Page No.
Date

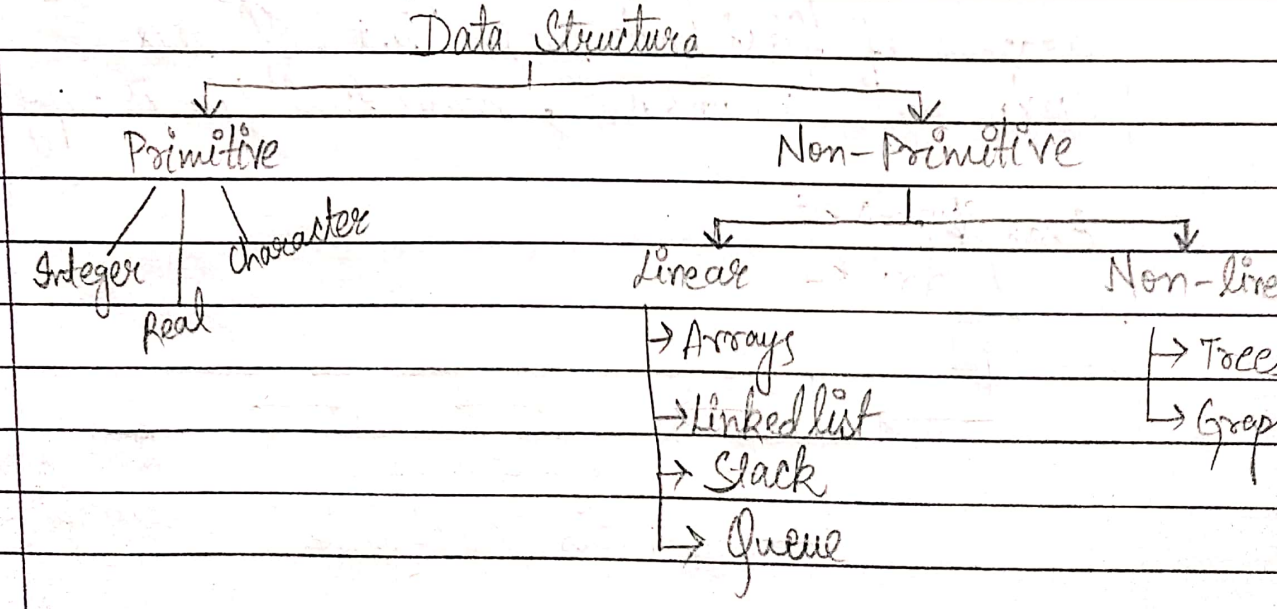
WHAT IS DATA STRUCTURES? Explain the categories of DS in detail? What are their applications?

In computer science, a data structure is a particular way of organizing data in a computer so that it can be used efficiently.

The logical or mathematical model of a particular organization of data is called a data structure.

Need of data structure

- It gives different level of organization data.
- It tells how data can be stored and accessed in its elementary level.
- Provide operations on group of data, such as adding an item, looking up highest priority item.



Date

Array : The simplest type of data structure is a linear (or one dimensional) array.

Example : A linear array A[8] consisting of numbers is pictured in following fig.

1	2	3	4	5	6	7	8

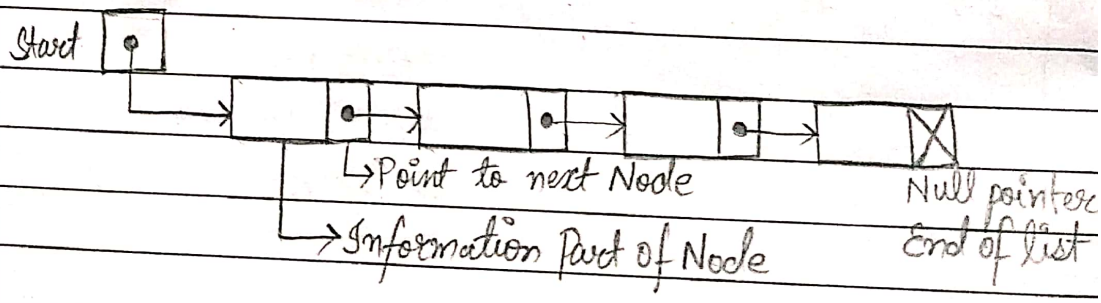
A[0] A[1] A[2] A[3] A[4] A[5] A[6] A[7]

```
int A[8] = {1, 2, 3, 4, 5, 6, 7, 8}
```

Linked List : A linked list or one way list is a linear collection of data elements, called nodes, where the linear order is given by means of pointers.

There is a special pointer Start/list contains the address of first node in the list. If this special pointer contains null, means that List is empty.

Example :



Page No.
Date

Tree: Data frequently contain a hierarchical relationship between various elements. The data structure which reflects this relationship is called a rooted tree graph or simply a tree.

Graph: Data sometimes contains a relationship between pairs of elements which is not necessarily hierarchical in nature, e.g. an airline flights only b/w the cities connected by lines. This data structure is called graph.

Queue: A queue, also called FIFO system, is a linear list in which deletions can take place only at one end of the list, the font of the list and insertion can take place only at the other end rear.

Stack: It is an ordered group of homogeneous items of elements. Elements are added to and removed from the top of the stack (the most recently added items are at the top of the stack). The last element to be added is the first to be removed (LIFO: Last In, First Out)

Date

Explain applications of data structures in detail.

Application of Arrays

- A basic application of Arrays can be storing data in tabular format.
For example, if we wish to store the contacts on our phone, then the software will simply place all our contacts in an array.

- Arrangement of leader-board of a game can be done simply through arrays to store the score and arrange them in descending order to clearly make out the rank of each player in the game.

- A simple question paper is an array of numbered questions with each of them assigned to some marks.

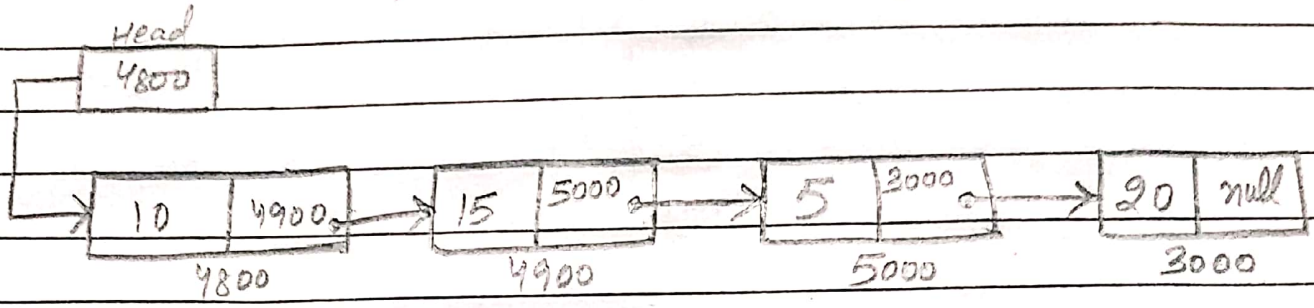
- 2D arrays, commonly known as, matrix are used in image processing.

- It is also used in speech processing, in which each speech signal is an array.

Page No.
Date

Application of Linked Lists

A linked list is a sequence data structure, which connects elements, called nodes, through links.



Images are linked with each other. So, an image viewer software uses a linked list to view the previous and the next images using the previous and next buttons.

Web pages can be accessed using the previous and the next URL links which are linked using linked list.

The music players also use the same technique to swith between music.

To keep the track of turns in a multiplayer game, a circular linked list is used.

Page No.
Date

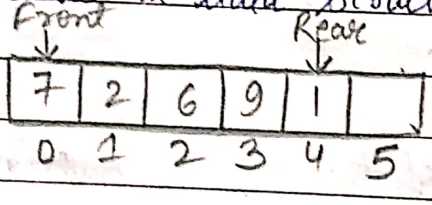
Application of Stack

A stack is a data structure which uses LIFO order.

