Algorithem and Data Structure.

2022.10.05.

contest of computing.

Input → Data process → output

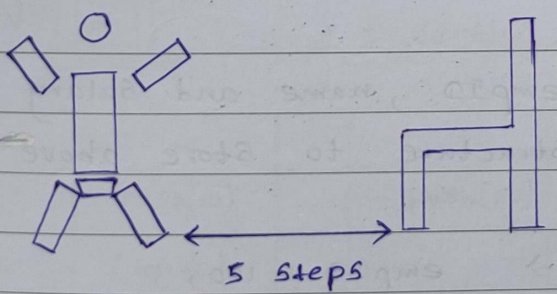
Comp computer program

Instructions + Data → Data Structure / variable arrays.

Algorithms

- Data Structures,
  - * It's a way of storing data or it's a memory arrangement.
- Algorithm,
  - * set of instructions in perticular order is an algorithm.

ex:- give the instructions to 'otto' to sit on the chair.



1) Turn left
2) walk
3) walk
4) walk
5) walk
6) walk
6) Turn Right
8) Turn Right
9) sit

Structures in 'c'

Instruction order → time and space.
→ we take 'time' as the main.

* Structures in C // This is an user defined Data type.
* Variable, array // This is a pre-defined Data type.

1) Variable

In a variable we can add only one data type.

ex:- `int age;` 25 age.

DT VN

2) Structure. (struct) ← keyword / structure tag.

```
Struct employee {
    int age;
    float salary;
};
```

- we can store variables under 'struct'.
- can add multiple components in different data types.
- 'struct' is a user define data type it's a complex data structure.
- we can store multiple variables under single data type (struct)

3) array

`char arr[3];`

T	O	M
0	1	2

4) Assume we need to store empID, name and salary on each employee. Declare a structure to store above

```
Struct emp {
    int empID;
    char name[20];
    float salary;
} e1, e2, ..;
```

↑ method 1

$e_1 \rightarrow$ empID = 100;

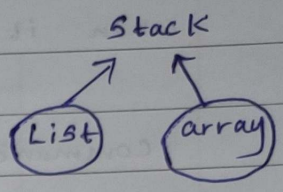
$e_2 \rightarrow$ empID = 200;

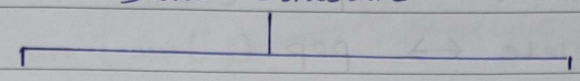
```
Struct employee {
    int empID;
    char name[50];
    float salary;
};
Struct employee emp1, emp2;
```

↑ method 2

Stack data Structure.

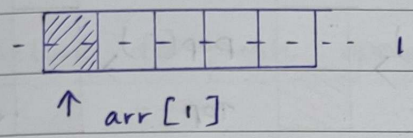
1) Stack (LiFo)





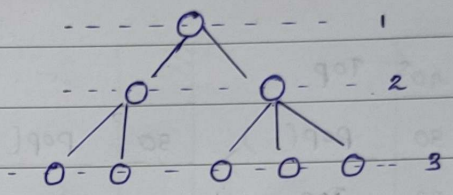
Linear

(Data allocate sequently | one after other, It's in the same level)



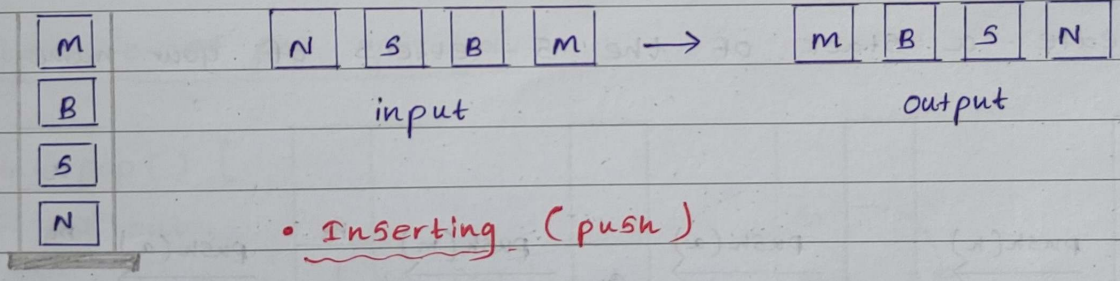
non- Linear

(have many level's)

