Data Structure using C

Date
DELTA Pg No.

✱ Difference b/w datatype and data structure -

* A datatype is the most basic and the most common classification of data.
The compiler gets to know the form or the type of information that will be used throughout the code. So, basically datatype is a type of information transmitted b/w the programmer and the compiler where the programmer informs the compiler about what type of data is to be stored and also tell how much space its required in the memory. Some basic examples are int, float, char, etc.

```
Void main()
{
  int a;
  a=5;

  float b;
  b=5.0;
}
```

```
  char c;
  c='A';

  char d[10];
  d="example";
```

Data structure

A data structure is a collection of different form and different types of data having a specific operation that can be perform. It is a collection of datatypes. It is a collection of organizing the item in terms of memory and also the way of accessing each item through some define logic. Some example of data

Scanned with CamScanner

structures are- stacks, queue, linked list, binary tree and many more.
Data structures perform such special operation like insertion, deletion and traversal. For example- You have to store data for many employees where each employee has its name, employee Id and a mobile no. So this kind of requires complex data which is data structure comprised of multiple primitive datatype.

Datatype	Data structure
* Datatype is a kind or form of a varible which is being used throughout the program. It define that the particular variable of a given datatype only.	* Data structure is the collection of different kinds of data that entire data can be represented using an object and can be used throughout the entire program.
* Implementation through datatype is a form of abstract implementation.	* Implementation through data structure is called concrete implementation.
* You can hold a value not data.	* You can hold different kinds and type of data within one single object.
* No problem of time complexity.	* Time complexity come into place when working with data structure.

Scanned with CamScanner

Date
DELTA Pg No.

Top-down and bottom-up design

* Structure/procedure oriented programming languages like C program language follow top-down approach whereas object-oriented programming languages like C++ and java programming language bottom-up approach.
* The top-down approach begins with high level design and ends with low-level design and development whereas bottom-up approach begins with low-level design or development and ends with high-level design.
* In top-down approach main function is written for and all such function are called from main function. Then, such function are written base on the required whereas in bottom-up approach code is developed for module and then these module are integerated with main function. Now a days both approaches are combined together followed in modern software design.

Debugging technique

Debugging is a process of detecting and removing a existing and potential error (also called as bug). In a software code that can cause it to behave unexceptedelly or crash. To prevent incorrect operation of the software on system debugging is used to find and resolved bug or defect. When various such system or module are tightly coupled the debugging become harder at

Scanned with CamScanner

any change in one module may cause more bugs to appear in another module. Sometimes it takes more times to debug a program than to code it.

Description:- The program debugging is a process of isolating and correcting the error.

* One simple & effective debugging tool is to take snapshot of program execution by inserting printf statement at point in the main program.
* The second technique is known as scaffolding. We insert the comment in the source code so when ever the data which is not required can be easily deleted.
* The third technique is "The Roll back". In back track they are able to crack the systems last configuration in which the system behave normally. In that condition if the system crashes it will automatically return roll back to its previous performing condition.

program testing

Testing is the process of finding bugs or error in a software product that is done manually by tester or can be automated. whereas in the debugging is the process of testing the bug found in testing state. The programmer or developer is responsible for debugging and it can't be automated.

Scanned with CamScanner

The tester never tester the error but rather find them and return to the programmer.

The view of data structure

* Abstract
* Implementation
* Application

Abstract or logic level

On this level, the programmer has to decide how the element are related to each other and what operation are needed. At this level the programmer has no concern about the representation of data in memory and how these operation are carried out. For ex- if you are a delhi and want to go Goa then you must know the appropriate route to know goa from delhi. specially when there are more than one route similarly in programming you must clear which data structure to be used in which or what kind of data.

Implementation level

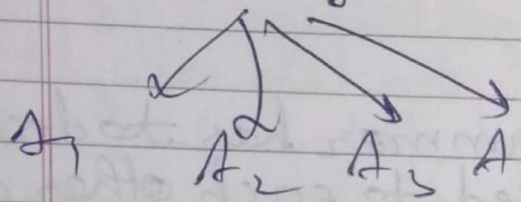
On the implementation level we do a specific way for representation of the data structure that is in implementation we convert an algorithm into and decide how much memory is to be allocated at static or dynamic time.

Scanned with CamScanner

Application level

On this level we use the data structure this is implemented at earlier level for ex- a Stack is to used to hold the processer that are waiting for different resources of the computer system in a multi-programming operating system.

Complexity



Efficiency finding on the basis of

① Time

Memory

$O(n)$

- ~~Big~~ highest upper bound

→ Big Oho (worst case complexity)

Big omega Ω (Best case n)

Big theta Θ (average case n)

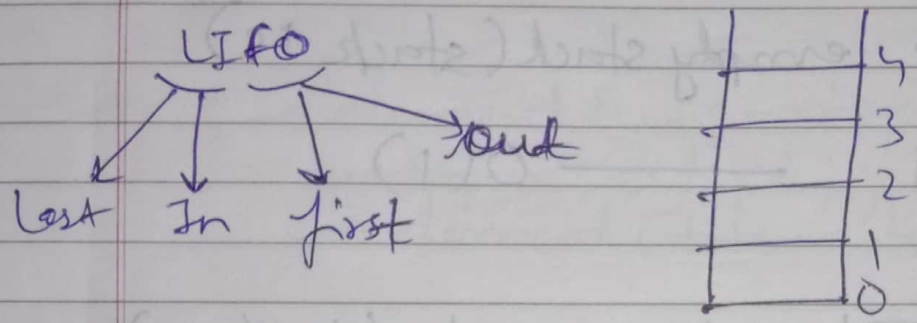
Example for (i=1; i<=n; i++)

Scanned with CamScanner

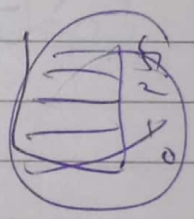
linear data structure or non primitive data

Stack

It is a linear data structure in which insertion (push) and deletion (pop) operation can be done from the same end that is the top of the stack and its follow lifo structure.



LIFO



Implementation of stack

operation- create empty stack

1) Is empty
2) Is full
3) push
4) pop
5) peek
6) size

static implementation → Array

Dynamic → link list

```
struct stack
{
 int top;
 int element[max];
};
 struct stack*;
```

Scanned with CamScanner

```
typedef struct stack
{
  int top;
  int element [max];
} stack;
```

① Void create empty stack (stack *s)

```
{
 s->top = -1
}
```

—— O(1)

② ~~int is empty~~ int is empty (stack *s)

```
{
  if( s->top == -1)
     return 1;
```

→ O(1)

```
else
return 0;
}
```

③ ⇒

```
int ~~is full~~ (stack *s)
{
  if( s->top == (Max-1))
     return 1;
  else
  return 0;
}
```

Scanned with CamScanner

(4) void push (stack *s, int auto)
{
s->top = s->top+1;
s-> element [s->top] = data)
}

# s-> element [++s->top] = data
s->top = s->top+1;
s-> element [++s->top] = data;

(5) int pop (stack *s)
{
int temp = s-> element (s->top);
s->top = s-> top--;
return temp;
}
return s -> element [s->top--];

(6) int peek (stack *s)
{
return s->element (s->top)
}

(7) int size (stack *s)
{
return s->top+1;
}

Scanned with CamScanner

Application of stacks

1. Reverse a string
2. checking a palindrome
3. Checking balanced parenthisization
4. Infix, prefix and postfix conversion
5. Postfix and prefix evaluation
6. Permutation generalization

* ALGORITHM

Swap (i^{th} symbol beginning ~~ends~~ i^{th} symbol from ~~last~~)

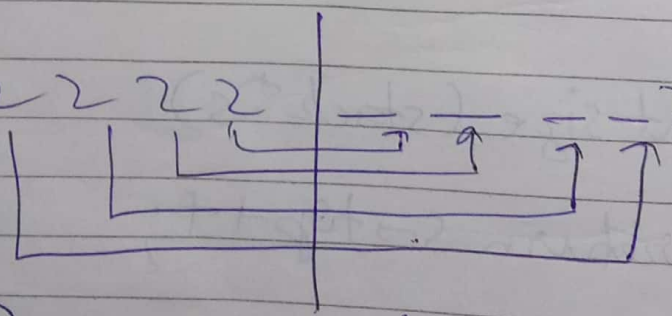
$1 \leq i \leq n/2 \rightarrow$ cost $O(n/2)$

Total cost = npush + npop

* checking a palindrome → even
→ odd

odd $a^R = a(aba)^R$ $(abba)^R = abba$

Even 2^4

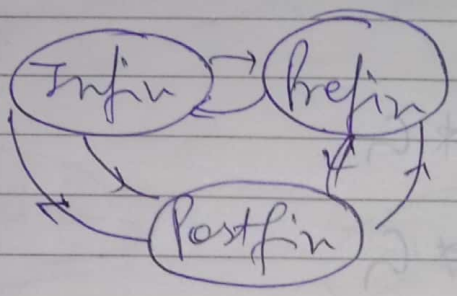


add 2^5 $\leftarrow^{En/2)}$ 2 2 2 2 2

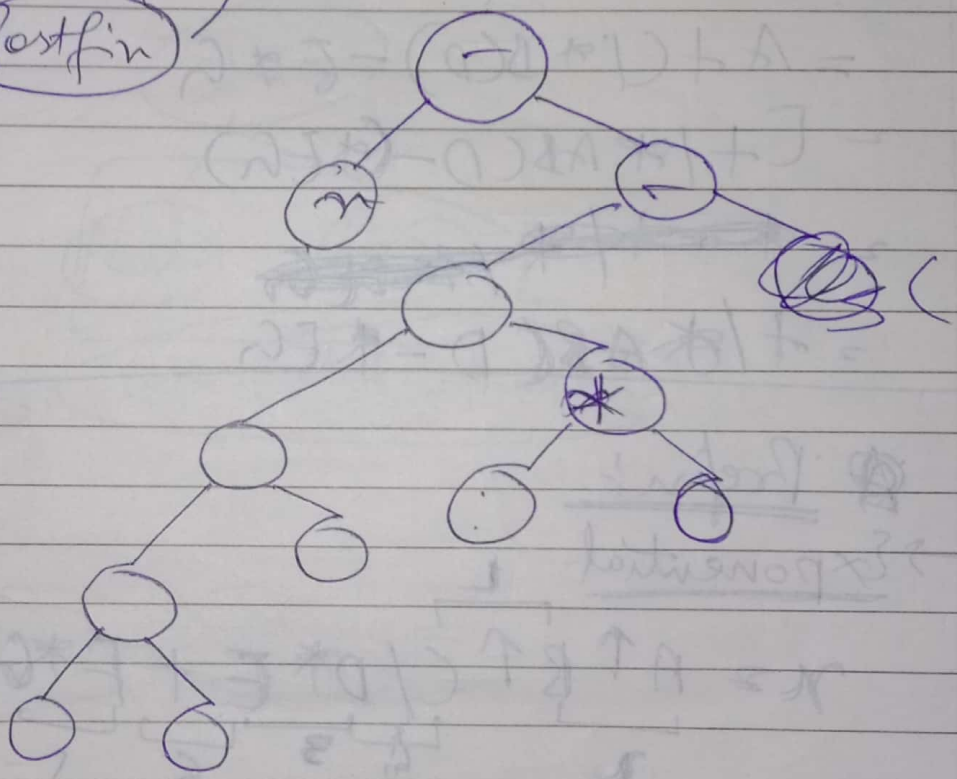
Scanned with CamScanner

Evaluation

Infix	Prefix	Postfix
A+B	+AB	AB+



x = A * B / (+ D * E - F



A + B * C / D - E * G

4 1 2 5 3

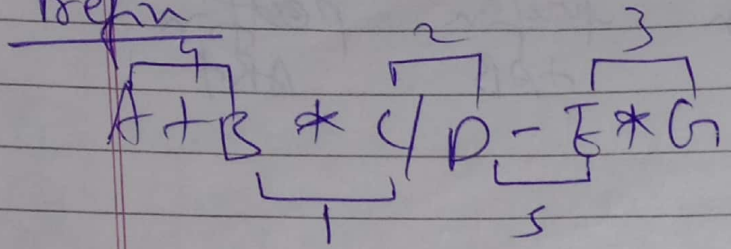
[(*BC)/D] - [

[A / BCD] - (*EG)

Scanned with CamScanner

Date
DELTA Pg No.

Prefix



A+(*BC)/D - E*G

= A+(/*BCD) - E*G

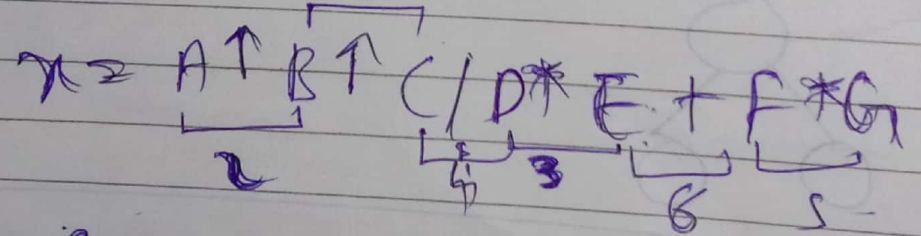
= [+/*ABCD-(*EG)

= */+ABCD-*EG

= +/*ABCD-*EG

Prefix:

→ Exponential



A↑(↑BC)/D*E+F*G

= (↑A↑BC)

Operations in Queue

* Insertion
* Deletion
* Traversal / Display
* Update

Insertion

→ In the insertion operation, the new element is inserted into the queue in the rear end.
→ It follows the following steps:-
1) checking the overflow.
2) If queue is empty then front will be set at 0.
3) Increment the rear and put the item in rear position.

Algorithm:-

Insertion (Q, item, size, rear, front)
// Q:- linear array or it is a queue represented by linear array.
// item:- Variable for holding the data element to be inserted.
// size:- Maximum no. of elements that Q can hold.
// rear:- index variable
// front:- index variable

Step 1 - if rear = size - 1
1) print it is overflow condition
2) otherwise goto step 2.

Step 2 - if rear = -1
1) front = front + 1 or front = 0
2) rear = rear + 1

Scanned with CamScanner

(2) Q[rear] = item

Step 3 - Exit.

Deletion

Deletion is nothing but incrementing the front value. It follows following steps :-

1) Underflow checking
2) checking for single element.
3) Incrementing front.

Algorithm

Delete (Q, item, front, rear)
// Q :- Queue represented by linear array.
// item :- item to be deleted.
// front :- index variable.
// rear :- index variable

Step 1 :- if front = -1
1) print it is underflow condition
2) otherwise goto step 2.

Step 2 :- else if front = rear and front ≠ -1
(i) print the deleted item is Q[front]
(ii) front = -1
(iii) rear = -1
(iv) otherwise goto step 3.

Scanned with CamScanner

Step3:- print the deleted item is Q[front].
front = front + 1

Step4:- Exit.

Traversal / Display

Visiting of elements starts from front end to till rear end by which we can say it follows FIFO principles.

Algorithm:-
display (Q, rear, front)
// Q:- It is a linear queue (existing one)
// rear :- index variable
// front :- index variable

Step1:- if front = -1
i) print underflow condition
ii) otherwise goto Step2

Step2:- for i = front to rear
print Q[i]
[End for]
[End if structure]

Step3:- Exit

Scanned with CamScanner

Update

It means modifying the existing data in the queue.