- Converting infix to postfix expressions.
- Undo operation is also carried out through stacks.
- Syntaxes in languages are parsed using stacks.
- It is used in many virtual machines like JVM

Application of Queue

A queue is a data structure which uses FIFO order

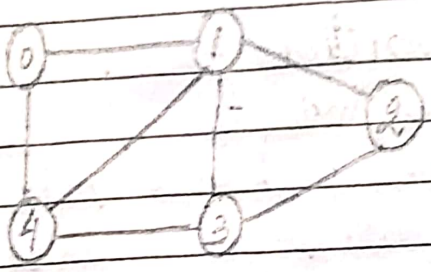


- Operating system uses queue for job scheduling.
- To handle congestion in networking queue can be used.
- Data packets in communication are arranged in queue format.

Page No.
Date

Application of graph.

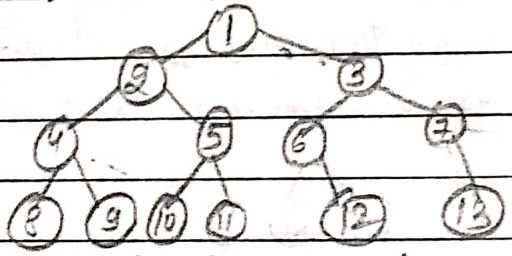
Graph is a data structure where data is stored in a collection of interconnected vertices (nodes) and edges (paths).



- Facebook's Graph API uses the structure of graphs.
- Google's Knowledge Graph also has to do something with Graph.
- Dijkstra algorithm or the shortest path first algorithm also uses graph structure to finding the smallest path between the nodes of the graph.
- GPS navigation system also uses shortest path APIs.

Applications of Tree

Trees are hierarchical structures having a single root node.



- Decision based algorithm is used in machine learning which works upon the algorithm of tree.
- Databases also uses tree data structures for indexing
- Domain Name servers (DNS) also uses tree structures

What is the difference between Primitive Data Structures and Non-Primitive Data Structures. Also Explain ADT

Primitive Data Structures

These are data structures that can be manipulated directly by machine instructions.
It is also called predefined data structures because these data structure can be handled at machine level instructions and we can't change or alter this.
In C language, the different primitive data structures are int, float, char, double.

Non-Primitive Data Structures

These are data structures that can not be manipulated directly by machine.
These are also called user-defined data structure.
Arrays, linked lists, trees, etc. are some of non-primitive data structures and are classified into linear data structures and non-linear data structures.

Page No.
Date

Abstract Data Type (ADT)

The design of a data structure involves more than just its organization.
The notion of a data structure in the abstract needs to be treated differently from what ever is used to implement the structure. The abstract notion of a data structures is defined in terms of the operations we plan to perform on the data.
Considering both the organization of data and the expected operations on the data, leads to the notion of an abstract data type. An abstract data type is a theoretical construct that consists of data as well as the operations to be performed on the data while hiding implementation.

For example, a stack is a typical abstract data type. Items stored in a stack can only be added and removed in certain order — the last item added is the first item removed. These operations are called pushing and popping.

Another example, the code to read from a keyboard is an ADT. It has a data structure, character and set of operations that can be used to read that data structure.

Date

What are pointer variables ? Which type of role play by pointer in data structures.

A pointer is a variable whose value is the address of another variable i.e., direct address of the memory location. Like any variable or constant, you must be declared a pointer before using it to store any variable address. The pointer might be belonging to any of the data type such as int, float, char, double, short, etc.

Need of pointers in Data Structures

Optimization of our code and improving the time complexity of one algorithm. Since using pointers helps to reduce the time needed by an algorithm to copy data from one place to another. Since it used the memory location directly, any change made to the value will be reflected at all the locations.

Another use of pointers is to control the program flow. This is implemented by control tables that use these pointers. These pointers are stored in a table to point to the entry point of each subroutine to be executed one after the other. These pointers reference the address of the various procedures. This helps while working with a recursive procedure or traversal of algorithms where there

is a need to store the location of the calling step.

The next need of pointers arises in the case of various secondary data structures such as linked lists or structures to point to the next memory locations in the list.

Explain Stacks & Queues with the help of operations that can be performed on them. Also mention their applications.

Stack : It is a linear data structure which follows a particular order in which the operations are performed.
The order may be LIFO (Last In First Out) or FILO

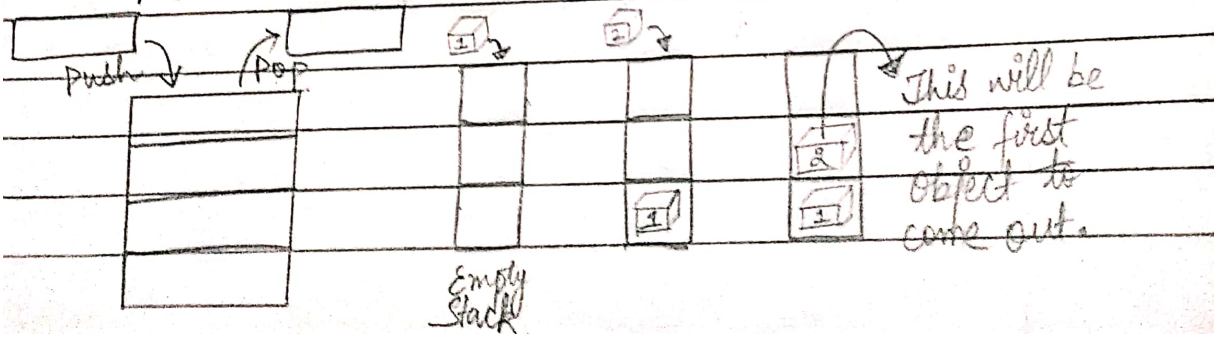
Mainly the following three basic operations are performed in the stack:

Push – Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.

Pop – Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.

Peek or Top – Returns top element of stack.

IsEmpty – Returns true if stack is empty, else false.



Applications of stack

- Expression Handling
  - ★ Infix to Postfix & Infix to Prefix Conversion
    The stack can be used to convert some infix expression into its postfix equivalent, or prefix equivalent.
  - ★ Postfix or Prefix Evaluation
    After converting into prefix or postfix notations, we have to evaluate the expression to get the result.
- Backtracking Procedures
  It is one of the algorithm designing technique.
- Another great use of stack is during the function call and return process.

Queue : It is a linear structure which follows a particular order in which the operations are performed.
The order is First In First Out (FIFO)

The difference b/w stacks and queues is in removing.
In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

Page No.
Date

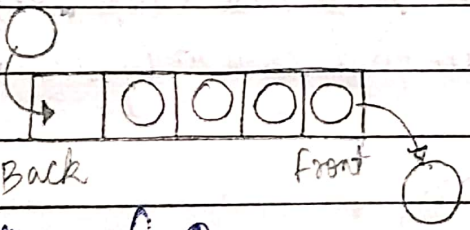
Basic Operations performed in Queue:

Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations of queue:

enqueue() - add (store) an item to the queue.
dequeue() - remove (access) an item from the queue.

peek() - Gets the element at the front of the queue without removing it.
isfull() - Checks if the queue is ~~empty~~ full.
isempty() - Checks if the queue is empty

In queue, we always dequeue (or access) data, pointed by front pointer and while enqueing (or storing) data in the queue we take help of rear pointer.



Application of Queue

- Waiting in line
- Waiting on hold for tech support
- Round Robin scheduling
- Job scheduling (FIFO Scheduling)
- Key board buffer.

Page No.
Date

What is garbage Collection?

Garbage Collection is a dynamic approach to automatic memory management and heap allocation that processes and identifies dead memory blocks and reallocates storage of reuse.
The garbage collector, attempts to reclaim garbage or memory occupied by objects that are no longer in use by the program.

What are arrays? Explain with the help of example.

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value i.e., the memory location of the first element of the array.

