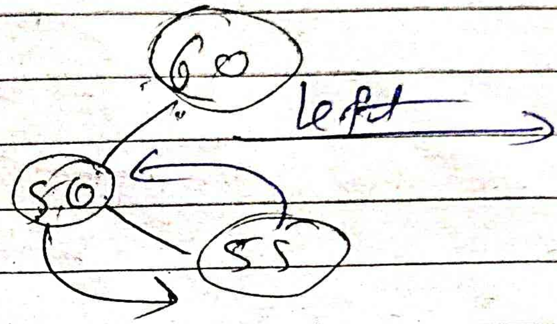


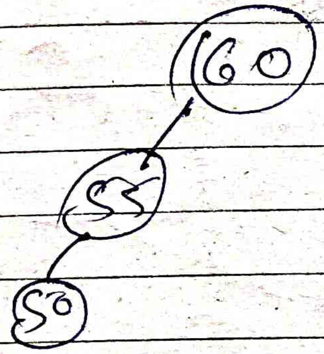


③ Left Right Rotation:

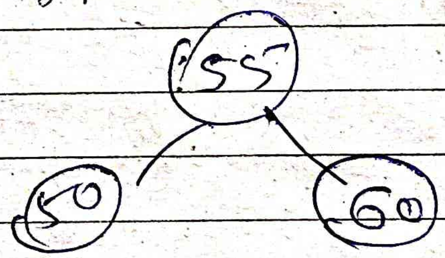


Left Rotation →

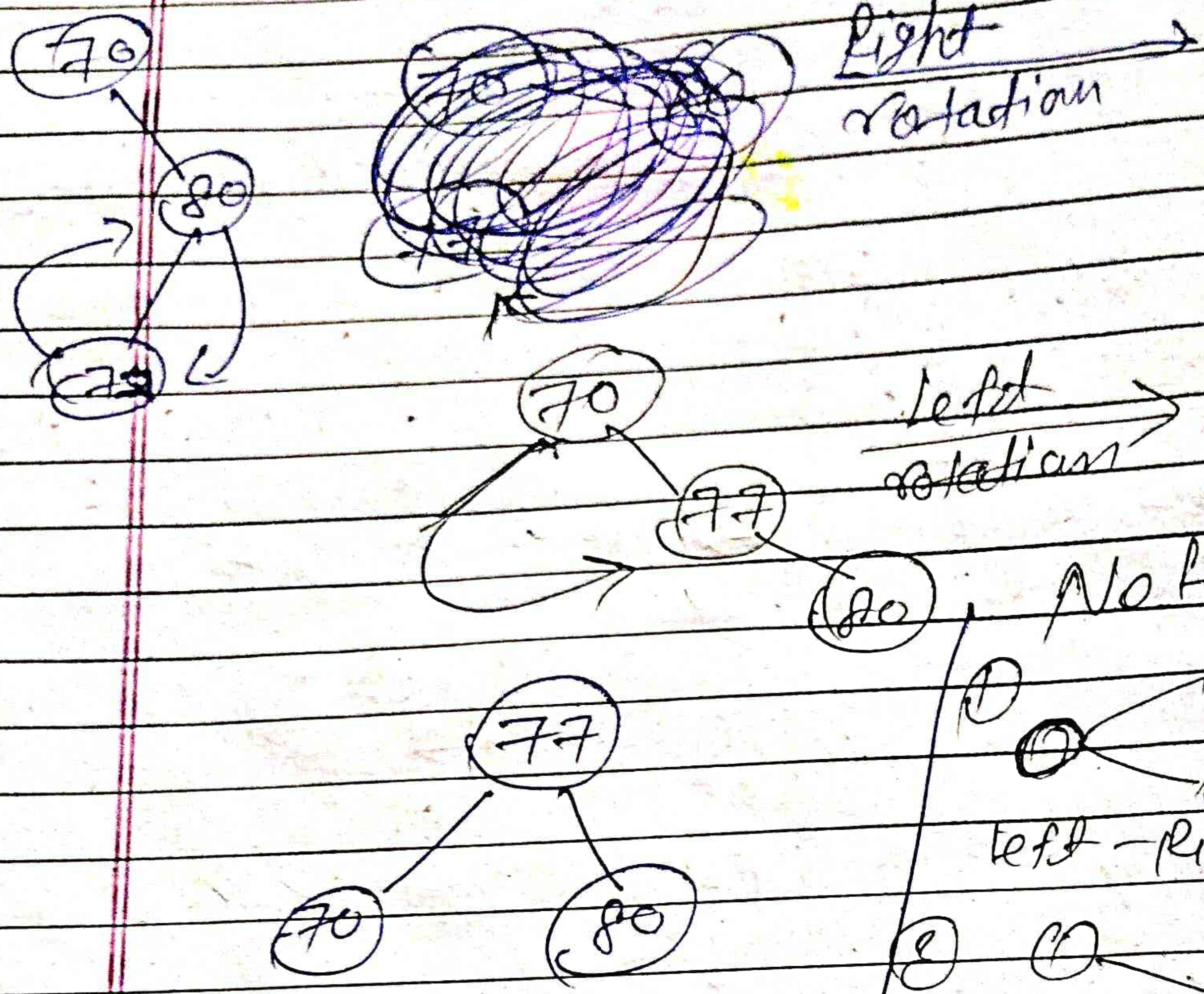


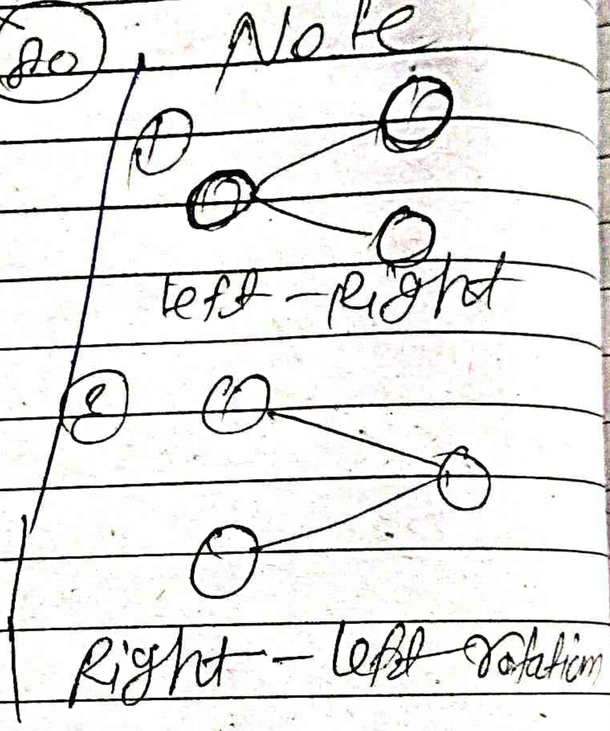


right rotation



4 Right left rotation

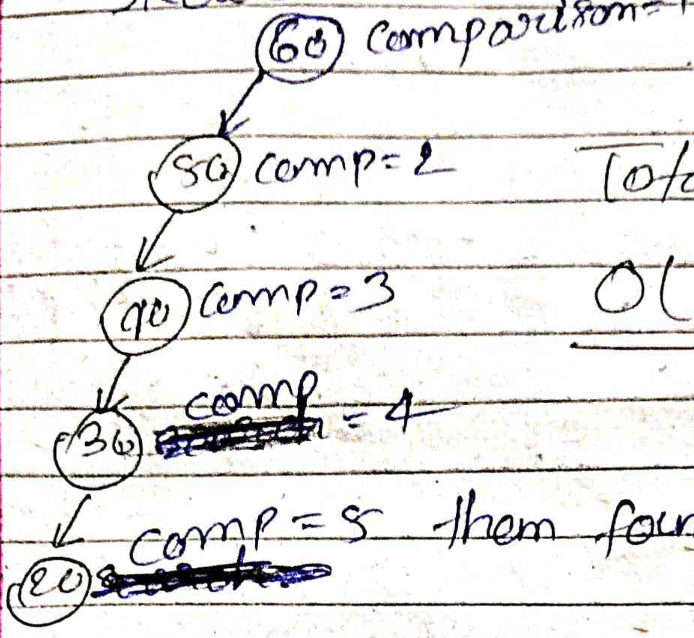




Q

* How can you say that a skewed does not give performance than AVL Tree.

ans → Skewed

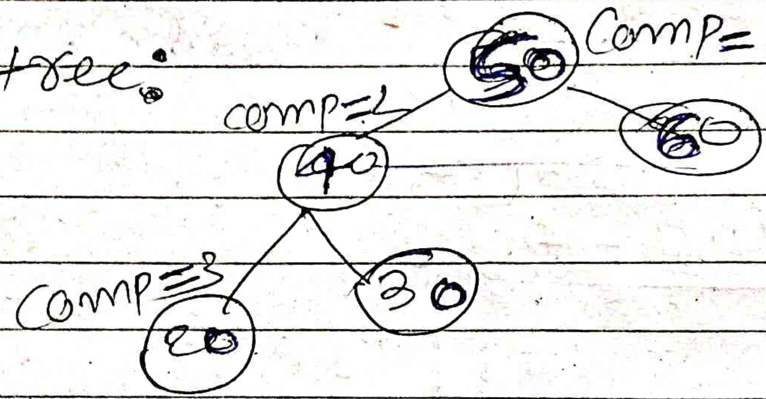


Search = 20

Total no of comparison = 5

O(n) (skewed)

AVL tree:



No of comparison = 3

$O(\log_2 n)$ (AVL Tree)

that's why we can say skewed does not give performance than AVL Tree.

Insertion] in BST
Deletion]
↑ Balanced

AVL Tree = Balanced BST

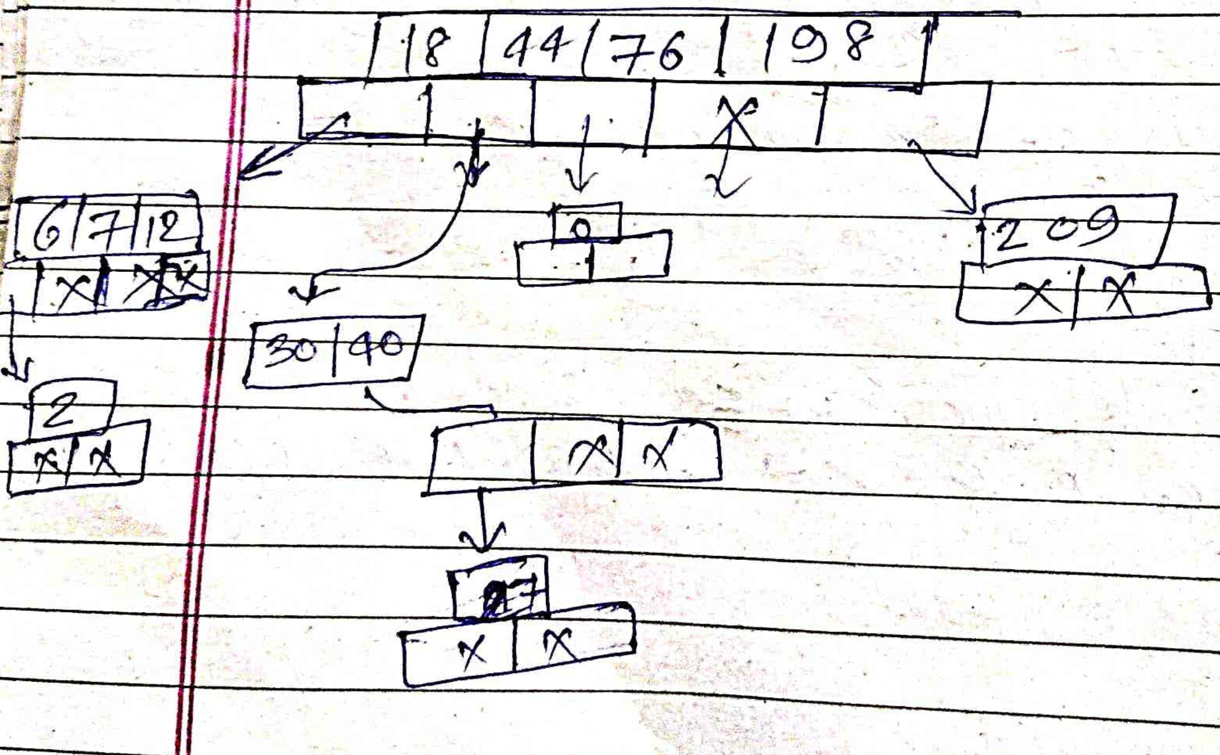
m-way search Tree: Every node can have at the most (m) child nodes.

