2025 Feb 8 | Module 2

Data structure and Algorithm (DSA)

by Turinumukiza Maurice.

Chap 1. Introduction

Generally DS (data structure) are how data are arranged.

Algorithm - steps by step on solving problem.

① **Data structure** - way of organizing and storing data in computer so that it can be accessed and modified efficiently.

For various application like DB, web Indexing web service...

Data $\xrightarrow{\text{process}}$ Information

Importance

- ✓ DS study how data are stored so that operations performed efficiently.
- ✓ Important to deal with large amount of data.
- ✓ conceptual and concrete way to organize data.

classifications

① . **Primitive DS**

↳ basic building blocks of data manipulat° in programming...

↳ store single value of data.

ex. • Integer
- floating point
- character
- Boolean

↳ Memory representation / storage of Primitive
⇒ Their memory are predefined and user can not change them.
ex Int { 2 bytes (32-bit)
4 bytes (64-bit)

nce b/n algorithm and promm. language
⇓
* ordered steps to solve the problem
whereas.
Promm. language
↳ collection of symbols and characters used to make set of instruction for solving problem.

(ii). Non-Primitive / complex / collection
↳ more complex DS that are derived from primitive datatypes.
↳ used to handle large amounts of data

classifications
a) Linear DS - elements are arranged sequentially or in linear order.
ex
- Array
- Linked List
- Stack
- Queues ... etc

b) Non-linear DS - data elt are not arranged sequentially or in linear order Instead, they are organized in a hierarchical order.
ex. Trees, Graphs ... etc

↳ **Storage representation**

requires more memory due to additional info they store such as pointers, references and metadata...

ex ⊗ Arrays

contiguous Memory allocation...

⊗ Linked lists.

Nodes with Pointers

↳ contains data and reference to a next node.

Examples simply linked list of 10 integer nodes

$$[\underbrace{4\text{ bytes}}_{\text{data}} + \underbrace{4\text{ bytes}}_{\text{reference}}] * 10 = 80\text{ bytes}$$

Frequently used operations on DS

a) Traversing - Accessing each record exactly once so that certain items in the record may be processed.

b) Searching

c) Inserting

d) Sorting

e) Merging (combining two records)

↳ two ~~int~~ sorted lists into single sorted list.

② . **Algorithm** - set of ordered steps or procedures necessary to solve problem.

problem

↓

Algorithm

↓

XCs of good algorithm

A good algorithm - performs efficiently to solve the problem it's intended to solve.

(i). Correctness

Correctly solve the problem

(ii). Efficiency → O - complexity

↳ Time complexity (min time taken)

↳ space complexity (memory used)

(iii). Finiteness

should start and have terminate point...

(iv). Clarity and simplicity

(easier to understand and implement)

(v) Generality.

(should be applicable to a broad range of input.)

(vi) Robustness

(able to handle unexpected (unusual inputs gracefully).

(vii) Maintainability

(easy to modify, change and update).

Chap 2. Arrays

→ collection of similar type of data elt stored at consecutive locations in the memory

Array dimensions

- direct° in wic you can vary the specs of an array's elements.
- It can be one or many dimension.

ex arr [4] → dimension (1D)

arr [2][5] → columns, → rows } 2Dimensions

↳ 2D Array declarations

SET Array_name = Array [row][column] of datatype

ex.

SET Array_num = Array [2][7] of Integer.

↳ 2D Array Initialization

Assigning values to the declared array:

Ex marks for 3 students in 4 courses

The array:

18	16	14	19
10	20	12	15
15	17	18	16

BEGIN

SET marks = Array [3][4] of Integer

marks[0][0] = 18

marks[0][1] = 16

marks[0][2] = 14

marks[2][3] = 16
END

Exercise

5 universities in kigali are displaying nb of students per year that are registered in 2021, 2022, 2023 and 2024.

a) How many rows and columns will be in the table that contains the displayed data?

{ 5 rows
  4 columns

b) Declare an array called population to contain them.

SET population = Array [5][4] of Integer.

c) Initialize the array population with nb of your choice.

```
BEGIN
SET population = Array [5][4] of Integer
population [0][0] = 150
population [0][1] = 200
        :
population [4][3] = 300
END
```

Accessing elements of 2D

There two ways:

- READ (storing data) ex READ Array-name [i][j]
- WRITE (displaying data) ex WRITE Array-name [i][j]

Example: Write algorithm of program that allows the user to enter (store) 2D elts. of 3 rows and 8 columns:

```
BEGIN
SET marks = Array[3][8] of Integer
Use variable i as Integer
Use variable j as Integer
FOR i=0 TO 2 DO
   FOR j=0 TO 7 DO
      READ marks[i][j]
   END FOR
END FOR
END
```

Exercise (Assignment)

Given 2D array of data
mat A = [8][8]

a/ Write algorithm to display all elt of matrix

```
BEGIN
SET matrix = Array[8][8] of Integer
Use variable i, j as Integer
WRITE "matrix A Elements are:"
FOR i=0 TO 7 DO
   FOR j=0 TO 7 DO
      WRITE matrix[i][j]
   END FOR
END FOR
END
```

b/ The array of students and subjects of 10 students (row) and 8 subjects (column)

① store grades for each student in array.

```
BEGIN
SET student_grades = Array[10][8] of character
Use variable i, j ase Integer
WRITE " Enter the subject grades"
FOR i=0 TO 9 DO
   FOR j=0 TO 7 DO
      READ student_grades[i][j]
   END FOR
END FOR
END
```

(ii). Displaying grades on screen in formatted manner (row by row).

```
BEGIN
SET student_grades = Array[10][8] of character
Use variable i, j as integer
WRITE " Students grades are :"
FOR i=0 TO 9 DO
   FOR j=0 TO 7 DO
      WRITE student_grades[i][j]
   END FOR
END FOR
END
```

Operations on Arrays

i) Insertion
ii) Deletion
iii) Merge
iv) Traversal
v) Find.

↳ Insert Element at the End of Predefined Prefilled Array:

Q. Analyze Below the execution :

```
BEGIN
  SET Array = array of Integer with the size N
  SET currentSize = Initial Nb of Elt.
  Use variable newElt as Integer

  PRINT "Enter the element to insert:"
  READ newElt

  IF currentsize < N THEN
     SET array[current size] = newElt
     SET current size = current size + 1
     PRINT "Element inserted successfully"
  ELSE
     PRINT "Array is Full."
  END IF
END
```

Q. Analyze:

Pseudo code: Insert Element at specified Position:

```
BEGIN
  SET array = array of Int with size N
  SET currentSize = Initial Nb of Elt
  Use variable newElt as Integer
  Use variable position as Integer

  PRINT "~~Et~~ Enter Element to insert"
  READ newElt

  PRINT "Enter the position to insert (0-based Index)"
  READ position
```

```
IF position < 0 OR position > currentsize THEN
    PRINT "Invalid position."
ELSE IF currentsize < N THEN
    FOR i= currentsize - 1 TO position STEP -1 DO
        SET array [i+1] = array [i] // shift elt to right
    END FOR
    SET array [position] = newElt // Insert new element
    SET currentsize = currentsize + 1
    PRINT "Element inserted successfully"
ELSE
    PRINT "Array is full"
END IF
END
```

Chap 3. Linked lists

→ They are linear DS where elt are stored in nodes and each node contains a reference (or link) to the next node in the sequence.
↳ allows efficiency insertion and deletion.

↳ Each node consist of :
- (*). Data (holds actual value)
- (*). Pointer (Link) [contains reference or link to the next node in the sequence.]

Types : (i). Singly
(ii). Doubly
(iii). Circularly.

(i). Singly linked list

- ✓ Each node contains data and pointer to the next node.
- ✓ Traversal is forward-only from head (start) to the tail (end).
- ✓ Memory-efficient for insertions and deletions at the head, but accessing elements by index requires linear traversal.

Head → 10 → 20 → 30 → 40 → NULL.

Algorithm in single linked list

a/ Creating and read an element in a NODE

⚠ Analyze below :

```
PROCEDURE  CreateNode (x)
    temp ← ALLOCATE (Node)
    temp.info ← x
    temp.next ← NULL

    IF Head = NULL THEN
        Head ← temp
    ELSE
        current ← Head
        WHILE current.next ≠ NULL DO
            current ← current.next
        END WHILE
        current.next ← temp
    END IF
END PROCEDURE
```

<u>Note</u>: Temp - temporary pointer used to create and initialize a new node.
Current - pointer used to traverse list

b) <u>Inserting at specific location in the list (After a Node</u>

⚠ Analyze:

Procedure InsertAfter (NodeData location, NodeData newval

Step 1: create a newNode with value newval

Step 2: If head = = NULL:

SET newNode → next = NULL

SET head = newNode

Else

Define node pointer temp and initialize to head

WHILE temp != NULL and temp → data != location

SET temp = temp → next

```
IF temp == NULL:
    Display "Given node not found"
ELSE
    Create newNode with value newval
    SET newNode -> next = temp -> next
    SET temp -> next = newNode
END/IF
```

c/ <u>Deleting a specific Node from list</u>

<u>Analyze</u>:

```
// Step 1. Check if list is Empty
IF Head == NULL THEN
    PRINT " List is Empty, Delete Impossible "
    EXIT ALGORITHM
END IF

// Step 2. Initialize pointers
temp1 = head
temp2 = NULL
// step 3. Traverse the list to find the node to delete
WHILE temp1 != NULL and temp1 -> Info != target DO
    temp2 = temp1.
    temp1 = temp1 -> next
END WHILE

// Step 4. Handle case where node to delete is not found
IF temp1 == NULL THEN
    PRINT " Given Node not found in the list! "
EXIT ALGORITHM
END IF

// Step 5. delete node
```

```
IF temp1 == head THEN
    // case 1. If temp1 is the first node
    head = head -> next
    delete temp1.
ELSE
    IF temp1 -> next == NULL THEN
        // case 2. If temp1 is the last node
        temp2 -> next = NULL
        delete temp1.
    ELSE
        // Case 3. If temp1 is an intermediate node
        temp2 -> next = temp1 -> next
        delete temp1.
    END IF
END IF
// Step 6. End of algorithm
EXIT ALGORITHM
```

09/02/2025

Deletion in linked list

1. check head, if head == Null
   (Empty, Deletion is not possible)
2. Else, head ≠ NULL (not empty)
   ex: Head → 2 → 5 → 9 → 20 → NULL
   ↑ temp1 ↑ temp2
3. Set temporary point at head and null to traverse for target value (delete 9)
   Using Loop until temp1 = 9
   Since,
   temp1 ≠ 7, temp2 = temp1, temp1 = temp1 -> next
   temp2 = head, temp1 = 2, 2 ≠ 9

temp 1	temp 2	target value (9)
temp1 = 2	temp 2 = head	2 ≠ 9
temp1 = 5	temp2 = 2	5 ≠ 9
temp1 = 9	temp 2 = 5	9 = 9

Terminate the iteration (END while / Loop).