How to declare an array?

datatype arrayname [arraySize];

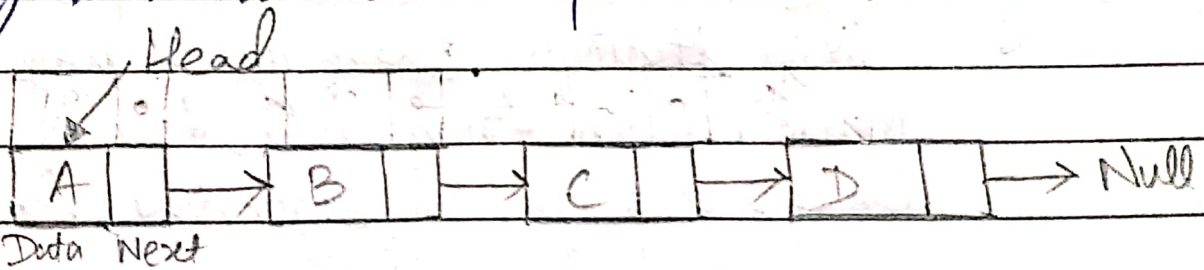
Example: float mark [5];
Here, we declared an array, mark, of floating-point type. And its size is 5. Meaning, it can hold 5 floating-point values.

int mark[5] = {19, 10, 8, 17, 9};

mark[0]	mark[1]	mark[2]	mark[3]	mark[4]
19	10	8	17	9

What is linked list? Explain with an example.

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. In simple 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. The first node is called a head and if the list is empty then the value of head is NULL.



What is the difference between static and dynamic allocation?

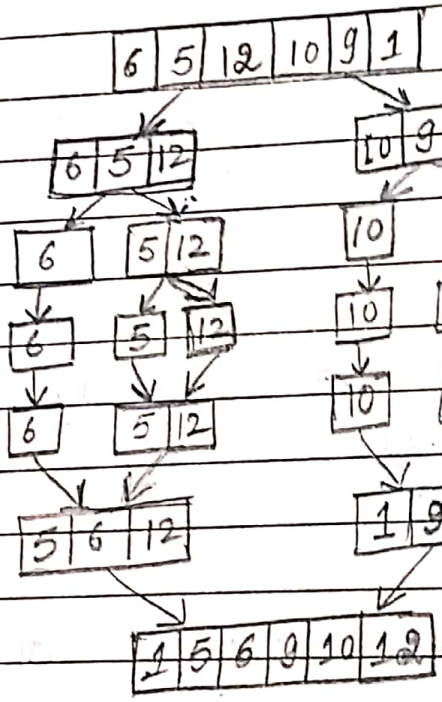
Static Memory Allocation	Dynamic Memory Allocation
• In this case, variables get allocated permanently	• In this case, variables get allocated only if your program unit gets active.
• Allocation is done before program execution.	• Allocation is done during program execution.
• It uses the data structure called stack for implementing static allocation.	• It uses the data structure called heap for implementing dynamic allocation.
• There is no memory reusability	• There is memory reusability and memory can be freed when not require
• Less efficient	• More efficient

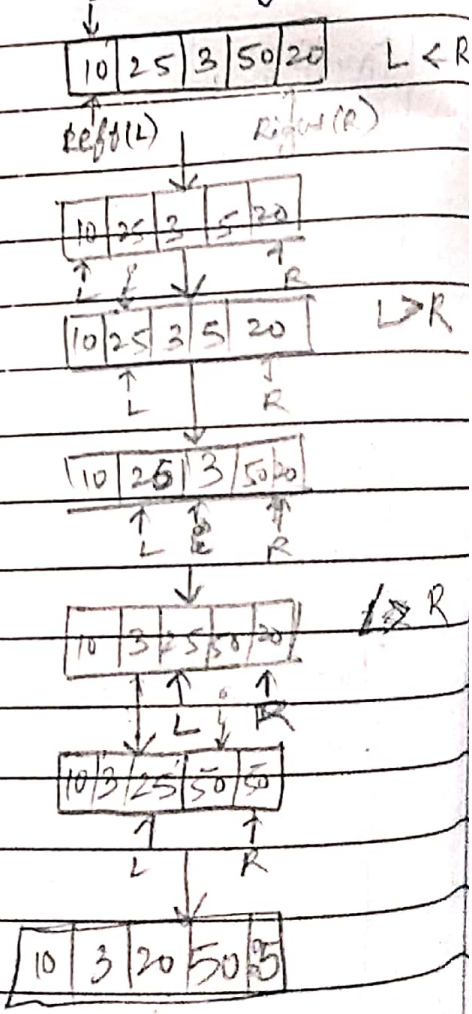
Page No.
Date

Basis of Comparison	Merge Sort	Quick Sort
Definition	It is a sorting technique based on divide and conquer technique. It first divides the array into two equal halves and then combines them in a sorted manner.	It is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. It is partitioned into 2 arrays one of which holds values smaller than the specified values, say pivot, based on which the partition is made & another array holds values greater than the pivot value.
Partition of elements in array	In merge sort, the array is parted into just two halves (i.e., n/2)	In quick sort, the array is parted into any ratio. There is no compulsion of dividing the array of elements into equal parts.
Worst-Case Complexity	O(n log n)	(O n2)
Usage with datasets	It can work well on any type of data sets irrespective of its size.	It can not work well with large datasets.

Efficiency	It is more efficient and works faster than quick sort in case of larger array size or datasets.	It is more efficient & works faster than merge sort in case of smaller array size or datasets.
Stability	It is stable as two elements with equal value appear in the same order in sorted output as they were in the input unsorted array.	It is unstable in this scenario. But it can be made stable using some changes in the code.
Preferred for	It is preferred for linked list.	It is preferred for arrays.

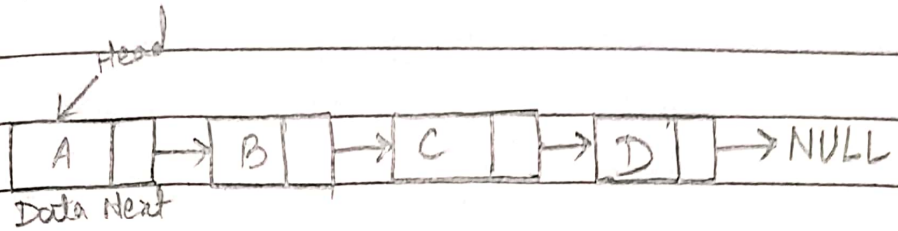
Example:





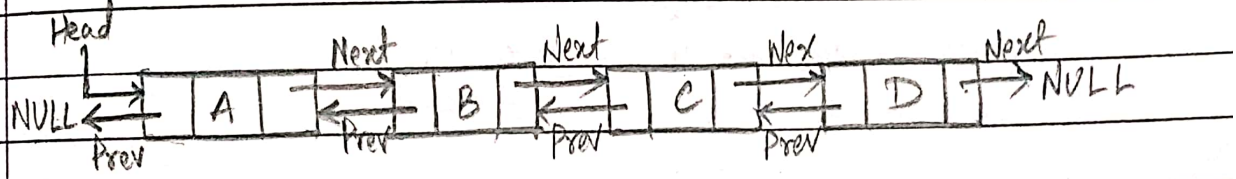
Page No.
Date

Linked List : It is a set of nodes where each node has two fields "data" and "link". The data field stores actual piece of information and "link" field is used to point to next node. Basically "link" field is nothing but address only.
It has nodes with only a data field. and next linked field.



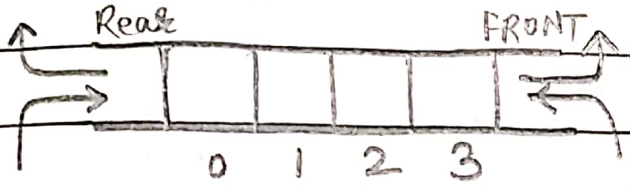
It occupies less memory than DLL at it has only two fields.
Less efficient access to elements.

Doubly Linked List : It contains an extra pointer, typically called previous pointer, together with next pointer and data which are there in linked list.
It has nodes with a data field, a previous link field and a next link field.