Algorithm:-

PROCEDURE update (Q, rear, front, pos, item)
// Q:- linear queue
// rear:- index variable
// front:- index variable
// pos:- position at which item will be updated.
// item:- updated item

Step1:- pos = pos-1

Step2:- if (pos >= front) and (pos <= rear)
i) Q[pos-1] = item
ii) otherwise goto step3

Step3:- print invalid position

Circular Queue

* Overflow condition:-

if front == (rear+1) % size then overflow condition occur.

* Overflow condition:-

if front == -1, then underflow condition occur.

Scanned with CamScanner

Operations in Circular Queue

* Insertion
* Deletion
* Traverse (Display)

Insertion

Inputting the element in the rear end is called insertion. It follows the following steps at the time of insertion:-

1) Overflow condition
2) checking the 1st element insertion i.e, is it the first element that is going to insert.
3) find the next rear position
4) Input the item or element in the rear position.

Algorithm:-

```
CQ - Insert (CQ, front, rear, size, item)
// CQ → It is a linear array, (Circular queue)
// front → initial value = -1, it is a index variable
// rear → initial value = -1, it is a index variable
// size → Size of CQ; i.e., maximum size
// item → The item that is to be inserted.

Step1:- If (front == (rear+1)%size)
        a) print overflow
        b) exit
        [End if]
Step2:- if (rear == -1)
        a) front = 0
        [End if]
```

Scanned with CamScanner

Step 3:- rear = (rear + 1) % size

Step 4:- CQ[rear] = item

Step 5:- exit.

Deletion

It means deleting the element in the front end.
It follows the following steps:-

1. checking for underflow (i.e. front == -1)
2. checking if there exist only one element. If its then initialize front = rear = -1.
3. otherwise increment the front value.
   i.e. front = (front + 1) % size.

Here before going for next front value, the element that is to be deleted should be stored in a variable.

Algorithm:-

PROCEDURE delete (CQ, rear, front, size, Pitem)
// CQ → Circular queue
// rear → index variable
// front → index variable
// size → It is the maximum no. of data element that circular queue (CQ) can hold
// Pitem → Element or item to be deleted.

Scanned with CamScanner

Step1:- [checking of underflow]
if (front = -1) then print underflow
else store the CQ[front] in another variable Ditem
if (front = rear) then
a) front = rear = -1
b) else front = (front + 1) % size
print the deleted item is Ditem
[End of if structure]

Step2:- Exit.

Traversal
Means visiting each element exactly once.

Algorithm:-
PROCEDURE Display (CQ, rear, front, i, size)
// CQ → Circular Queue
// rear → Index variable
// front → Index variable
// i → loop control variable or counter variable
// size → max. no. of elements that a queue can hold.

Step1:- if front = -1 then print overflow

Step2:- if (rear ≥ front)
for (i = front; i != (rear + 1); i = (i + 1) % size)
print CQ[i]

Scanned with CamScanner

Step3:- if (front > rear)

i) for (i=front; i<=size-1; i++)
print (a[i]).

ii) for (i=0; i<=rear; i++)
print (a[i]).

Step4 :- Exit.

Scanned with CamScanner

Stack

1. LIFO
2. Same end is used to insert and delete elements
3. Only one pointer can use
5. Push and Pop
6. Top == -1 (empty condition)
8. top == max-1 full condition
8. It does not have variants.
9. Simpler implementation

Queue

1. FIFO
2. One end is used for insertion i.e rear end and another end is used for deletion of elements i.e front end
4. Two pointer used in simple queue case
5. Enqueue and dequeue
6. front == -1 || front == rear+1
8. Rear == Max-1
9. It has variants like circular queue, priority queue, doubly ended queue
9. Comparatively complex implementation

Scanned with CamScanner

PSC

Unit-1

* Structured programming :- It is the programming paradigm aimed at improving clarity, quality and development time of the computer program by making extensive use of subroutine, block structure, for and while loop - In contrast to using tests and jumps such as go to statement structured programming is most frequently used with deviations.

* Program testing

It's a proces of running the program on sample data chosen to find errors if they are present. It is clear that the compiler can detect only syntax and semantic errors. All remaining errors can be detected at run time only. Therefore it is necessary for the programmer to detect all such errors.

Basically there are two type of testing method:-

(1) The Unit-test method → In this, program testing is done at subprogram level where a subprogram is designed to perform only one task.

(2) An integration test method ⇒ After testing of all the subprograms the data should be tested according to the specification of the complete program, without regard to initial details of the

program such type of testing is made to find out the interfacing problem between different subprograms.

Documentation :- It is a written tests and comments that makes a program easier for others to understand, use and modify for small programs, it is not necessary to have a detailed documentation because one can keep all details in one's mind and so needs documentation only to explain program to someone else.

Program Testing	Program Debugging
(1) Testing is a process in which the program is validated	(1) It is a process in which errors are removed.
(2) It is a positive activity that checks the requirement of program specification.	(2) It is a negative activity in the sense that it tracks unknown errors and removes them
(3) It is said to be complete when all desired verification among specifications are found valid.	(3) It is finished when there is no error present.
(4) It is a planned process in which a programmer has to make plan how and when to test in a program.	(4) It is not a planned process actually it is stemmed from testing.

Scanned with CamScanner

(5) Testing can be perform in early stages as well as later stages of program development	(5) It begins after the coding of algorithm because it requires an executable program.

Top-down approach

→ It takes the whole software system as an entity and then decomposes it to achieve more than one sub-systems or components based on some characteristics.

→ Top-down design is more suitable when the software solution needs to be designed from scratch and specific details are unknown.

Bottom-up approach

→ It starts with the beginning of most basic or primitive components and proceeds to higher level components that uses there lower level components.

→ If a system is built from an existing software, then bottom up approach is more suitable as it starts on some existing components.

Scanned with CamScanner

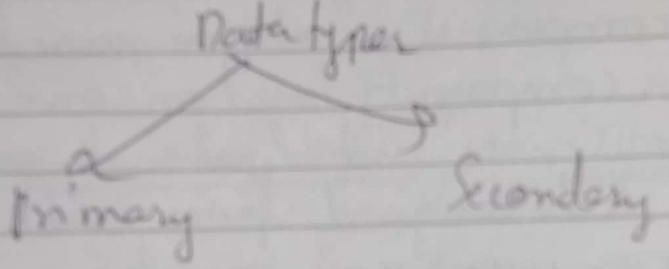
PSC

Unit-2

Datatype

Datatype is a term, which is used to refer the kind of data that variable may hold in programming language. The general form of class of data items is known as data type i.e.

Such as :- int, long int, Unsigned int, unsigned char, signed char.



Primary :- The data types that are not composed of other data types known as primary. Such as Integer, float and char.

Secondary :- Data types which are composed of primary data types called secondary data types, they are defined by the programmer, that's why it is sometimes called user-defined datatype. Such as - array, structure, pointers, union, etc.

Data object :- It is term used to represent a set of elements. data object may be static or infinite.

Scanned with CamScanner

Date
DELTA Pg No.

Data structure

The specific way of arranging the data in an effiient manner.

Data structure

(1) Built-in data structure:- Which are initially provided by the higher level language called built in data structure.
Ex- arrays, structures, etc

(2) User-defined data structure:- The data structure which are formed by the ~~use~~ user with the help of built-in data structure.
Ex- stack, queue, graph, tree.

(3) Linear data structure:- The data structure whose elements are processed sequentially one by one are called linear data structure. Ex- stack, queue, linked list

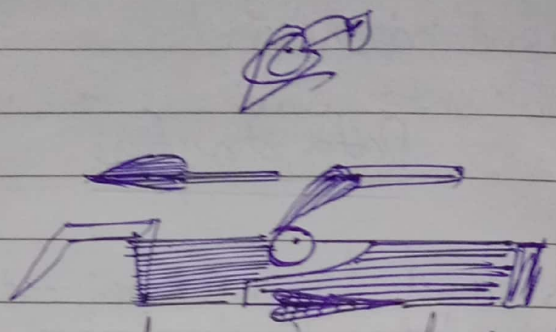
(4) Non-linear data structure:- The data structure whose elements are not processed sequentially.
Ex- Graph tree

Scanned with CamScanner

Date
DELTA Pg No.

Data structure operations

1. Traversing
2. Inserting
3. Sorting
4. Searching
5. Deleting
6. Merging



1. Traversing :- we mean processing of each data item in the data structure exactly once from first element to last element.

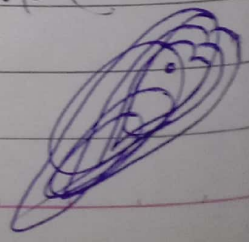
2. Inserting :) By We mean addition of data item in the data structure at any position, last position or first position.

3. Sorting :) We mean sort the data item into ascending order, descending order or according to the requirement of programma.

4. Searching :) We mean finding an data item from a set of numbers.

5. Deleting :) We mean, we mean delete an data item in a data structure from any position.

6. Merging :) We mean combining of two data structure into one data structure.

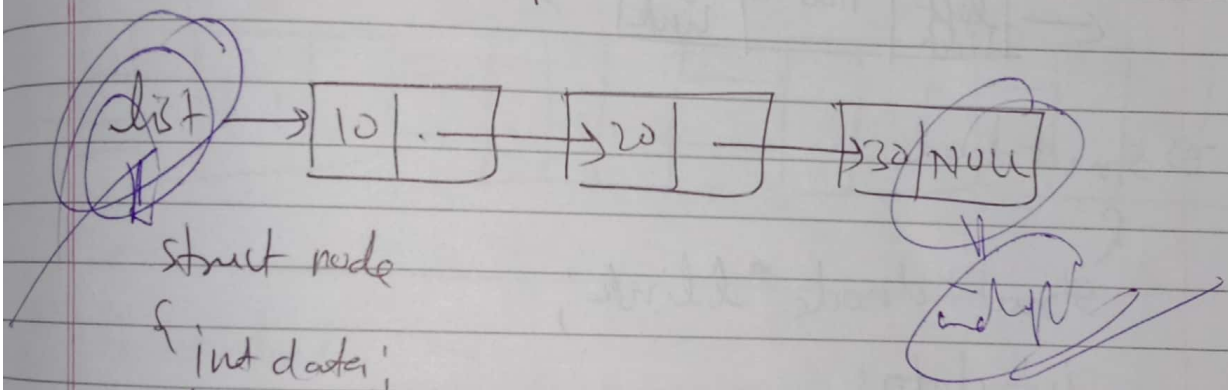


Scanned with CamScanner

Unit - 3

linked list

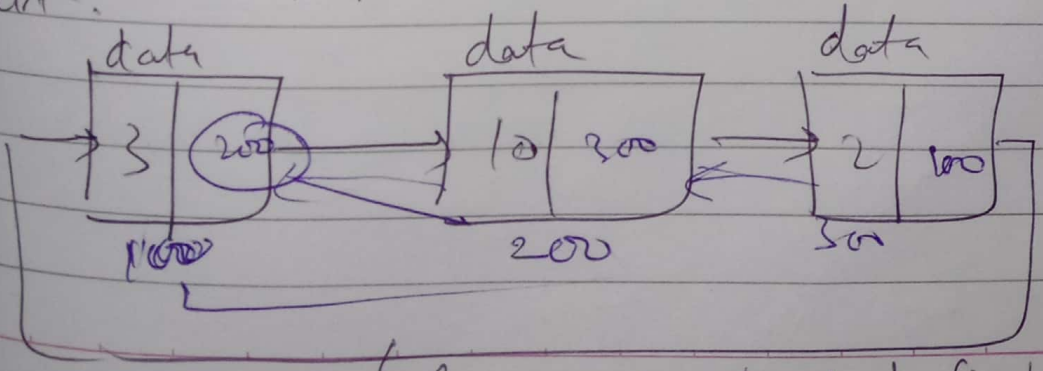
- A dynamic list in which each node combined with a pointer that links it to the next node is called linked list.
- The node of a linked list can be store anywhere in the memory.
- We access the linked list by saving the address of first node. In the pointer variable say list. This pointer called the external pointer variable to the list.
- The link field to the last node contains a NULL to indicate the end of the list.




```
struct node
{ int data;
  struct node *link;
};
```

Circular linked list

A linked list in which the last node containing the address of first node is known as circular linked list.



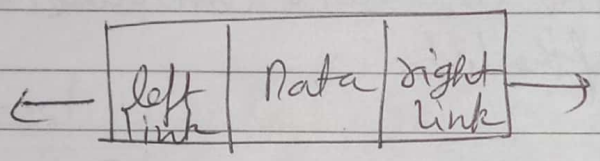
Last element points back to first

Scanned with CamScanner

Doubly linked list

The linked list that may traverse in either one direction or in both direction. A doubly linked list has two pointa one to point to the previous node and another for next node.

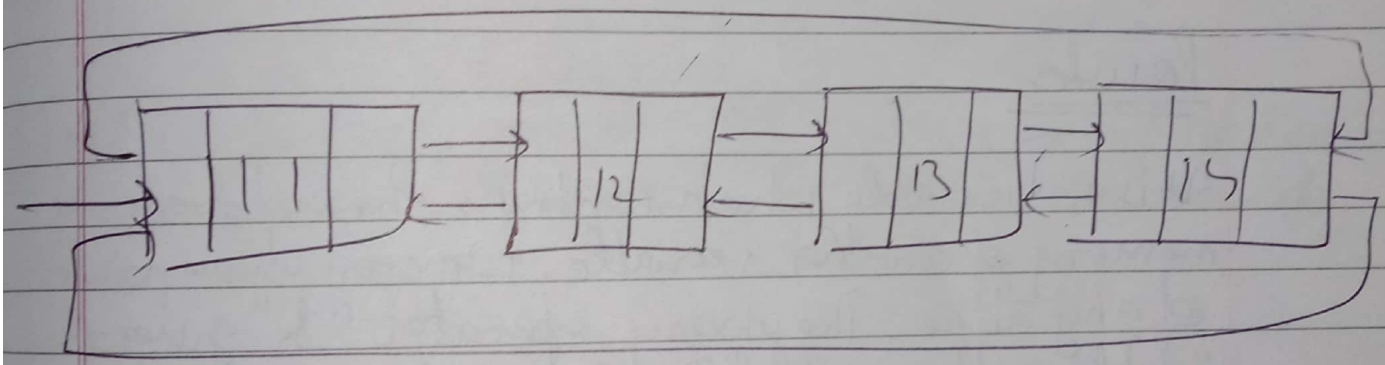
- left link - link to predecessor or left node
- datafield
- Right link - link to sucessor node or right node.



⇒
```
struct dnode
{
  struct dnode *llink;
  int data;
  struct dnode *rlink;
};
```

Circular doubly linked list

In circular doubly linked list, instead of storing a NULL, we will store the address of first node and last node in the right link and left link of last and first node respectively.




```
Struct dnode
{
 struct dnode *llink;
 Int data;
 Struct dnode *rlink;
};
```

Scanned with CamScanner

Application of linked list

1. Linked list can be used to implement stack, queues, graph.
2. Useful for dynamic memory allocation
3. Real life example - persons standing for food in mess.
4. Songs playing in media player.
5. Circular linked list is used in our computer where multiple application are running.

Pointer

It is a variable which contains the address in memory of another variable. We can have pointer of any type. The unary operator "&" gives the "address of variable". The Indirection or dereference operator "*" gives the content of an object pointed to by a pointer.

malloc() and calloc()

Malloc() function allocate space in byte.