⚠ case Element is not in the list

For ex delete 22, it will continue:

temp1 = 20	temp 2 = 9	20 ≠ 22
temp 1 = NULL	temp 2 = 20	NULL ≠ 22

⇒ condition to End / loop since temp1 = NULL then "Deletion not possible, since item to delete not found".

Otherwise,

(4). Delete element by checking its position

(i). First node (temp1 == head)?

(ii). last node (temp1 → next = Null)?

(iii). Intermediate node

{ set temp2 → next = temp1 → next
delete temp 1 (delete 9) }

Therefore, new list:

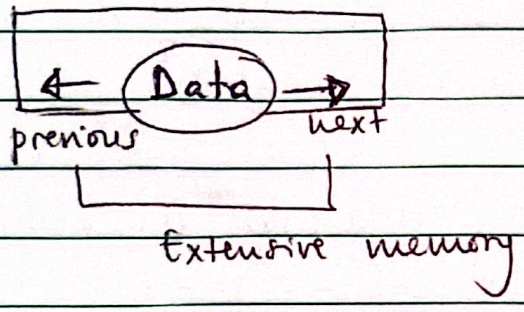
Head → 2 → 5 → 20 → NULL

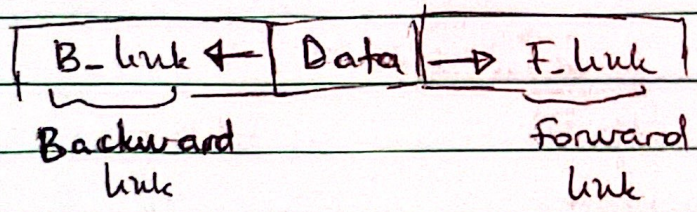
d) <u>Displaying a single linked list. (Analyze)</u>

```
PROCEDURE    Display ( head)
        current := head
        While current != NULL DO
                print current.info // current.data)
                current := current.next
        END WHILE
END PROCEDURE
```

②. Doubly linked

- Each node has data, pointer to next and pointer to previous.
- Allow bi-directional traversal (forward and backward)
  head ⇌ tail.
- More memory intensive due to extra ~~node~~ pointer per node but allows efficient insertion and deletion at both ends and easier traversal in both directions





Algorithm for Doubly

①. Inserting at specific location in the list (Analyze)

PROCEDURE InsertAfter (NodeData locat^n, NodeData newva

Step 1. Create newNode with newval

Step 2. If head == NULL:

SET newNode → previous = NULL

SET newNode → next = NULL

SET Head = newNode.

Step 3. ELSE:

Define two node pointers temp1 and temp2

temp1 = head

WHILE temp1 != NULL AND temp1 -> data != location:
SET Temp1 = temp1 -> next.
IF temp1 == NULL:
Display "Given node is not found in the list! Insertion not possible"
ELSE:
SET temp2 = temp1 -> next
SET newNode -> previous = temp1.
SET newNode -> next = temp2
IF temp2 != NULL:
SET temp2 -> previous = newNode
SET temp1 -> next = newNode

Summary:
1. creating new node with value new value
2. check if list is empty.
(head == head)
3. Iterate through the list to find location at which you want to insert your new node.
4. check the position ~~of~~ for entering newvalue
5. Use temp pointers to swapp, point link and insert newNode.

Head -> 2 ⇄ 4 ⇄ 6 ⇄ 8 -> NULL

2. Deleting a specific node from the list

PROCEDURE Delete (NodeData value to delete):
Step 1: IF head == NULL:
DISPLAY "list is Empty"
Terminate.
Step 2: Define node pointer and initialize to head
temp1 = head.

Step 3. WHILE temp1 != NULL and temp1 -> data != value:
SET temp1 = temp1 -> Next
Step 4: IF Temp1 == NULL:
DISPLAY " Given node not found "
Terminate
Step 5: IF Temp1 -> previous == NULL and temp1 -> next == NULL:
SET head = NULL // Single node
DELETE Temp1 (Free (temp1))
Step 6: IF Temp1 -> previous == NULL:
SET Head = temp1 -> next
IF temp1 -> next != NULL:
SET temp1 -> next -> previous = NULL
DELETE temp1 (Free (temp1))
Step 7: IF temp1 -> next == NULL:
SET temp1 -> previous -> next = NULL
DELETE temp1 (Free (temp1))
Step 8: ELSE:
SET temp1 -> previous -> next = temp1 -> next
SET temp1 -> next -> previous = temp1 -> previous
DELETE temp1 (Free (Temp1))

③. Displaying Doubly (Analyze)
Display Linked (head):
(i). IF head == NULL:
Display " List is Empty "
Terminate.
(ii). temp = head
(iii). Display "Null <="
(iv). WHILE temp != NULL:
Display temp -> data
IF temp -> next != NULL:

temp = temp → next

(v). Display "---> NULL"

Summary

1. check if list is not empty, otherwise
2. Iterate through the List using temp pointer
3. display data element

3. Circular linked

- Formed by connecting last node to the First node creating a circular loop.
- Singly or doubly linked versions exist.
- No Null value.

Algorithm on circular

1. Inserting at specific location in the list (after node). InsertNode (head, location, value);

Step 1 { Create newNode with value newvalue

Step 2 { Check if head == Null : (Empty list)
head = newNode
newNode → next = head
return.

Step 3.
temp = head

Step 4. traverse through the list
WHILE Temp → next != head and temp → data != location:
temp = temp → next

Step 5. (location not found)
IF temp → next == head and temp → data != location
Display "not found"
return.

Step 6. // Insert new node at the end.
IF temp → next == head:

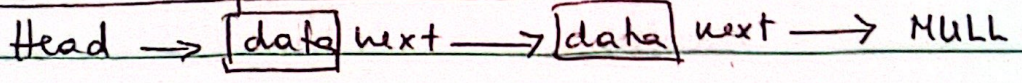
temp → next = newNode
Newnode → next = head.
return.

Step 7. // Insert newNode in between.
newNode → next = temp → next
temp → next = newNode.
// End algorithm.

(ii). Deleting a specific Node from the list
DeleteNode (head, valueToDelete):
Step 1. Check if list is empty
Step 2. Iterate through list to find value to delete.
Step 3. Delete according to the position of the value to delete

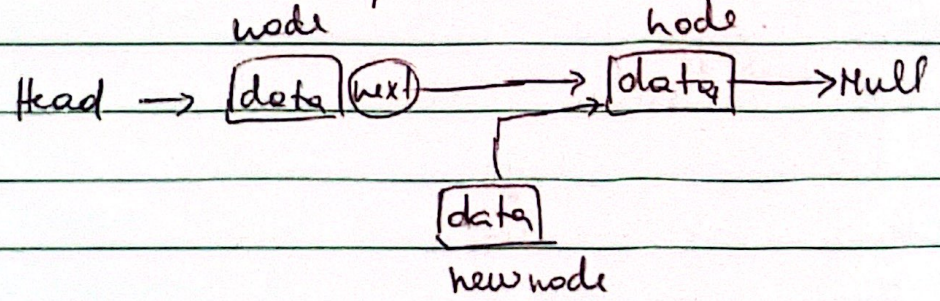
(iii) Displaying elements from the list.

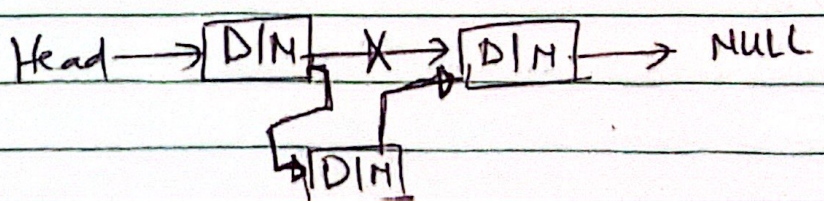
Linked List Representation.



Basic operations

(i). Insertion operation





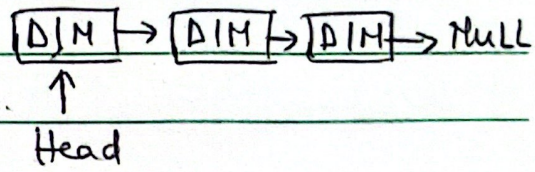
1st Allocate the newNode
2nd NewNode.next → RightNode
3rd LeftNode.next → NewNode
4th Disconnecting leftNode to RightNode

Exercise . Diagramatically show steps to add; using the linked list provided, illustrate all the steps necessary to:

a) add new node at the start

b) Insert new node in the middle

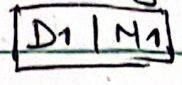
c) Remove the node located in the middle



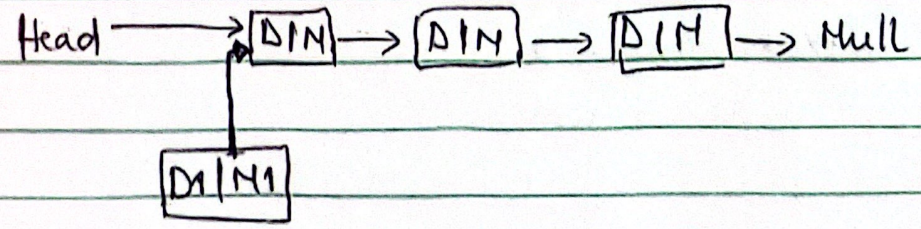
Soln

a) Adding new node at the start.