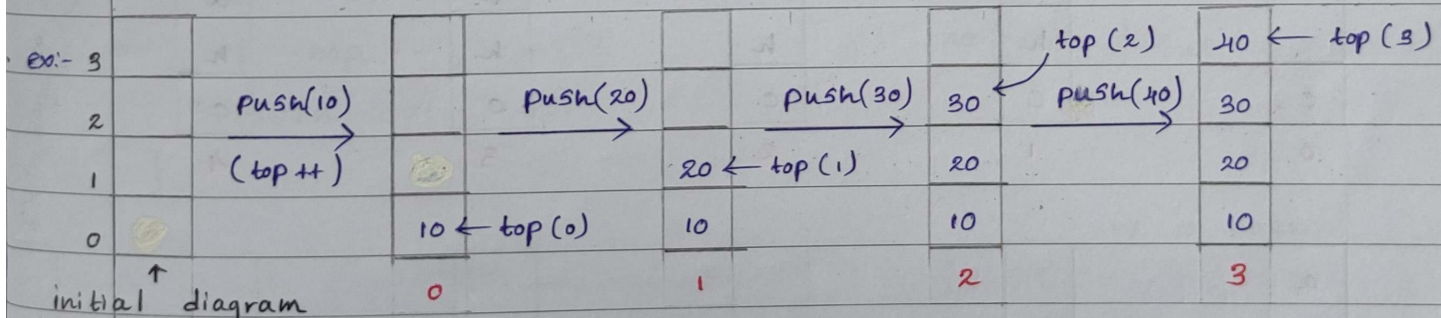


levels

* A Stack is known as a data set.
* A Stack can made by An array or a list.
* Stack method → first in last out (LTFO)



• Inserting (push)



initial diagram

top (-1)

arr 10, 20, 30, 40

int top; print '0' or '1' or '2' or '3'

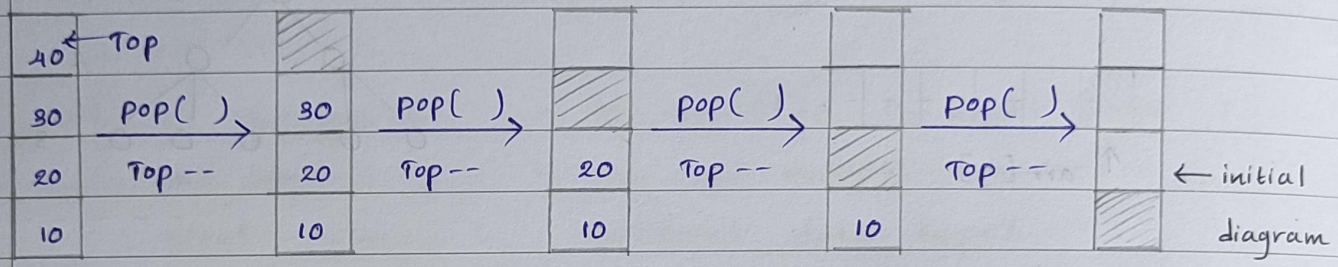
arr [top]; '10' or '20' or '30' or '40'

* Top is always an index.

* inside the stack it is → 0, 1, 2, 3, ...
when it's outside the stack ..., -3, -2, -1 ←

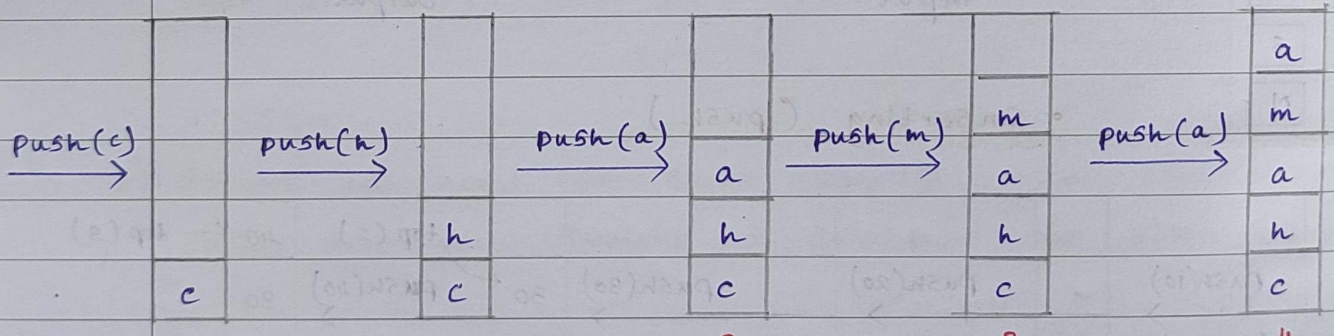
command, insert → push ( );
delete ↔ pop ( );