Syntax ⇒ * malloc (size);

Calloc() ⇒ Allocate spaces for "n" no. of items where an item is of "size" byte and store 0 in reserved area.

Syntax: *Calloc (n, size);

Scanned with CamScanner

Date
DELTA Pg No.

Static memory allocation	Dynamic memory allocation
(1) performed at static or compile time.	(1) performed at dynamic or run time.
(2) Assigned to stack	(2) Assigned to heap.
(3) Size must be know at compile time.	(3) Size may be unknown at compile time
(4) Fist in fist out	(Eg) No particular order of assignment.
(5) It is best if require size of memory known in advance.	(5) It is best if we don't now about how much memory require.

Advantages of linked list

(1) These are dynamic data structure thus, they can grow or shrink during executing of a program.
(2) Many complex application can be easily carried out with linked lists.
(3) Efficient ~~mory~~ memory utilization.
(4) Insertion and deletion are easier and efficient.

Scanned with CamScanner

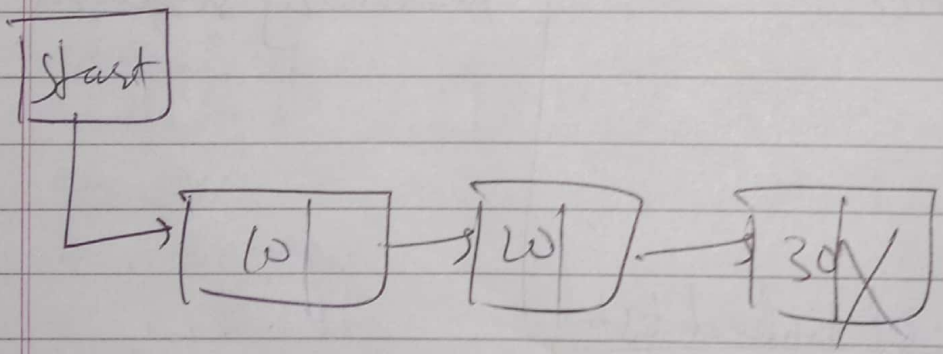
Disadvantages of linked list

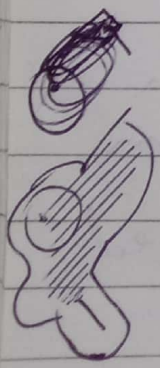
1. More memory :- If the no. of fields are more, then more memory space is needed.
2. Access to an arbitary data item is little bit cumbersome and also time-consuming.

Singly-linked list

It is one in which all nodes are linked together in some sequential manner. Hence it is also called linear linked list. Clearly it has the beginning and the end.

The problem with this list we cannot access the predecessor of node from the current node. This can be overcome in doubly linked lists.





Scanned with CamScanner

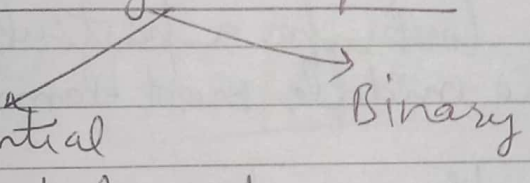
Date
DELTA Pg No.

Unit-7

Searching

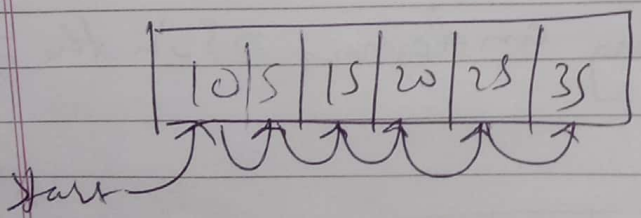
- It is the process of finding a given value position in a list of value.
- It decide whether a search key is present in the data or not.
- It is the algorithmic process of finding a particular item in a collection of items.
- It can be done on internal or on external data structures.

Searching technique



Sequential search

- It is also called as linear search
- It search starts at the begning of the list and checks every element of the list.
- It is a basic and simple search algorithm.
- It compares the element with all other elements in the list. If the element is matched, it returns the value index, else it returns -1.



It searches an element or value from an array till the desired element or value is not found. If we search the element 25, it will go step by step in a sequence order. It searches in a

Scanned with CamScanner

sequence order. Sequential search is applied on the unsorted or unordered list when there are fewer elements in a list.

Binary Search

- This used for searching an element in a sorted array.
- It is a fast search algorithm with run-time complexity of O(log n)
- Binary search works on the principle of divide and conquer.
- This technique looks for a particular element by comparing the middle most element of the collection.
- This useful when there are large number of elements in an array.

5	10	15	20	25	30

- The above array is sorted in ascending order. As we known binary search is applied on sorted lists only for fast searching.

for ex - if searching an element 25 in the 7-element array.

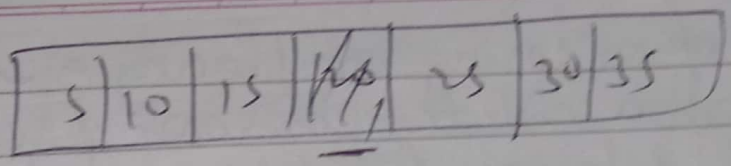
Search element: 25

5	10	15	20	25	30	35

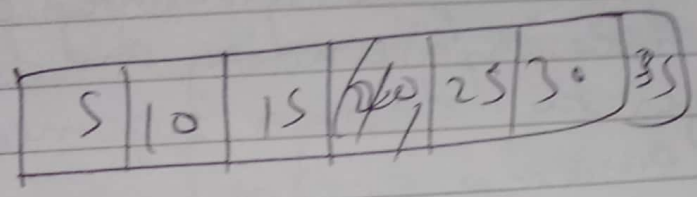
Scanned with CamScanner

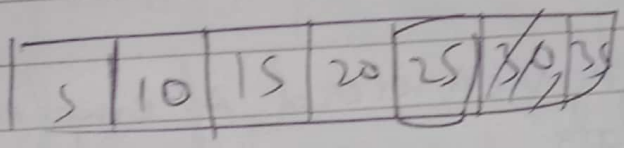
Date
DELTA Pg No.

Start with middle element

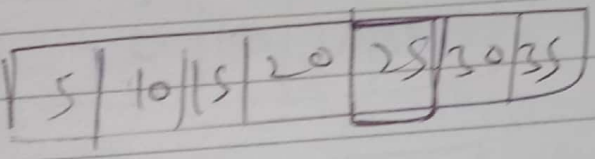


25 > 20

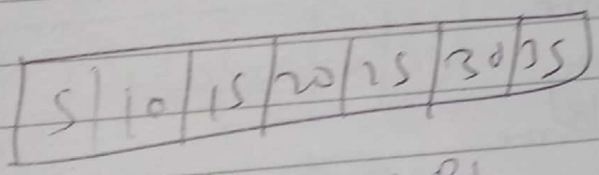




25 < 30



Element found



Working structure of Binary search

Binary Searching starts with middle element. If element is equal to the element that we are searching then return type. If the element is less than then move to the right of the list or if the element is greater than then move to the left of the list. Repeat this, till you find an element.

Scanned with CamScanner

Sorting

It is a process of ordering or placing a list of elements from a collection in some kind of order. It is nothing but storage of data in sorted order. Sorting can be done in ascending and descending order. It arranges the data in a sequence which makes searching easier.

for ex - Suppose we have a record of employee. It has following data.

Employee No.
Employee Name
Employee Salary
Department Name

Here, employee no. can be taken as key for sorting the records in ascending or descending order. Now, we have to search a employee with employee no. 116, so we don't require to search the complete record, simply we can search b/w the employee with employee no. 100 to 120.

Sorting technique depends on the situation:

(1) Execution time of program that means time taken for execution of program

(2) Space that means space taken by the program

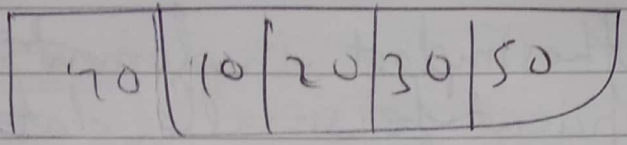
~~Bubble Sort~~

10 56

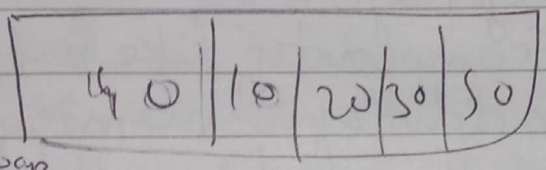
Bubble Sort

$O(n^2)$ → Worst
$O(\log n)$ → Best

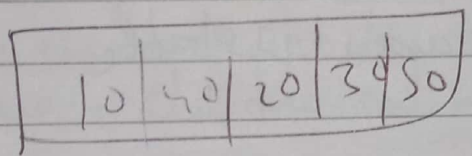
- This type of sorting is used for sorting 'n' (no. of items) elements.
- It compares all the elements one by one and sorts them based on their values.



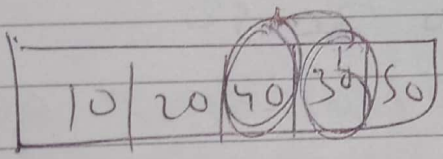
Starts with first two
40 > 10 swap



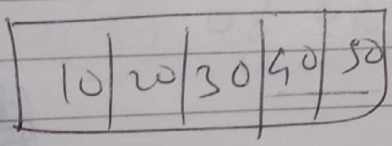
40 > 20



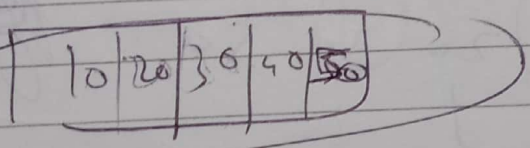
40 > 30



so 40



Sorted array

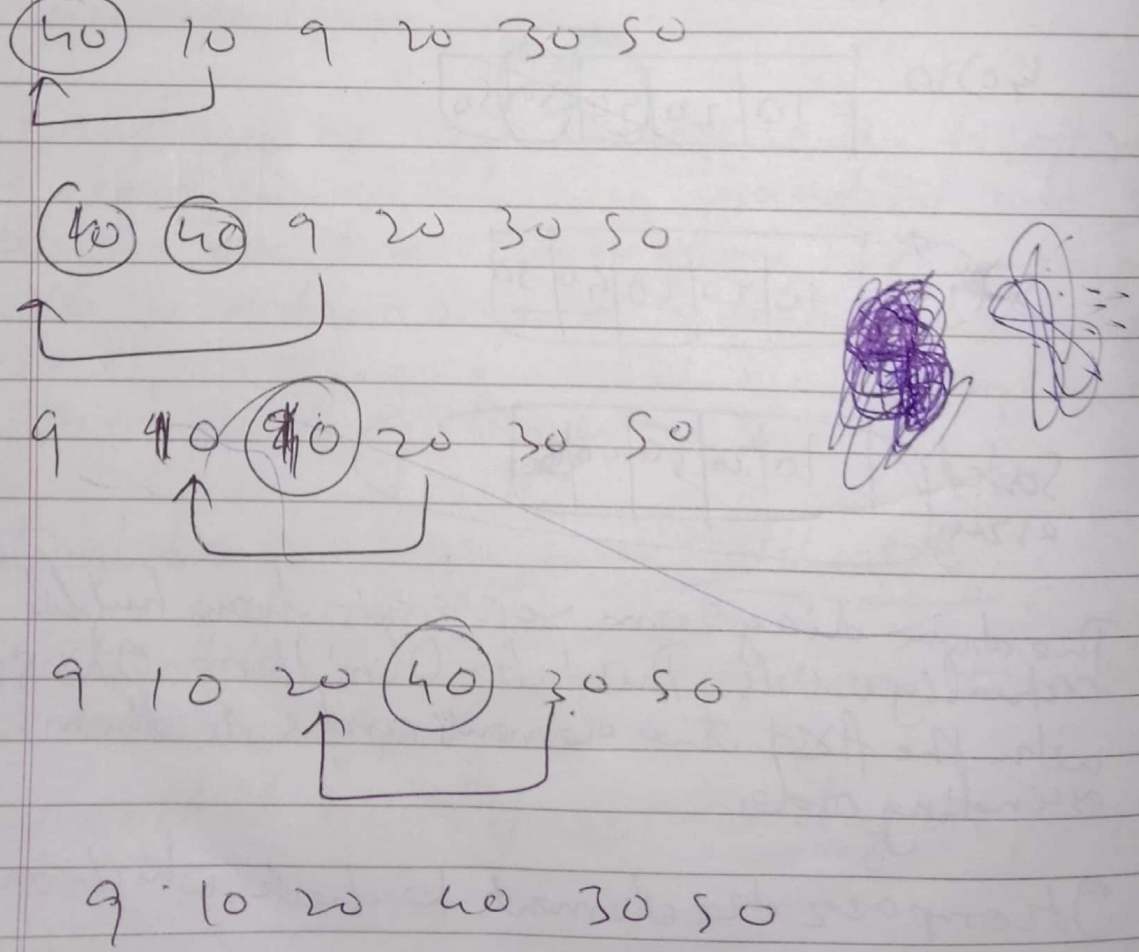


- The above diagram represents how bubble sort actually works. This takes $O(n^2)$ time. This starts with the first two elements and sorts them in ascending order.
- Compare the element to check which one is greater.

Scanned with CamScanner

Insertion Sort

- It is a simple sorting algorithm
- This sorting method sorts the array by shifting elements one by one.
- It builds the final sorted array one item at a time.
- It has one of the simplest implementation
- This sort is efficient for smaller data sets but is inefficient for larger lists.
- It has low space complexity like bubble sort.
- It requires single additional memory space.
- It does not change the relative order of elements with equal keys because it is stable



Scanned with CamScanner

It works like the way starts with the second element or key. The key is compared with the elements ahead of it and is put it in the right place.

Selection Sort

This a simple sorting algorithm which finds the smallest element in the array and exchanges it with the element in the first position. Then finds the second smallest element and exchanges it with the element in the second position and continues until the entire array is sorted.

40 30 9 20 10 50

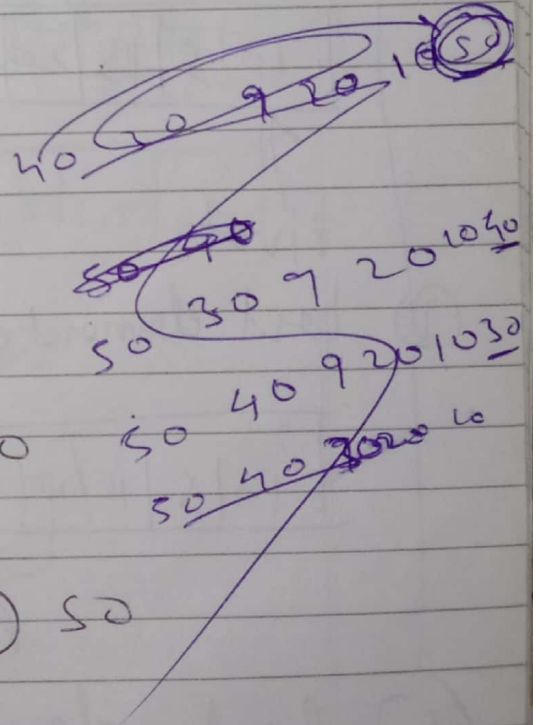
4 30 (9) 20 10 50

9 30 40 20 (10) 50

9 10 40 (20) 30 50

9 10 20 40 (30) 50

9 10 20 30 40 50

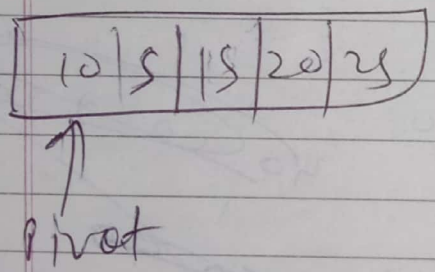


Quicksort

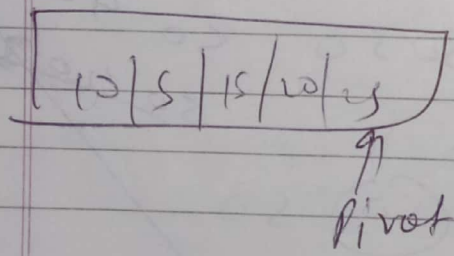
→ It also known as partition-exchange sort based on the rule of divide and conquer.
→ It is a highly efficient sorting algorithm.
→ It is the quickest comparison-based sorting algorithm.
→ It is very fast and requires less additional space, only $O(\log n)$ space is required.
→ Quick sort picks an element as pivot and partition the array around the picked pivot.