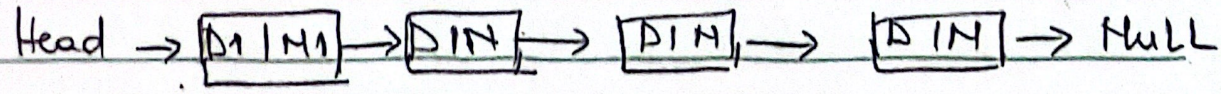
1st step. Allocating new node:



2nd step NewNode.Next → Right Node



3rd step. ~~left~~ Head.next → NewNode



b) Inserting new node in the middle of list

Head → [D1|N1] → [D|N] → [D|N] → [D|N] → Null

Head → [D|N] → [D|N] → [D|N] → Null

1st. Allocate new node [D2|N2]

2nd step. NewNode.next → RightNode.

3rd step. LeftNode.next → newNode;

Head → [D|N] → [D|N] → [D2|N2] → [D|N] → NULL

c) Remove node located in the middle.

node1 node2 node3

Head → [D|N] → [D|N] → [D|N] → NULL

1st step: node1.next → node3

node1 node2 node3

Head → [D|N] → [D|N] → [D|N] → Null

2nd step: delete node 2.

node1 node3

Head → [D|N] → [D|N] → Null

Answer:

a)

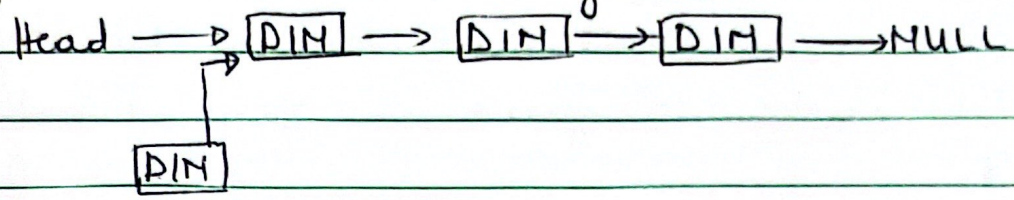
1st step. Allocate the new node

Head → [D|N] → [D|N] → [D|N] → Null

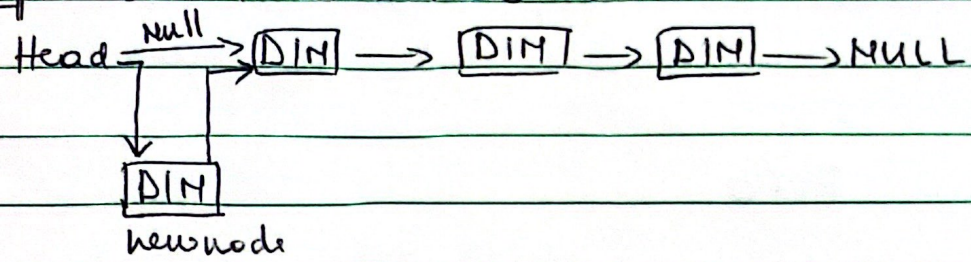
[D|N]

new node

2nd step: newNode → next = RightNode



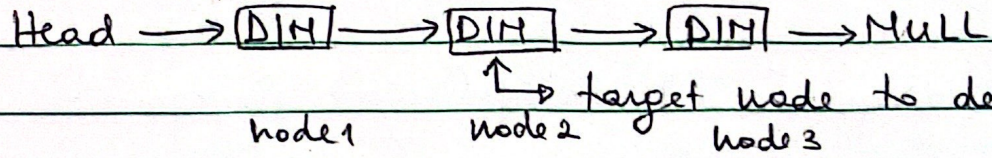
3rd Step: set Head = NewNode.



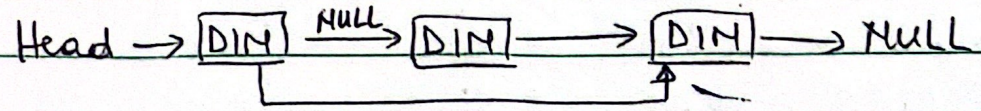
4th step: Therefore:

c) Deleting node

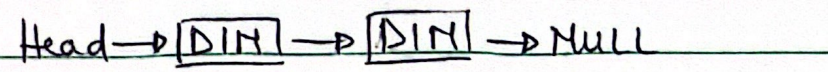
1st: Identify node to delete.



2nd: node 1 → next = node 3



3rd: delete node 2.



4th: End...

(ii). Deletion operation

leftNode → next = rightNode

~~left~~ targetNode → previous = NULL

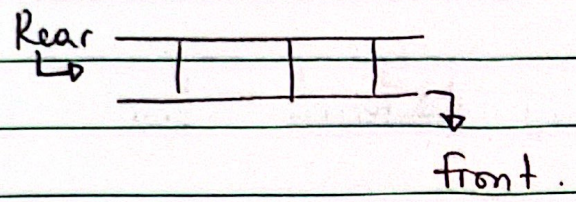
(iii). Reverse operation

Head to end ~~at~~ fail to start

Chap 4. Queue

- abstract DS somewhat similar to stacks
- Unlike stacks, a queue is open at both its ends.
  One ends is always used to insert (enqueue)
  Other used to remove data (dequeue).

- Follows FIFO



Basic operations

↳ may involve initializing or defining the queue, utilizing it then completely erasing it from memory.

↳ { enqueue () - add (store) in queue
  { dequeue () - remove (access) an item from queue

✓ peek () - get elt at front without removing it.
✓ isfull () - checks if the queue is full
✓ isempty () - checks if the queue is empty

(i). Peek () :

```
BEGIN PROCEDURE PEEK
    return queue [front]
END PROCEDURE
```

(ii). isfull ()

```
BEGIN PROCEDURE ISFULL
    if rear equals to MAXSIZE
        return true.
```

```
        else
            return False
        end if
    end procedure.
```

Ex. bool isfull () {

```
        if (rear == MAXSIZE - 1)
            return TRUE
        else
            return FALSE }
```

iii) IsEmpty ()

```
    Begin Procedure isempty
        IF front is less than MIN OR front < rear
            return true
        else
            return False
        end if
    end procedure.
```

ex. bool isempty () {

```
        if ( front <0 || front < rear)
            return TRUE
        Else
            return False }
```

Q. Enqueue Op. (adding on queue)

(i). check if the queue if is full

(ii). If the queue..

Algorithm:

```
    Procedure queue (data):
        If queue is full
            return overflow
        end if
```

```
    rear = rear + 1.
    queue [rear] ← data
    return True.
end procedure
```

ex:

```
int queue (int data)
    if (isfull ())
        return 0;
    rear = rear + 1;
    queue [rear] = data;
    return 1;
end procedure.
```

(b). **Dequeue op**

→ remove element at the front.

Algorithm

```
procedure dequeue
    if dequeue is empty
        return underflow
    endif
    data = queue [front]
    front = front + 1
    return true
end procedure
```

Exercise

Consider an empty queue with max size of 5. Perform the following sequence of operations on the queue and display the queue after each operation:

First, ~~eq~~ enqueue 101, then ~~eq~~ enqueue 102 and ~~eq~~ enqueue 103.

After that, dequeue an element.

Next, enqueue 104, and perform peek operation to check the front element.

Following this, dequeue an element, then enqueue 105 and enqueue 106.

Finally, perform another dequeue operation. Provide the state of the queue after each operation.

After performing all of above operations, Explain what happens when you attempt to enqueue 107 to the queue.

Soln x

BEGIN

op 1 (step 1)

		103	102	101

enqueue (101, 102, 103)

step 2.

			103	102

		104	103	102

peek () = return 102

Step 3.

106	105	104	103	102

enqueue (105, 106)

Step 4. enqueuing 107 will return overflow error since the queue is full.

Another question

Sat 15/02/2025

Chap 5. Stack

A stack is an abstract data type (ADT), commonly used in most programm languages.

- It behaves like a real world stack
  - ex deck of cards or pile of plates and so on.
- Uses LIFO (Last In First Out)

Operations

- push () = Add
- pop () = remove
- isempty () = check emptiness
- is full() = check if full
- peek () = return top element

stack representation

based on LIFO principle

Algorithm :

1. peek()

```
begin procedure peek
        return stack [top]
end procedure
```

ex:
```
int peek() {
        return stack[top];
}
```

2. isempty()

```
begin procedure isempty
        if top < 1
                return true,
        else
                return false
```

```
    end If
  end procedure.
```

ex:

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

For C programming

③. push ()

```
begin procedure push: stack, data
    if stack is full
        return null
    endif
    top ← top + 1
    stack [top] + data
end procedure.
```

④. pop ()

```
begin procedure pop: stack
    if stack is empty
        return null.
    endif
    data ← stack [top]
    top ← top - 1
    return data
end procedure
```

⑤. isfull ()

```
begin procedure isfull
    if top is MAXSize - 1
        return true
    else
```

Chap 6 Sorting

a) bubble sort
b) Merging
c) selection
d). Insertion
e) Quick sort

①. Bubble sort

↳ compares two adj elt and swaps them until they are ~~[illegible]~~ in intended order.

↳ algorithm

```
begin
    for k = 1 to (n - 1) by 1 do //nb of comparison
        for j = 0 to (n-k-1) by 1 do
            if (a[j] > a[j+1]) then
                set temp = a[j]
                set a[j] = a[j+1]
                set a[j+1] = temp
            endif
        end for
    end for
end
```

array dimension

Ex Apply the algorithm

$a = [5, 3, 8, 1]$

$k = 1$ to 3

$k=1$ $\left[\begin{array}{l} J = 0 \text{ to } 2 \\ 0 : (A[0] > A[1])? \\ 1 : (A[1] > A[2])? \\ 2 : (A[2] > A[3])? \end{array}\right.$

$k=2$ $\left[\begin{array}{l} J = 0 \text{ to } 1 \\ 0 : (A[0] > A[1])? \\ 1 : (A[1] > A[2])? \end{array}\right.$

$k=3$ $\left\{\begin{array}{l} J = 0 \text{ to } 0 \\ 0 : (A[0] > A[1])? \end{array}\right.$

Exercise

consider : {10, 9, 8, 7, 6, 5, 4, 3, 2, 1}

Sort the array element using bubble sort algorithm. (Asc order)

Soln : $n = 10$ thus $k = n - 1$ $(k = 9)$

For $k = 1$ to 9

$k = 1$ (1st pass)

J = 0 to 8

J

0 (9, 10, 8, 7, 6, 5, 4, 3, 2, 1) swapp

1 (9, 8, 10, 7, 6, 5, 4, 3, 2, 1) swapp

2 (9, 8, 7, 10, 6, 5, 4, 3, 2, 1) swapp

3 (9, 8, 7, 6, 10, 5, 4, 3, 2, 1) swapp

4 (9, 8, 7, 6, 5, 10, 4, 3, 2, 1) swapp

5 (9, 8, 7, 6, 5, 4, 10, 3, 2, 1) swapp

6 (9, 8, 7, 6, 5, 4, 3, 10, 2, 1) swapp

7 (9, 8, 7, 6, 5, 4, 3, 2, 10, 1) swapp

8 (9, 8, 7, 6, 5, 4, 3, 2, 1, 10) swapp

$k = 2$ (2nd pass)

J = 0 to 7

⇒ 7 8

↳ Bubble sort complexity

Time complexity

Best	$O(n)$	already sorted
worst	$O(n^2)$	already desc to make asc
Average	$O(n^2)$	mixed.