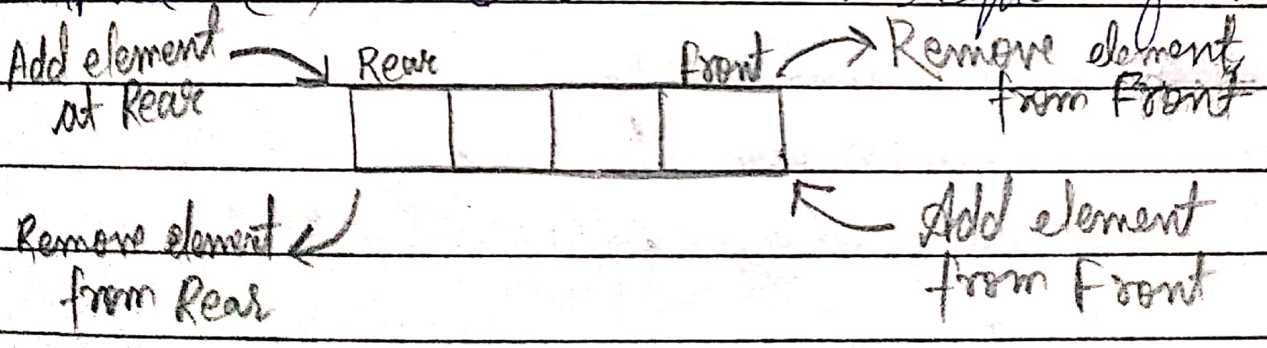
In doubly linked list, the traversal can be done using the previous node link or the next node link.
It occupies more memory than linked list as it has 3 fields.
More efficient access to elements.

Deque is a generalized version of queue data structure that allows insertion and deletion at both ends.
It is also known as double ended queue.
It can be used both as stack and queue as it allows the insertion and deletion operations on both ends.



Operations on Deque:

insertFront( ): Adds an item at the front of Deque.
insertRear( ): Adds an item at the rear of Deque.
deleteFront( ): Deletes an item from front of Deque.
deleteRear( ): Deletes an item from rear of Deque.
getFront( ): Gets the front item from queue.
getRear( ): Gets the last item from queue.
isEmpty( ): Checks whether Deque is empty or not.
inFull( ): Checks whether Deque is full or not.



// Deque implementation.

```
#include <stdio.h>
#define Max 10
void addfront(int*, int, int*, int*);
void addRear(int*, int, int*, int*);
int delfront(int*, int*, int*);
int delRear(int*, int*, int*);
void display(int*);
int main()
{
  int arr[MAX];
  int front, rear, i, n;
  front = rear = -1;
  for(i=0; i<MAX; i++)
    arr[i] = 0;
addRear(arr, 5, &front, &rear);
addfront(arr, 12, &front, &rear);
addRear(arr, 11, &front, &rear);
addFront(arr, 5, &front, &rear);
addRear(arr, 6, &front, &rear);
addfront(arr, 8, &front, &rear);
printf("\n Elements in a deque: ");
display(arr);
i = delfront(arr, &front, &rear);
printf("\n remove item: %d", i);
printf("\n Elements in a dequeu after deletion: ");
display(arr);
addRear(arr, 16, &front, &rear);
addRear(arr, 7, &front, &rear);
```

```
printf("\n Elements in a deque after addition:");
display(arr);
insertRear(arr, &front, &rear);
printf("\n Inserted Elem %d", i);
printf("\n Elements in a deque after deletion:");
display(arr);
n = count(arr);
printf("\n Total number of elements in deque: %d", n);
}
void addfront(int *arr, int item, int *pfront, int *prear)
{ int i, k, c;
if (*pfront == 0 && *prear == MAX-1)
{ printf("\n Deque is full. \n");
  return;
}
if (*pfront == -1)
{ *pfront = *prear = 0;
  arr[*pfront] = item;
  return;
}
if (*prear != MAX-1) {
  c = count(arr);
  k = *prear + 1;
  for(i=1; i<=c; i++)
  { arr[k] = arr[k-1];
    k--; }
  arr[k] = item;
  *pfront = k;
  (*prear)++;
}
```

```
else
{ (*pfront);
  arr[*pfront] = item;
}
}
void addfront(int *arr, int item, int *pfront, int *prear)
{ int i, k;
  if (*pfront == 0 && *prear == MAX-1)
  { printf("\n Deque is full.\n");
    return; }
  if (*pfront == -1) {
    *prear = *pfront = 0;
    arr[*prear] = item;
    return; }
  if (*prear == MAX-1) {
    k = *pfront - 1;
    for (i = *pfront-1; i < *prear; i++)
    { k = i;
      if (k == MAX-1)
        arr[k] = 0;
      else
        arr[k] = arr[i+1]; }
    (*prear)--;
    (*pfront)--;
  }
  (*prear)++;
  arr[*prear] = item;
}
int delfront(int *arr, int *pfront, int *prear)
{ int item;
```

```
if (*pfront == -1)
{ printf("\n Deque is empty.\n");
  return 0; }
item = arr[*pfront];
arr[*pfront] = 0;
if (*pfront == *prear)
 *pfront = *prear = -1;
else
 (*pfront)++;
 return item; }
int delRear (int *arr, int *pfront, int *prear)
 { int item;
if (*pfront == -1)
 { printf ("\n: Deque is empty.\n");
   return 0; }
item = arr[*prear];
arr[*prear] = 0;
 (*prear)--;
 if (*prear == -1)
  *pfront = -1;
  return item; }
void display (int *arr)
 { int i;
printf ("\n front : ");
for (i=0; i<MAX; i++)
  printf ("%d", arr[i]);
  printf ("  : rear"); }
int count (int *arr) {
  int c=0, i;
for (i=0; i<MAX; i++)
 { if (arr[i] != 0)
     c++; }
 return c; }
```

Date

Dynamic Memory Allocation: It is when an executing program requests that the operating system give it a block of main memory. The program then uses this memory for some purpose. Usually the purpose is to add a node to a data structure. In object oriented languages, dynamic memory allocation is used to get the memory for a new object.

The memory comes from above the static part of the data segment. Programs may request memory & may also return previously dynamically allocated memory. Memory may be returned whenever it is no longer needed. Memory can be returned in any order without any relation to the order in which it was allocated. The heap may develops holes where previously allocated memory has been returned between blocks of memory still in use.

A new dynamic request for memory might return a range of addresses out of one of the holes. But it might not use up all the hole, so further dynamic requests might be satisfied out of the original hole.

Page No.
Date

Evaluate prefix.

1) Put the pointer P at the end of the end.
2) If character at P is an operand push it to stack.
3) If the character at P is an operator pop two elements from the stack. Operate on these elements according to the operator, and push the result back to the stack.
4) Decrement P by 1 and go to point (3) as long as there are characters left to be scanned in the expression.
5) The result is stored at the top of the stack, return it.
6) End

Evaluate Postfix

1) Create a stack to store operands or values.
2) Scan the given expression and do the below operations for every scanned element:
   → If the element is a number, push it into stack.
   → If the element is a operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack.
3) When the expression is ended, the no. in the stack is the final answer.

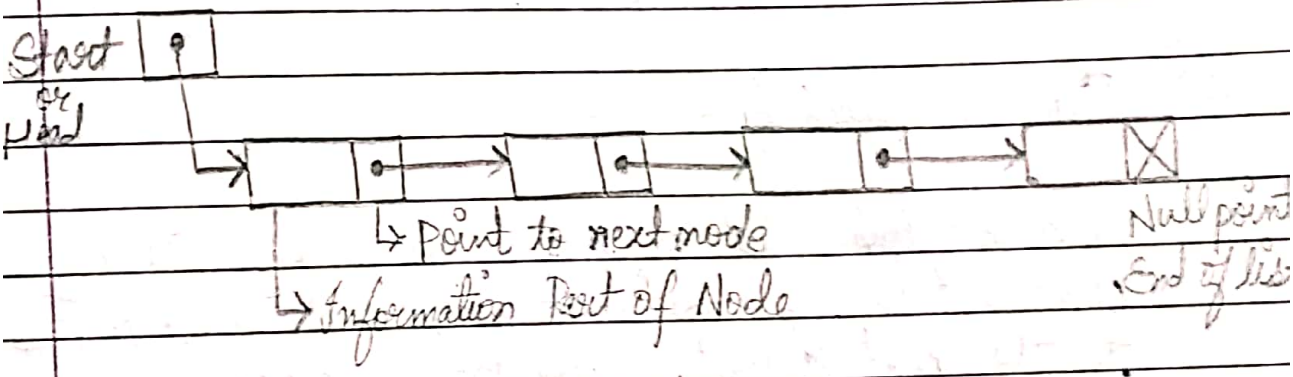
Page No.
Date

Linked List

A linked list is a linear data structure or linear collection of data elements whose order is not given by their physical placement in memory. Instead, each element points to the next.