There are different versions of quick sort which chose the pivot in different ways:-

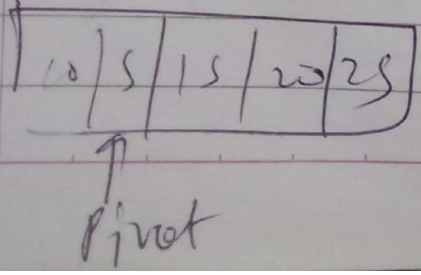
(1) First element as pivot



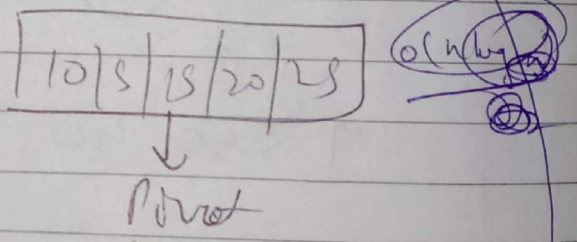
(2) Last element as pivot



(3) Random element as pivot



(4) Median as pivot

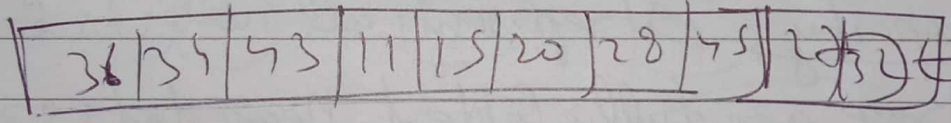


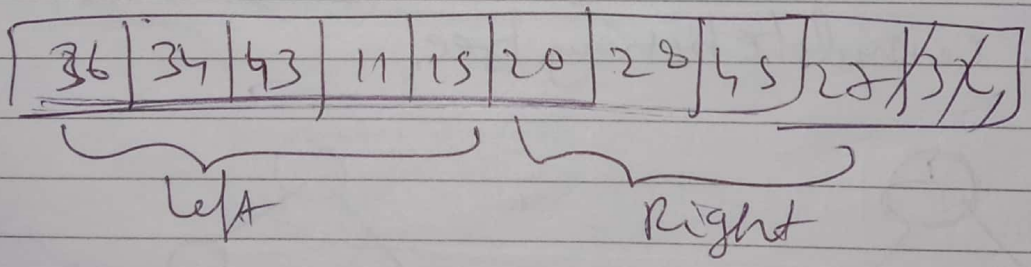
Scanned with CamScanner

Date
DELTA Pg No.

Algorithm for Quick Sort

Step 1: Choose the highest index value as pivot.
Step 2: Take two variables to point left and right of the list excluding pivot.
Step 3: Left points to the low index
Step 4: Right points to the high index
Step 5: While value at left < (less than) pivot move right
Step 6: While value at right > (Greater than) pivot move left
Step 7: If both Step 5 and step 6 does not match, swap left and right
Step 8: If left = (less than or Equal to) right, the point where they met is new pivot.





Scanned with CamScanner

Date
DELTA Pg No.

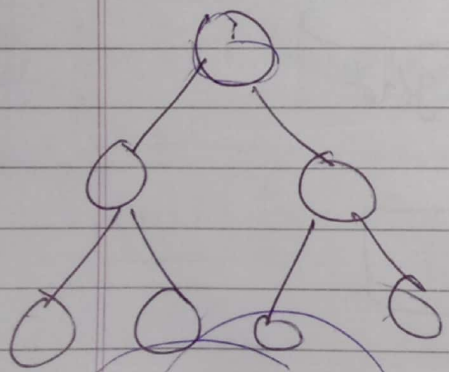
Heap Sort

- It is a comparison based sorting algorithm
- It is a special tree-based data structure.
- It is similar to selection sort. The only difference is, it finds largest element and places it at the end.
- This sort is not a stable sort. It requires a constant space for sorting a list.
- It is very fast and widely used for sorting.

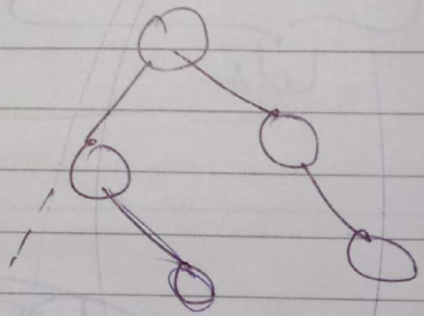
It has two properties

1. Shape property
2. Heap property

Shape property: It represents all nodes or levels of the tree are fully filled. Heap data structure is a complete binary tree.



Complete Binary tree



In-complete binary tree

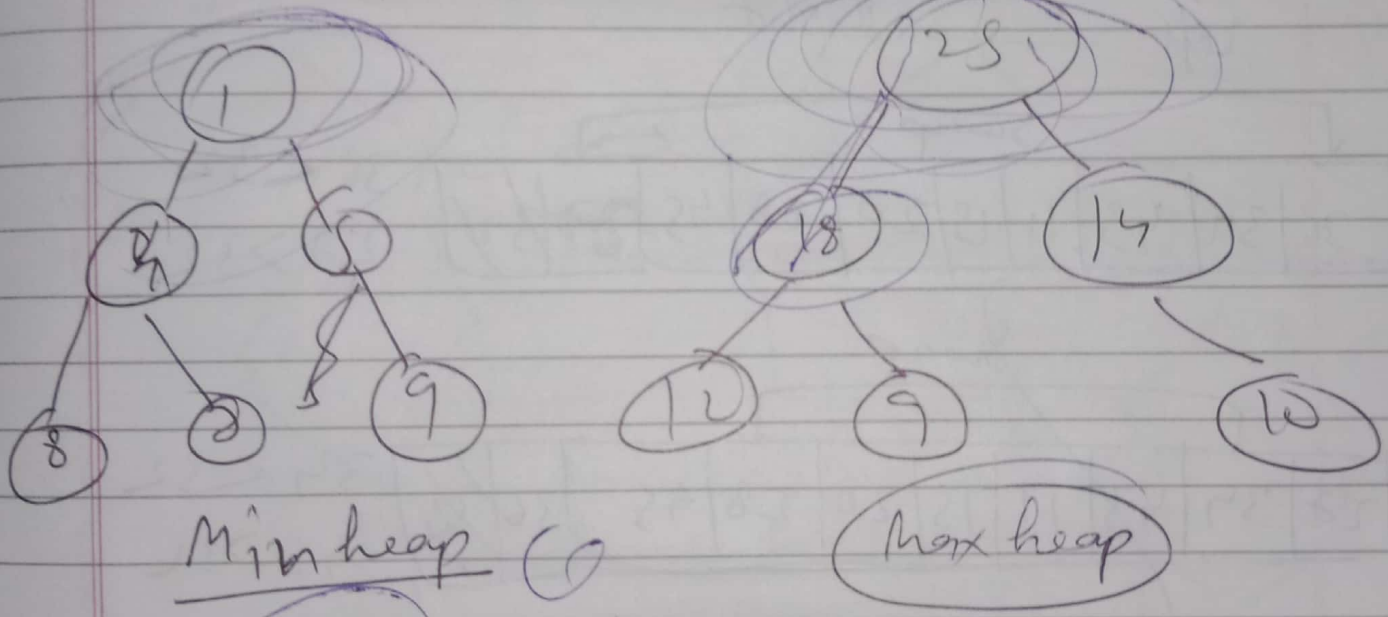
Scanned with CamScanner

Heap property :- It is a binary tree with special characteristics. It can be classified into two types :-

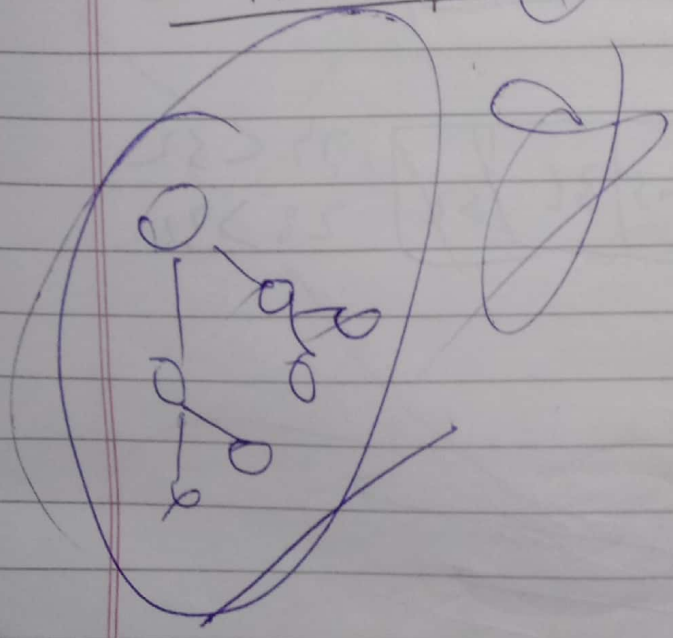
1) Max heap
2) Min. heap

Max :- If the parent nodes are greater than their child nodes

Min :- If the parent nodes are smaller than their child nodes



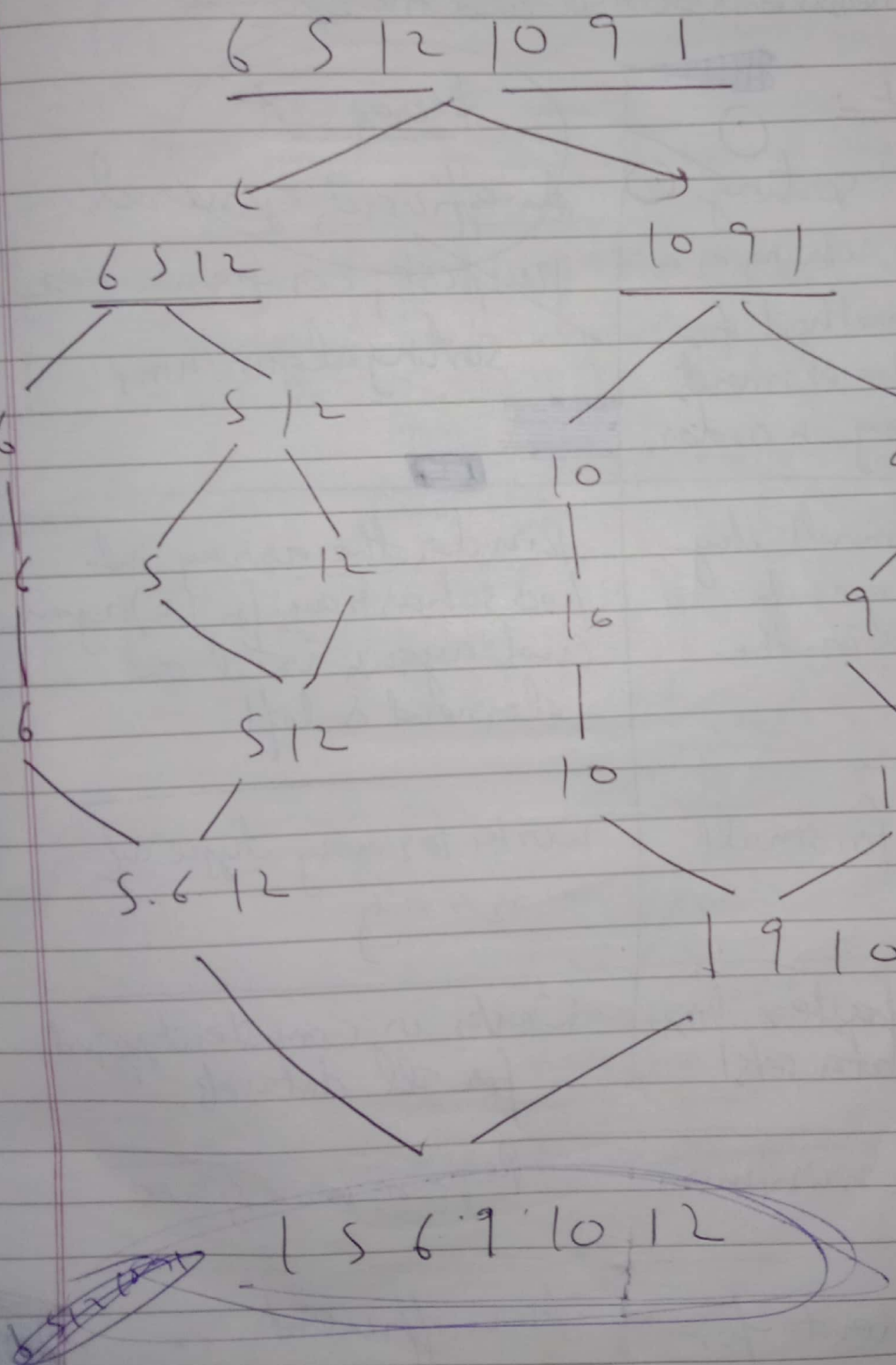
Min heap



Scanned with CamScanner

Merge Sort

- It first divides the array into equal halves and then combines them in a sorted manner.
- It is based on divide and conquer technique.



Scanned with CamScanner

Date
DELTA Pg No.

I

* first divide the list into the smallest unit (1 element), then compare each element with the adjacent list to sort and merge the two adjacent lists. finally all the elements are sorted and merged.

Quicksort :-	Merge sort
An efficient sorting algorithm, serving as a systematic method for placing the elements of an array in order.	An efficient, general purpose, comparison-based sorting algorithms.
Sorts the elements by comparing each element with the pivot.	Divides the array into two subarrays, half the size again and again until one element is left.
Suitable for small arrays.	Works for any type of array.
Works faster for small data sets.	Works in consistent speed for all datasets.
Requires minimum space.	Requires more space.
Not efficient for large arrays.	More efficient.

Scanned with CamScanner

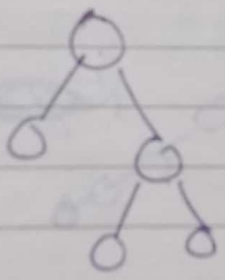
Binary Tree and its types

Binary Tree → A rooted tree in which every node has atmost 2 children that is 0, 1, 2 children are possible.