Space complexity $O(1)$ → only one extra temp memory required

↳ Applications

1. Complexity doesn't matter (Higher complexity negotiable or neglected)
2. Short and simple code is better

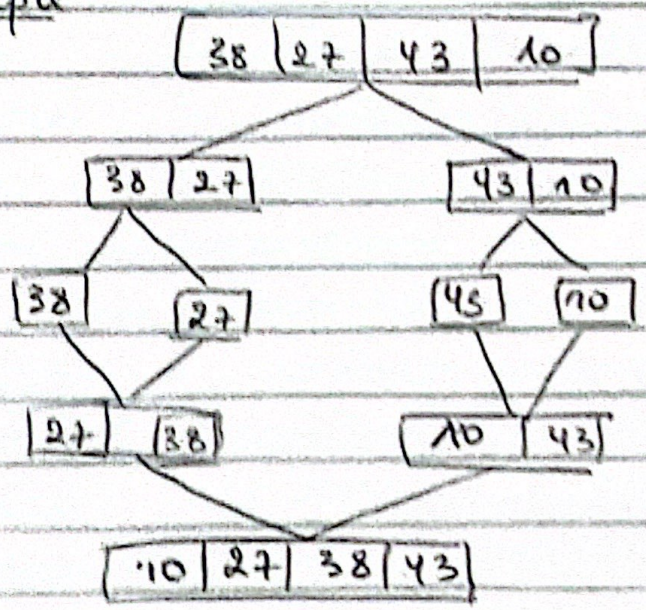
Example:

```
void bubblesort (int arr[], INT size[]).
{ for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp; }
}}}
```

② . Merging Sort

→ follows divide and conquer.

→ example



MergeSort (A)

(i) If size of A ≤ 1, return A

(ii). Else

(a). Find the middle index: mid = length(A) / 2

(b). Divide A into 2 halves

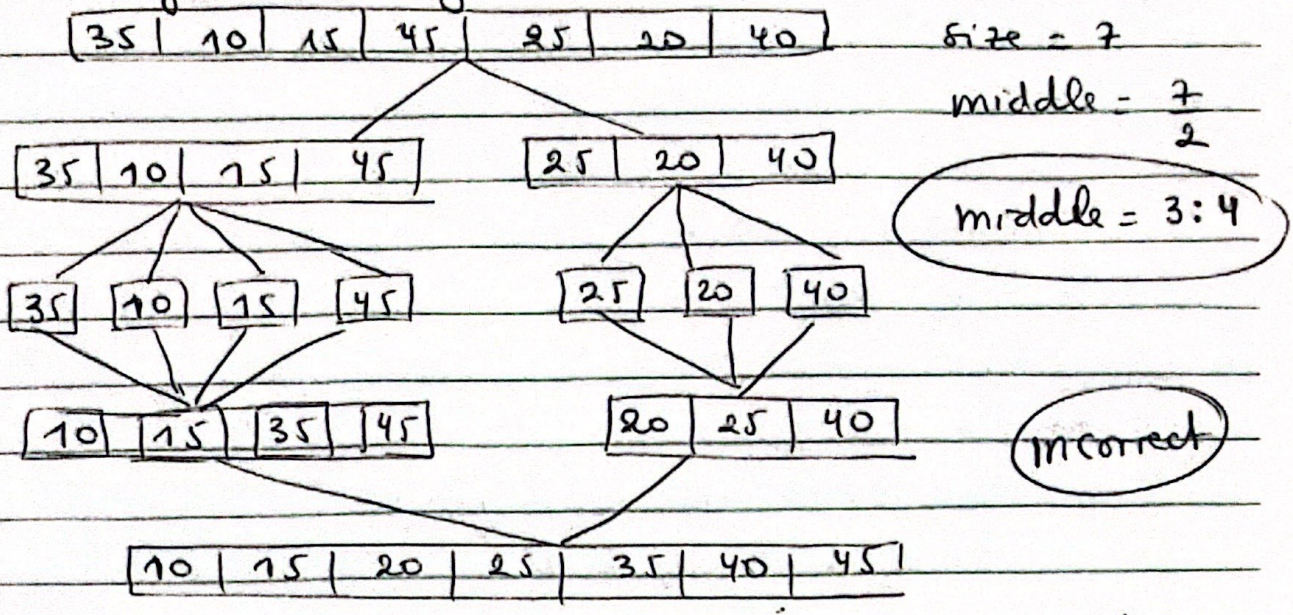
left side and right side

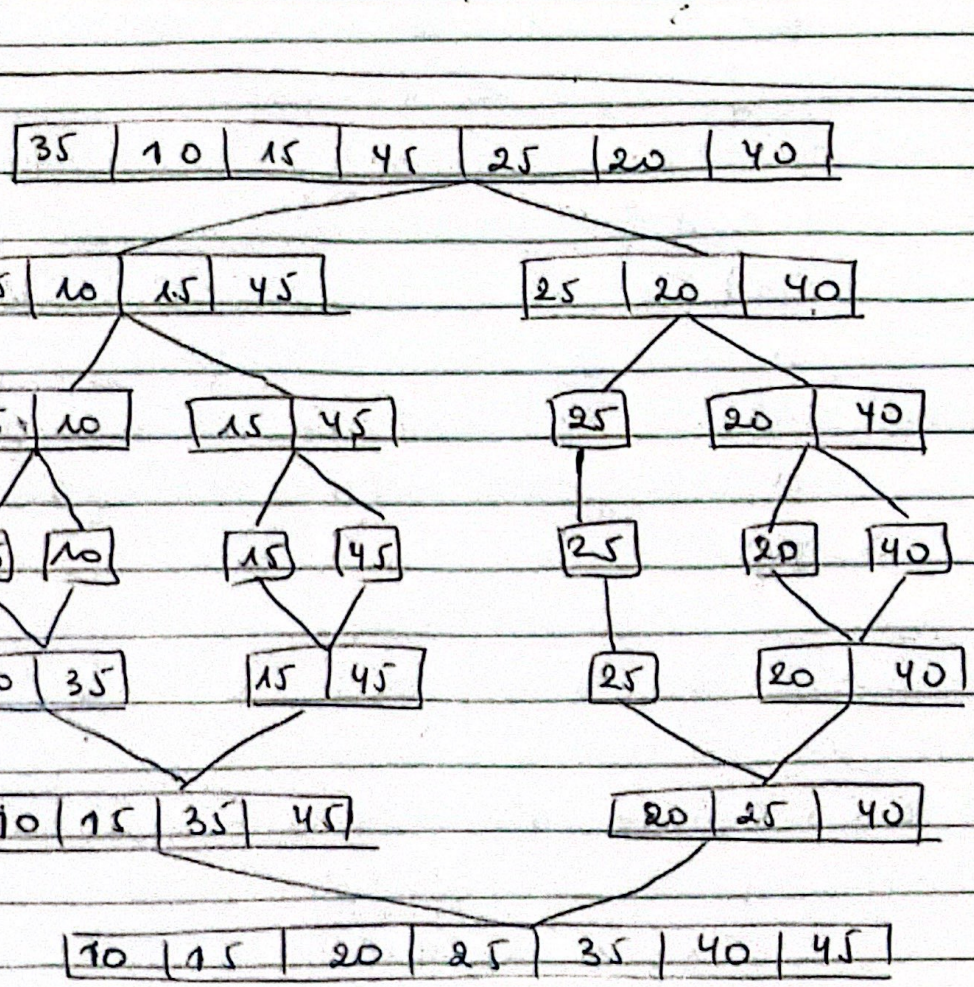
compare element by another

until you sort it.

Exercise

Show merge sort algorithm for:





↳ **Complexity**:

1. **Time**
   - best $O(n \log n)$
   - worst: $O(n \log n)$
   - Average: $O(n \log n)$

② Space : O(n) since each element will require extra space.

⚠ Time complexity is lesser compared to bubble sort

③. Selection sort

→ algorithm:

```
Begin
    For i = 0 to n-1 do:
        Small sub = i
        For j = i+1 to n-1 do:
            if A[j] < A[small sub]
                small sub = j
            endif
        end for
        temp = A[i]
        A[i] = A[small sub]
        A[small sub] = Temp.
    end for
end.
```

Apply on: Arr [7, 1, 4, 2, 3]

Soln: n = 5

①. For i = 0 to 4

1st pass: i = 0, small sub = 0

j = 1 to 4

1: (A[1] < A[0]) ! small sub = 1

2: (A[2] < A[1]) : small sub = 2

3: (A[3] < A[2]) : small sub = 3

4: (A[4] < A[3]) : small sub = 4

No Review

temp = A[i] ⇒ A[0] = 7

A[0] = A[4] ⇒ A[0] = 3

A[4] = Temp ⇒ A[4] = 7

Review

before [7, 1, 4, 2, 3]

swapp

result of 1st swap : [3, 1, 4, 2, 7]

2nd pass ~~swapp~~ : i = 1, smallsub = 1

j = 2 to 4

Review

2 : (A[2] < A[1]) ⇒ smallsub = 2

3 : (A[3] < A[2]) ⇒ smallsub = 3

4 :

⚠ Selecting the smallest element and compare to other elements...