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

There is a special pointer start / list containing the address of first node in the list. If this special pointer contains null, means that list is empty.



Some terms used in linked list:

Link : Each link of a linked list can store a data called an element.

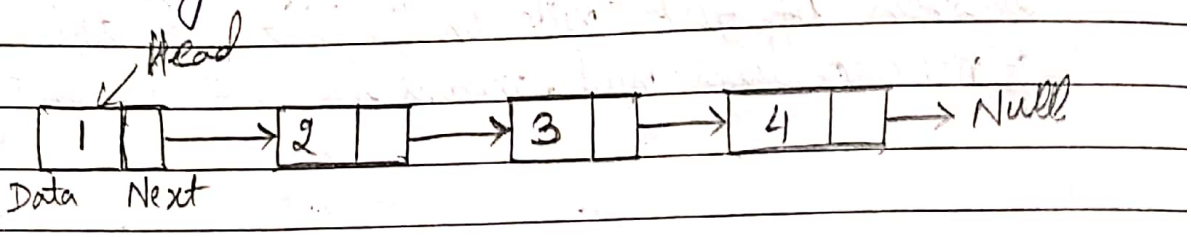
Next : Each link of a linked list contains a link to the next link called Next.

Linked list : A linked list contains the connection link to the first link called first.

Types of Linked List

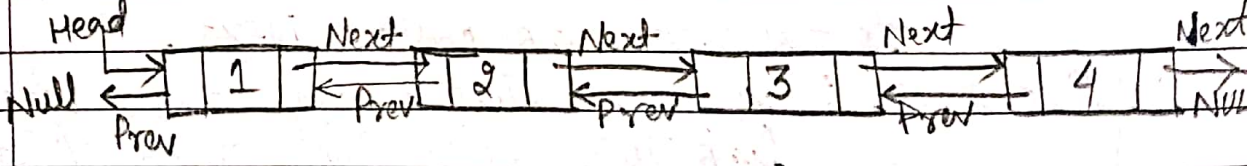
i) Simple Linked list / Singly Linked List

It is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type. The node contains a pointer to the next node means that the node stores the address of the next node in the sequence.
A single linked list allows traversal of data only in one way.



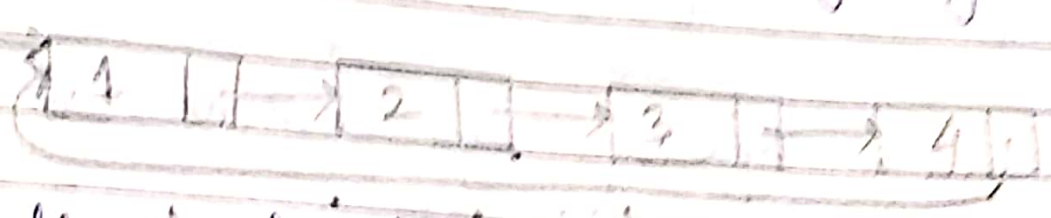
ii) Doubly Linked List

A doubly linked list or a two-way linked list is more complex type of linked list which contains a pointer to the next as well as the previous node in sequence. Therefore, it contains 3 parts i.e., data, a pointer to the next node and a pointer to the previous node. This would enable to traverse the list in the backward direction as well.



iii) Circular linked list

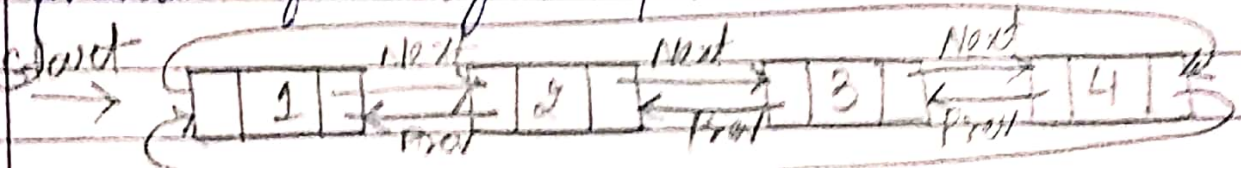
A circular linked list is that in which the last node contains the pointer to the first node of the list. While traversing a circular linked list, we can begin at any node and traverse the list in any direction forward & backward until we reach the same node we started. Thus a circular linked list has no beginning and no end.



iv) Doubly circular linked list

A doubly circular linked list or a circular 2-way linked list is a more complex type of linked list that contains a pointer to the next as well as the previous node in the sequence.

The difference b/w the doubly linked list & circular doubly linked list is the same as that b/w a singly linked list and a circular linked list. The circular doubly linked list does not contain null in the previous field of the first node.



Following are the basic operations supported by a list

Insertion - Adds an element at the beginning of the list

Deletion - Deletes an element at the beginning of the list.

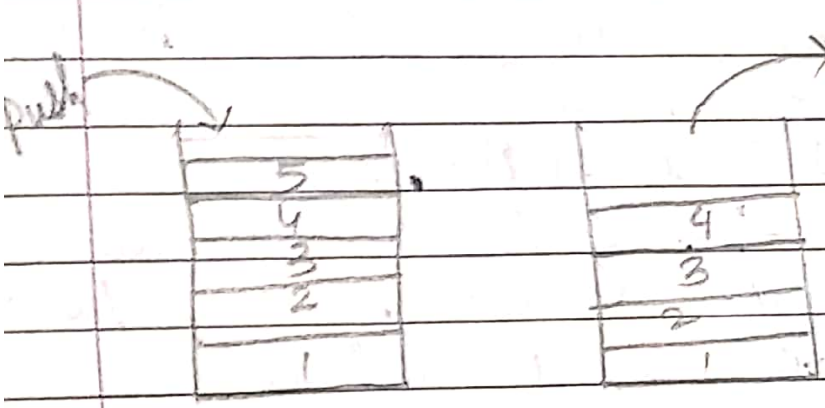
Display - Displays the complete list.

Search - Searches an element using the given key.

Delete - Deletes an element using the given key.

Stack: It is a linear data structure which follows a particular order in which the operations are performed.
The order may be LIFO (Last In First Out) or FILO (First In First Out)

Mainly the following basic operations are performed in the stack:
Push: Adds an element / item in the stack. If the stack is full then it is said to be an overflow condition.
Pop: Removes an item / element from the stack. The items are popped in reversal order in which they are pushed. If the stack is empty then the condition is underflow.



Stack: Last In First Out

A stack can be implemented by means of Array, structure, pointer and linked list.
Stack can either be a fixed size one or it may have a sense of dynamic resizing

To use a stack efficiently, we need to check the status of stack as well. For this, following functionality is added to stacks:

peek() - get the top data element of the stack, without removing it.

isEmpty() - Check if stack is empty.

isFull() - Check if stack is full.

Every time we maintain a pointer to the last Pushed data on the stack. As this pointer always represents the top of stack. Top points provides top value of the stack without removing it.

Now, here are the following steps involving pop operation:

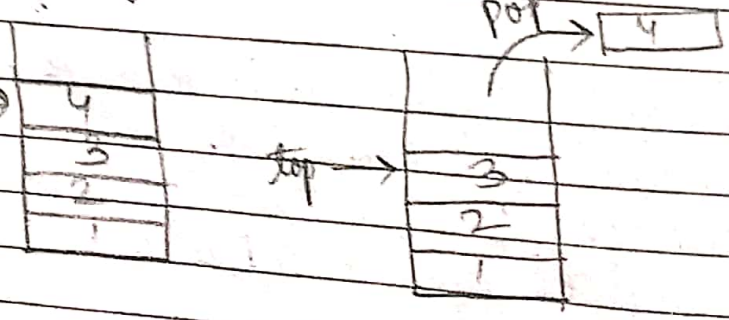
Step 1: Check if stack is empty

Step 2: If the stack is empty, produces an error and exit.

Step 3: If the stack is not empty, accesses the data element at which top is pointing.

Step 4: Decreases the value of top by 1.