5 way search tree

Ex → m=5, every node can have max 5 childnodes.

key = m-1
key = 5-1 = 4

$\boxed{\text{key} = 4}$



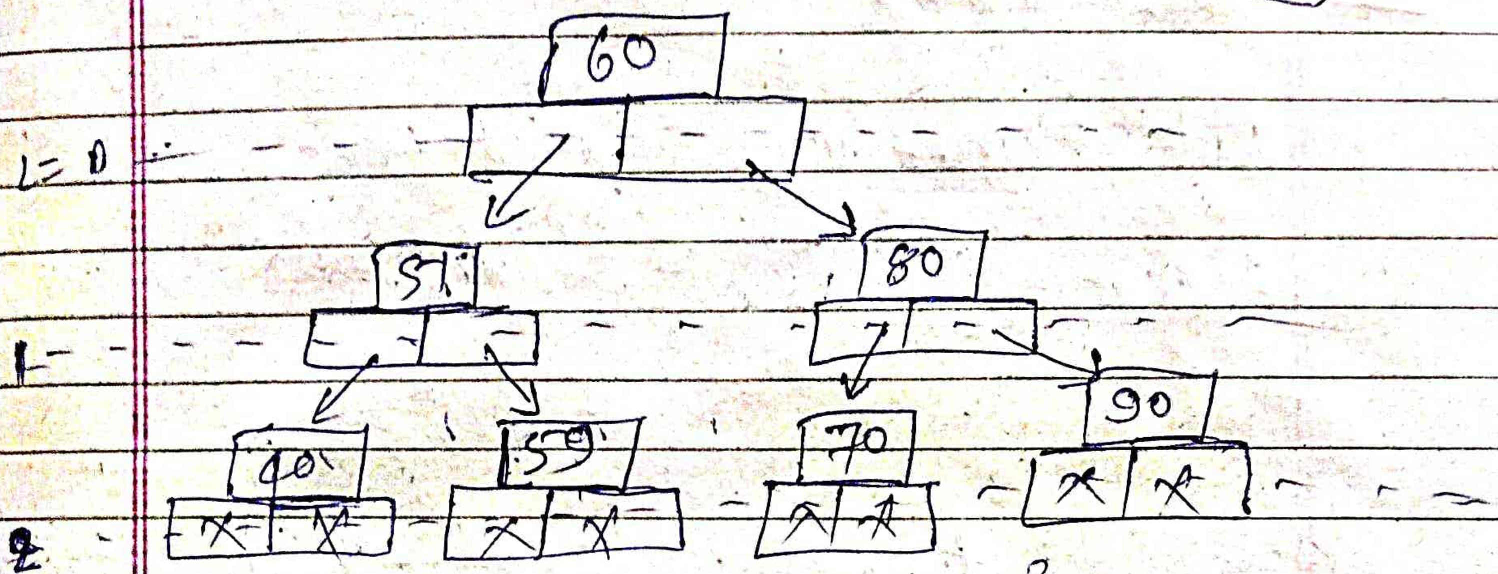
7 Marks — B-tree make ~~a graph of a~~ an m-way search tree as balanced tree by maintaining a height close to $\log_m(n+1)$

Analysis of m-way Search tree:

2-way Search Tree → key = m-1

key = 2-1

$\boxed{key = 1}$



No of nodes ⇒ $n = 7 = 2^3 - 1$

Height = No of levels + 1 ⇒ 2+1 = 3

$n = 7 = 2^3 - 1$

$7 = 2^3 - 1$

$n = m^h - 1$

Height = 2+1 = 3

$n = m^h - 1$

$m^h = n+1$

Taking ~~[crossed out]~~ $\log_m$ both side

$\log_m(m^h) = \log_m(n+1)$

$h \cdot \log_m m = \log_m(n+1)$ $\{\log_2 2 = 1$

$h = \log_m(n+1)$

$\boxed{h = \log_m(n+1)}$

Application → file indexing in disk

Conclusion

m-way search tree may vary

$h \Rightarrow \log_m(n+1) \longleftrightarrow n$ (max)

* B - Tree

Rules ⇒ m - way search tree

1. Root $\xrightarrow{\text{at least}}$ 2 child nodes
$\xrightarrow{\text{at the most}}$ m child nodes.

2. The internal nodes $\xrightarrow{\text{at least}}$ $\lceil \frac{m}{2} \rceil$ child nodes
(except root)

Note
$\lceil 2.5 \rfloor$
= 2

m = 5

$\lceil \frac{5}{2} \rceil = \lceil 2.5 \rceil$. upper value = 3

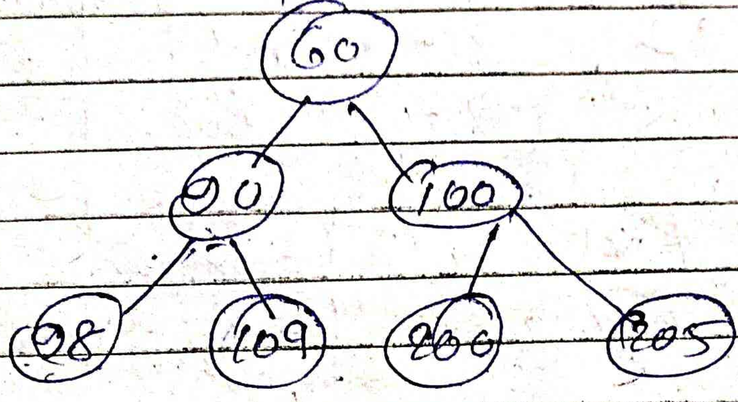
3. keys = (m-1) } way.

4. All leaf nodes should be at the same level.

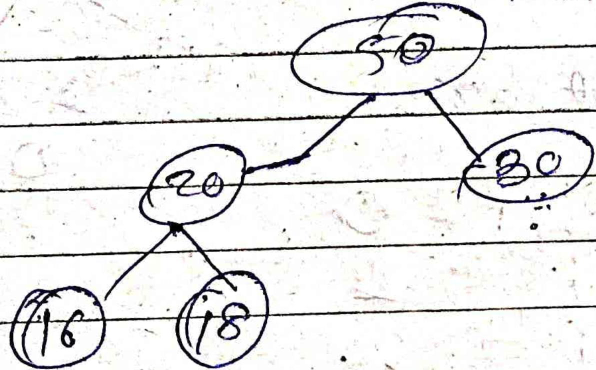
* Heap Trees → min Heap
→ Max Heap

It is that binary tree

Root ~~equal~~ min Heap



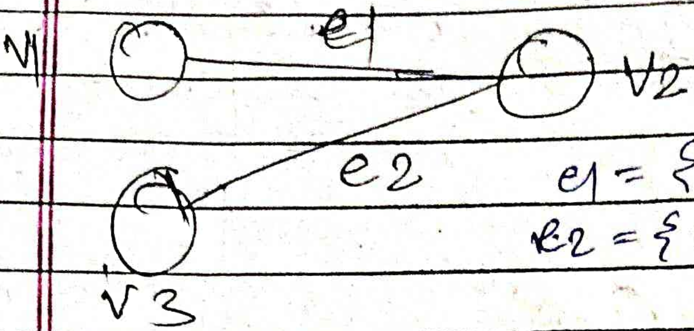
max Heap Root



Graphs

* $G = (V, E)$ $V = \{v1, v2, v3\}$
$E = \{e1, e2\}$

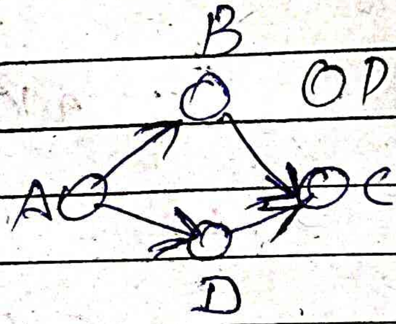
Term:
~~Divided graph~~
~~undivided graph~~
directed graph
undirected graph



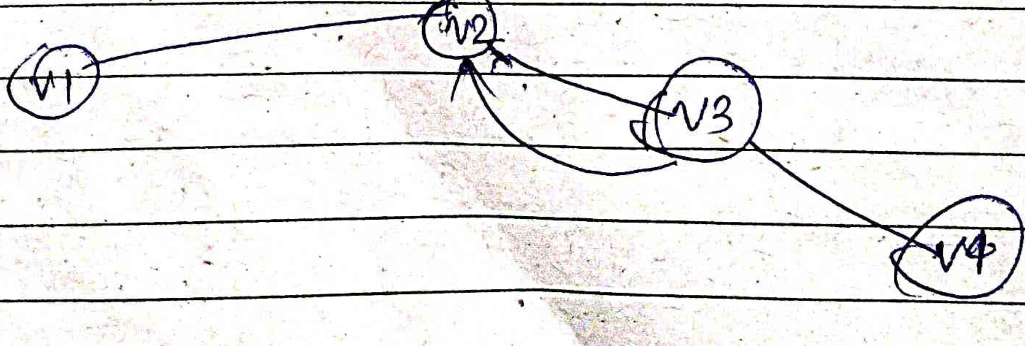
$e1 = \{v1, v2\}$
$e2 = \{v2, v3\}$

$[v1, v2, v3] = ✓$ source
$[v3, v2, v1] = ✗$ sink

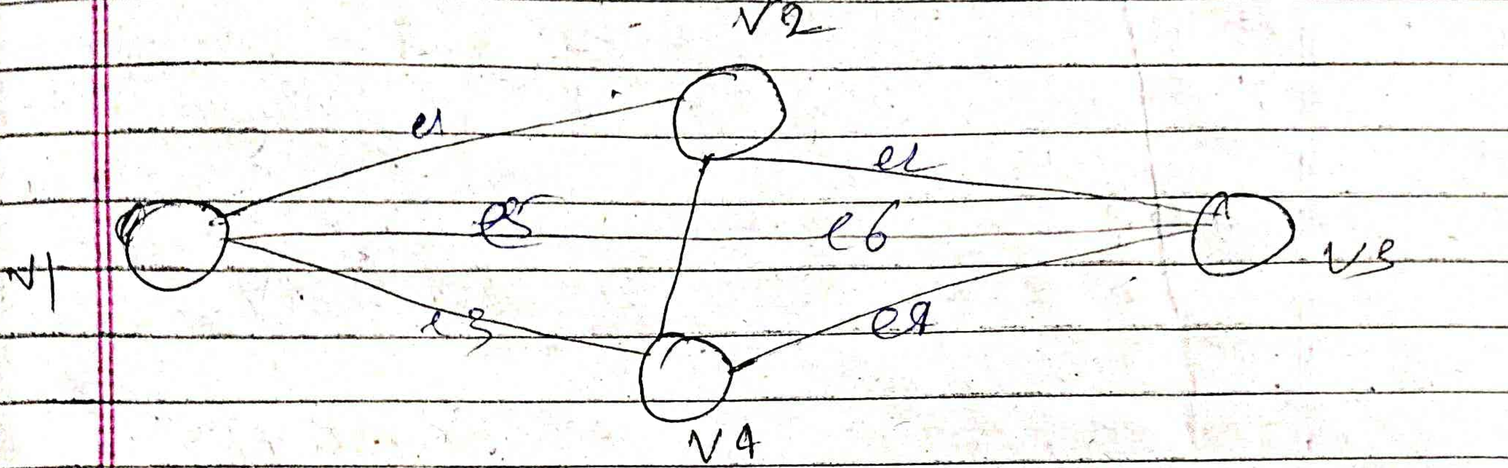
No of edges of A = 2
degree of A =
in degree   out degree



Indegree (A) = 0
Out deg (A) = 2
Indeg (c) = 2 } sink
out deg (c) = 0
indeg (b) = 0
out deg (b) = 0



Cycle: It is simple closed path.

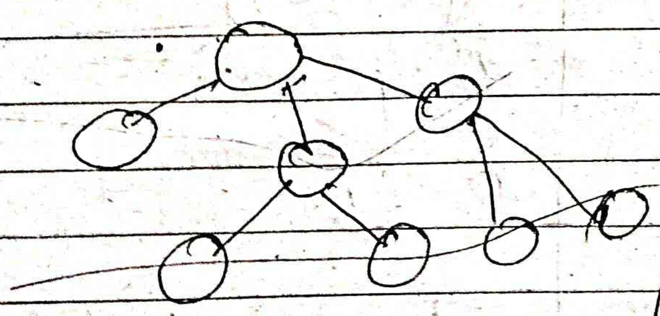


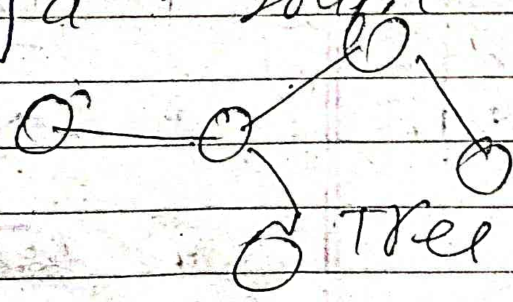
No of vertices $= n = 4$
No of edges in a complete graph $= e$

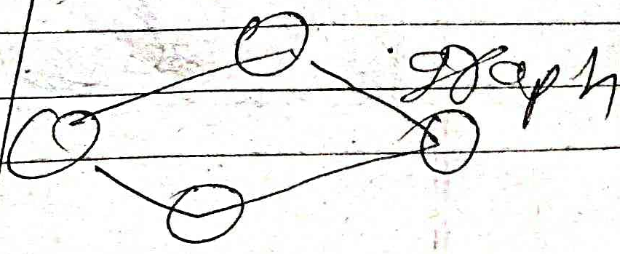
Q Complete a graph of vertices 4 and compute the edges of a complete graph.

$$\frac{n(n-1)}{2} = \frac{4(4-1)}{2}$$
$$\frac{4 \times 3}{2} = \frac{12}{2} = 6$$

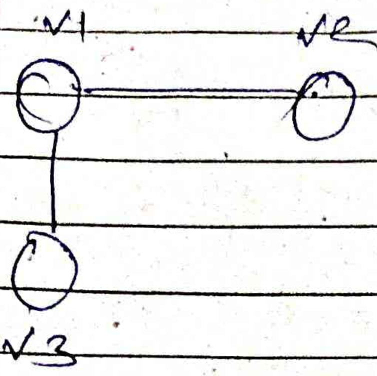
Tree or Tree graph | a graph



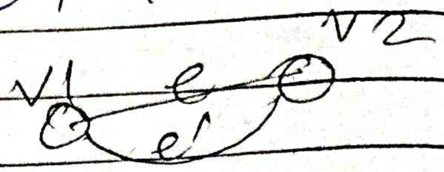


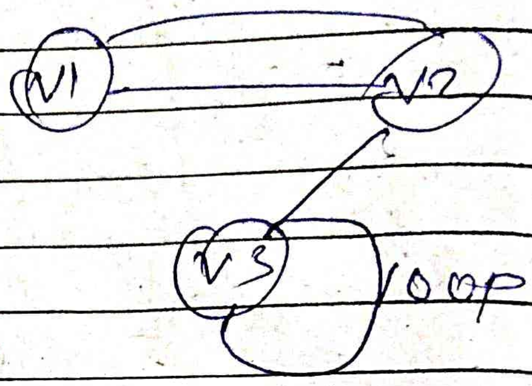


* Multigrap : Allows (I) multiple edges
(II) loop



graph





multiple

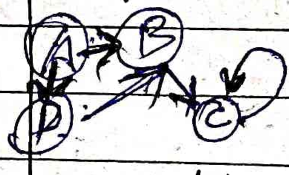
* Representation of graph

Sequential rep (2D-Matrix)

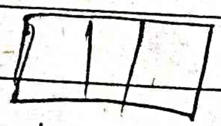
linked repre
linked list (Adjacency list)

Adjacency matrix

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



Node	Adjacency list
A	B, D
B	C
C	C
D	B



Nodelist

edgelist

Date / /
Page No.