Exercise

Consider N = 10, arr[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1}

Sort array elm using selection sort :

Soln

For i = 0 to 9 , smallsub = 0

{10, 9, 8, 7, 6, 5, 4, 3, 2, 1}

swap

{1, 9, 8, 7, 6, 5, 4, 3, 2, 10}

⊗ i = 1 , small sub = 1

{1, 9, 8, 7, 6, 5, 4, 3, 2, 10}

swap.

{1, 2, 8, 7, 6, 5, 4, 3, 9, 10}

⊗. i = 2, small sub = 2.

{1, 2, 8, 7, 6, 5, 4, 3, 9, 10}

swapp

Arr [10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1]

For i = 0 to 9,

j = i + 1 to 9

i = 1, A[i] = 10

[10] [9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1]

A[small sub] = 1

A[i] > A[small sub], True

→ swap

[1] [9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 10]

⊛. A[i] = 9 (i = 1)

[1 | 9] [8 | 7 | 6 | 5 | 4 | 3 | 2 | 10]

A[small sub] = 2

A[i] > A[small sub], True (swap)

[1 | 2] [8 | 7 | 6 | 5 | 4 | 3 | 9 | 10]

⊛. A[i] = 8 (i = 2)

[1 | 2 | 8] [7 | 6 | 5 | 4 | 3 | 9 | 10]

A[small sub] = 3

A[i] > A[small sub], True (swap)

[1 | 2 | 3] [7 | 6 | 5 | 4 | 8 | 9 | 10]

⊛. A[i] = 7 (i = 3)

[1 | 2 | 3 | 7] [6 | 5 | 4 | 8 | 9 | 10]

A[small sub] = 4

A[i] > A[small sub], True (swap)

[1 | 2 | 3 | 4] [6 | 5 | 7 | 8 | 9 | 10]

⊛. A[i] = 6 (i = 4)

[1 | 2 | 3 | 4 | 6] [5 | 7 | 8 | 9 | 10] A[small sub] = 5

A[i] > A[small sub], True (swap)

1	2	3	4	5	6	7	8	9	10

↳ Complexity

① Time : best : $O(n^2)$
worst : $O(n^2)$
Average : $O(n^2)$

② Space : $O(1)$ — temp memory for (only one) swapp.

④. Insertion Sort

```
Begin
  For i = 1 to n-1
      j = i
      Do while (j > 0) and (A[j] < A[j-1])
          temp = A[j]
          A[j] = A[j-1]
          A[j-1] = temp
          j = j - 1 // (j--)
      End do
  End for
End
```

If

Comparing to the previous and swap

```
Begin
  For i = 1 to n-1
      For j = i , j > 0 , j--
          if A[j] < A[j-1]
              temp = A[j]
              A[j] = A[j-1]
              A[j-1] = temp
          end if
      end for
```

↳ Complexity

→ same as bubble sort

$j = 1$

$A[j] > A[j-1]$, true

swapp

Exercise

Use insertion sort to sort:

10	15	25	30	35	20	45	40

10	15	25	30	35	20	45	40

10	15		25	30	35	20	45	40

10	15	25		30	35	20	45	40

10	15	25	30		35	20	45	40

10	15	25	30	35		20	45	40

10	15	25	30	20	35	45	40

Exercise:

5. Quick sort Illustration

a. choose the pivot and compare placing small nb to left to the right.

7	-2	4	1	6	5	0	-4	2

pivot

Either last, middle or start pivot element.

Sunday 16/02/2025.

Exercises (individual assignment)

Consider Numbers = [19, 14, 37, 13, 105, 42, 7, 55, 22]

i. Provide steps to sort this using

a) Bubble sort

b) Merge sort

c) Selection sort
d) Quick sort
e) Insertion sort

(ii). compare algorithm complexity of the above type of sorting.

Soln:

(a). Bubble Sort

$n = 9$

For $k = 1$ to 8

1st pass:

[29, 14, 37, 13, 105, 42, 7, 95, 22]

29 > 14, swap

[14, 29, 37, 13, 105, 42, 7, 95, 22]

29 > 37, False

Chap 7. Search using complex data structure

Def.

Searching - process of finding the locat° of given element in set of elements.

2 types

1. linear search
2. Non-Linear / Binary search

(A). Linear search

also known as sequential search

→ goes element by element

→ algorithm:

* linear search (a, n, item, loc)

arr, size of arr, to search, return location

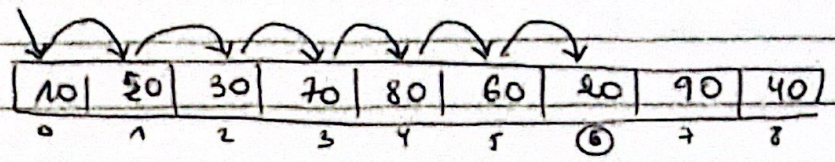
Begin

for i = 0, to (n-1) by 1 do

if (a[i] == item) then

```
        Set loc = i
        print loc
      end if
    end for
  end
```

Ex Find 20 :



Found at index 6
Location ↑

Complexity

Time : O(n)
↳ iterates through whole list of array.

best case : O(1) - when found at first index
worst case : O(n) - when found at the end or not present.

②. Binary search

↦ divide the search interval in half while dividing you begin with interval covering the whole array. (compare the item with middle)
↦ applied to the sorted array.
↦ algorithm

```
Procedure Binary search (arr , n, item) returns integer
    beg := 0          //initialize variables
    end := n-1
    mid := 0
    loc := -1
```

```
//Binary search loop
while (beg <= end) and (arr[mid] != item) do
        mid := (beg + end) / 2
    if item < arr[mid] then
            end := mid - 1   // left half
    else
            beg := mid + 1   // right half.
end while
//check if item is found
if beg > end then
        loc := -1   // item not found
else
        loc := mid   // Item found at index loc
end if
return loc
end procedure
```

<u>Example</u> Find 20

10	50	30	70	80	60	20	90	40

<u>Soln</u>:

Sorted:

10	20	30	40	50	60	70	80	90

beg = 0 , end = 8 , mid = (0+8)/2 -> mid = 4

if item < arr[mid] (arr[4])

20 < 50, then.

Consider lefthalf:

end = 4 - 1

end = 3

⋮

Example 2 use binary search to search 23

2	5	8	12	16	23	38	56	72	91

Soln : beg = 0 , end = $(\overset{9}{n-1})$, mid = (0 + 9)/2 = 4.5

1st

mid = 4

if item < arr[mid] left half

else item > arr[mid] right half

23 > arr[4]

23 > 16 → right half

2nd

~~Set beg = 0, end~~

Set beg = (mid + 1)

beg = 5 , end = 9 , mid = $(\overset{14}{5+9})/2$ = 7

item < arr[mid]

23 < arr[7]

23 < 56 , (True) left half.

3rd

Set beg = 5, ~~mid~~ end = mid - 1 , mid = $(\overset{11}{5+6})/2$ = 5.5

end = 6 mid = 5

item < arr[mid]

(23 < 23) False

arr[mid] = item

⇒ Item 23 is found at Index 5

Complexity

↳ Time

↳ space

Chap 8 Apply algorithm to solve complex mathematical functions

① Intro Quadratic eqn :

algorithm:

```
Input : coefficients a, b and c (real nb)
        delta = b^2 - 4ac
        if delta > 0 :
            root 1 = (-b + sqrt(delta)) / 2a)
            root 2 = (-b - sqrt(delta)) / 2a)
        if delta = 0 :
            root = -b / 2a
        if delta < 0 :
            realPart = -b / 2a
            imaginaryPart = sqrt(abs(delta)) / 2a
            root 1 = realPart + imaginaryPart * i
            root 2 = realPart - imaginaryPart * i
        Print root 1, root 2
    End
```

② Recursive function

→ calls itself during its execution.

Ex : ✓ summing list of nb
✓ Factorial function
✓ Fibonnacci series
✓
✓
✓

ⓘ. summing list of nb

List = [1, 3, 5, 7, 9]

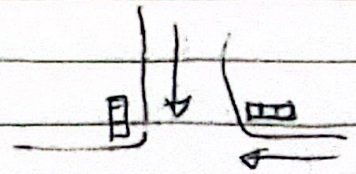
algorithm :

```
Def list sum (numList)
    the sum = 0
```

```
For i in Numlist:
    thesum = the sum + i
    return thesum
print (list sum ([1,3,5,7,9]))
```

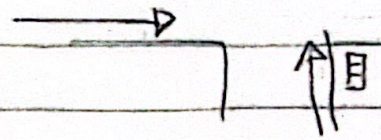
0 1 2 3 4 5

0 1 1 3 5



(ii). Factorial function

Pseudo code A



```
double factorial (int n) {
        double fact1;
        fact1 = 1;
    for (int i=1; i<=n; i++)
            fact1 = fact1 * i;
    return (fact1); }
```

or

Pseudo code B

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

(iii). Fibonnacci Series function

Pseudo code:

```
Begin Fibonnacci (n):
    Initialize an array fib of size n
    set fib[0] = 0, fib[1] = 1
    for for i from 2 to n-1 do:
            fib[i] = fib[i-1] + fib[i-2]
    return fib
End
```

Exercise : Apply Fibonacci series algorithm to find its 10 first terms.

n = 10

fib [10] = {0, 1, 1, 2, 3, 5, 8, 13, 21, 34}

fib[0] = 0

fib[1] = 1

For i = 2, to 9

i = 2, fib[2] = 0 + 1 = 1

i = 3, fib[3] = 1 + 1 = 2

i = 4, fib[4] = 2 + 1 = 3

i = 5, fib[5] = 3 + 2 = 5

i = 6, fib[6] = 5 + 3 = 8

i = 7, fib[7] = 8 + 5 = 13

i = 8, fib[8] = 13 + 8 = 21

i = 9, fib[9] = 21 + 13 = 34

End.

(iv). <u>Palindrome funct^{e}</u>

↳ word that is spelled the same from both ends.