Types of Binary Tree →

* Full / Proper / Strict
* Complete Binary tree.
* perfect Binary tree (Complete + full)
* Degenerate Binary tree

⇒ **Full / Proper / Strict** ⇒ each node have either 0 or 2 children

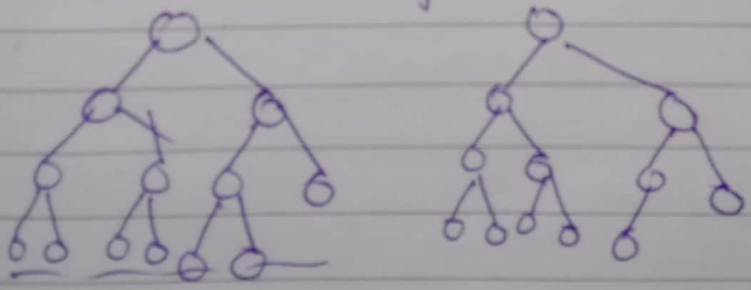


no. of leaf node = no. of internal node + 1

3 = 2 + 1

3 = 3

⇒ **Complete Binary tree** :- if all the level are completely filled (except possibly the last level) and the last level has node as left as possible

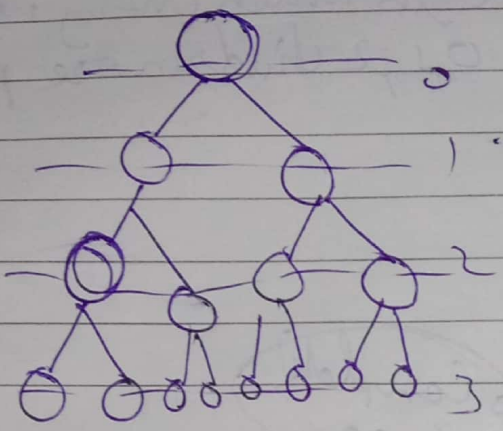


Scanned with CamScanner

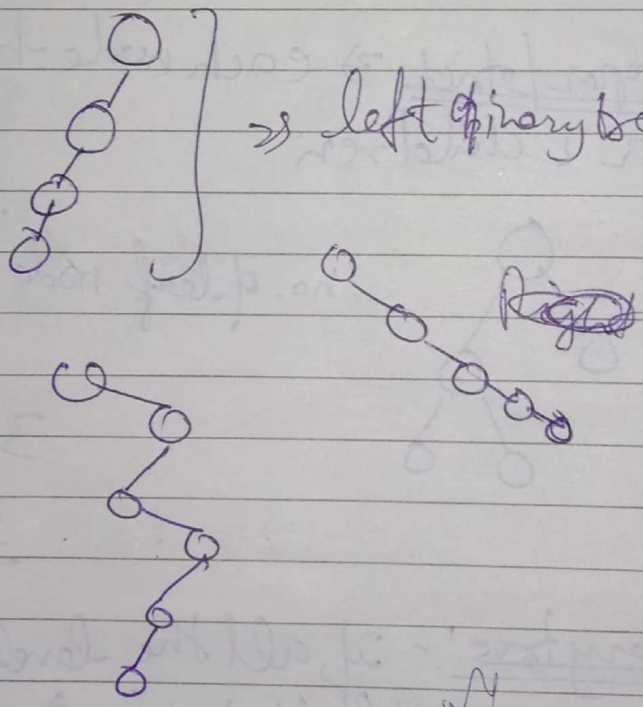
(CBT) and FULL Binary Tree

Date
Page No.

⇒ Perfect Binary Tree ⇒ all internal nodes have 2 children and all leaves are at same level.



⇒ Degenerate binary tree ⇒ all the internal nodes are having one child.



Scanned with CamScanner

Applications of Binary Tree

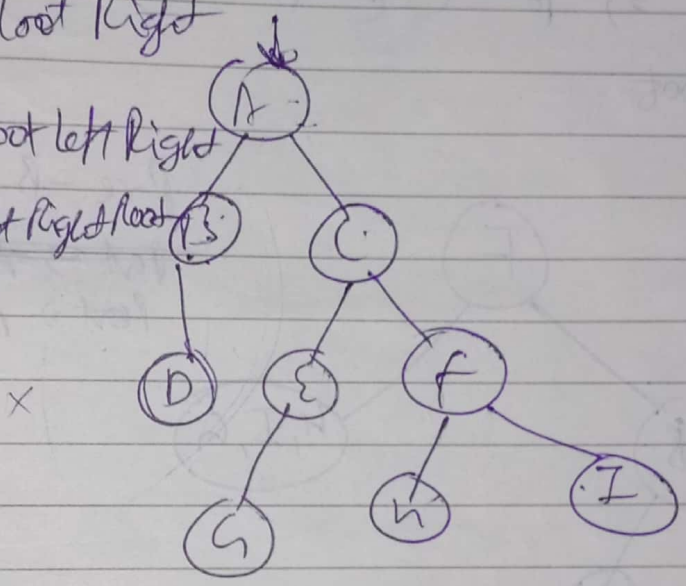
1) To use in the algebraic equation
2) To use in encoding technique ⇒ Huffman's technique.

Tree Traversals (Binary Tree)

Inorder :- Left Root Right

Preorder :- Root Left Right

Postorder :- Left Right Root



Inorder :- B D A G E C H F I

Preorder :- A B D C E G F H I

Postorder :- D B G E H I F C A

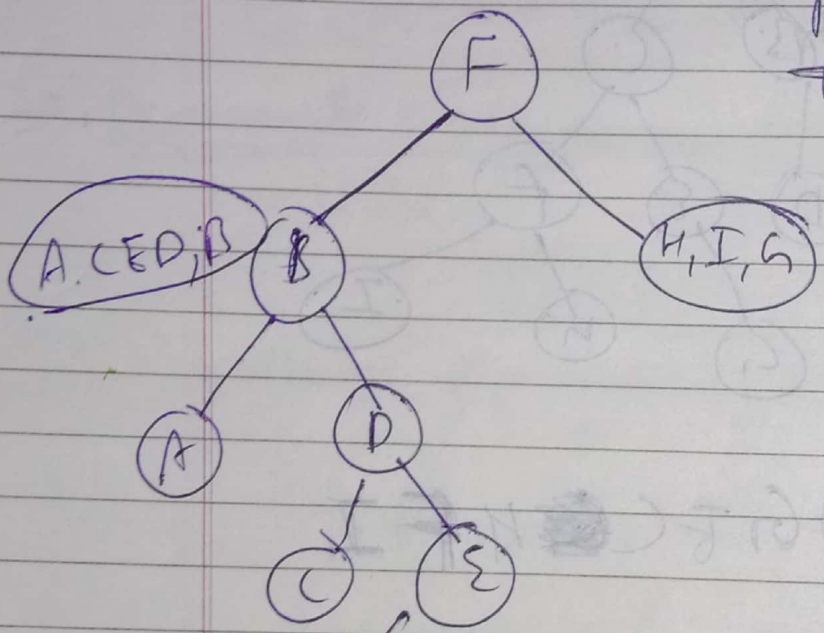
Scanned with CamScanner

Construct Binary tree given Preorder and Postorder.

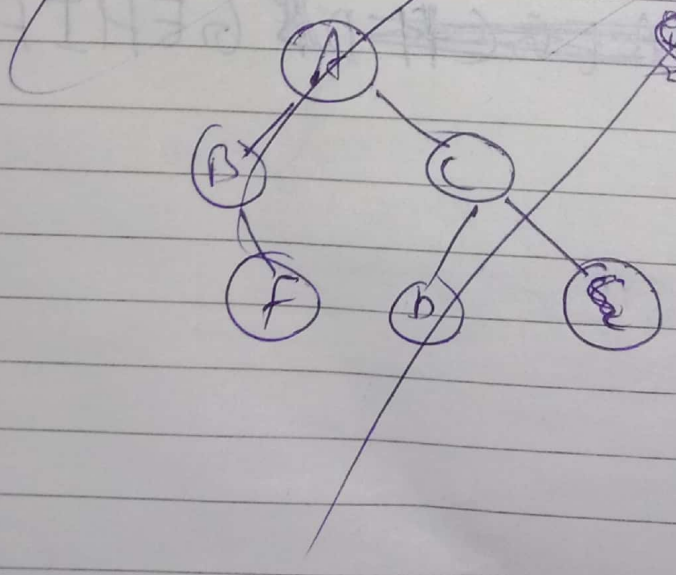
Preorder → F B A D C E G I H
Root left Right

Postorder → A C E D B H I G F
left Right Root

Pre - B A D C E
Post → A, C, E, D, B



Insertion and Deletion in a Binary Tree



Scanned with CamScanner

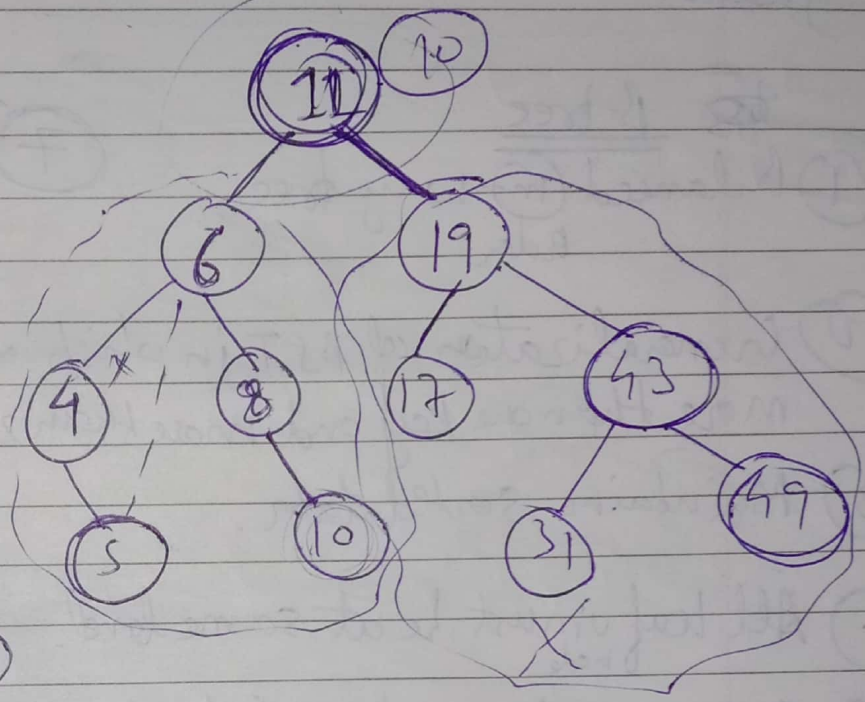
0/1/2 children

Date
DELTA Pg No.

Binary search Tree

Inserting left to right

11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31



Delete (8)

1. 0 children
2. 1 children
3. 2 children

Delete (31)

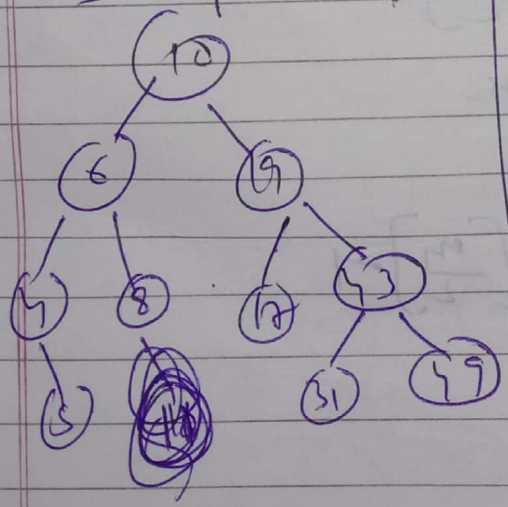
Delete (4)

Delete (11) → replace (19) by (17)

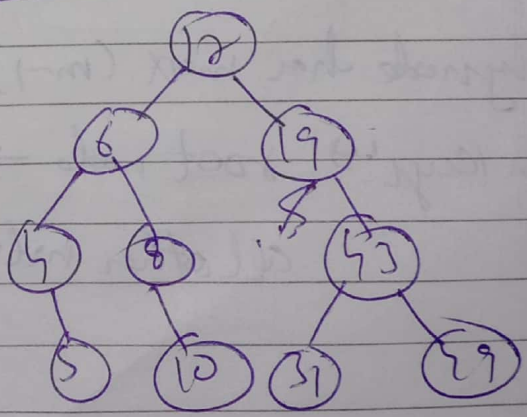
(i) inorder predecessor

(ii) inorder successor

Inorder predecessor (target element)



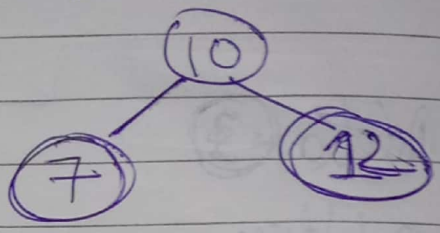
Inorder sucessor (smallest element)



Scanned with CamScanner

Binary Search Tree

A binary tree in which for each node, value of all the nodes in left subtree is lesser or equal and value of all the nodes in right subtree is greater.



B-tree

(1) Balanced (m) way tree
order

(2) Generalization of BST in which a node can have more than one key and more than 2 children

(3) Maintain sorted data.

(4) All leaf node must be at same level.

(5) B-tree order m has following properties:-

→ Every node has max m children

→ Min. children → leaf → 0
root → 2
Internal node = $\left\lceil \frac{m}{2} \right\rceil$

→ Every node has max. (m-1) keys.