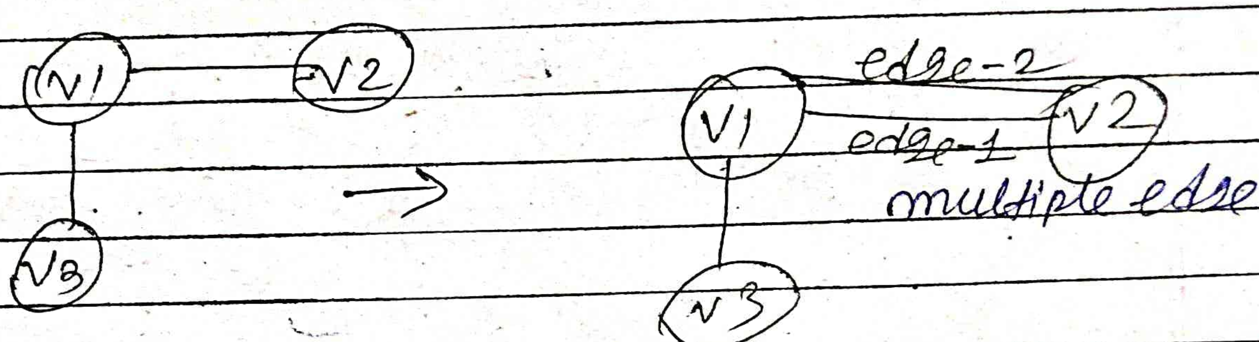
(pg9) Analyze the features that make a graph as a multigraph.

ans ⇒ To make a graph as a multigraph

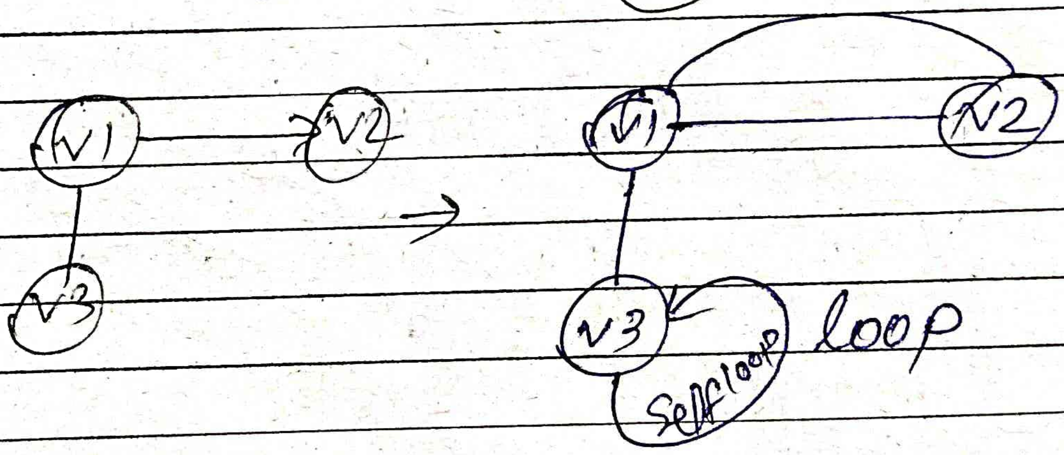
(i) graph should have multiple edges between two nodes.

(ii) graph can also have self loops.

(i)

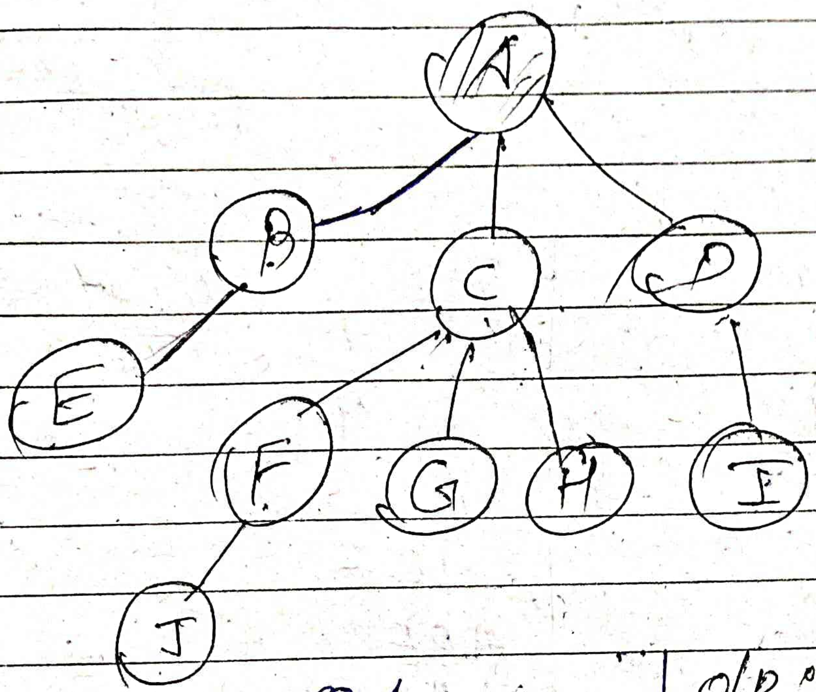


(ii)



* Apply BFS tech. [queue]

queue Q1 queue Q2

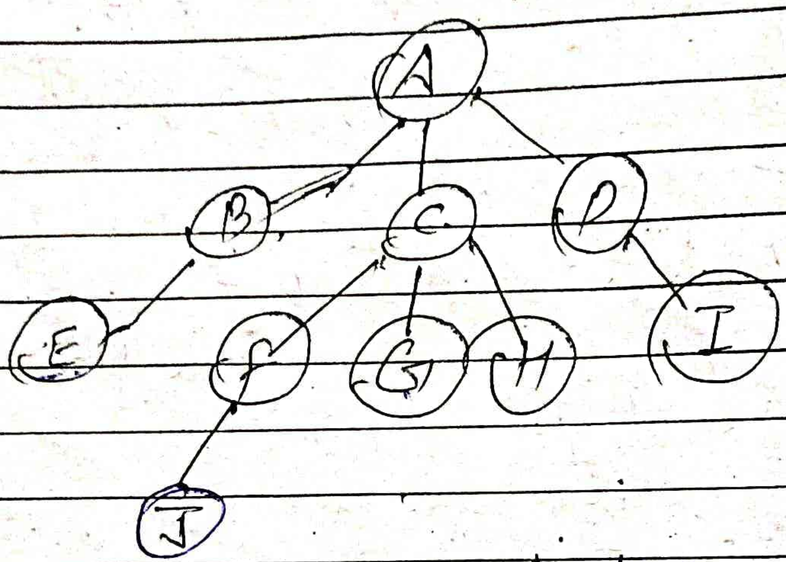


Queue Q1	o/p:	queue Q2
A	ϕ	
B C D	A	
C D E	A B	
D E F G H	A B C	
E F G H I	A B C D	

	ABCDE
FGHI	ABCDEF
GHIJ	ABCDEFG
HIJ	ABCDEFGH
IJ	ABCDEFGHI
J	ABCDEFGHIJ

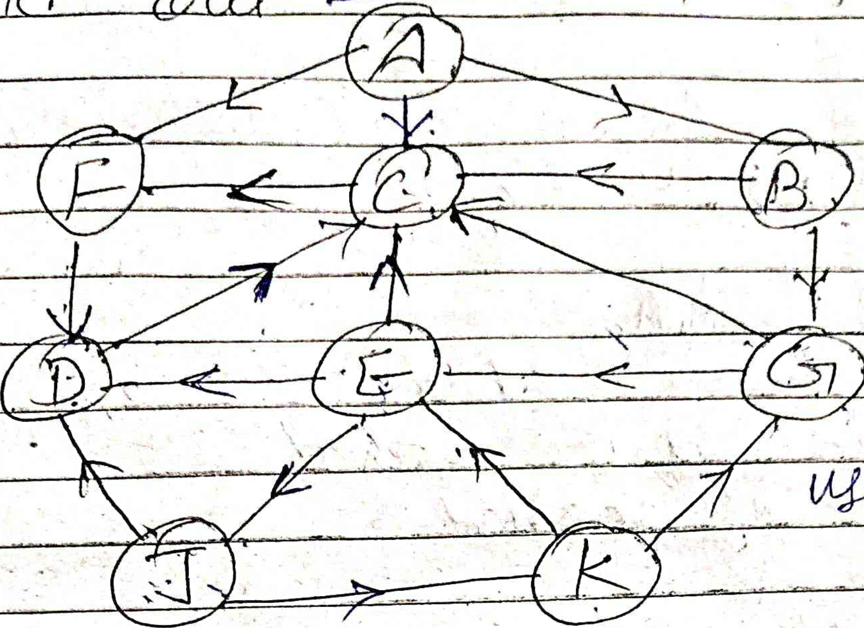
* Apply DFS [stack]

only one stack require



stack	o/p: print command
A	ϕ
D C B	A
D C E	AB
D C	ABE
D H G F	ABEC
D H G J	ABECF
D H G	ABECFJ
D H	ABECFJG
D	ABECFJGH
I	ABECFJGHD
	ABECFJGHDI

* find out minimum path from node A to J



use BFS

Q1

A

F C B

C B D

B D

D G

G

E

J

K

O/P :

Φ

A

AF

AFC

AFCB

AFCBD

AFCBDG

AFCBDGE

AFCBDGEJ

↓

Back Track

A ~~F~~ ~~C~~ B ~~D~~ G E J

A → B → G → E → J

Minimum path → A → B → G → E → J

* Hasing (saving the memory)

* Hasing techniques [Hash functions]
→ H(K) : K — L

Hashing
(1) Division method
(2) Mid - squre method
(3) folding method

k = 3205
h = k mod m
h = 3205 mod
To represent 4 digit no into two digit no mod h with m
m → prime no.
m → closest to maximum value and greater the record of the member
> m : 9699 (97)
> record
> no of employee prime no होना चाहिए
> 68

97) 3205 (04
X

(ii) Mid-Squre method : $K = 3205 \rightarrow K^2$

$K^2 = 10272025$

72

(iii) Folding method : (i) $32|05 = 32 + 05 = 37$ without reversing

(ii) reversing 3205

$32 + 50 = 82$ with reversing

E.g. (i) $7148 = 71 + 48 = 119 = 19$

(ii) $7148 = 71 + 84 = 155 = 55$

E.g. $3232 = 32 + 23 = 55$

* Collision in Hashing:

$$\boxed{\begin{array}{l} H(k) = h \\ H(k') = h \end{array}}$$

$H(K) \rightarrow$ folding

$H(7148) \rightarrow 55$

$H(3332) \rightarrow 55$

Coll resolution tech : (i) linear probing

(ii) quadratic Probing

(iii) Double ~~Reu~~ Hashing

practicle formula

① Linear probing : | $h, (h+1)\%N, (h+2)\%N$

$h, h+1, h+2, h+3,$

② Quadratic probing : | $h, (h+1^2)\%N, (h+2^2)\%N$

$h, h+1^2, h+2^2, h+3^2$

$h, h+1, h+4, h+9$

double hashing

③ $H(K) = h$

$H'(K) = h'$

$h, h+h', h+2h', h+3h'+ -$

$h, (h+h')\%N, (h+2h')\%N$ ..

---

① linear probing

$n=8$

	A	B	C	D	E	X	Y	Z
h :	4	8	2	11	4	11	5	1

Mem loc :

$(4+1)\%11$ $(11+1)\%$

1	2	3	4	5	6	7	8	9	10	11
X	C	Z	A	E	Y		B			D

Open Hashing chaining

Open addressing (closed Hashing)
[To resolve collision: involve probing]

Efficiency:

	A	B	C	D	E	X	Y	Z			
H(K)	4	8	2	11	4	11	5	1			
	X	C	Z	A	E	Y		B			D
M loc.	1	2	3	4	5	6	7	8	9	10	11

maximum = 11

Step: 1

$\boxed{n = 8, \quad mloc = 11}$

$$\text{load factor} = \frac{n}{m} = \alpha$$

$$= \frac{8}{11} \Rightarrow \boxed{\alpha = 0.73}$$

Step: 2 Compute $S(\alpha)$ & $U(\alpha)$

$S(\alpha)$ → It is the average no of probes for successfull search
$U(\alpha)$ → It is the average no of probs for unsuccessfull search.