<u>Ex</u>: anna, ifi, is, malayalam, 121 and so on

→ <u>algorithm to check</u>

```
Begin
    int n, num, ldig, rev = 0
    Print "Insert number"
    Read num
    n = num
    While (num != 0):
        ldig = num % 10
        rev = (rev * 10) + ldig
        num = num / 10
    if (n == rev)
        print "This is palindrome"
    else
        print "This is not palindrome"
End
```

$21 \underline{|10}$
1

Exercise : Use palindrome algorithm to check if 134567 is palindrome.

Soln : n = 0 rev = 0

num = 134567

ldig = 0

While num ≠ 0

dig = num % 10

dig = 134567 % 10 = 7

rev = (rev * 10) + 7

= (0 * 10) + 7

rev = 7

num = 134567 / 10 = 13456

$5 \underline{|3}$
$3 \quad 1$
2

$82 \underline{|10}$
$80 \quad 8$
2

$80 \underline{|10}$
$80 \quad 8$
0

① num ≠ 0 (13456 ≠ 0)

dig = 13456 % 10 = 6

rev = (rev * 10) + 6

= (7 × 10) + 6

rev = 76

num = 13456 / 10 = 1345

② num ≠ 0 (1345 ≠ 0)

dig = 1345 % 10 = 5

rev = (76 × 10) + 5

rev = 765

num = 1345 / 10 = 134

③. num ≠ 0 (134 ≠ 0)

dig = 134 % 10 = 4

rev = (765 * 10) + 4

= 7654

num = 134 / 10 = 13

④ num ≠ 0 (13 ≠ 0)

dig = 13 % 10 = 3

rev = (7654 × 10) + 3 ⇒ rev = 76543

76540

num = 13 / 10 = 1

⊛. num ≠ 0 (1 ≠ 0)

dig = 1 % 10 = 1

rev = (76543 * 10) + 1

= 765431

num = 1 / 10 = 0

⊛ num ≠ 0 (num ≠ 0)

* (rev = num)!

(765431 ≠ 134567)

Henceforth, 134567 is not palindrome.

To check string

Step 1. Input s (string)

Step 2. len = 0, Flag = 0

Step 3. while (s[len]! = NULL):

```
        len ++
```

Step 4. i = 0, j = len - 1

Step 5. while (i < (len/2) + 1):

```
        if (s[i] == s[j]):
            flag = 0
        else:
            flag = 1
        i++, j--
```

Step 6. if (flag == 0):

```
        print key is palindrome
    else:
        print key is not a palindrome
```

Step 7. End

Example Use palindrome algorithm to check "anna"

soln: s = "anna"

~~len = 4 (length (s))~~ ~~[len(s)]~~ len = 4

flag = 0

while s[len] ≠ NULL

len = 4 + 1 = 5

i = 0; j = 5 - 1 ⇒ j = 4

while (0 < (len/2) + 1)

while (0 < 3)

0 : (s[0] == s[4])! (s[i] == s[j])?

1 : (s[1] == s[3])!

2 : (s[2] == s[2])!

Henceforth, anna is palindrome.

(v) Euclid's GCD

↳ Greatest Common ~~Denominator~~ Divisor)

Begin

let a, b denote the numerator and denominator respectively. let 'gcd' denote for both a and b

if (a == b || b == 0)

return gcd = 1;

lcm = (a * b) / gcd

display gcd and lcm

Stop;

~~or~~

else { r = (a % b);

while (r < 0 || r > 0) {

a = b;

```
        b = r;
        r = (a % b);
    }
    gcd = b;
}

return gcd;
```

Exercise:

Use above algorithm to calculate the GCD and LCM of 2, 4 and 18.

soln: $a = 2$ $c = 18$

$b = 4$

$\boxed{\begin{array}{l} GCD = 2 \\ LCM = 18 \end{array}}$

⊗ $a \neq 0$, $b \neq 0$, $c \neq 0$

⊗ else: $r = (2 \% 4)$

$r = 2$

```
while (2 < 0 || 2 > 0)
        a = 4
        b = 2
        r = (4 % 2)
        r = 0
```

$gcd = 2$

$lcm = (2 \times 4) / 2$

$= 4$

Q2. Apply GCD algorithm to calculate The GCM (GCD) and LCM of $a = 56$ and $b = 15$.

Chap 9. DDA and Bresenham's Algorithms

Sat 01/03/25

- DDA - Digital Differential Analyzer.
  - methods for line drawing in computer graphics.

DDA - calculates intermediate points using slope.

Bresenham's - uses decision parameter

Ex: DDA Algorithm.

Step 1. Input 2 endpoints (x_1, y_1) and (x_2, y_2)

Step 2. calculate $dx = x_2 - x_1$ and $dy = y_2 - y_1$

Step 3. obtain nb of steps by using conditions:

if abs(dx) > abs(dy) then ~~steps~~
steps = abs(dx)

else
steps = abs(dy)

Step 4. ~~Decrement~~ Determine the increment values of x and y as

xinc = dx/steps and
yinc = dy/steps

Step 5. plot starting point (x, y)

Step 6. for k = 1 to steps in increments of 1

Step 7. plot next point by generating it as

x = x + xinc

y = y + yinc

Step 8. End for

Step 9. stop.

Example: Given points $P_1(2,3)$ and $P_2(7,5)$ calculate all ~~points~~ intermediate points using DDA and then plot it on cartesian plan.

Sol:

① Input $P_1(2,3)$ and $P_2(7,5)$

② $dx = X_2 - X_1 = 7 - 2 = 5$
$dy = Y_2 - Y_1 = 5 - 3 = 2$ } variations.

③ [abs (dx) > abs (dy)] True

Hence steps = abs(dx)

steps = 5

④ Increments:

$X_{inc} = dx/steps = 5/5 = 1$

$Y_{inc} = dy/steps = 2/5 = 0.4$

⑤ Plot starting point $P_1(2,3)$

⑥ For k = 1 to steps:

k = 1 to 5:

$X = X + X_{inc}$

$Y = Y + Y_{inc}$

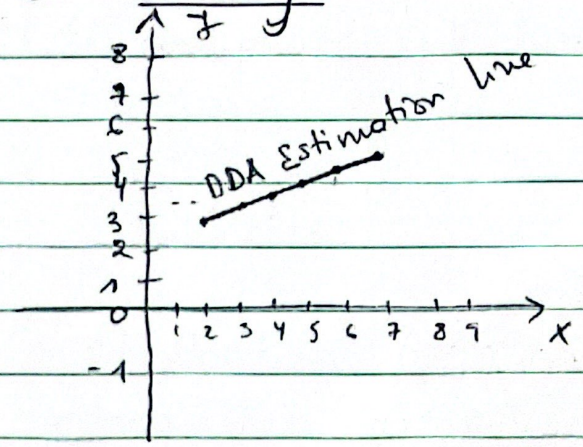
⑦

x	2	3	4	5	6	7
y	3	3.4	3.8	4.2	4.6	5

1 2 3 4

steps.

⑧ Plotting



Ex: Bresenham's Algorithm

step 1. Input 2 end points and store the left endpoints in (x_0, y_0)

step 2. Load (x_0, y_0) into frame and plot first point

step 3. calculate dx, dy, $2dy$ and $2dy - 2dx$ and obtain the starting value for decision parameter.

as $P_0 = 2dy - dx$ // decision parameter

step 4. At each x_k along the line starting at $k=0$ check the following test:

if $P_k < 0$, the next point to plot is $(x_k + 1, y_k)$ and

$$P_{k+1} = P_k + 2dy$$

otherwise, the next point to plot $(x_k + 1, y_k + 1)$ and $P_{k+1} = P_k + 2dy - 2dx$

step 5. Repeat step 4 dx times

Ex:

1. Input $P_1(2,3)$ and $P_2(7,5)$
2. Load $(x_0, y_0) \Rightarrow (2,3)$
3. calculate dx, dy, $2dy$ and $2dy - 2dx$

$dx = x_2 - x_1 = 5$

$dy = y_2 - y_1 = 2$

$2dy = 4$

$2dy - 2dx = 4 - 10 = -6$

$P_0 = 2dy - dx$

$= 4 - 5$

$P_0 = -1$

4. $P_0 < 0$ $(-1 < 0)$ True: $(x_k + 1, y_k)$

next point $(x_0 + 1, y_0) \Rightarrow (3,3)$

$P_k + 1 = P_k + 2dy$

$P_0 + 1 = P_0 + 2dy$

$\cancel{-1+1} = -1 + 4$

$P_1 = 3$

$P_1 > 0$ $(3 > 0)$ True: $x_k + 1$, y_k

Class work

Given endpoints: $(x_1, y_1) = (2,3)$

$(x_2, y_2) = (9,8)$

a) Apply the DDA to find the next point to draw a line from (2,3) and (9,8)

b) Apply Bresenham's algorithm to find next point to draw a line from (2,3) to (9,8)

Soln:

a) **DDA**

(i). Input $P_1(2,3)$ and $P_2(9,8)$

(ii). $dx = 9 - 2 = 7$
$dy = 8 - 3 = 5$ } variation.

(iii). [abs (dx) > abs (dy)]

(7 > 5) True; Then

steps = abs (dx)

steps = 7

(iv). **Increment**:

x-inc = dx / steps = 7/7 = 1

y-inc = dy / steps = 5/7 = 0.7142 ≃ 0.7

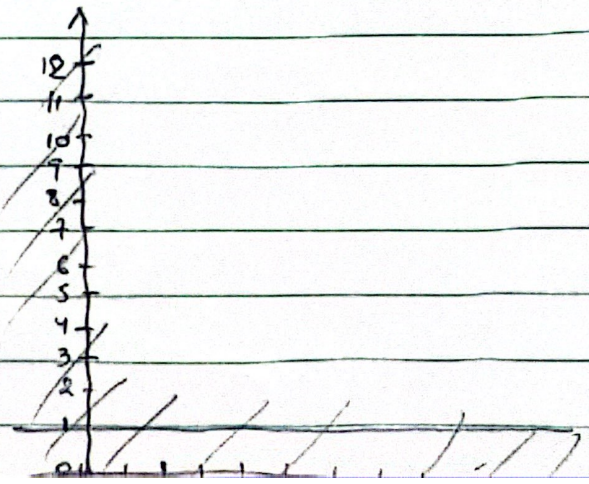
(v). For k = 1 To steps = 7:

x = x + x-inc

y = y + y-inc

x	2	3	4	5	6	7	8	9
y	3	3.7	4.4	5.1	5.8	6.5	7.2	8

(vi). Plot and End.