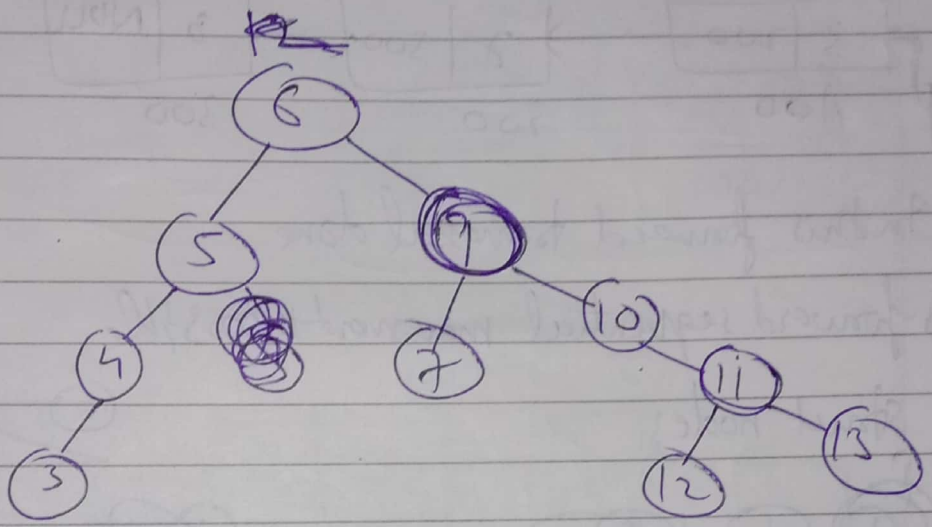
→ Min keys → root node → 1
all other nodes → $\left\lceil \frac{m}{2} \right\rceil - 1$

Scanned with CamScanner

BST

Insertion

6, 5, 9, 10, 11, 13, 7, 3, 4, 12

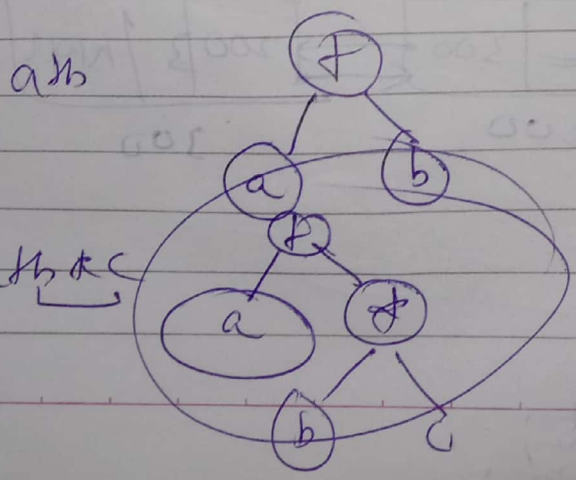


Application of tree

1) storing naturally hierarchical data → e.g → file system

2) Organize data for quick insertion, search, deletion
eg: B-S-T

3) Network Routing algorithm



Scanned with CamScanner

Algorithm for PUSH on the stack

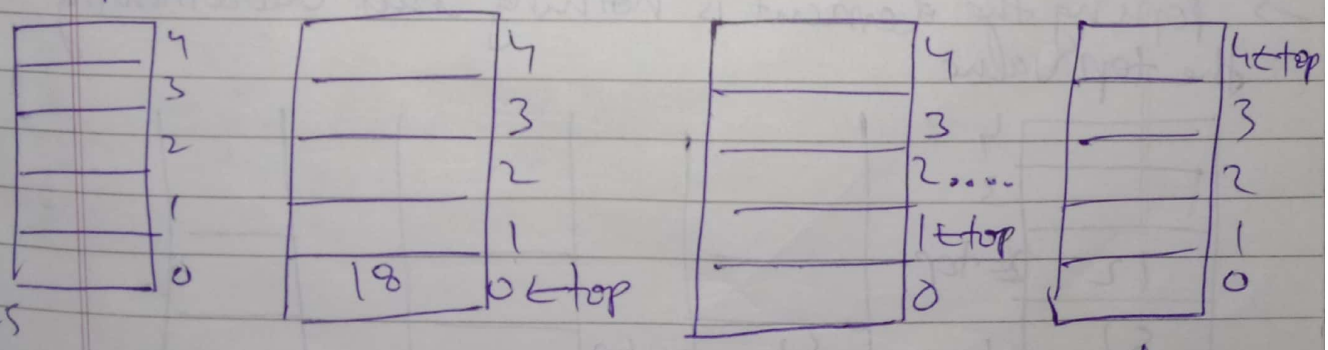
PUSH (STACK, size, top, item)

(i) stack :- It is a linear array
(ii) size :- size of the stack
(iii) top :- It is an integer variable, which holds the index of last element in the stack.
(iv) item :- The new element that is to be pushed into the stack.

Step 1 : if (top == size-1)
(a) print overflow
otherwise goto step 2.
Step 2 :- top = top + 1
Step 3 : STACK [top] = item
Step 4 :- exit.

Push operation

The following steps should be followed at the time of pushing any new element into the stack.
(i) checking of overflow.
(ii) Incrementation of the top value i.e, top++.
(iii) putting the new item/ element in the top position i.e., stack [top] = item.



Page 5

(a) -1 ← top
Empty stack

(b) 0 ← top
1st element pushed

(c) 1 ← top
2nd item pushed

(d) 4 ← top
5th item pushed

Scanned with CamScanner

Here, after pushing the 5th element into the array, if we want to push another item, then we will encounter the overflow situation because top = size-1.

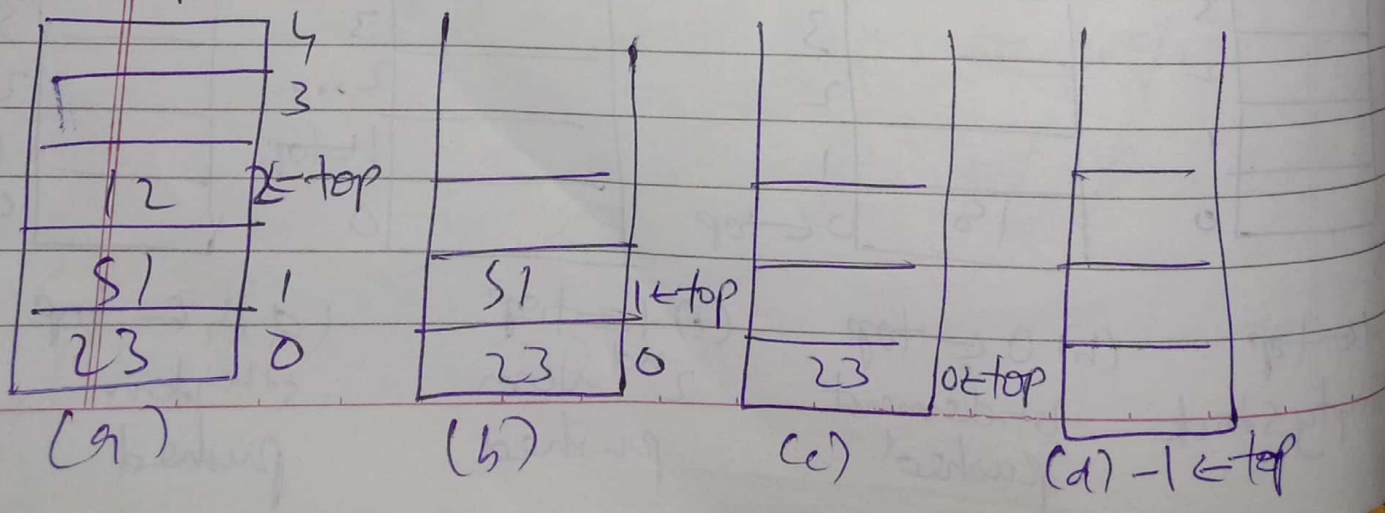
C-procedure:-

```
void push (int stack[], int item)
// Here "top" is declared as global variable
{
  if (top == size-1)
    {printf("Overflow");
     exit();
    }
else
  {
   top = top+1;
   stack [top] = item;
  }
}
```

Pop operation

→ Deleting the topmost element in the stack is called as pop operation.

→ Poping the element is nothing but decrementing the top value



Scanned with CamScanner

→ Consider the above figure, where initially only 3 elements were there in the stack and one by one element has been deleted in the subsequent steps a, b, c. At last fig (d) indicates that there exist no element in the stack so top = -1.

→ At the time of poping operation the following steps to be followed -

(i) checking the underflow condition

(ii) Decrementing the top value by 1.

Algorithm:

pop (stack, top, Ditem)

i) stack :- It is a linear array

ii) top :- It is an integer variable, which holds the index of last element in the stack.

iii) Ditem :- Before deleting or decrementing the top value the element will be stored in Ditem.

Step1 :- if (top == -1)
print "underflow" otherwise go to next step.

Step2 :- Ditem = stack [top]

Step3 :- top = top - 1

Step4 :- print the deleted item is Ditem.

Step5 :- exit

Scanned with CamScanner

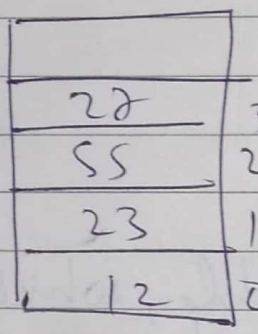
Peep operation

→ Extracting the required element from a given position of a stack is called peep operation

→ for checking whether the given position from where the element is to be extracted, is valid one or not, formula is top - pos + 1.

→ If (top - pos + 1 < 0) then it is a in-valid position

Example



pos = 3

So, top - pos + 1

= 3 - 3 + 1

= 1 > 0

So, position is valid

→ if pos = 5

top = 3 then top - pos + 1

= 3 - 5 + 1 = -1 < 0 i.e position is invalid

→ if pos = 4 then top - pos + 1

top = 3 = 3 - 4 + 1 = 0 is a valid position

Scanned with CamScanner

Algorithm

PEEP(stack, pos, item)

1) stack :- linear array
2) pos :- position from which element is to be peeped
3) item :- variable which will hold peeped element.

Step1 :- if (top - pos + 1 < 0)
(a) Print invalid position
(b) otherwise goto step2.

Step2 :- (a) item = stack [pos - 1]
↳ Assign stack [pos - 1] to item
(b) Print "The value of peeped item".

Step3 :- Exit.

Scanned with CamScanner

~~Atomic to Mol using~~

PSC

Postfix evaluation

Scan the given postfix expression from left to right symbol by symbol.

If the current scanned symbol is an operand push the operand into stack.

If the current scanned symbol is an operator:-

(i) Applied to pop operation on the stack.

(ii) Apply the operater and push back into the stack.

Repeat step 2 and step 3 until whole expression is scanned.

Pop the result out from the stack.

2 + 3 * 4 - 7 + 6 / 2

(1)

2 + 3 4 * - 7 + 6 / 2

2 + 34 * - 7 + 6 / 2

234 * + - 7 + 6 / 2

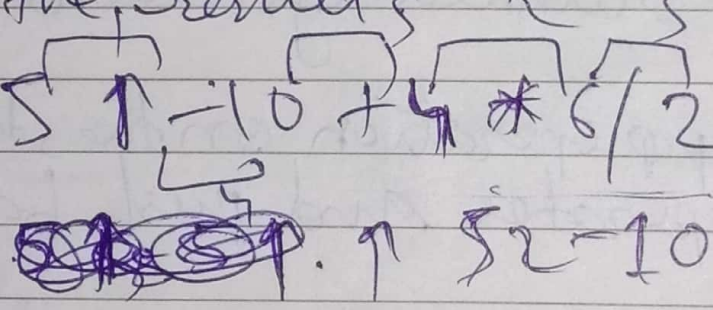
234 * + 2 - 6 / 2 +

My NoteBook Sign.

Scanned with CamScanner

Prefix Evaluation

* Scan the given prefix expression from right to left
* If the scanned symbol is an operand push the symbol into the stack.
* If the scanned symbol is an operator
  (i) Apply two times pop operation.
  (ii) Apply operator and push the result back into the stack.
* Repeat step 2 and step 3.
* Pop the result.



(↑ 5 2) =

+ - ↑ 5 2 10 / * 4 6 2

5 ↑ - 10 + 4 * 6 / 2

5^2 - 10 + 4 * 6 / 2

My NoteBook Sign.

Scanned with CamScanner

$= 25 - 10 + 4 \times 3$

$= 25 - 10 + 12$

$= 25 + 2$

$= 27$

Generating Permutation

1) 123

2) 34

3) 231

Sol: n = 3. What is possible sequence.

① 1 2 3

② 1 3 2

③ 2 1 3

④ 2 3 1

⑤ 3 1 2

⑥ 3 2 1

Sign.

Scanned with CamScanner

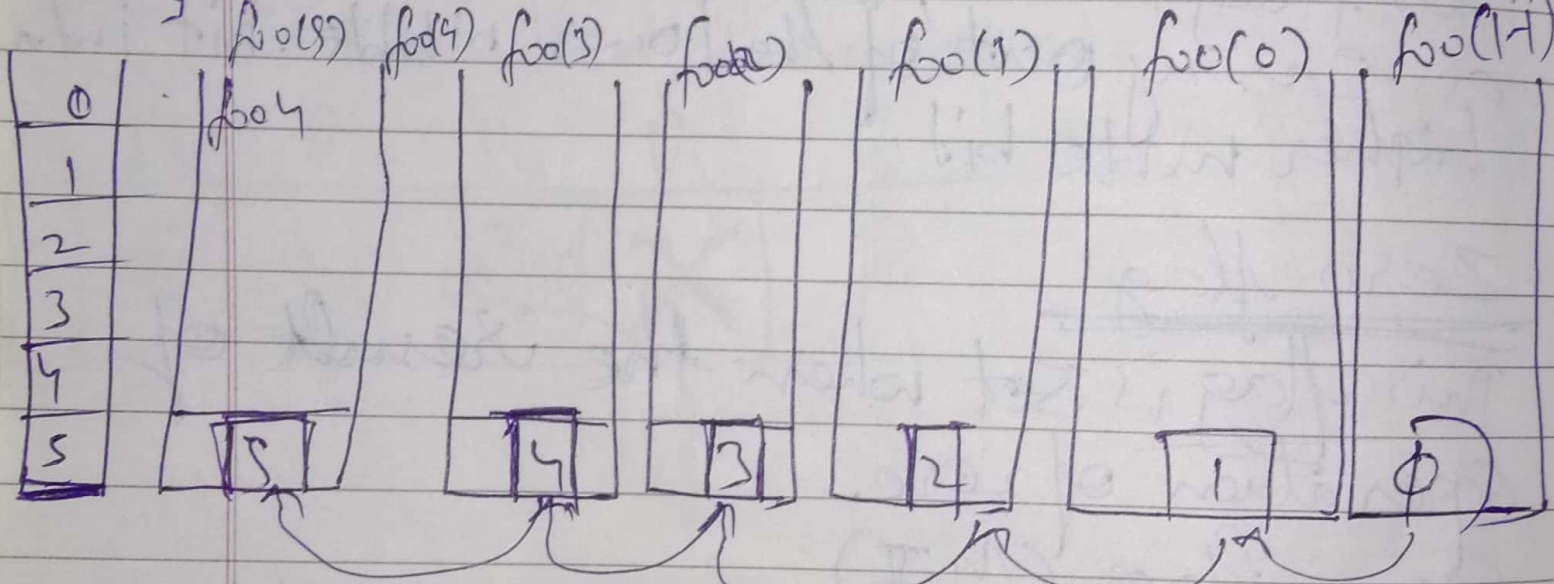
DSC

→ Iterative

→ Recursion

```
void foo(int n)
{ if(n<0)
  else
  { foo(n-1)
  printf("%d", n)
  }
}
```

3 3

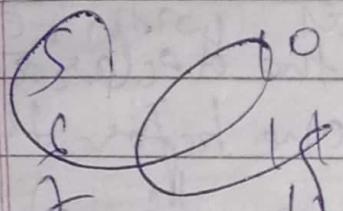


pop the result from the stack:-

| 0 | 1 | 2 | 3 | 4 | 5 | → Output

Stack → A stack is a non-premitive linear data structure, it is an ordered list in which addition of new data item and deletion of already existing data item is done from

My NoteBook Sign
Scanned with CamScanner


from any one end, known as top of the stack. As all the deletion and insertion in a stack is done from top of the stack the last element will be the first to be removed added from the stack. Ex- A common model of a stack is plate arrange in a marriage party.

Stack implementation: Stack can be implemented in two ways:

* Static implementation
* Dynamic implementation

Static implementation → State implementation uses arrays to create stack. The static implementation is a very simple technique but it is not flexible way of creation. In the size of the stack has to be declared during program design after that the size can not be vary. Moreover the static

My NoteBook

Scanned with CamScanner

implementation is not too efficient with respect to memory utilisation. As the declaration of Array for implementing is done before the start of the operation (At program design time). Now if there are two element to be stored in the stack the statically allocated memory will be wasted. On the other hand, if there are more no. of element to be stored in the stack then we can't be able to change the size of the array to increase its capacity.

Dynamic implementation :- The dynamic implementation is called as linked list representation and it uses pointer to implement the stack type of data structure.

Operations of stack

The basic operation that can be performed on stack are as follows :-

① Push :- The process of adding a new element with the top of the stack which called as push operation. Pushing an element in the stack is box adding of element, at the

new element will be inserted at the top after every push operation. The top is incremented by 1. In case the array is full and no new element can be accomodated it is called as stack full condition. The condition is called as stack overflow.