Remember

$S(\alpha) = ?$
$U(\alpha) = ?$

$$S(\alpha) = \frac{1}{2}\left[1 + \frac{1}{(1-\alpha)}\right] = 2.35$$

$$U(\alpha) = \frac{1}{2}\left[1 + \frac{1}{(1-\alpha)^2}\right] = 7.36$$

$S(\alpha) = \frac{1}{2}[1 + \frac{1}{\ }$

$S = \frac{2.35}{n}$, $U = \frac{7.36}{m}$

$\boxed{S = 0.29}$ $\boxed{U = 0.669}$

A B C D E X Y Z

$$S = \frac{1 + 1 + 1 + 1 + 2 + 2 + 2 + 3}{8(m)}$$

$$S = \frac{13}{8} \quad \boxed{S = 1.625} \approx \boxed{1.63}$$

$K \rightarrow \frac{1}{2}$ केलिए जगह खोजना होगा

$1.63 < 2.35$

$S < S(\alpha)$

$$U = \frac{\overset{1}{7} + \overset{2}{6} + \overset{3}{5} + \overset{4}{4} + \overset{5}{3} + \overset{6}{2} + \overset{7}{1} + \overset{8}{2} + \overset{9}{1} + \overset{10}{1} + \overset{11}{8}}{11(m)}$$

$$U = \frac{40}{11} \quad U = 3.63$$

$\boxed{U < U(\alpha)}$

$3.63 < 7.36$

(*) open chaining (hashing)

Records - A B C D E X Y Z

4 8 2 11 4 11 5 1

$n = 8$, $m = 11$

link	
1	8
2	3
3	0
4	~~1,~~ 5
5	7
6	0
7	0
8	2
9	0
10	0
m = 11	~~4~~ 6

	Info	Link
1	A	0
2	B	0
3	C	0
4	D	0
5	E	1
6	X	4
7	Y	0
8	Z	0
9		0
10		0
11		

faster method than open hashing but not space efficient because it using memory to save the pointer.

efficiency

$S(\lambda) = 1 + \frac{1}{2}\lambda$ $\quad U(\lambda) = e^{-\lambda} + \lambda$, $\lambda = \frac{n}{m}$

$S_1 = ?$ $\quad U = ?$ $\quad e \rightarrow$ Euler's no - 2.718

$\boxed{\lambda = \frac{8}{11}}$ $\boxed{\lambda = 0.73}$

$S(\lambda) = 1 + \frac{1}{2} \times 0.73$, $U(\lambda) = e^{-\lambda} + \lambda$

$S(\lambda) = 1.36$

$\boxed{S(\lambda) = 1.37}$

$U(\lambda) = 0.48 + 0.73 = 1.21$

A B C D E X Y Z

$$S = \frac{2 + 1 + 1 + 2 + 1 + 1 + 1 + 1}{m = 8}$$

$S = \frac{10}{8}$ $\boxed{S = 1.25}$

$S(\alpha)$ S

$1.37 > 1.25$

efficient

$$U = \frac{\overset{1}{1} + \overset{2}{1} + \overset{3}{1} + \overset{4}{1} + \overset{5}{1} + \overset{6}{1} + \overset{7}{1} + \overset{8}{1} + \overset{9}{1} + \overset{10}{1} + \overset{11}{1}}{m = 11}$$

$U = \frac{11}{11}$ $U = 1$

$U(\alpha)$ U

$1.21 > 1$

efficient

Analysis of load factor:

Open Addressing closed hashing	Chaining open-hashing
$\boxed{\text{Hash Table}}$	n
$\boxed{m \geq n}$ $11 = m$	m
$\alpha = \frac{n}{m} \leq 1$	$\boxed{m < n}$
	$\alpha = \frac{n}{m} > 1$

Rehashing:

(4) Quick sort Algorithm

step (1) 25, 57, 48, 37, 12, 92, 86, 33

Set first element as pivot

Step (2) dn →

dn = 0

A[dn] > pivot

UP = 7

UP will decrement till A[UP] <= pivot

Step (3) if Up > dn

swap A[dn] with A[UP]

but if UP <= dn

swap A[UP] with pivot

*

8 25, 57, 48, 37, 12, 92, 86, 33

pivot dn → ← up

pivot ← 25, 57, 48, 37, 12, 92, 86, 33

dn up

57>25 12<=25

pivot ← 25, 12, 48, 37, 57, 92, 86, 33

dn up

pivot ← 25, 12, 48, 37, 57, 92, 86, 33

up dn

12<=25 48>25

up has crossed dn so up<=dn

then swap A[up] with pivot

sorted	unsorted
12, 25	48, 37, 57, 92, 86, 33

pivot ← 48, 37, 57, 92, 86, 33 {apply quick sort again}

dn → up

pivot ← 48, 37, 57, 92, 86, 33

dn up

57>48 33<=48

pivot 33 48, 37, 33, 92, 86, 57

↑ dn ↑ up

Pvt ← 48, 37, 33, 92, 86, 57

up dn

33<=48 92>48

up has crossed dn

up <= dn so swap it with pivot

A[UP] swap with pivot

[33, 37], 48, [92, 86, 57]

unsorted unsorted

pivot 33, 37

dn up

pivot 33, 37

dn

up

dn has crossed dn

Up <= dn

then A[up] swap with pivot

[33, 37] 48 [92 86 57]

sorted unsorted

pivot 92, 86, 57

dn up

pivot 92, 86, 57 + ∞

dn

Up

57 < 92

[57, 86], [92]

unsorted sorted

UP crossed dn

swap A[UP] with pivot

Up <= dn

pivot 57, 86

dn up

pivot 57, 86

dn

up

Up crossed dn up <= dn

A[up] swap with pivot

[57, 86]

sorted

finally we got

$\boxed{12, 25, 33, 37, 48, 57, 86, 92}$

sorted list

Time complexity of quicksort (normal/average case)

$= O(n \log n)$

worst case $= O(n^2)$

(5) Merge Sort:

1	2	3	4	5	6	7	8	9	10
310	285	179	652	351	423	861	254	450	520

compute mid $= \left\lfloor \frac{low + high}{2} \right\rfloor$

low $= 1$

high $= 10$ mid $= \left\lfloor \frac{1+10}{2} \right\rfloor \Rightarrow$ mid $= 5$

low high

$\boxed{1, 10}$

$\boxed{1, 5}$ $\boxed{6, 10}$

310, 285, 179, 652, 351 | 423, 861, 254, 450, 520

~~Do same~~

Do same process

till individual

element are not

obtained.

310, 285, 179, 652, 351

low = 1
high = 5

mid = $\left[\frac{1+5}{2}\right]$ $\boxed{mid = 3}$

1, 2, 3 | 4, 5
310 285 179 | 652 351

$\boxed{1,10}$

$\boxed{1,5}$ $\boxed{6,10}$

$\boxed{1,3}$ $\boxed{4,5}$

310, 285, 179

$l = 1, h = 3$

$m = \left[\frac{1+3}{2}\right]$ $\boxed{m = 2}$

310, 285 | 179

$\boxed{1,3}$

$\boxed{1,2}$ $\boxed{3,3}$

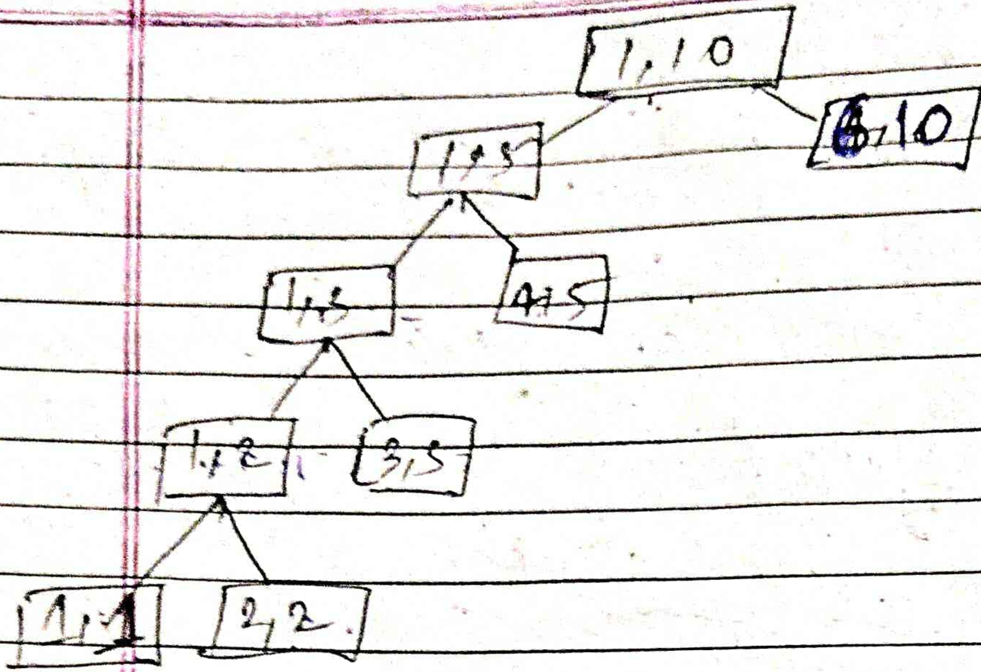
310, 285

$l = 1$ $h = 2$ $m = \frac{3}{2} = m = 1$

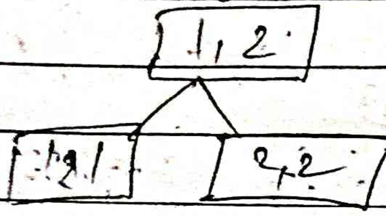
310 | 285

$\boxed{1,2}$

$\boxed{1,1}$ $\boxed{2,2}$



merge
individual element



310 | 285

merge

~~310, 285~~ 285, 310

1, 2 | 3, 3

285, 310 | 179

merge

179, 285, 310

~~split~~ 4, split 4, 5

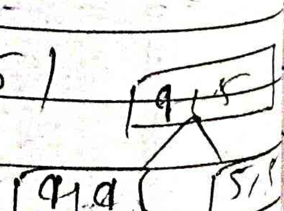
4 5

652, 351

l = 4
h = 5

m = $\frac{9}{2}$ $\boxed{m = 4}$

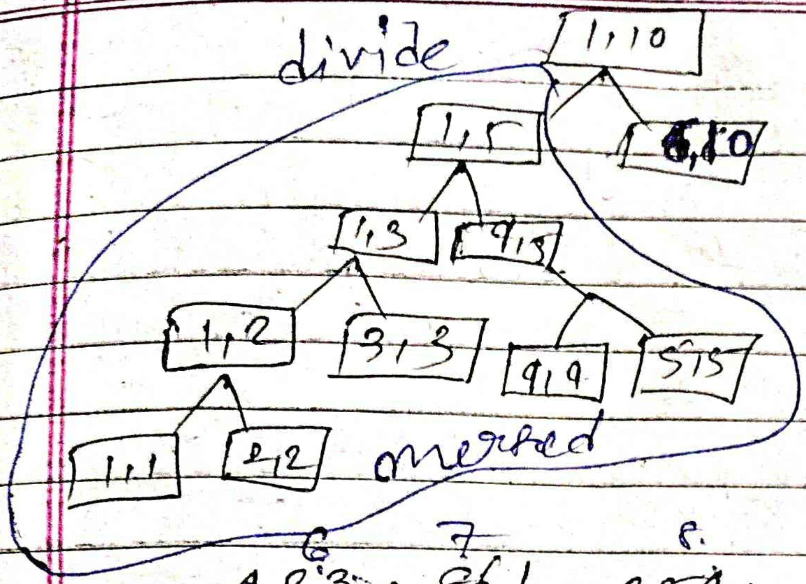
652 | 351



merge

merge

179, 285, 310, 351, 652

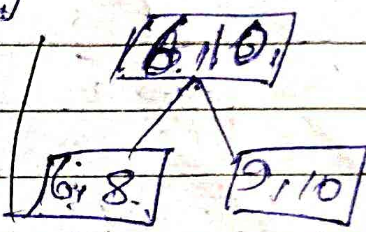


6, 7, 8, 9, 10
423, 861, 254, 450, 520

$l = 6$
$h = 10$

$m = \left\lfloor \frac{6+10}{2} \right\rfloor$ $\boxed{m = 8}$

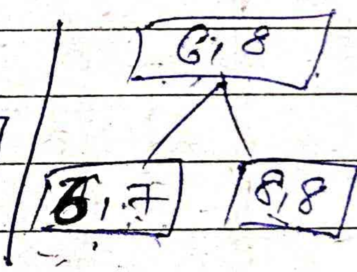
6, 7, 8	9, 10
423 861 254	450 520



6, 7, 8
423 861 254

$l = 6$
$h = 8$
~~m = 6~~

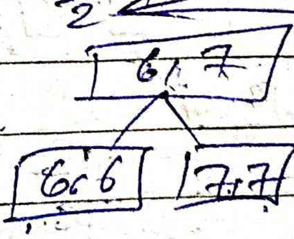
$m = \left\lfloor \frac{6+8}{2} \right\rfloor$ $\boxed{m = 7}$



6	7
423	861

$l = 6$
$h = 7$

$m = \left\lfloor \frac{6+7}{2} \right\rfloor$ $m = \frac{13}{2}$ $\boxed{m = 6}$



6	7
423	861

merged

423, 861	8 254

merged

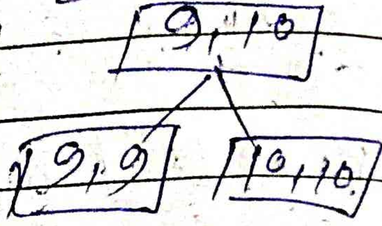
6 7 8
$\boxed{254, 423, 861}$

Split | 9, 10

| 450 , 520 |

$l = 9$ $\quad m = \frac{10+9}{2}$ $\quad m = \frac{19}{2}$ $\quad \boxed{m = 9}$