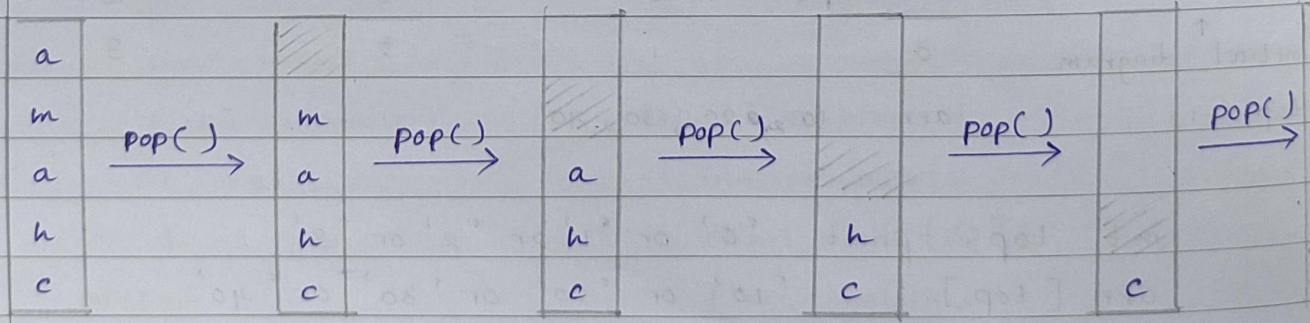
deleting (pop)



* pop( ) will remove the top most element.
* we cannot Delete values 40 isn't visible.

ex:- create a stack of the 5 letters of your name.





Stack Structure declaration.

```
# define size 6
    Struct stack {
        int arr [ six ];
        int top ;
    } st ;
```

Structure arr[ ]

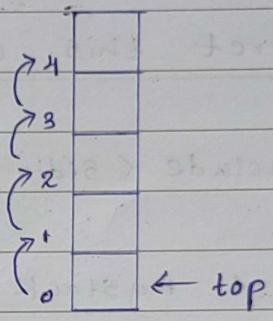
st. top

Structure variable.

Structure for push operation.

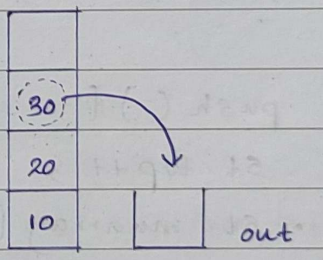
```
Void push ( int element ){
        st. top ++ ;
        st. arr [ st. top ] = elem;
}
```

normally in 'c' arr [top] = 10;



Structure of pop operation

```
int pop ( ) {
    int out ;
    out = st. arr [ st. top ];
    st. top -- ;
    return out ;
}
```



18. 10. 2022

* Array have random access variables. Struct don't have.

Array (RA)

↓

Stack (~~RA~~)

> check for empty and full Stack.

```
int st empty( )
{
    if ( st.top == -1 )
        return 1;
    else
        return 0;
}
```

```
int st Full( )
{
    if ( st.top >= size-1 )
        return 1;
    else
        return 0;
}
```

Q) recorret this code.

```
# include <stdio.h>

Struct mystack {
    int myarray[5];
    • int top = -1;
} st;
```

int top (we don't declare element here like int top = -1;)

```
• void push( ) {
    st.top++;
    • st.myarray[top] = ele;
    printf ( " Item Added to index %d \n ", st.top );
}
```

void push (int. element) need to add element.

st.my array [st.top] = ele ;