(2) Pop :- The process of deleting of element from the top of the stack is called as pop operation. After every pop operation the stack is decremented by 1 there is no element in the stack and the pop is performed then this will result to stack underflow condition.

Stack terminology :-

(1) Content :- The environment in which the function execute includes argument variable, local variable and global variable. All the content except the global variable is stored in stack frame.

(2) Stack frame :- The data structure containing all the data (argument, local variable, return address) needed each time a procedure or function is called.

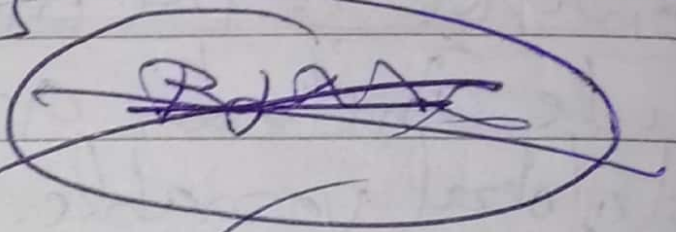
(3) Max Size :- The term is not a standard one, we use this term to refer the maximum size of the stack.

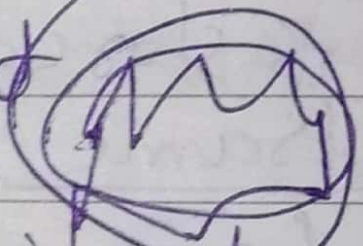
(4) top :- This term refer to the top of the stack. The stack top is used to check stack overflow or underflow condition. Initially the top stores (-1). This assumption is taken so that whenever an element is added to the stack the top is first incremented and then item is inserted into the location currently indicated by the top.

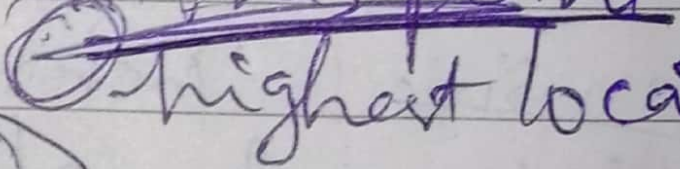
My NoteBook
Sign.
Scanned with CamScanner

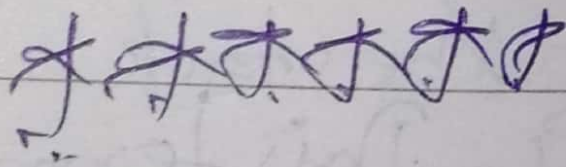
(5) Stack empty or underflow :- This is the situation when the stack contain no element. At this point of a stack. The top is present at the top bottom of the stack.

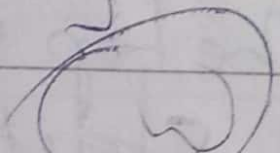
(6) Stack Overflow :- This is the situation when the stacks become full and no more element can be push on to the stack. At this point the stack top is present at the highest location of the stack.











Scanned with CamScanner

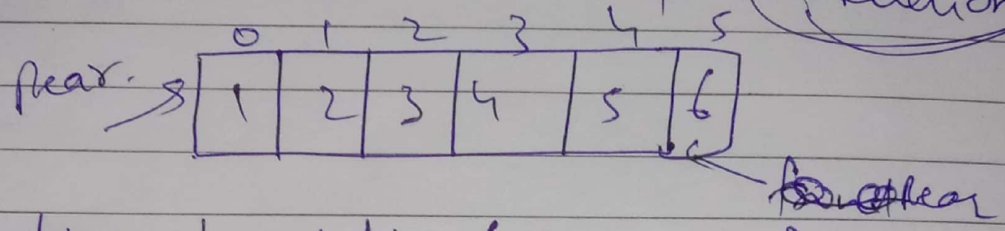
DSC [Data Structure Using C]

Date
DELTA Pg No.

Queues

Queues or other list queues can also be implemented in two ways :-

first in first out → Rear → Insertion
first in first out → front → Deletion



(i) static implementation (using array)
(ii) Dynamic implementation (using pointer)

If queue is implemented using array we must be sure about the exact no. of element we want to store in the queue because we have to declare the size of the array at design time or before the processing start. In this case the beginning of the array will become the front for the queue and the last location of the array will act as rear for the queue. The following relation gives the total no. of elements present in the queue while implementing using array.

$$F - R + 1$$

→ Queue operation

1) Create Empty-Queue
2) Is Empty
3) Is full
4) Insert

Scanned with CamScanner

5) Relate
6) Size

```
typedef struct Queue
{
  int f,r;
  int element [max];
}
Struct Queue *q
```

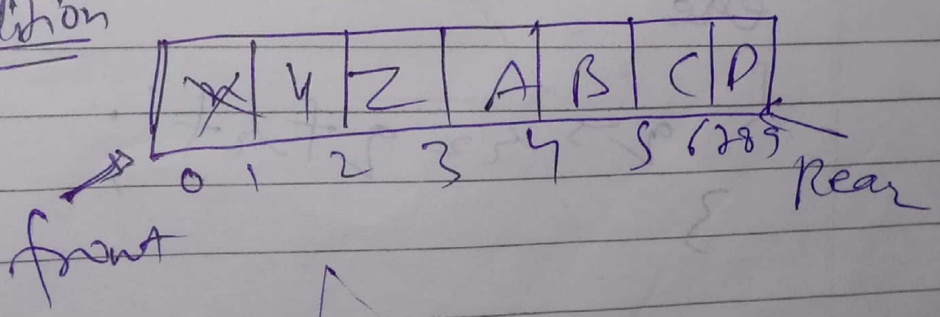
→ Void create empty_queue (queue *q)

```
{
  q->f = q->r = -1
}
```

Underflow condition

```
int Is_empty(queue *q)
{
  if(q->f==q->r==-1)
  return 1;
  else
  return 0;
}
```

Overflow condition



Scanned with CamScanner

```
int isfull (queue *q)
{
if (q->f==0) && (q->r == MAX-1)
return 1;
else
return 0;
}
```

→ Void insertion (queue *q; int data)

```
{
if (q->f == q->r == -1)
    q->f = q->r = 0;
else if (q->f != 0) && (q->r == MAX-1)
{
for (i = q->f; i <= q->r; i++)
{
q->element [i - q->f] = q->element [i];
q->r = q->r - q->f + 1;
q->f = 0;
}
else
{
q->r = q->r + 1;
}
```

front to below

swapping

Date
DELTA Pg No.

q→ element [q→r] = data;

Deletion

→ Void deletion (queue *q)
{
int temp = q→element[q→f];
if (q→f == q→r)
{
q→f = q→r = -1;
return 0;
}

else
{
q→front = q→f+1
return temp;
}

Ex

DSC

Date
DELTA Pg No.

Circular Queue

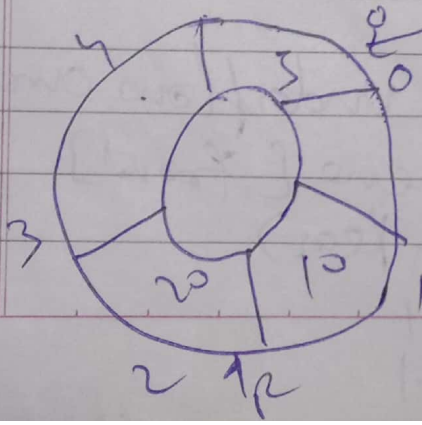
In which the insertion of new element is done at the very first location of the queue. If the last location of the queue is full then the next element will be inserted at the very first location. A circular queue overcome is the problem of unutilise space in linear queue implemented as array.. A circular queue also has a front and rear to keep the track of the element to be deleted. The below instruction are as under.

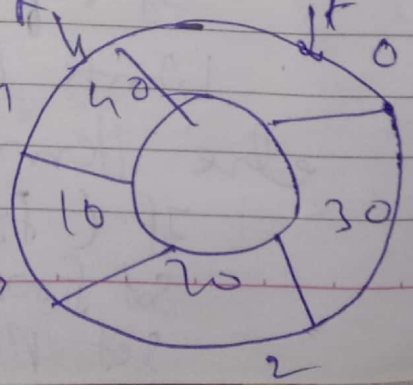
1. Front is always be pointing to the first element.
2. If the front is equal to the rear the queue is empty.
3. Each time the new element is inserted the rear is incremented by 1.
   R = R + 1
4. Each time an element is deleted the front is equal to front + 1

Rear = (Rear + 1) % MAX_SIZE
Queue(Rear) = Value

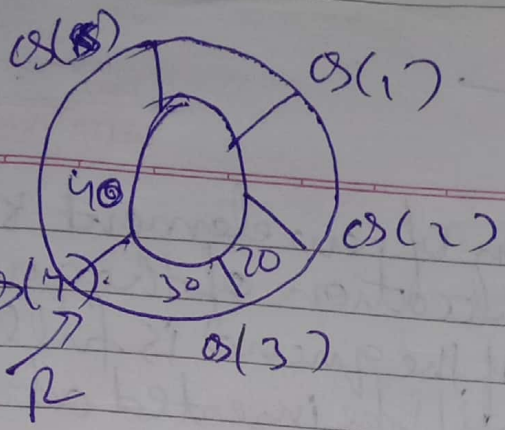
Rear = (Rear + 1) % MAX SIZE

(4 + 1) % 5 = 0





Scanned with CamScanner



INSERT (QUEUE [MAX SIZE, ITEM])
If (FRONT == (Rear+1) % MAXSIZE
Write Queue overflow & Exit
else: Take the value
If (FRONT==-1)
Set front = REAR=0
else
Rear = ((Rear+1) % MAXSize)
Queue[Rear] = value;
[End-If]

Relation

(1) DELETE (Queue [MAXSIZE, ITEM]).

If (f==-1)
Write Queue underflow and exit
else item = Queue[front]
If (front == Rear).
Set front = -1
Set Rear = -1

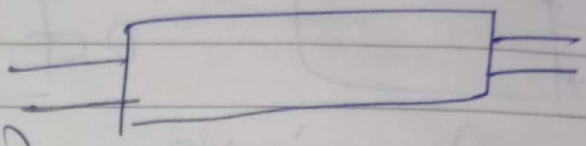
Scanned with CamScanner

else :- front = (front + 1) % MAXSIZE

exit.

Variants of Queue

Dequeue
(Double ended Queue)



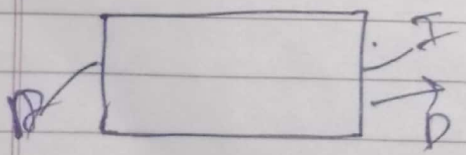
Dequeue is called as double ended queue.

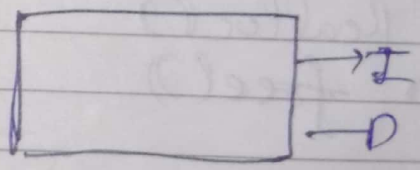
(*) The insertion and Deletion can be done from both the ends. In b/w insertion and deletion is not allowed.

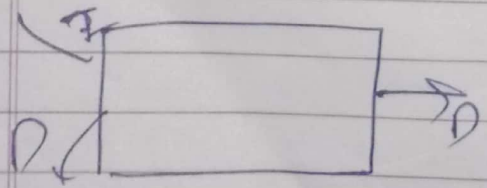
If we want to implement stack or dequeue then restrict the other end operation and it will behave as stack.

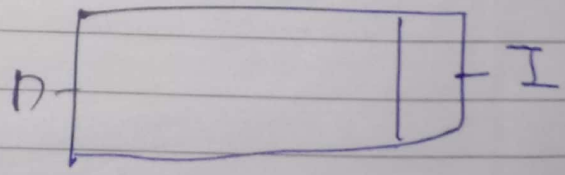
from one side restrict the insertion from the other side restrict the deletion.

(i) Input restricted dequeue



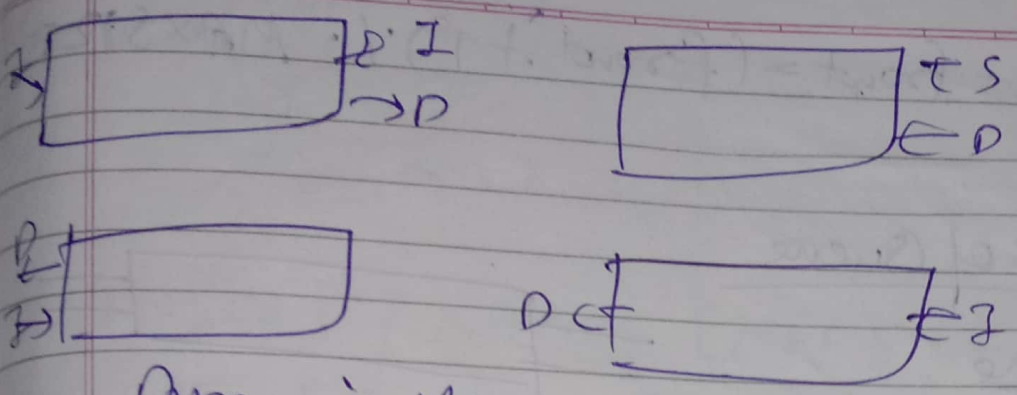






(ii) Output restricted dequeue.

Scanned with CamScanner

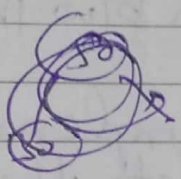


Dynamic Memory

- Heap
- Reign
- Heapify
- Max Heap
- Min Heap

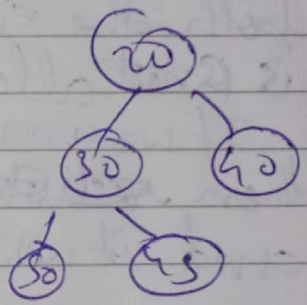
Max heap

50, 30, 40, 20, 45



Min heap

20, 30, 40, 50, 45



Dynamic memory allocation

- Malloc()
- Calloc()
- Realloc()
- free()

Scanned with CamScanner

Date
DELTA Pg No.

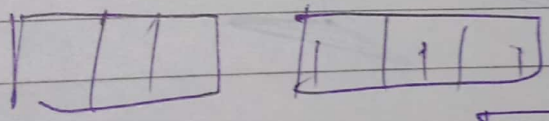
→ Malloc() it represents the memory allocation at runtime. The malloc() it include the memory size. If the memory is available it allocate the memory. If the memory is unavailable it returns null. The memory given to you is in the continuous allocation by default the O.S return the void pointer. The allocated address may contain garbage value so we can reasigned it.

~~Calloc()~~ :- → free previous allocated memory

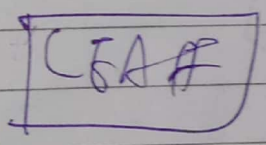
→ Realloc() :- → In realloc we perform the following steps :-

1. Add copy to new block
2. Old block if destroyed
3. New block address is returned.
4. Leaves the previous memory.

int * = (int *) malloc (pointer the)



How much memory allocated core leaf.



Scanned with CamScanner

Tree

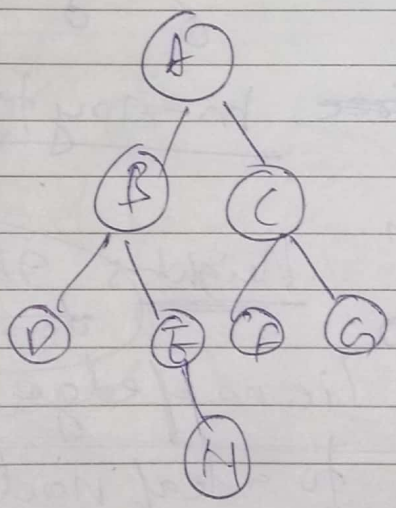
Tree is a complete m-ary tree. A tree is connected acyclic graph with n-vertex and n-1 edges.