$h = 10$



9 | 10

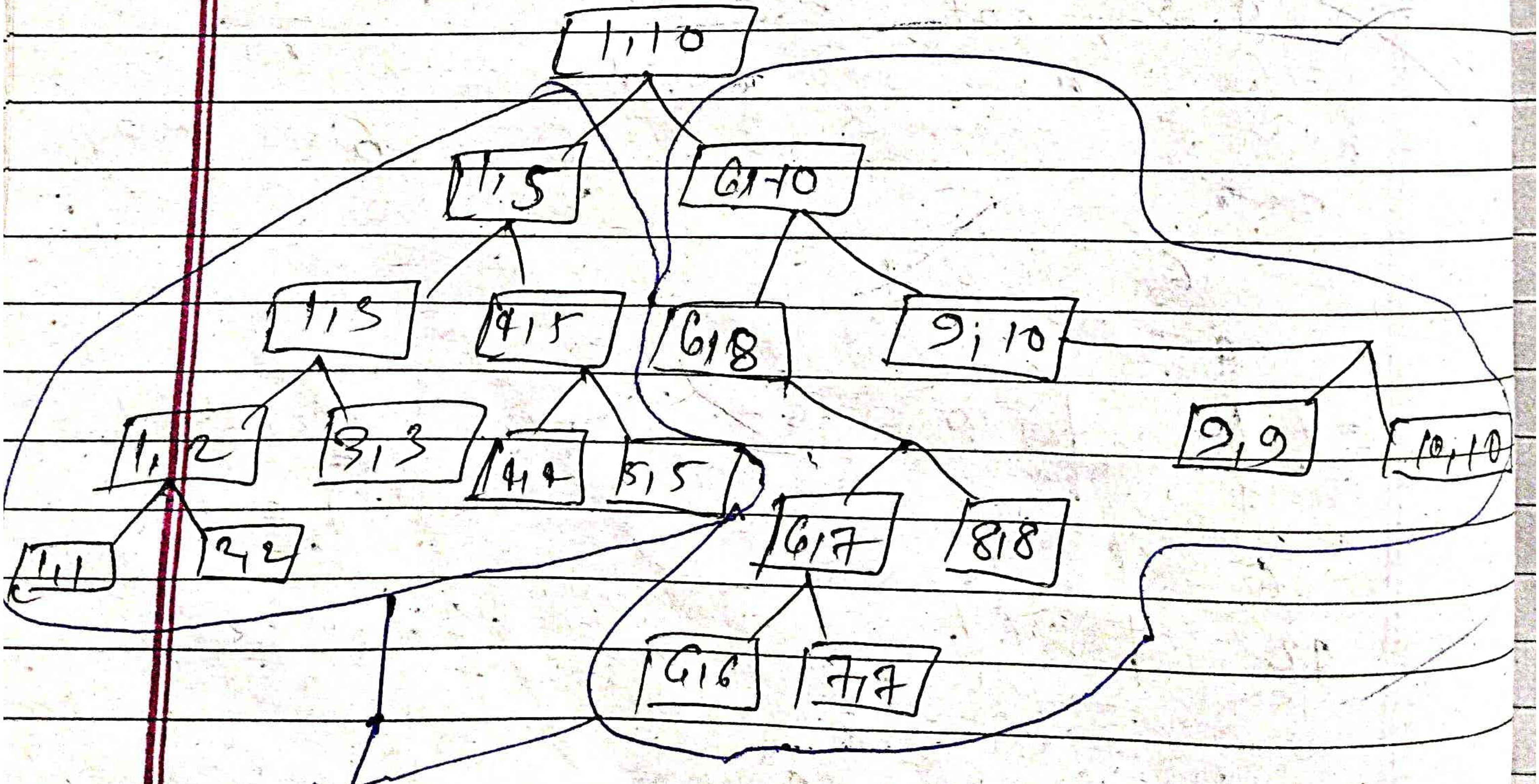
450 | 520

merged

6, 7, 8

254, 423, 861

merged {254, 423, 450, 520, 861}



finally merge both

finally merging the two list

179, 285, 310, 351, 652 | 254, 423, 450, 520, 861

for merging process an extra array is required

179, 285, 310, 351, 652 | 254, 423, 450, 520, 861

i i i i i | j j j j j

Compare i & j and increment them

if i < j insert i in new array & increment i & again compare | if j < i insert j in new array and increment j & again compare

[179, 254, 285, 310, 351, 423, 450, 520, 652, 861]

final merged & sorted Array / list

Time Complexity of Merge sort = O(n log n) in normal, Average, worst case.

but not space efficient because it takes one extra array to merge both list.

(6) Heapsort

40, 80, 35, 90, 45, 50, 70.

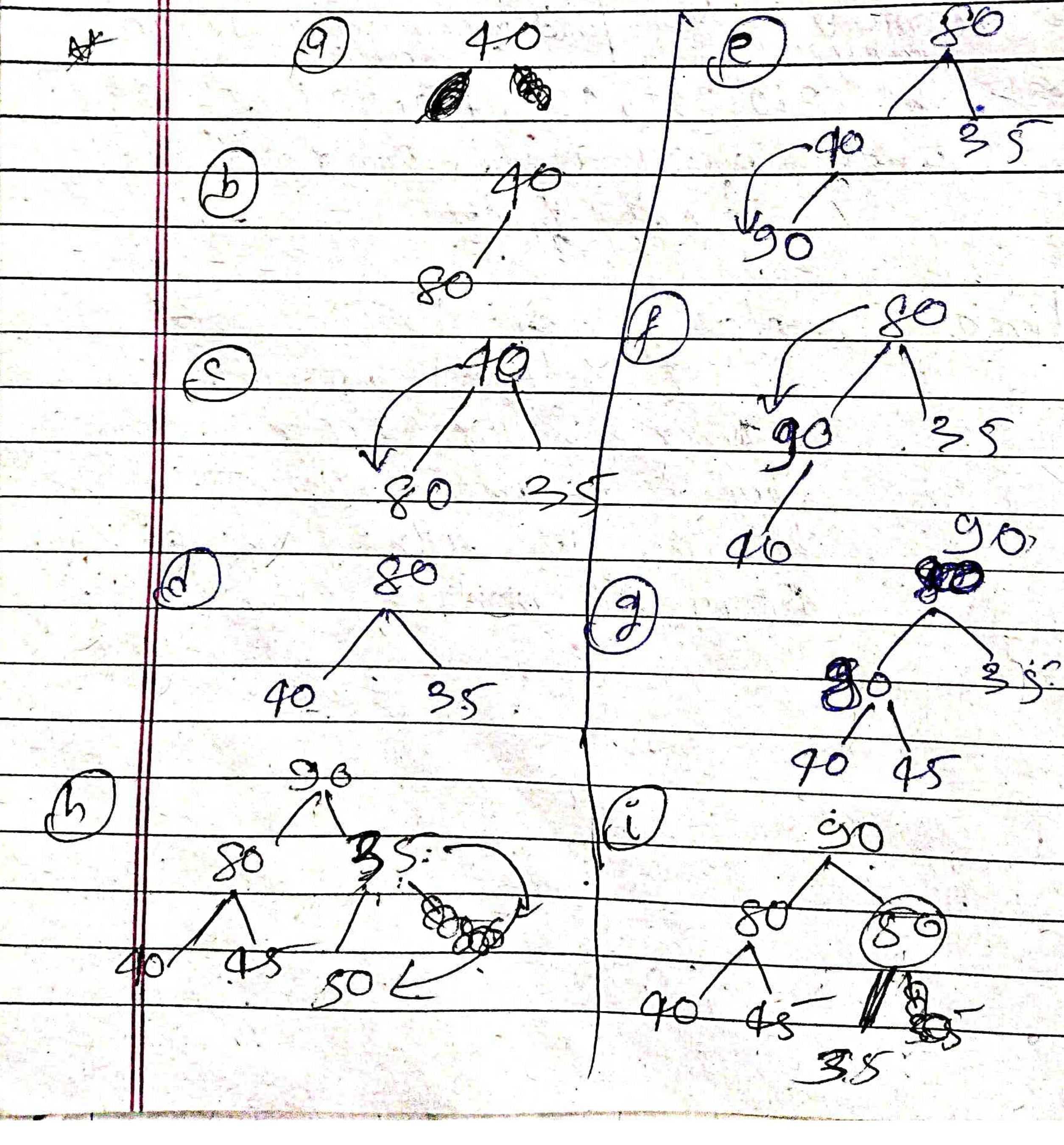
phase (1) create max Heap

(a) 40

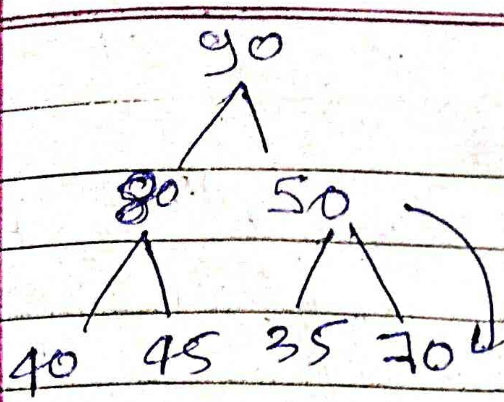
(b) 40
80

{ compare left & right child
→ whichever is greater
→ compare it with parent
→ swap (if required) to get max heap }

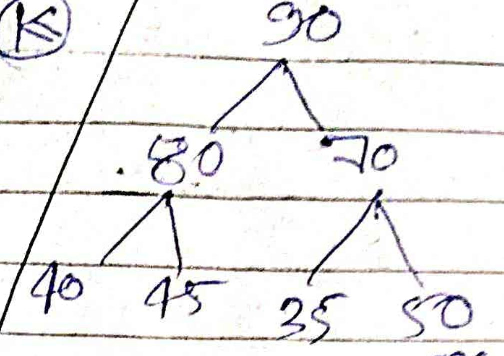
Note: Insert left & Right
Delete right & left



(j)



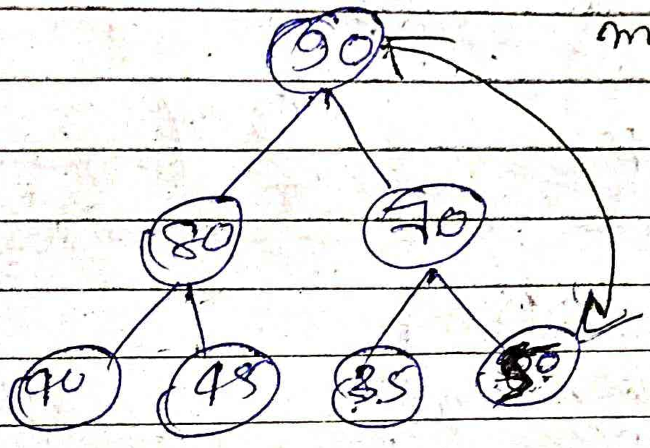
(K)