(b). Bresenham's Algorithm

(1). Input $P_1(2,3)$, $P_2(9,8)$

(2). Load (x_0, y_0)

$(2,3)$

(3). Calculate dx, dy, $2dy$, $2dy - 2dx$

$P_0 = 2dy - dx$

$dx = 9 - 2 = 7$

$dy = 8 - 3 = 5$

$2dy - 2dx = 10 - 14 = -4$

$2dy = 10$

$P_0 = 2dy - dx = 10 - 7 = 3$

(4). For $k = 0$ to $dx = 7$: $(0, 1, 2, 3, 4, 5, 6)$

* $(P_k < 0)$ False

* $(P_k > 0) \Rightarrow k = 0$, $P_0 = 3$ $(P_0 > 0)$

point 1 $(3, 4)$

$P_1 = P_0 + 2dy - 2dx$

$= 3 + (-4)$

$P_1 = -1$

* $k = 1$, $(P_1 < 0)$ True $P_1 = -1$

point 2 $= (4, 4)$

$P_2 = P_1 + 2dy$

$= -1 + 10$

$P_2 = 9$

* $k = 2$, $(P_2 > 0)$.

point 3 $(5, 5)$

$P_3 = P_2 + 2dy - 2dx$

$= 9 + (-4)$

$P_3 = 5$

* $k = 3$, $(P_3 > 0)$

point 4 $(6, 6)$

$P_4 = P_3 + 2dy - 2dx$

$= 5 + (-4)$

$P_4 = 1$

* $k = 4$, $(P_k > 0)$

point 5 $(7, 7)$

$P_5 = P_4 + 2dy - 2dx$

$= 1 + (-4)$

$P_5 = -3$

* $k = 5$, $(P_k < 0)$ $P_5 = -3$

point 6 $(8, 4)$

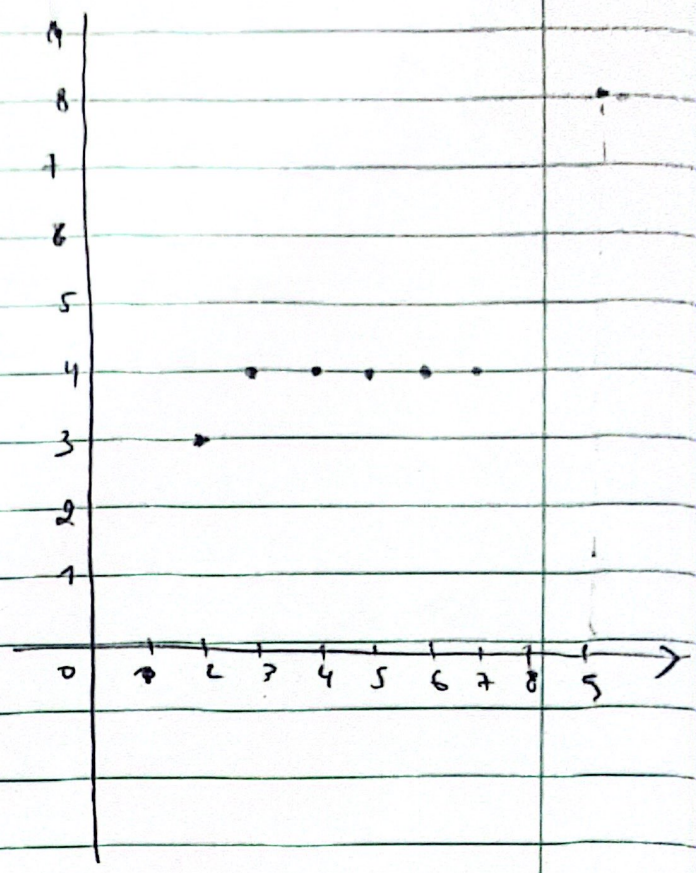
$P_6 = P_5 + 2dy$

$= -3 + 10$

$P_6 = 7$

* $k = 6$ $(P_k > 0)$

point 7 $(9, 8)$



⑤. Table of points

x	2	3	4	5	6	7	8	9
y	3	4	4	5	6	7	8	8

Banker's Algorithm

↳ algorithm used in OS to ensures that the system does not enter an unsafe state where processes may deadlock due to conflicting resource requests.

It's primarily used in systems where multiple processes are competing for a finite number of resources.

Summary of Brehensham's (Rewind)

① Input P_1, P_2

② load initial point (x_0, y_0)

③. Calculate: dx, dy, $2dy$, $2dy - 2dx$, $\overbrace{P_0 = 2dy - dx}^{\text{decision parameter}}$

4. for k = 0 to dx do.

if $P_k > 0$:

newpoint $(x_k + 1, y_k)$

$P_{k+1} = P_k + 2dy$

else:

newpoint $(x_k + 1, y_k + 1)$

$P_{k+1} = P_k + 2dy - 2dx$

5. End for

6. End.

↳ 4 types of data structures

DS used to implement Banker's are: (typ

- available
- Max
- Allocation
- Need (request) ⇒ need <= available
  ↳ Max - Allocation

1. Available

→ nb of available instances of each resource in the system at a given point of ~~type~~ time.

2. Max

3. Allocation

4. need.

Example

Considering a system of five processes P_0 thro' P_4 and three resources A, B, C.

Resource type A has 10 instances

B has 5 instances

C has 7 instances.

Suppose at time t_0 following snapshot of the system has been taken:

Q1. what will be the content of the Need matrix ?

⚠ Safe state of processor.

Process	Allocation	Max	Available	need
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2	2 0 0 (P_2)	1 2 2
P_2	3 0 2	9 0 2	2 1 1 (P_3)	6 0 0
P_3	2 1 1	2 2 2	0 0 2 (P_4)	0 1 1
P_4	0 0 2	4 3 3	0 1 0 (P_0)	4 3 1
			3 0 2 (P_3)	

Q2. Is the system in safe state ?
If yes, the what is the safe sequence ?
Apply ~~no~~ the safety algorithm on the given system.

Sol:

Safe sequence :

* $i=0$, P_0, need = [7, 4, 3], available = [3, 3, 2]
  need > available
  P_0 wait. (must)
* $i=1$, P_1, need [1, 2, 2], available = [3, 3, 2]
  need < available
  P_1 can be kept in a safe sequence,
  Update work
  Work = Available + allocation
  = [3, 3, 2] + [2, 0, 0]
  Work = [5, 3, 2]

⊛ $i = 2$, P_2 need $[6, 0, 0]$, available $[5, 3, 2]$

need > available

P_2 must wait.

⊛. $i = 3$, P_3, need $[0, 1, 1]$, available $[5, 3, 2]$

need < available, update work

work = ~~Available~~ + allocation

work =

work $= [5, 3, 2] + [2, 1, 1]$

work $= [7, 4, 3]$

⊛. $i = 4$, P_4, need $[4, 3, 1]$, available $[7, 4, 3]$

need < available, update work

work = + allocation

$= [7, 4, 3] + [0, 0, 2]$

$= [7, 4, 5]$

⊛ $i = 0$, P_1, need $[7, 4, 3]$, available $[7, 4, 5]$

need < available, update

work = + allocation

$= [7, 4, 5] + [0, 1, 0]$

work $= [7, 5, 5]$

⊛. $i = 2$, P_2, need $[6, 0, 0]$ available $[7, 5, 5]$

need < available, update

work = + allocation

$= [7, 5, 5] + [3, 0, 2]$

work $= [10, 5, 7]$

⇒ Safe sequence $= P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$

Max $= [10, 5, 7]$

<u>Exercice</u>:

A system with 5 processors use data structures for allocated, maximum and available resources

as shown in the table. These structures monitor current allocat°, max resource needs and available resources to manage process operations efficiently.

Process	Allocated				max				Available			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1	0	0	1	2	0	0	1	2	2	1	0	0
P_2	2	0	0	0	2	1	1	1				
P_3	0	0	3	0	6	6	5	4				
P_4	2	3	5	4	4	3	5	6				
P_5	0	3	3	2	0	6	5	2				

a) Compute the need matrix D.S

b) Determine all possible safe sequence

<u>Soln</u> :

a)

Process	need			
	R_1	R_2	R_3	R_4
P_1	0	0	0	0
P_2	0	1	1	1
P_3	6	6	2	4
P_4	2	0	0	2
P_5	0	3	2	0

need = max − allocat°

(b). <u>safe sequence</u>

(*). $i = 1$, P_1, need $[0,0,0,0]$, available $[2,1,0,0]$

need < available,

P_1 executed / processed

Update available = available + allocation

$= [2,1,0,0] + [0,0,1,2]$

$= [2,1,1,2]$

(*). $i = 2$, P_2, need $[0,1,1,1]$, available $[2,1,1,2]$

need < available

P_2 processed

update available = + allocation

$= [2, 1, 1, 2] + [2, 0, 0, 0]$

$= [4, 1, 1, 2]$

⊗. $i = 3$, P_3, need $[6, 6, 2, 4]$, available $[4, 1, 1, 2]$

need > available

P_3 must wait (enqueue)

⊗. $i = 4$, P_4, need $[2, 0, 0, 2]$, available $[4, 1, 1, 2]$

need < available

P_4 processed / executed

new available = + allocat^{n}

$= [4, 1, 1, 2] + [2, 3, 5, 4]$

$= [6, 4, 6, 6]$

⊗. $i = 5$, P_5, need $[0, 3, 2, 0]$, available

need < available

P_5 executed

update available = + allocation

$= [6, 4, 6, 6] + [0, 3, 3, 2]$

$= [6, 7, 9, 8]$

⊗ $i = 3$, P_3, need $[6, 6, 2, 4]$, available

need < available

P_3 executed / processed

Update available = + allocat^{n}

$= [6, 7, 9, 8] + [0, 0, 3, 0]$

$= [6, 7, 12, 8]$

$\Rightarrow$ safe sequence $= P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_3$

Chap 10. Tree

→ data collection which is organized in hierarchical structure and thus is a recursive definition.