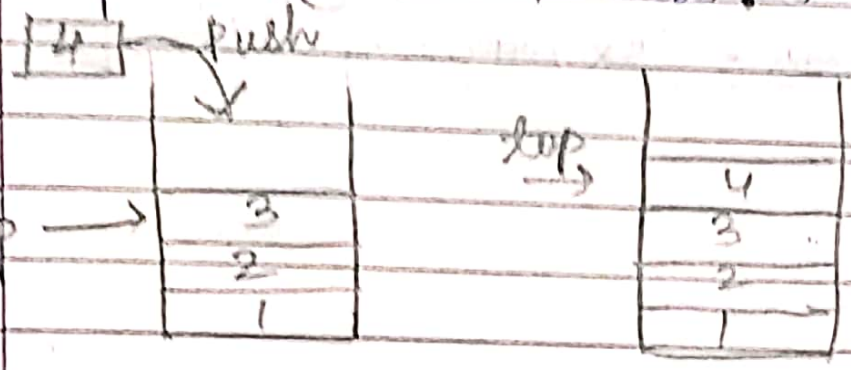
Step 5: Return success.



Page No.
Date

Now, following steps involved in push operation:

Step 1: Check if the stack is full.
Step 2: If the stack is full, produces an error of overflow condition and exit.
Step 3: If the stack is not full, increments top to point next empty space.
Step 4: Adds data element to the stack location, where top is pointing.
Step 5: Returns success.

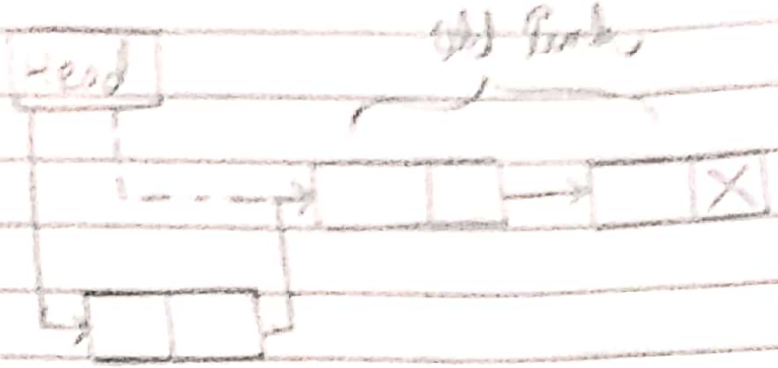


Applications of stack

i) Expression handling: The stack can be used to convert some of infix to postfix or postfix to infix expression. As well as infix to prefix & prefix to infix expression.

ii) Backtracking Procedure: It is one of the algorithm designing technique.

iii) Stack is also used during the function call and return process.



Insertion in linked list

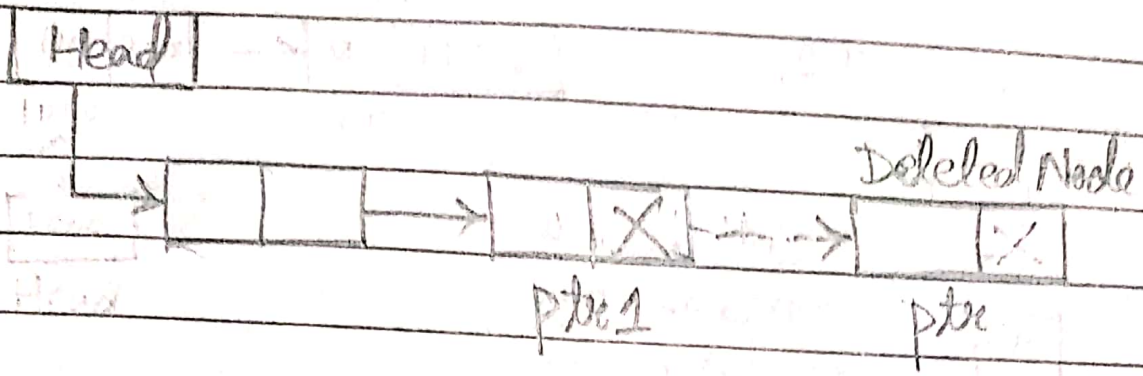
Head is the pointer pointing to the old link.
Pointer is the pointer pointing to the new link.

Allocate the space for the new node and store data into the data part of the node.
Make the link part of new node pointing to the existing first node of the list.
At the last, we need to make new node as the first node of the list.

Step 1 : If Ptr = NULL
Overflow
Step 2 : Set New_Node = Ptr
Step 3 : Set Ptr = Ptr → Next
Step 4 : Set New_Node → Data = VAL
Step 5 : Set New_Node → Next = Head
Step 6 : Set Head = New_Node
Step 7 : Exit

Page No.
Date

Deletion in linked list

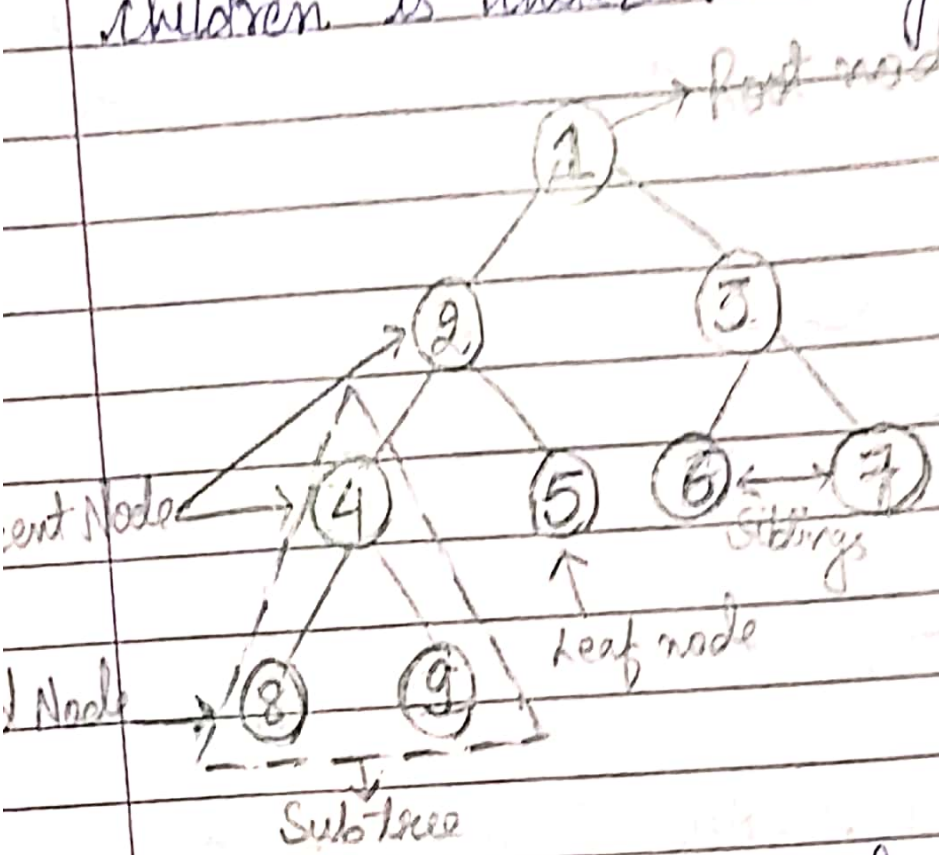


Head is the pointer pointing to the previous list

Step 1: If Head = NULL
Underflow
Step 2: Set PTR = Head
Step 3: Repeat steps (4) and (5)
while PTR → Next != NULL
Step 4: Set PREPTR = PTR
Step 5: Set PTR = PTR → Next
Step 6: Set PREPTR → Next = NULL
Step 7: Free PTR
Step 8: Exit

Binary Tree

A tree whose elements have atmost two children is called a binary tree.



A node in binary tree has 0, 1 or 2 successor

A parent can have atmost Two children i.e., left child, right child.

The data of left child is less than the parents data / root node's data.

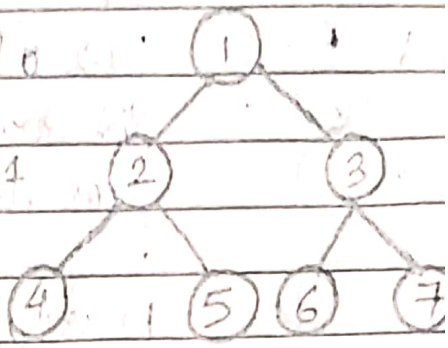
The data of right child is greater than the parent's data / root node's data.