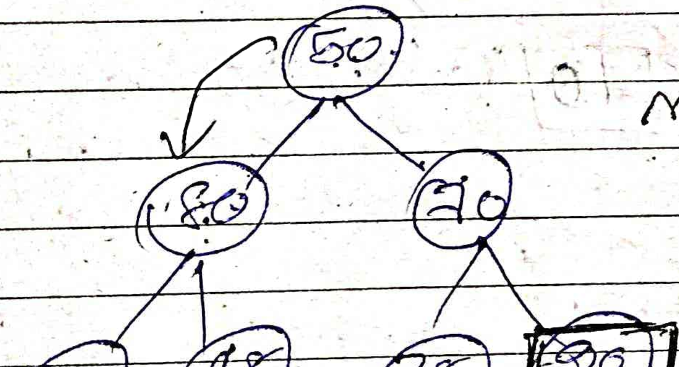
Max Heap is obtained

phase (2) Repeat Deleting Root & swap with last element

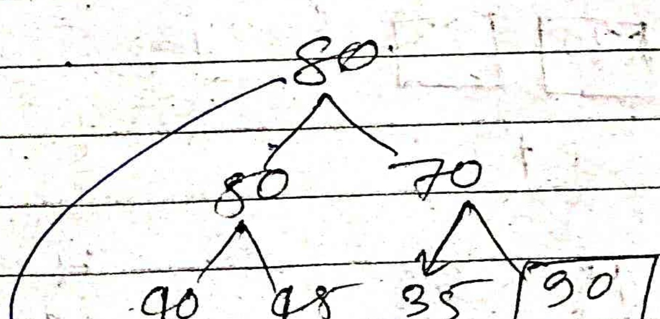
make max Heap

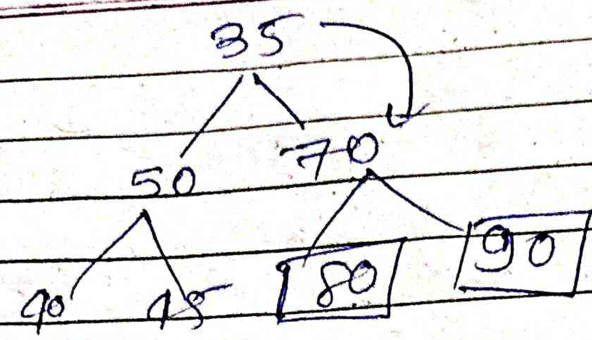


again make max Heap

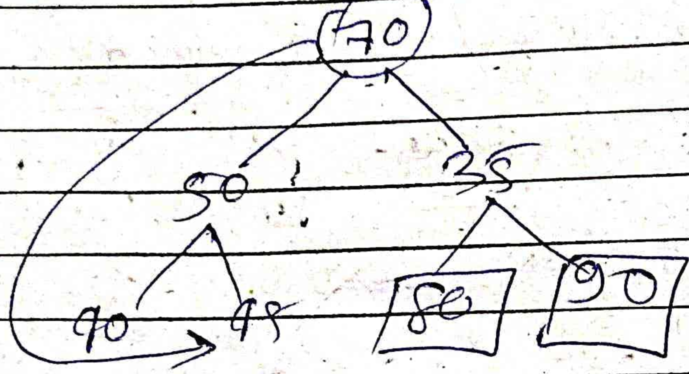


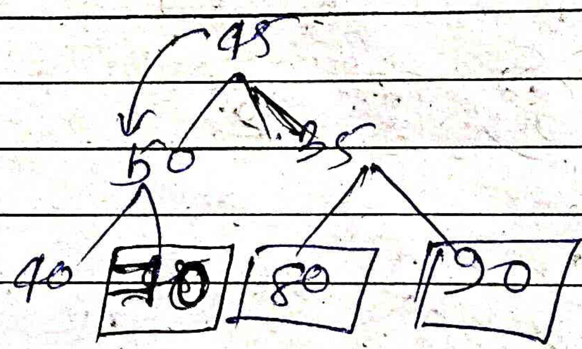
sorted



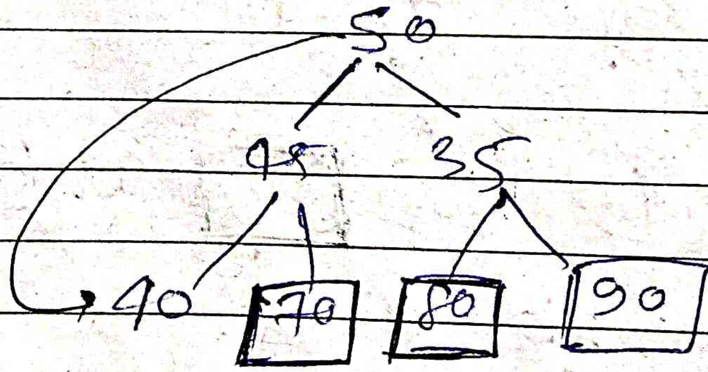


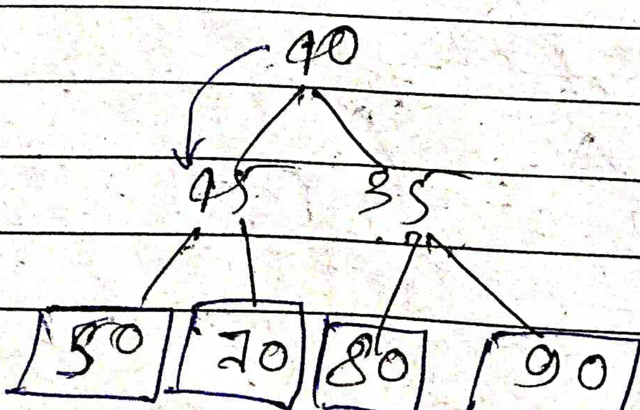
Make max Heap again



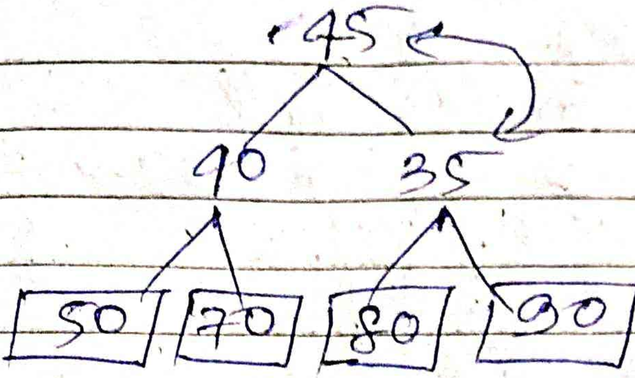


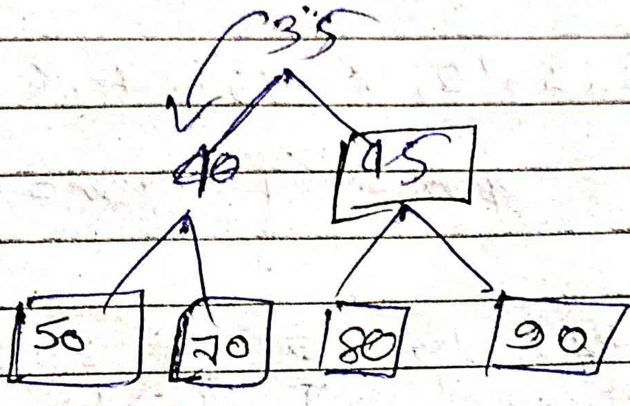
make max Heap again



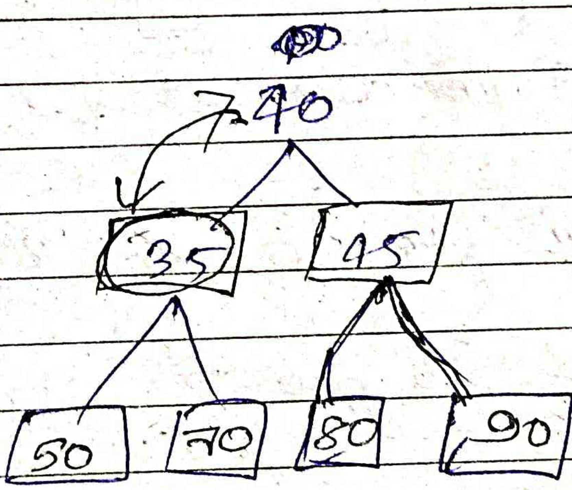


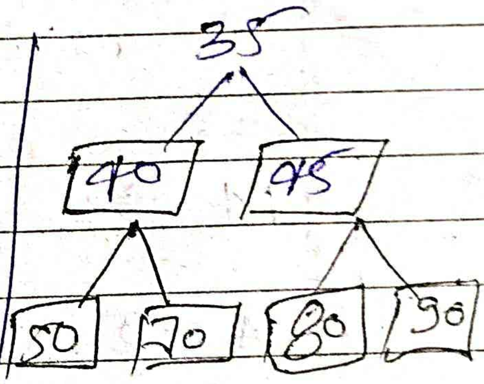
make max Heap again



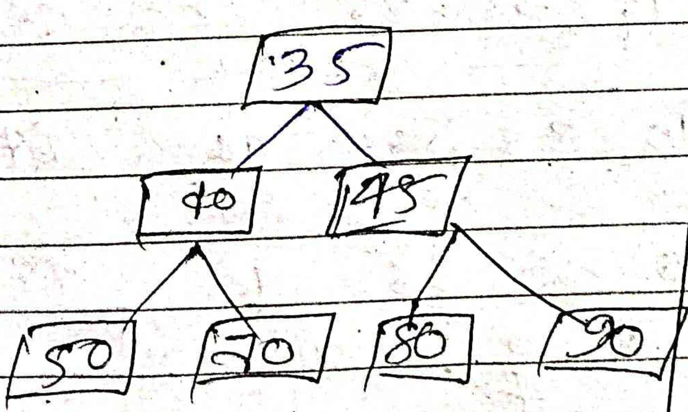


max Heap again





Time complexity of Heap sort
→ O(n log n) in all cs
no extra space required



Sorted array

(7) Shell sort :

(1) By Donald Shell
choice of incremant a :

$\left[\frac{N}{2}\right], \left[\frac{N}{4}\right], \left[\frac{N}{8}\right], \text{----} 1$

No of elements.

Q) 35, 33, 42, 10, 14, 19, 27, 44

$N=8$ $\quad$ span $= \left[\frac{8}{2}\right]$ $\quad$ span $= 4$.

35 33 42 10 14 19 27 44

swapping the elements if required,

pass 1

14 33 42 10 35 19 27 44

14 19 42 10 35 33 27 44

14 19 27 10 35 33 42 44

No swap

14 19 27 10 35 33 42 44

14 19 27 10 35 33 42 44

Next span $= \frac{8}{4} = 2$.

14 19 27 10 35 33 42 44

14 19 27 10 35 33 42 44

14 10 27 19 35 33 42 44

14 10 27 19 35 33 44 44.