→ In tree DS, if we have N nb of nodes then we can have max N-1 of links

Ex :

↳ Terminology

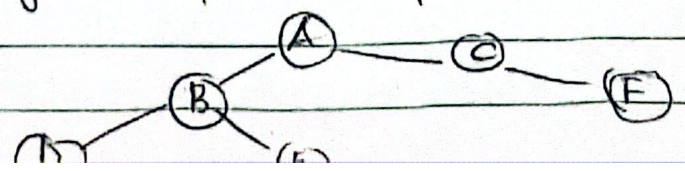
- ✓ root (top node)
- ✓ edge
- ✓ parent
- ✓ siblings (node with same parent.
- ✓ leaf (node with no child)
- ✓ Internal node (at least one child)
- ✓ degree (nb of children)
- ✓ level
- ✓ Height (total nb of edges from leaf node to particular node)
- ✓ depth (total nb of edges from root node to particular node)
- ✓ sub-tree (tree within another tree)

Binary Tree representations :

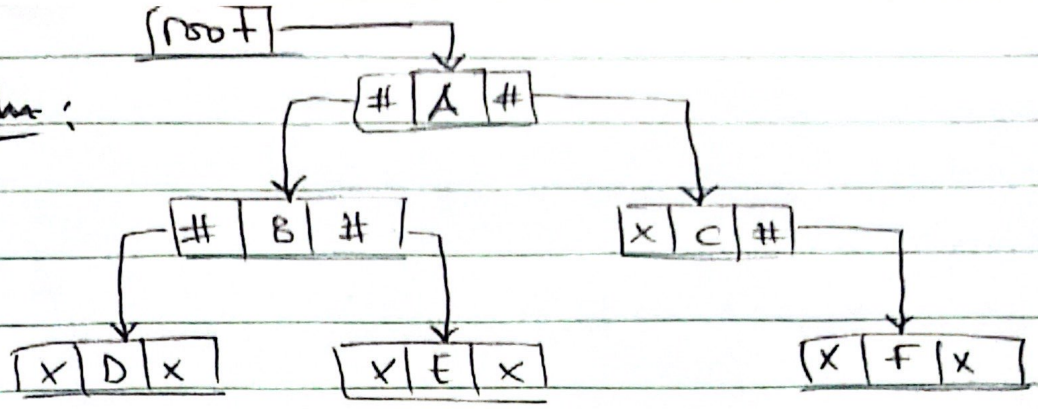
- ✓ Linked list
- ✓ array representation.

a) Linked list of Binary Tree

Example consider binary tree T in the figure of Binary tree.



Soln :

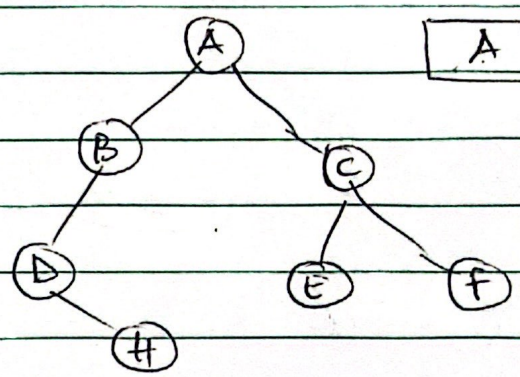


(a). Array representation :

A	B	C	D	E	-	F

(b). Array representation

ex :



A	B	C	D	-	E	F	-	H		

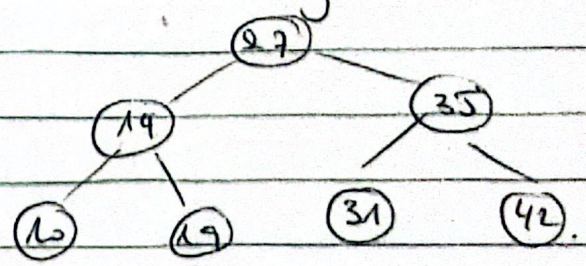
Tree traversals

- ✓ In-order (left - root - right)
- ✓ pre-order (Root - left - right)
- ✓ post order (left - right - root)
- ✓ BFS (by level

Exercises

Traverse the following tree using :

a) In-order
b) pre-order
c) post-order
d) BFS



Soln :

a) In-order : 10 → 14 → 19 → 27 → 31 → 35 → 42

b) pre-order

27 → 14 → 10 → 19 → 31 → 38 → 42.

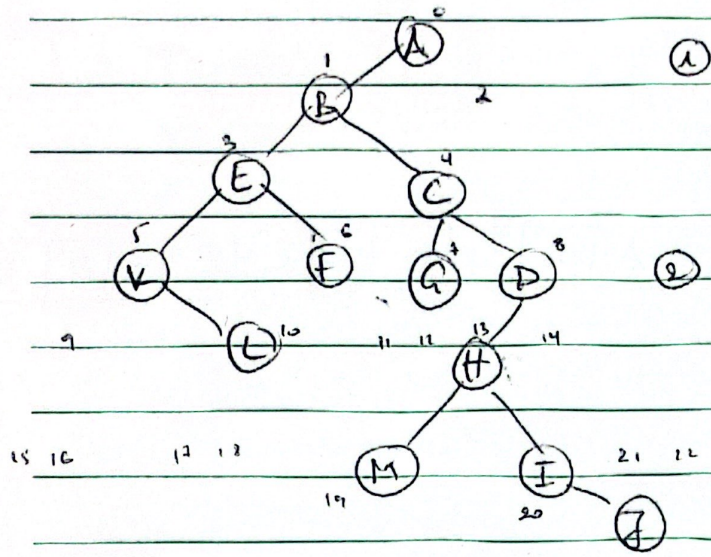
c) Post-order

10 → 19 → 14 → 31 → 42 → 35 → 27

d) BFS :

27 → 14 → 35 → 10 → 19 → 31 → 42

Exercize 1



① . Represent the tree using:

ⓐ linked list of binary tree

ⓑ array representation

② . traverse the tree using:

ⓐ. In-order

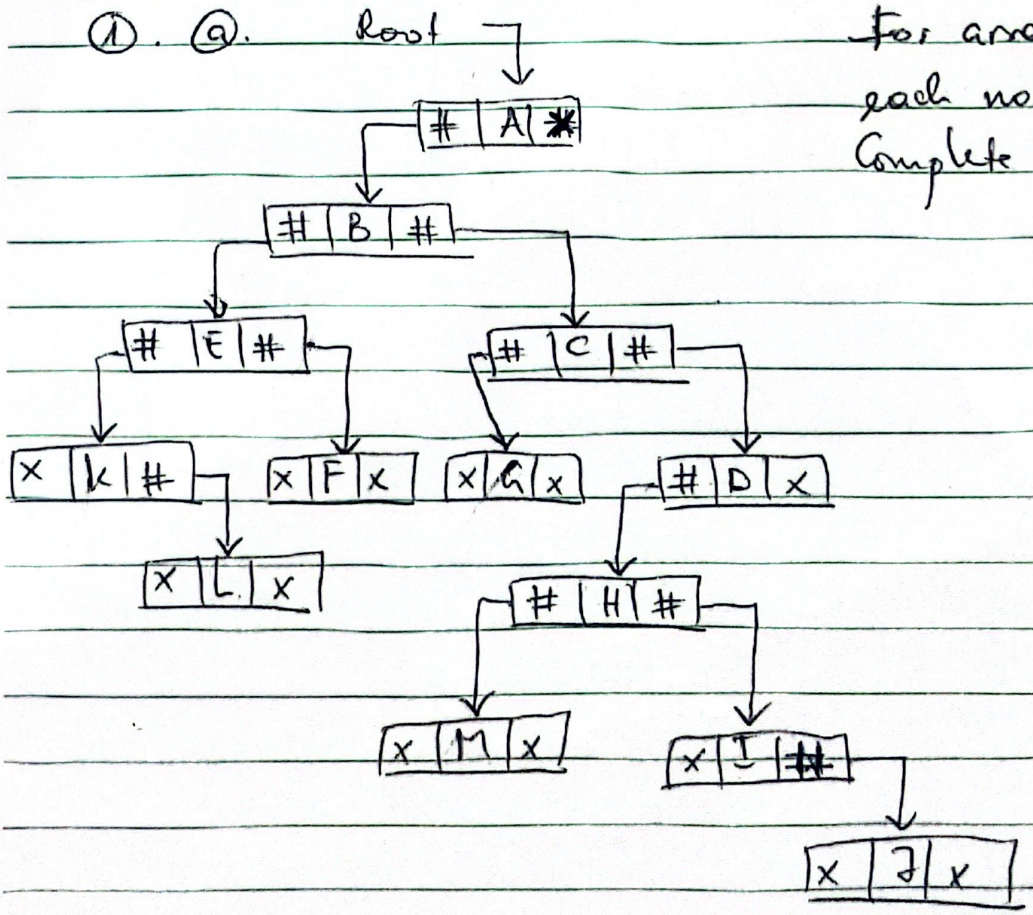
ⓑ. pre-order

ⓒ. post-order

ⓓ. BFS.

Soln :

① . ⓐ.



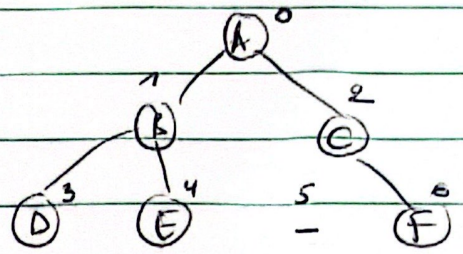
for array each node should have Complete tree (sub tree)

b) Array:

A	B	-	E	C	k	F	G	D	→	L	-	-	-	-	H	-	-	-	M	I	-	-	-	J	-	-

② a) In-order:

L → k → F → E →



A	B	C	D	E	-	F

②. a) In-order: (left - root - right)

~~k → L → E → F → B → I → J → M → H → D → G → C~~

~~k → L → E → F → B → I → J → M → H → G → C → D → A~~

k → L → E → F → B → G → C → M → H → I → J → D → A

b) pre-order (root - left - right)

A → B → E → k → L → F → C → G → D → H → M → I → J

c) post-order (left - right - root)

L → k → F → E → G → M → J → I → H → D → C → B → A

d) BFS

A → B → E → C → k → F → G → D → L → H → M → H → J