Nodes having the same parent are called siblings.

Page No.
Date

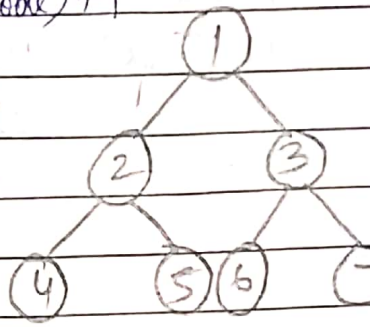
Properties of binary tree

(i) No. of internal nodes = (leaf node - 1)



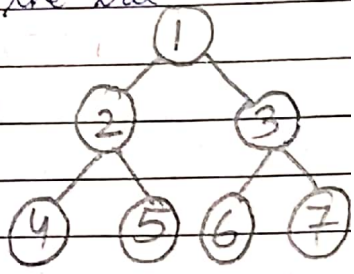
In this figure:
Leaf node = 4
Therefore,
No. of internal nodes = (leaf node - 1)
= 3

(ii) Total number of nodes = (2 * internal node) + 1



In this figure;
Internal node = 3
Therefore,
Total no. of nodes = (2*3) + 1
= 7

(iii) Max number of nodes = $(2^{H+1} - 1)$
where; H is the height of the tree
So, height of the tree = 2
⇒ Maximum no. of nodes = $(2^{H+1} - 1)$
$= 2^{2+1} - 1$
$= 7$



(iv) Minimum no. of nodes = (H+1)
⇒ Minimum no. of nodes = 3

(v) Maximum number of nodes at any level 'L' = 2^L
⇒ If level, L = 3
⇒ Max. no. of nodes = $2^3 = 8$

Page No.
Date

Algorithm: In CS, whenever we want to solve some computational problem then we define a set of steps that need to be followed to solve that problem. These steps are known as algorithm.

Properties of good algorithm:

- Correctness: An algorithm is said to be correct if for every set of input it halts with the correct output. If you are not getting the correct output for any particular set of input, then your algorithm is wron
- Finiteness: Generally, people ignore but it is one of the important factors in algorithm evaluation. The algorithm must always terminate after a finite numbe of steps. For example, in the case of recursion & loop, your algorithm should terminate otherwise you will end up having a stack overflow and infinite loop. scenario respectively.
- Efficiency: An efficient algorithm is always used. By the term efficiency, we mean to say the

i) The algorithm should efficiently use the resources available t the system.
ii) The computational time should be as less as possible.
iii) The memory used by the algorithm should also be as less as possible.

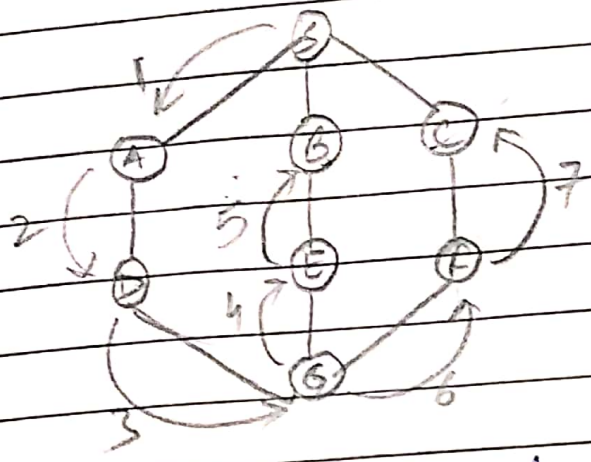
Page No.
Date

The efficiency of an algorithm is mainly defined by two factors i.e., space and time. A good algorithm is one that is taking less time and less space, but it is not possible all the time. There is trade-off b/w time and space. If you want to reduce the time, the space might increase. Similarly, if you want to reduce the space, then the time may increase. So, you have to compromise with either space or time.

Space Complexity: It is an algorithm denotes the total space used or needed by the algorithm for its working, for various input sizes. When you are creating a variable then you need some space for your algorithm to run. All the space required for the algorithm is collectively called the space complexity of the algorithm.

Time Complexity: It is the number of operations an algorithm perform to complete its task with respect to input size. The algorithm that performs the task in the smallest number of operations is considered the most efficient one.

Depth first search algorithm traverse a graph in a depthward motion & uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



In this example; DFS algorithm traverse from S to A to D to G to E to B first, then to F and lastly to C. It employs the following rules.

Rule 1: Visit the adjacent unvisited vertex. Mark it visited. Display it. Push it in stack.

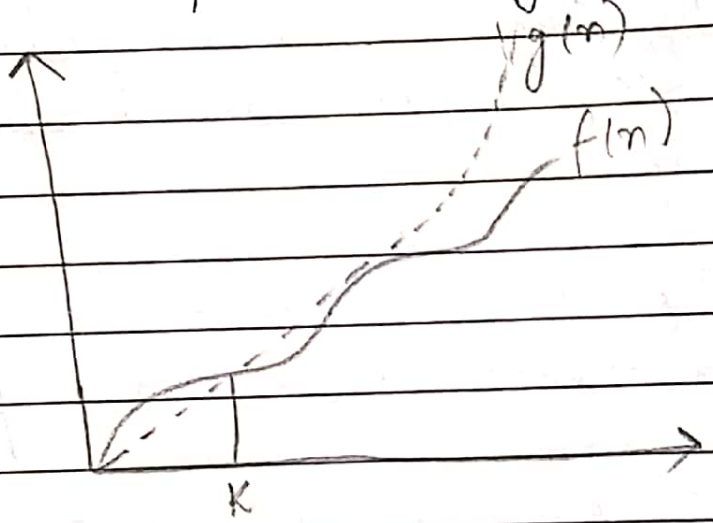
Rule 2: If no adjacent vertex is found, pop up a vertex from the stack (It will pop up all the vertices from the stack, which do not have adjacent vertices).

Rule 3: Repeat Rule 1 and Rule 2 until the stack is empty.

Page No.
Date

Big O notation is a mathematical notation that describes the limiting behaviour of a function when the argument tends towards a particular value or infinity. It is a member of a family of notations.

The notation O(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complet



for example, for a function f(n)

$O(f(n)) = \{g(n) : \text{there exists } c>0 \text{ and } n_0 \text{ such that } f(n) \leq c.g(n) \text{ for all } n > n_0\}$

Page No.
Date

Heap data structure is generally taught with heapsort. Heapsort algorithm has limited uses because quicksort is better. Heap data structure itself is enormously used. Following are the applications of Heap

i) Priority Queues: It can be efficiently implemented using binary heap because it supports insert(), delete() & extractmax(), decreasekey() operations in O(logn) time.

ii) Order statistics: The heap data structure can be used to efficiently find the kth Smallest element in an array.

iii) It is used in many famous algorithms such as Dijkstra's algorithm for finding the shortest path.

iv) Heaps are also used when we want to be able to access the maximum or minimum element very quickly.

v) Apart from needing the extra storage for the heap, heapsort is among efficient sorting algorithms.

File : It is a collection of records related to each other. The file size is limited by the size of memory and storage medium.

These are two important features of file:

→ File activity : It specifies percent of actual records which proceed in a single run.

→ File volatility : It addresses the properties of record changes. It helps to increase the efficiency of disk design than type.

File Organization : It ensures that records are available for processing. It is used to determine an efficient file organization for each base relation. For example, if we want to retrieve employee records in alphabetical order of name. Sorting the file by employee name is a good file organization. However, if we want to retrieve all employees whose marks are in a certain range, a file is ordered by employee name would not be a good file organization.

There are three types of organizing the file:

→ Sequential access file organization.

Storing & sorting in contiguous block within files on tape or disk is called as sequential access file organization.

In sequential access file organization, all records are stored in a sequential order. The records are arranged in the ascending or descending order of a key field.

→ Direct access file organisation
It is also known as random access or relative file organization. In direct access file, all records are stored in direct access storage device, such as hard disk. The records are randomly placed throughout the file. The records does not need to be in sequence because they are updated directly and rewritten back in the same location. The file organisation is useful for immediate access to large amount of information. It is used in accessing large databases. It is also called hashing.