14 19 27 10 35 33 42 44

Performing insertion sort on

→ 14 27 35 42

→ 19 10 33 44

→

14	27	35	42

Sorted →

14	27	35	42

Sorted →

14	27	35	42

pass-2 sorted →

14	27	35	42

19	10	33	44

19	10	33	44

10 19 33 44

10 19 33 44

10 19 33 44

10	19	33	44

So sorted result of pass 2

14 10 27 19 35 33 42 44

pass 3: span = $\frac{8}{8} = 1$

14 10 27 19 35 33 42 44

performing insertion sort on the elements

14 10 27 19 35 33 42 44

14 10 27 19 35 33 42 44

10 14 27 19 35 33 42 44

10 14 27 19 35 33 44 44

10 14 19 27 35 33 44 45

10 14 19 27 35 33 44 45

10 14 19 27 33 35 44 45

10 14 19 27 33 35 44 45

10 14 19 27 33 35 44 45

Time complexity of shell sort is

$\Theta(n(\log n)^2)$

Practice Q. Shell sort

Q 54 26 93 17 77 31 44 55 20

N = 9   Span = $\frac{9}{2}$ = 4

1st pass

54 26 93 17 77 31 44 55 20

No swapping   swapping

54 26 93 17 20 31 44 55 77

No swap

No swap

54 26 44 17 20 31 93 55 77

No swap

54 26 44 17 20 31 93 55 77

No swap

Next Span = $\frac{9}{4}$ = 2

54 26 44 17 20 31 93 55 77

Insertion on

54 44 20 93 77

26 17 31 55

54 44 20 93 77

sorted

44 54 20 93 77

sorted

44 54 20 93 77

20 44 54 93 77

Sorted

20 44 54 93 77

20 44 54 93 77

sorted

20 44 54 93 77

20 44 54 77 93

sorted

26 17 31 55

sorted

26 17 31 55

17 26 31 55

17 26 31 55

17 26 31 55

sorted

20 17 44 26 54 31 77 55 93

Next span = $\frac{2}{2} = 1$

20 17 44 26 54 31 77 55 93

20 17 44 26 54 31 77 55 93

20 17 44 26 54 31 77 55 93

17 20 44 26 54 31 77 55 93

17 20 44 26 54 31 77 55 93

17 20 44 26 54 31 77 55 93

17 20 26 44 54 31 77 55 93

sorted

17 20 26 44 54 31 77 55 93

17 20 26 40 54 31 77 55 93

17 20 26 31 40 54 77 55 93

17 20 26 31 40 54 77 55 93

17 20 26 31 40 54 77 55 93

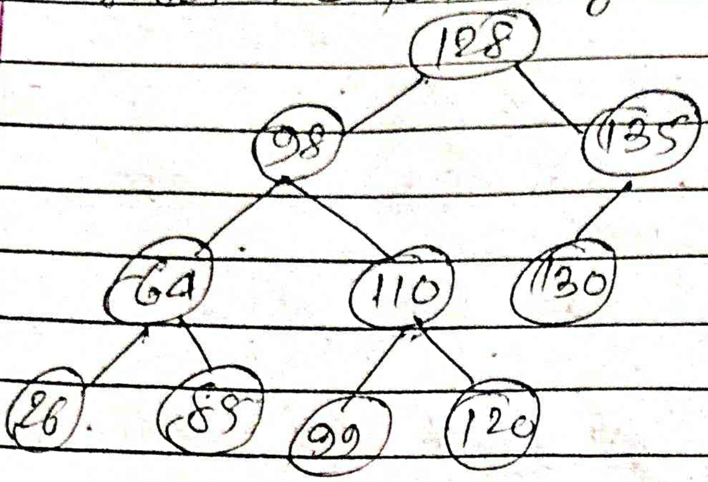
17 20 26 31 40 54 55 77 93

17 20 26 31 40 54 55 77 93

Sorted

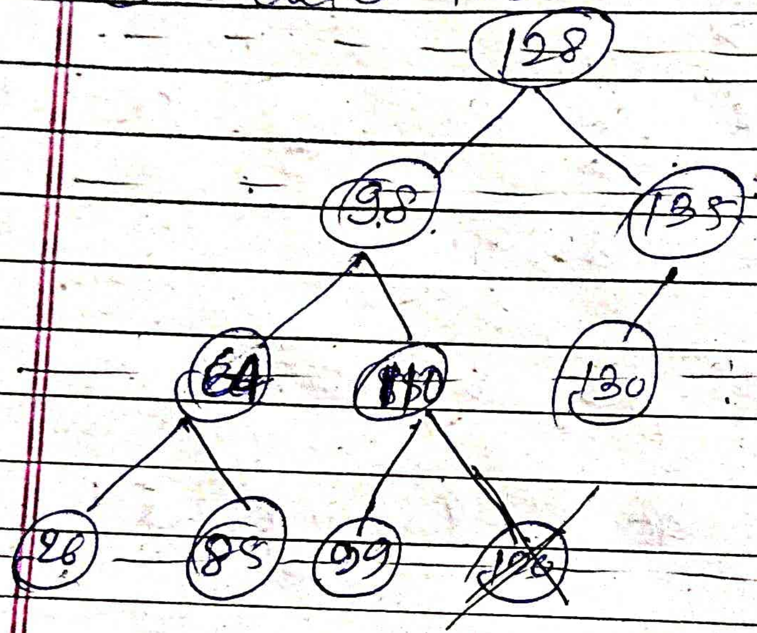
Time complixty $O(n^2)$ $(n \log n)$

Q45) Consider the following AVL search tree



(i) Delete 120
(ii) Delete 64
(iii) Delete 130
(iv) insert 102

ans ⇒ (1) Delete 120



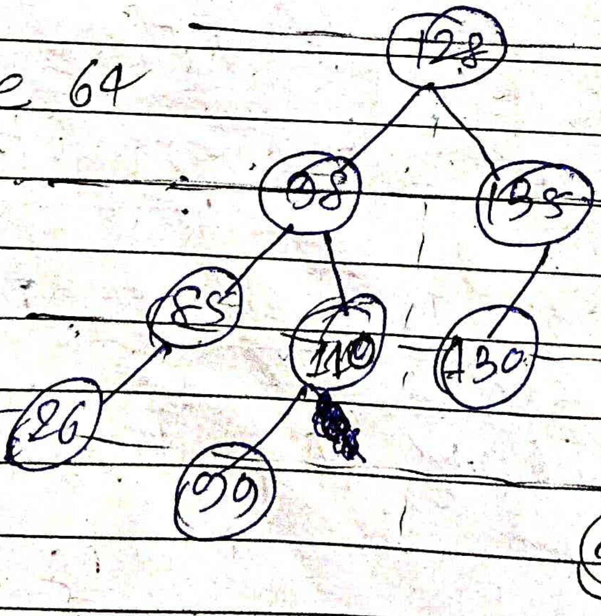
$|h(T^L) - h(T^R)|$

$= |3 - 2|$

$= 1$

No rotation is required
It is already AVL tree

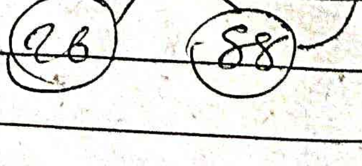
(2) Delete 64



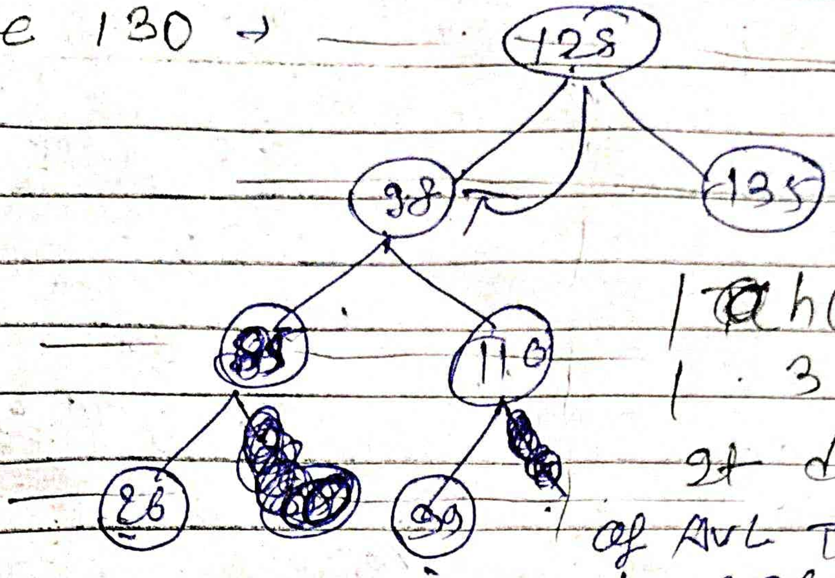
$|h(T^L) - h(T^R)|$

$|3 - 2| = 1$

No rotation is required
It is already AVL tree



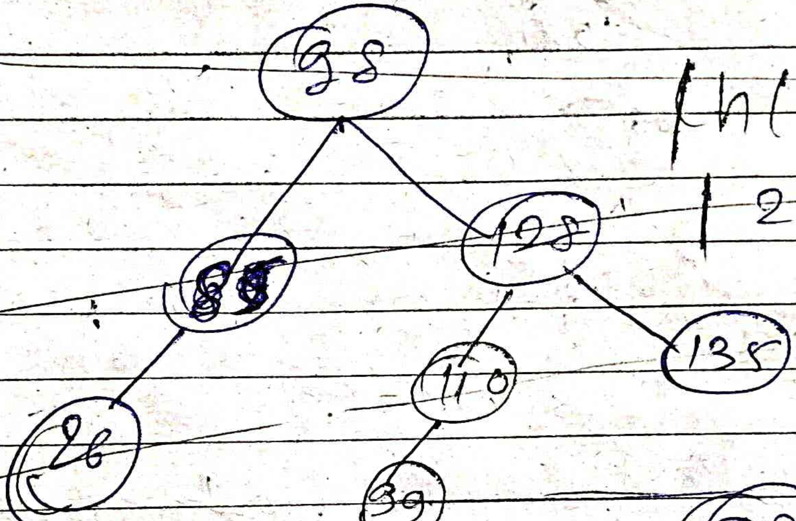
(3) Delete 130 →



$|h(T^L) - h(T^R)|$

$|3 - 1 = 2$

It does not remain of AVL tree we need to perform right rotation to balance the tree.

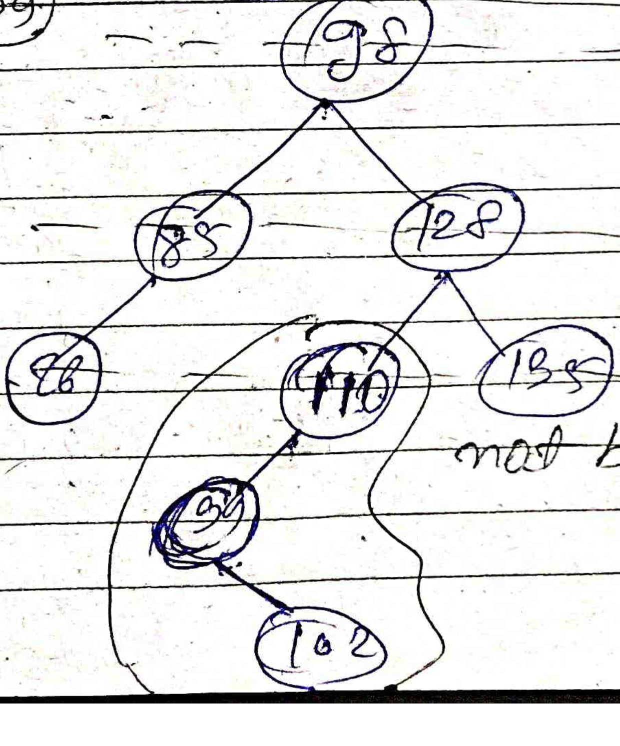


$|h(T^L) - h(T^R)|$

$|2 - 3| \Rightarrow |-1| = 1$

This tree is now balanced.

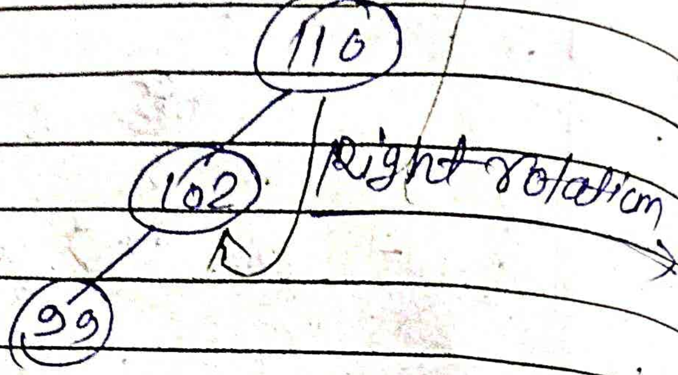
insert 102 :

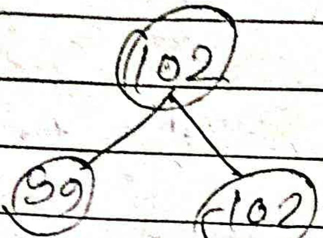


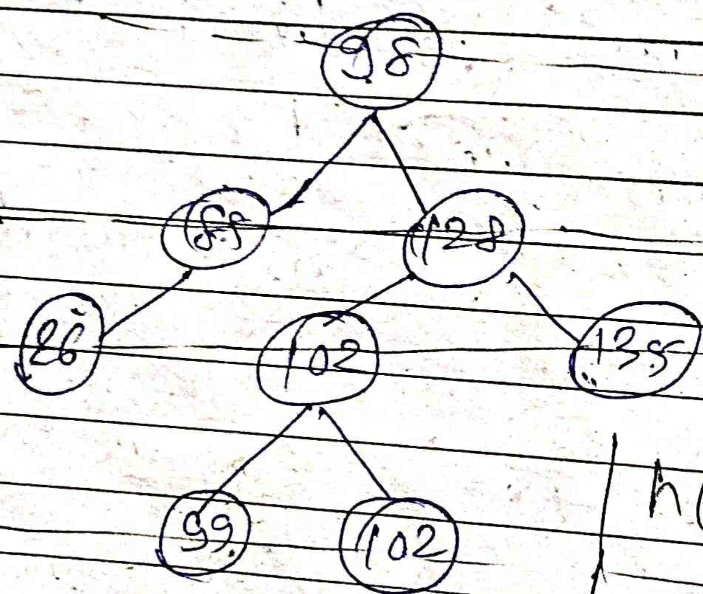
not balance

Left-Right-rotation

Left rotation







$$|h(T^L) - h(T^R)|$$

$$|2 - 3| = |-1| = 1$$

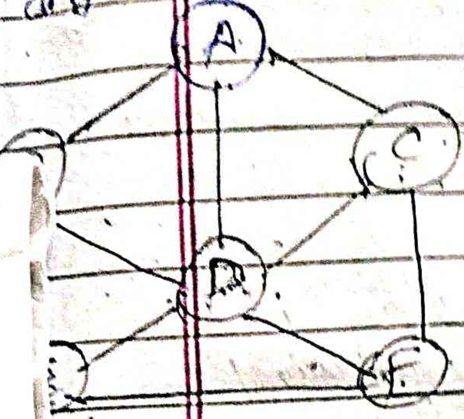
This is a balanced tree an AVL tree

practice more from magma's pdf

Date / /
Page No.

Traverse the following graph using BFS starting from node B.

ans ⇒



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

Q1	output Q2
	∅
B	BE
EDA	BED
DAF	BEDA
AFC	~~BEDAB~~
FC	BEDAF
C	BEDAFC
	$\boxed{\text{BEDAFC}}$

PYQ Illustrate the use of adjacency matrix and list to represent a graph in memory

PYQ

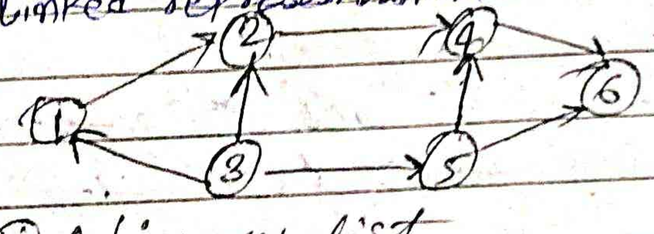
Sequential representation

(i) Adjacency matrix

	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	0	0	1	0	0
3	1	1	0	0	1	0
4	0	0	0	0	0	1
5	0	0	0	1	0	1
6	0	0	0	0	0	0

The adjacency matrix uses more space but allows for constant time to check whether edge exists b/w two vertices.

linked representation



(ii) Adjacency list

Node	Adjacency List
1	2
2	4
3	1, 2, 5
4	6
5	4, 6
6	•

The A.L. is more space efficient but requires more time for check whether edge exists b/w vertices.

(PYS) specify the situation(s) when operating system performing the garbage collection mention the steps involved in performing collection.

ans => operating system perform garbage collection in the following situation.

- when a process terminates: when a process terminates, the operating system reclaims/collect all the memory that was allocated to that process.

- when a program requests more memory: If a program requests more memory than the operating system may perform garbage collection.

steps involve in performing G.C collection

Step:1 → G.C will sequentially visit all nodes in memory and mark all nodes which are being used in program.

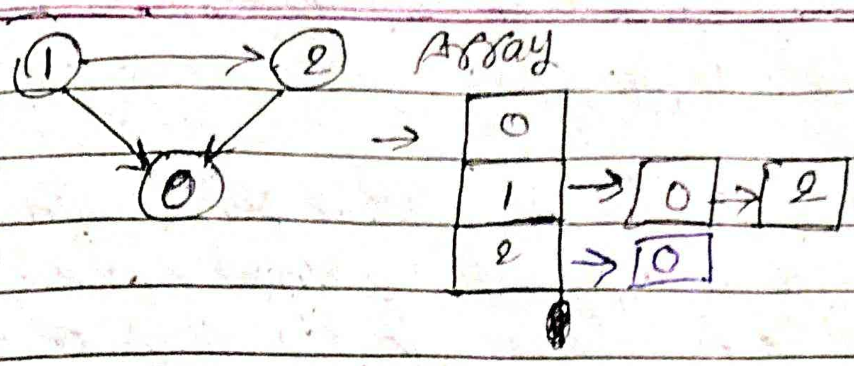
Step 2: It will collect all unmarked node and place them in free storage area.

(PYS) what is adjancemcy list?

ans => An adjancemcy list is a data structure used to represent a graph where each node in the graph stores a list of its neighboring vertices.

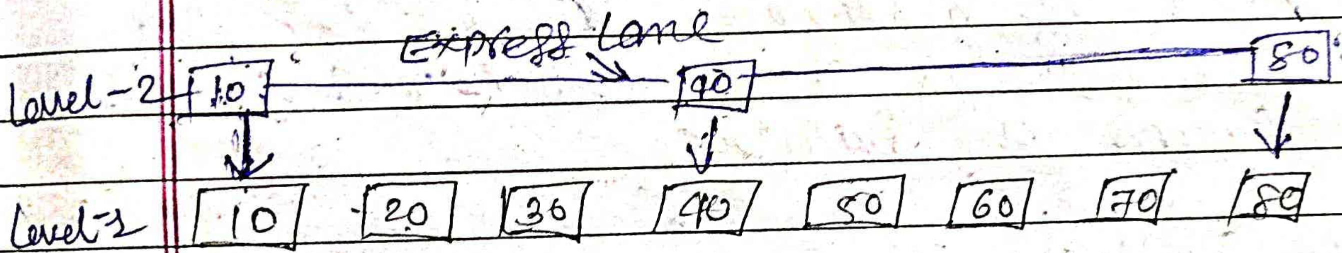
Date / /
Page No.

Ex→



(P49) what do you mean by skip list?

ans→ A skip list is a data structure that allows for efficient search, insertion and deletion of elements in a sorted list. It skips over many of items in the full list in one step, that's why It is called skip list.



(P49) why Balancing is important in Binary search Tree.
- Faster search time
- More efficient for insertion & deletion.
- Reduced memory usage.
- Time complexity O(log₂ n)

(P49) How rehashing is done?

ans→ Step 1: Create a new hash table with twice the no of buckets of the old table.

Step 2: Iterate over the elements in the old table and add them to the new table using their new hash values.

Steps 3: Discard the old table.

NAME:- RAUSHAN KU.

Rehashing is the process of increasing the storage a hashmap and redistributing the elem to new buckets.

117

* What is rehashing & when is rehashing

ans ⇒ Rehashing is a technique used in hash tables to improve performance as the number of elements in the table increases.

* when rehashing is done.
Rehashing is done when the load factor exceeds a certain threshold.

$$\boxed{\text{load factor} = \frac{\text{no of element in the table}}{\text{no of buckets}}}$$

* Need of hashing: hashing is need
ans ⇒ for index and retrieve information from a database.

PYQ Records: P Q R A B C V W X
H(K): 5 4 5 6 8 11 11 1 4

suppose the records are entered into the table T in the above order.

(a) Examine the efficiency of the given hash function with linear probing as the collision resolution technique.

(b) formulate the memory organization if the records are stored using chaining.

Date / / Page No.