Binary Tree

A rooted tree in which every node has almost 2 children that means 0, 1, 2 children are possible.

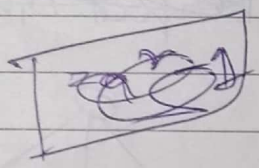
Depth of the tree

The number of edges on the path from the node to the root node is called as depth of the tree.

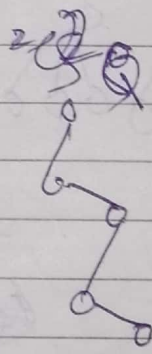


No.

Maximum no. of binary tree



2^n

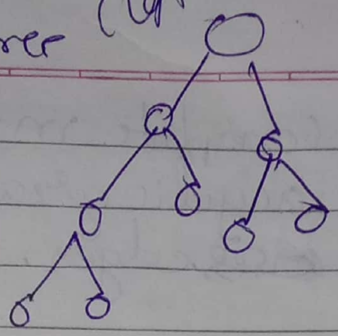


$= n+1$

Date
DELTA Pg No.

Kag(4092)→(1)

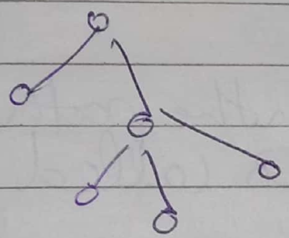
Complete Binary Tree (left occupied)

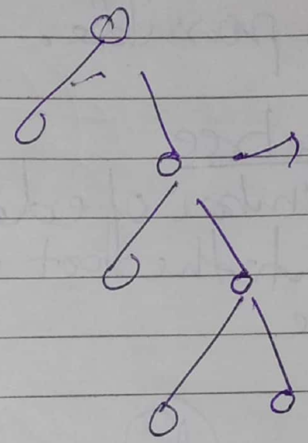


full Binary tree

fixed B-T (0, 2) children.

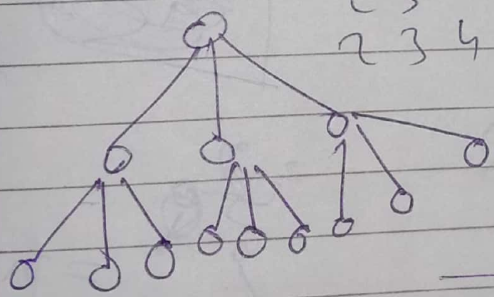
~~Strict binary Tree~~





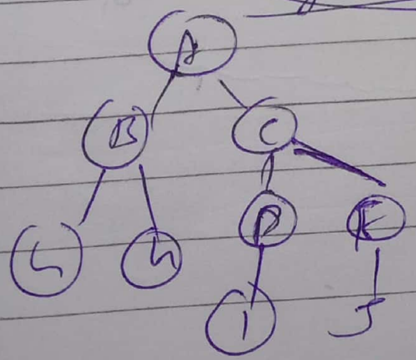
~~Perfect Binary Tree~~. m-ary tree

2 3
2 3 4



Height of the longest downward path (i.e. no. of edge) from node to a leaf node

The height of a binary tree is the height of root node.



height of node C
= 2

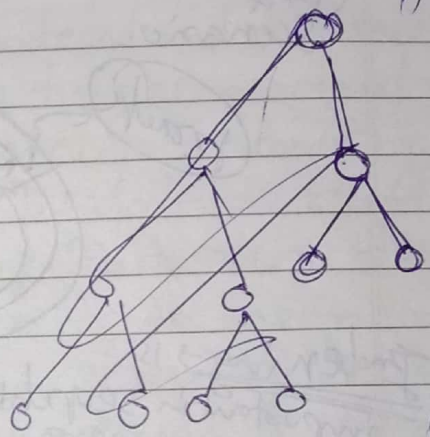
(1) Max no. of nodes

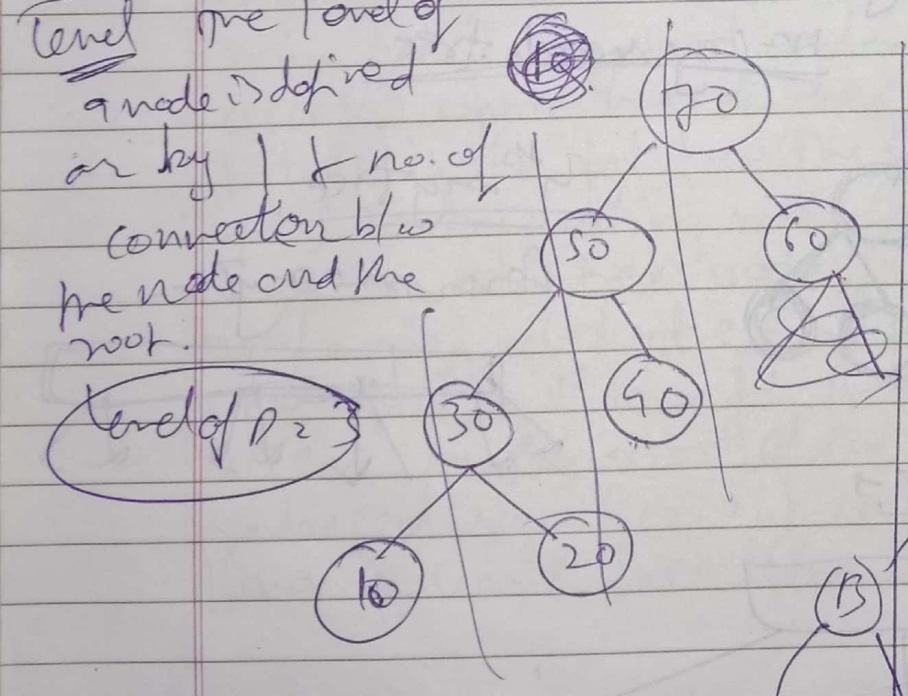
$n = 2^{n+1} - 1$

min, $n = n+1$

B.T = BST

m-bare tree

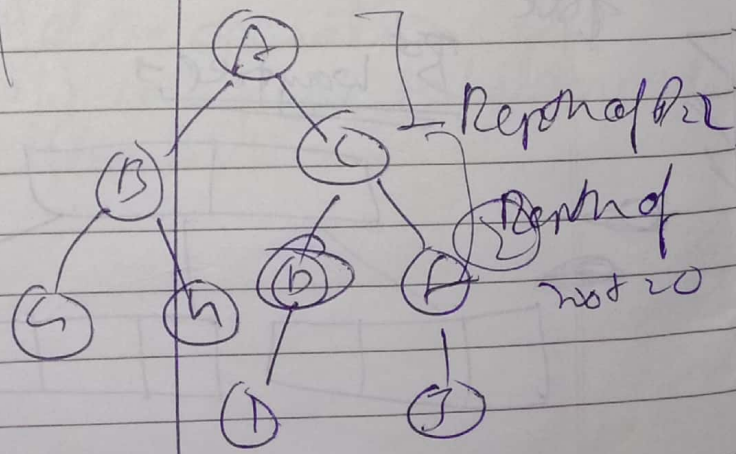




Level :- The level of a node is defined as by 1 + no. of connection b/w the node and the root.

Level of D = 3

Depth :- The depth of a node is the longest upward path from the node to the root node



Depth of root 20

Date
DELTA Pg No.

B Tree → disk known height balancing Tree.

B Tree is defined as well balancing search tree. In most of the other search balancing tree like AVL. It is assume that everything is in the main memory to understand the use of B tree we must think of the huge amount of data that cannot fit in main memory, then the no. of tree is high. The data is read from disk in the form of block. This access time is very high compared to main memory access time. The main idea of using b tree is to reduce the no. of disk access. Most of the tree operation (search, Insert, delete, max, min, etc) required O(h) disk access where h is the height of the tree. The b-tree is the fat tree. The height of b-tree is kept low by putting maximum possible keys in a B-tree nodes. Generally, a b-tree node size is kept equal to the disk block size. Since, h is low for B-tree the total disk access for most of the operation are reduced significantly compare to balanced binary search tree like AVL.

10, 20, 30, 40 | 90, 40, 50

Date
DELTA Pg No.

Properties of B-tree :-

* All leaves are at same level.
* A b-tree is defined by the term minimum degree 't'. The value of t depend upon the disk block size.
* Every node except root must contain atleast t-1 key. Root may contain minimum one heap.
* All nodes (including root) may contain at most 2t-1 keys.
* The no. of children of a node is equal to the no. keys in it plus one.
* All keys of a node are sorted in increasing order. The child b/w the two keys k_1 and k_2 contain all leaf in the range from k_1 and k_2.
* The b-tree grows and shrinks from the root which is unlike in B.S.T. In Binary search tree grow downwards and also shrink from downwards. Binary search
* Like other balanced tree time complexity to search, insert and delete is order of log n.

Scanned with CamScanner

Introduction

If a memory is allocated before the execution of a program which is fixed and cannot be changed. Here is a special data structure called linked list that provide a more flexible storage system and it does not require the use of array.

For stack and queue we employed array to represent them in computer memory then we choose array representation. It is necessary to declare in advance the amount of memory to be utilize. When we rarely take up array all the memory may not be used even though allocated.

Linked list

Linked list are special list of some data element linked to one another. The logic ordering is represented by having each element pointed to the next element. Each element is called as node. We have two parts, info part which stores the information and pointer part which points to next element.

Scanned with CamScanner

Date DELTA Pg No.

elements are always added to back and removed from the front.

operations of Queues

~~Enqueue~~

front (delete)

Get the front item from queue

Rear (front)

Get the last item from queue.

~~Algorithm~~ Overflow condition in Queue

Enqueue → overflow

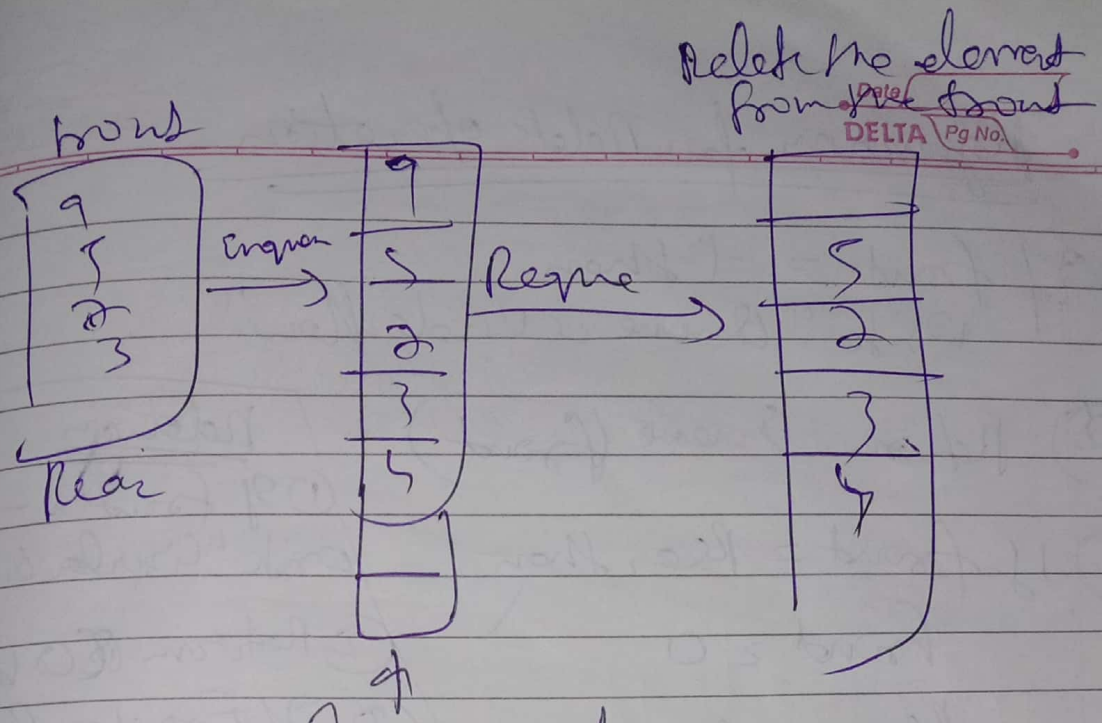
Dequeue → Underflow

if(rear = arrays.S;2)
print
print(rear) = element

Enqueue: Adds an element to the back of the queue. if the queue is not full otherwise it will print "Overflow".

Dequeue: Removes the element from the front of the queue if the queue is not empty otherwise it will print "underflow.

if(front == rear)
print (underflow
else
queue (front = 0),
front +1),



Algorithm for Insert operation

(1) If Rear >= SIZE - 1 then
Write "Queue is Overflow"

(2) Rear = Rear + 1

(3) Queue[Rear] = X

(4) If FRONT = -1, then
FRONT = 0.

Scanned with CamScanner

Algorithm for Delete operation

(1) If front = -1 then
write "Queue is Underflow"

(2) Return Queue (front)

(3) If front = Rear then
front = 0
Rear = 0
else
front = front + 1

Deletion

(1) If front = -1, then
write "Circular Queue is empty")

(2) Return (Q(front))

(3) If front = Rear then
front = Rear = -1

(4) If front = Size - 1 then
front = 0
Else
front = front - 1

Circular Queue Insertion

(1) If Rear = Size - 1 then
Rear = 0
Else
Rear = Rear + 1

(2) If front = Rear then
write "Circular Queue Overflow");

(3) Q[Rear] = x

(4) If front = -1, then front = 0

Scanned with CamScanner

B tree

→ Balanced m-way tree,

→ Maintain sorted data

→ All leaf node must be at same level,

→ Generalization of BST in which a node can have more than one key and more than 2 children

Properties

(1) Every node has max. m children

(2) ~~Min. children~~ Every node has maximum m-1 keys

(3) Min. children → leaf → 0
root → 2
internal node = $\lceil \frac{m}{2} \rceil$

(4) Min. key → root node → 1
all other node → $\lceil \frac{m}{2} - 1 \rceil$

Scanned with CamScanner

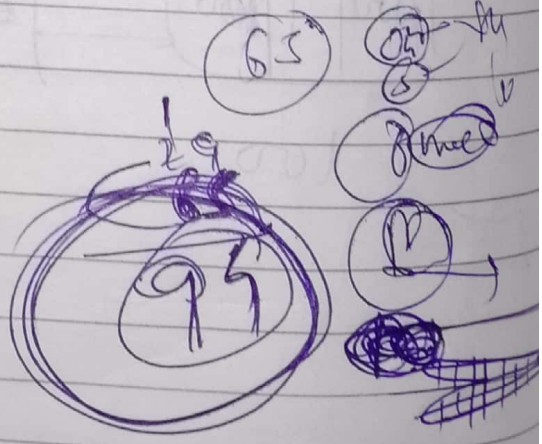
~~Doubly~~

Application of linked list

- Songs playing in a media player
- Person standing for food in a mess
- It is used to implement in stack and Queues.
- It is also used as dynamic memory allocation.
- Circular queue is used in our computer when multiple application are running

~~Disadvantages~~ Advantages

- Dynamic list
- More efficient utilization
- Insertion and Deletion are easier.
- More complex application are also easily done with the linked list.



Scanned with CamScanner