→ Indexed sequential access file organisation
It combines both sequential file and direct access file organisation. In indexed sequential access file, records are stored randomly on a direct access devices such as magnetic disk by a primary key. This file have multiple keys. The data can be read either sequentially or randomly using the index. The index is stored in a file & read into memory when the file is opened.

Objective of file organisation
It contains an optimal selection of records i.e., records can be selected as fast as possible. To perform insert, delete or update transaction on the records should be quick & easy. The duplicate records cannot be included as a result of insert, update or delete. For the minimal cost of storage, records should be stored efficiently.

Page No.
Date

Ans 3.a) AVL trees are balanced trees whose worst case time complexity is equivalent to binary search.

AVL tree was invented by Adelson Velsky and Landis in 1962. So tree was named in honour of its inventors.

AVL tree mainly depends on two factor:
i) It should be a binary tree
ii) Its balanced factor should be 0, 1, -1

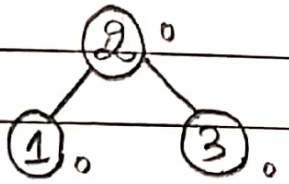
Balanced factor mean,

Balanced factor = $H_L - H_R$

where: H_L → height of left subtree

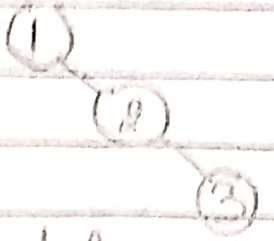
H_R → height of right subtree

Example:



Balanced

In this tree node 3 and 1 has zero balanced factor because they are leaf node and balance factor of 2 is also zero because the height of the left subtree is also 1 and height of right subtree is also one so difference is zero. Hence, This tree is balance or we can say it is an AVL tree.

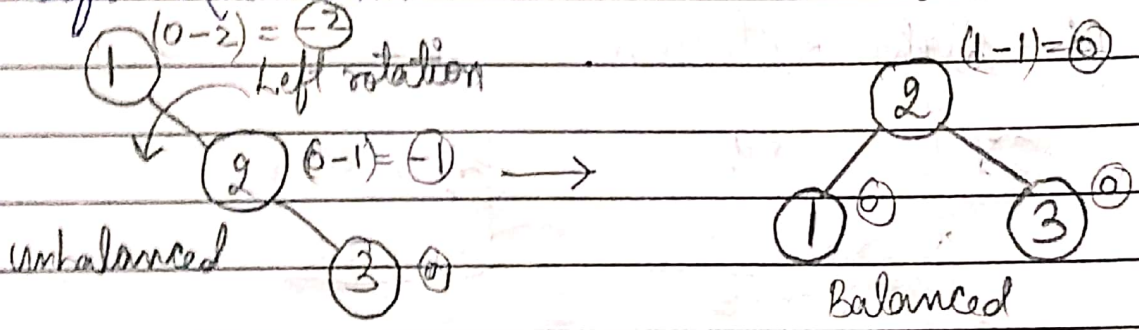


Unbalanced

In this figure the balanced factor of 3 is zero as 3 is a leaf node. Balanced factor of 2 is -1 because the difference of height of left tree and height of right tree is -1.
But the balanced factor of 1 is -2 because the difference of height of left tree and height of right tree is -2.
Hence this tree is unbalanced / Unbalanced.

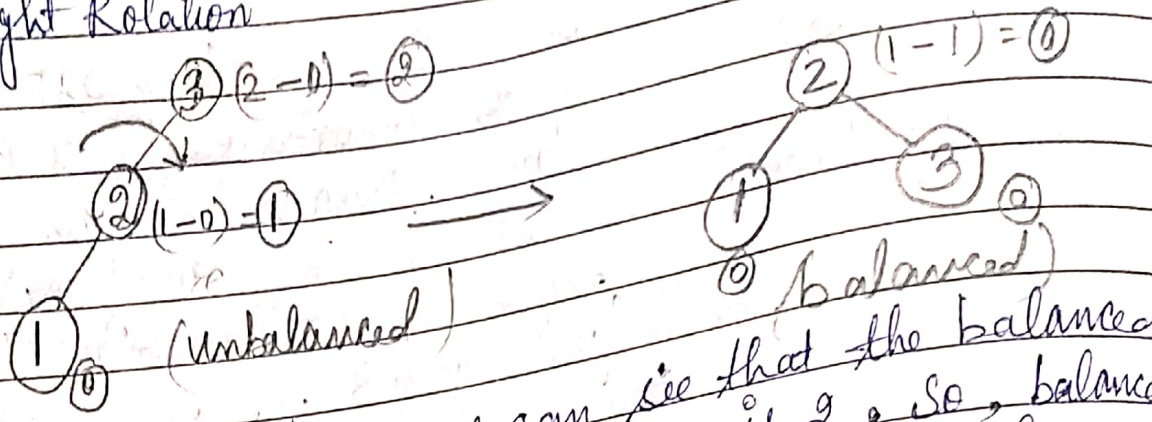
To make these type of unbalanced tree as balanced tree then the following rotation is done:

i) Left Rotation.



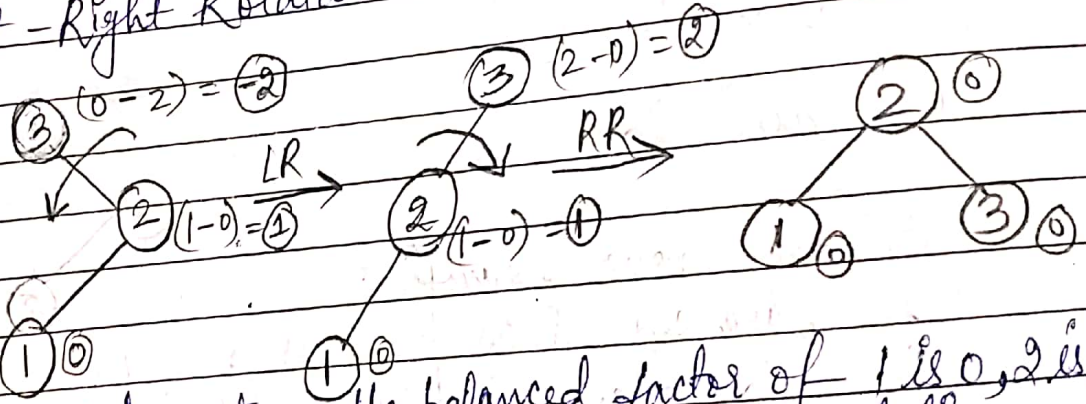
In the above example:
the balanced factor of 3 is 0, 2 is -1 and 1 is -2
So it is unbalanced because of the balanced factor of 1 is -2. Hence, we have to do a rotation i.e., left rotation to balanced this tree.
After left rotation the balanced factor of 1 is zero and 2 is zero and 3 is also zero hence the tree is balanced.

i) Right Rotation



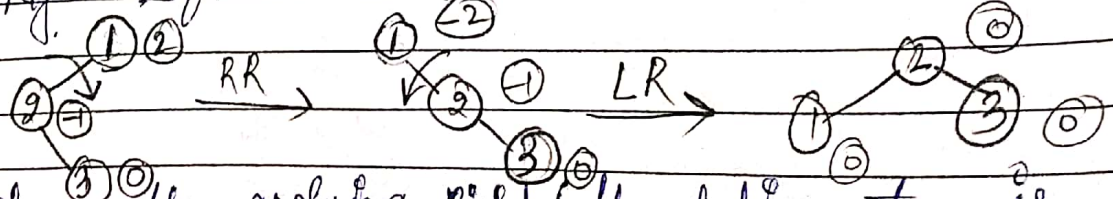
In the first figure, we can see that the balanced factor of 1 is 0, 2 is 1 and 3 is 2. So, balance factor of 3 is 2 therefore it is unbalanced. So we have to do a right rotation so that tree becomes balanced. Hence after applying right rotation, the balanced factor of 1 is 0, 2 is 0, 3 is 0 hence it is balanced tree.

iii) Left-Right Rotation.



In this above tree the balanced factor of 1 is 0, 2 is 1 and 3 is -2. Then we have to do a left rotation first after left rotation the tree is still unbalanced so we do a right rotation. Now, after left-right rotation tree is balanced.

iv) Right-Left Rotation.



Similarly, after applying Right Left rotation tree is