(9) B Q '
V W Q P R A B X C
mlod : 1 2 3 4 5 6 7 8 9 10 11

Step: 1 $n = 9$, $m = 11$

load factor $\alpha = \frac{n}{m}$ $\alpha = \frac{9}{11}$ $\boxed{\alpha = 0.81}$

Step: 2 $S(\alpha) = ?$ $U(\alpha) = ?$

$S(\alpha) = \frac{1}{2}\left[1 + \frac{1}{(1-\alpha)}\right]$, $U(\alpha) = \frac{1}{2}\left[1 + \frac{1}{(1-\alpha)^2}\right]$

$S(\alpha) = \frac{1}{2}\left[1 + \frac{1}{1-0.81}\right]$, $U(\alpha) = \frac{1}{2}\left[1 + \frac{1}{(1-0.81)^2}\right]$

$\boxed{S(\alpha) = 3.13}$, $\boxed{U(\alpha) = 14.35}$

P Q R A B C V W X

$S = \frac{1 + 1 + 2 + 2 + 1 + 1 + 2 + 2 + 6}{n(9)}$

$S = \frac{18}{9}$ $\boxed{S = 2}$

S S(α)
2 < 3.13
efficient

U = 1 2 3 4 5 6 7 8 9 10 11

$\frac{3 + 2 + 1 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 4}{m(11)}$

$U = \frac{38}{11}$ $\boxed{U = 3.45}$

U U(α)
3.45 < 14.35 efficient

(b)

	link
1	8
2	0
3	0
4	~~8~~ 9
5	~~1~~ 3
6	4
7	0
8	5
9	0
10	0
11	~~1~~ 7

	Info	link
1	P	0
2	Q	0
3	R	1
4	A	0
5	B	0
6	C	0
7	V	6
8	W	0
9	X	~~0~~ 2
10		
11		

(c)

efficiency $n = 9, m = 11$

load factor $\alpha = \frac{n}{m}$ $\alpha = \frac{9}{11}$ $\boxed{\alpha = 0.81}$

$\boxed{S(\alpha) = 1 + \frac{1}{2}\alpha}$ $\boxed{U(\alpha) = e^{-\alpha} + \alpha}$

$S(\alpha) = 1 + \frac{1}{2} \times 0.81$, $U(\alpha) = 0.4459 + 0.81$

$\boxed{U(\alpha) = 1.25\cancel{9}}$

$S(\alpha) = 1 + 0.40$

$\boxed{S(\alpha) = 1.40}$

$S = $ A Q A A B C V W X

$\frac{2 + 2 + 1 + 1 + 1 + 2 + 1 + 1 + 1}{n(9)}$

$S = \frac{12}{9}$ $\boxed{S = 1.33}$ | $S(\alpha)$: S

$1.40 > 1.33$

efficient

Date / /
Page No.

$$U = \frac{\overset{1}{1} + \overset{2}{1} + \overset{3}{1} + \overset{4}{1} + \overset{5}{1} + \overset{6}{1} + \overset{7}{1} + \overset{8}{1} + \overset{9}{1} + \overset{10}{1} + \overset{11}{1}}{m(11)}$$

$U = \frac{11}{11}$. $\boxed{U = 1}$

U(d)		U
1.25	>	1

efficient

(PYQ) List the areas of applications of data structure.

ans ⇒
- Databases : data storage and retrieval.
- operating system
- Compiler design
- Artificial intelligence
- Game development
- Machine learning
- Blockchain

(PYQ) with the help of code snippet explain Complexity analysis.

ans ⇒
```
int main()
{ int i, n=8;            // Hello 8 times.
for (i=1; i<=n; i++)
{ cout << "Hello" << endl;
} return 0;
} // The time complexity of above code is O(n)
```
because hello is printed n times on the screen

Auxiliary space : O(1)

you can give one more example or can your own in 4 marks.

(PYQ) which data structure is ideal to perform Recursion operation and why?

ans ⇒ Stack data structure is ideal for perform recursion operation because it allow for easily ~~maintain~~ main-tain functions calls and returns. Each function call is pushed onto stack and when a function completes it is popped off the stack.

(PYQ) why it is said that searching a node in a binary search tree is efficient than that of a simple binary tree?

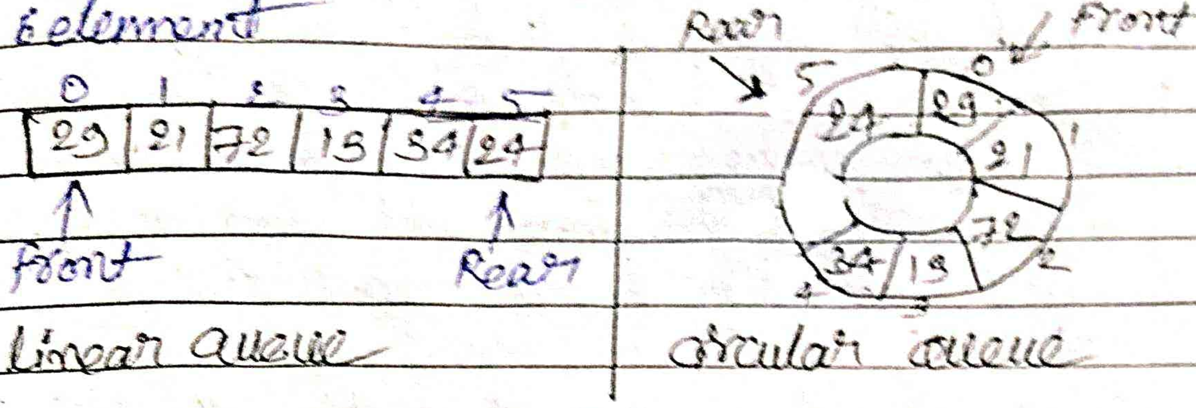
ans ⇒ Searching a node in a binary search tree (BST) is more efficient than ~~a~~ simple binary tree because a BST has a specific property: for each node, all in its left subtree have values less than the node, and all nodes in its right subtree have values greater than the node. This property allows for a more efficient search.

(PYQ) Discuss how circular queue overcomes limitations over linear queue with example:

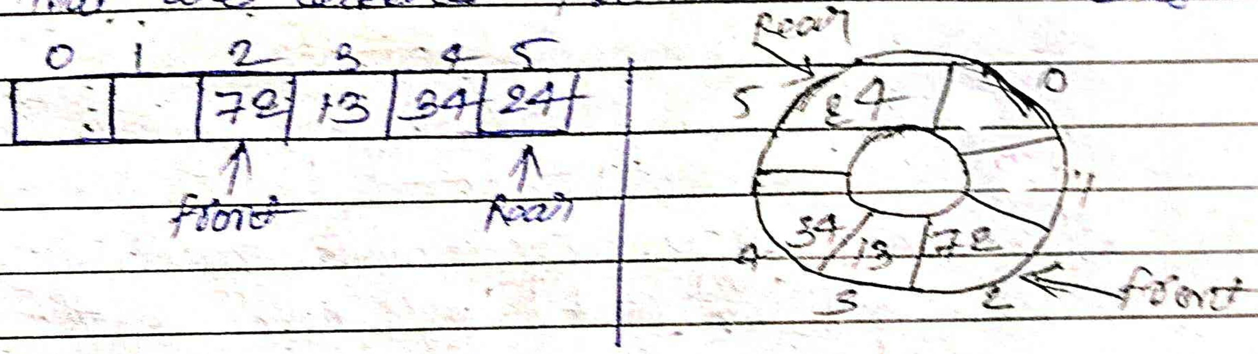
ans ⇒ A circular queue overcome limitations over linar queue by addressing the problem of wasted space at the front of the queue. In a linear queue, when elements are dequeued the front space becomes unusable.

POT example.

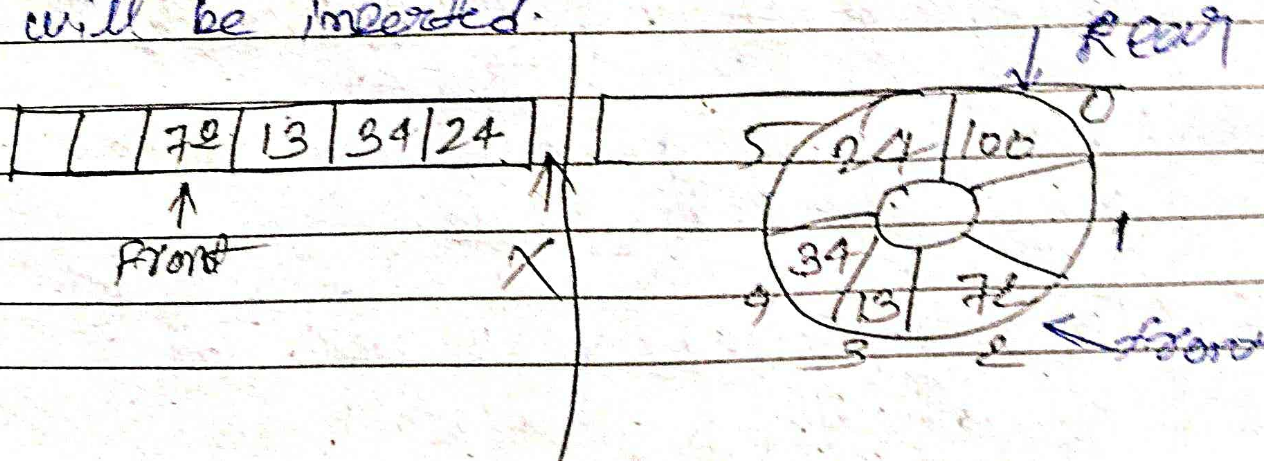
consider both linear and circular queue of size 6 element



- when the dequeue operation is performed on the both the queue's consider the first 2 elements that are deleted from bothe the queue.



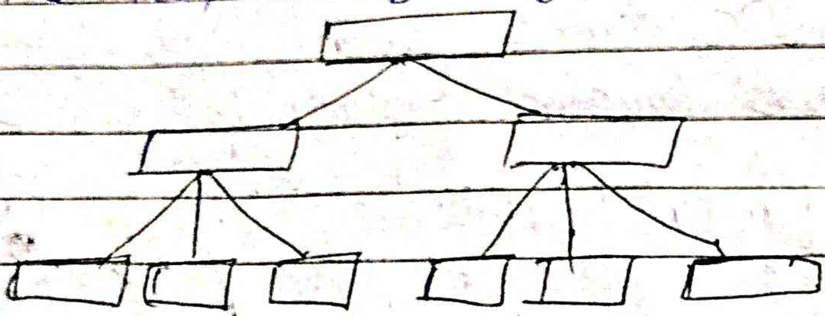
- Now enqueue operation is performed: consider an element with a value of 100, the insertion of element 100 in linear queue is not possible but in circular queue rear will be gone at the 0 th index and value will be inserted.



(Q49) write a brief Note on B-trees.
B-tree also called m-way-tree. It is a self-balanced Tree in which every node contains multiple keys and has ~~more~~ atleast two child and at the most m childs.

B-Tree of order m has the following properties

① All the leaf nodes must be at same level.



② All nodes except root node must have at least $(\lceil \frac{m}{2} \rceil - 1)$ keys and at the most $(m-1)$ key.

③ All the key value in a node must be in ~~ascen~~ ascending order.

④ B-Tree have at least 2 child nodes and at the most m child nodes.

⑤ Insertion of a node in B-Tree happens only at leaf Node.

⑥ Time complexity of B-Tree in
'searching, inserting, deleting' → O(log n)

Q10) what is hashing, need of hashing? Discuss the various hash functions.

ans ⇒ Hashing is the process of generating a fixed-size output from variable size input using mathematical formulas known as hash functions.

Need of hashing:

- Data retrieval • for index
- retrieve information from a database

hash functions

There are many hash functions

1. Division Method.
2. Mid square Method.
3. folding Method.

(i) Division method: In division method hash function divides the value K by M and then uses the remainder obtained as hash value

K = 9699

99 → Not prime
98 → Not prime
97 → prime

97) 9699 (0
960

K = 1276 M = 11

h(1276) = 1276 mod 11

1276 % 11 = 0

(ii) Mid square method: square the value of the key → K^2 and extract the middle r digits as the hash value.

Ex ⇒ K = 60, K^2 = 3600 = 60

(iii) folding method: It involve two method
(i) without reverse
(ii) with reverse

(i) Ex→ 32/05 $32+05=37$

(ii) Ex→ with reverse: 32/05 → 32/50 →
$32+50=82$

Ex (i) 7148
$71+48=\cancel{1}19=19$
Ex→ (ii) $71+84=\cancel{1}55=55$

* open ~~hashing~~ addressing is a techiniaue to resolve collision in hashing.

ans→ In linear probing the table is searched sequentially that starts from the original location of the hash if in case the location that we get is already occupied, then we check for the next location.

Ex→

Index	value

~~$H(k) = 5 \% 5 = 0$~~

Linear probing : $h, (h+1)\% N, (h+2)\% N, ---$

Ex→

A	B	C	D	E	X	Y	Z
4	8	2	11	4	11	5	1

	X	C	Z	A	E	Y		B		-	D
Mloc →	1	2	3	4	5	6	7	8	9	10	11

for E ⇒ $(h+1)\% N \Rightarrow (4+1)\% 11 = 5\% 11 = 5$

for X ⇒ $(h+1)\% N \Rightarrow (11+1)\% 11 = 12\% 11 = 1$.

for y $= (h+1)\% N \Rightarrow (5+1)\% 11 = 6\% 11 = 6$

for z $= (h+1)\% N = (1+1)\% 11 = 2\% 11 = 2$ X

$(h+2)\% N = (1+2)\% 11 = 3\% 11 = 3$

pyq Describe quadratic probing as a technique to resolve collision in hashing

ans→ Quadratic probing is a collision resolution technique in hash tables, when a collision occurs, quadratic probing searches for the next slot by using a quadratic function.

$H(k) = h, (h+1^2)\% N, (h+2^2)\% N, (h+3^2)\% N ---$

Ex→

A	B	C	D	E	X	Y	Z
4	8	2	11	4	11	5	1

NAME: RAUSHAN KU...

Date / /
Page No.

X	C		A	E	Y		B		Z	D
1	2	3	4	5	6	7	8	9	10	11

Mloc →

for E → $(h+1^2)\%N = (4+1^2)\%11$
$= 5\%11 = 5$

for X → $(h+1^2)\%N = (11+1)\%11 = 12\%11 = 1$

for Y → $(h+1^2)\%N = (5+1)\%11 = 6\%11 = 6$

for Z → $(h+1^2)\%N = (1+1)\%11 = 2\%11 = 2$ X

again $(h+2^2)\%N = (1+4)\%11 = 5\%11 = 5$ X

$(h+3^2)\%N = (1+9)\%11 = 10\%11 = 10$

* double hashing: In double hashing two hash functions are used the formula of double hashing is

$h(k) = h, (h+h')\%N, (h+2h')\%N, \ldots$