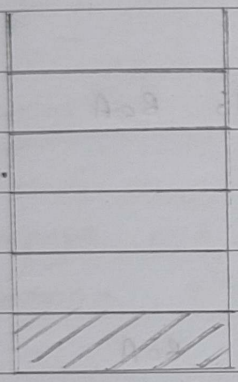
```
• int pop (int ele ) {
    int out = myarray [ st.top ];
    printf ( " popped out the item at index %d ", st.top );
} → next page.
```

int pop { we don't add element here

```
int main () {
    st.top = -1;
    push (10);
    pash (20);
    push (30);
    pop ();
retunn 0;
}
```

```
int pop() {
    int out;
    out = myarray [st.top];
    st.top --;
printf ("..... statement ....."")
    return out;
}
```

* main method is alone. and all the declaration are on outside the main.



> Stack → Evaluate postfix notifications.

infix → x + y prefix +xy postfix xy+

operator — operands

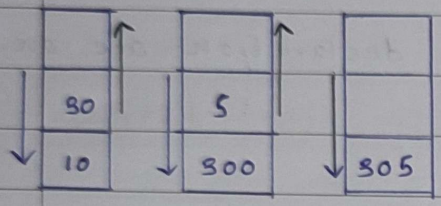
ex:- 10 30 * 5 + (postfix)

* In this case, these kind of eguation is also solved using a stack. There are few steps for that process.

IF the operand comes first
push ( ) it to the stack.

ELSE IF the operator comes.
pop ( ) → A
pop ( ) → B
} pop 2 times.

AND B o A
After push (B o A)

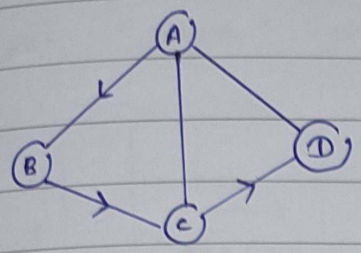


push (10);
push (30);
pop ( ); → 30
pop ( ); → 10
Ⓡ ← 10 * 30 push (300); BoA
push (5);
pop ( ); → 5
pop ( ); → 300
Ⓡ ← 300 + 5 push (305); BoA

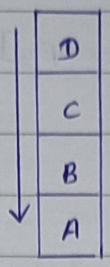
ex:- 100 200 + 2 / 5 * 7

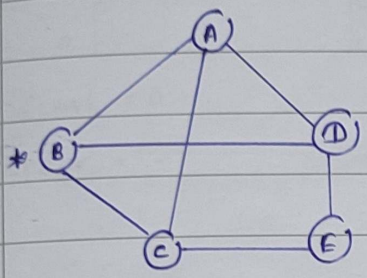
push (100);
push (200);
pop ( ); → 200
pop ( ); → 100
100 + 200 push (300); BoA
push (2);
pop ( ); → 2
pop ( ); → 300
300 / 2 push (150); BoA
push (5);
pop ( ); → 5
pop ( ); → 150
150 * 5 push (750); BoA
push (7);
pop ( ) → 7
pop ( ) → 750
750 + 7 push (757); BoA

Depth first search



A, B, C, D





B A C E D

* This is commonly using in graph data structure.

2022.10.26.

Tutorial 1

01) Compare and contrast stacks and queues in term of adding, removing, size and emptiness.

Stacks	queues.
• use 'push' command to add data. (Inserting)	• Inserting element is known as the 'enqueue.' operation
• use 'pop' command to remove data. (deleting)	• deleting element is known as the 'dequeue'
• This structure implements and follows Last in first out principle (LIFO)	• This structure implements and follow First In First out (FIFO) principle.
• Stack is a linear form of data structure.	• Queue' also can refer as a linear form of data structure.
• user can delete and insert elements from only one side of a list	• In this, a user can insert elements from only one side of the list.

2)
```
void push (int element) {
        st.top ++;
        st.arr [st.top] = element; }
int main() {    push (7);
                push (6);
                push (3);
                push (5);
                push (9);
                push (0);
}
```

03)
```
pop( ); → 0
pop( ); → 9
pop( ); → 5
pop( ); → 3
push (2);
```

04)
```
int st empty( )
{
    if (st.top == -1)
        return 1;
    else
        return 0;
}
```

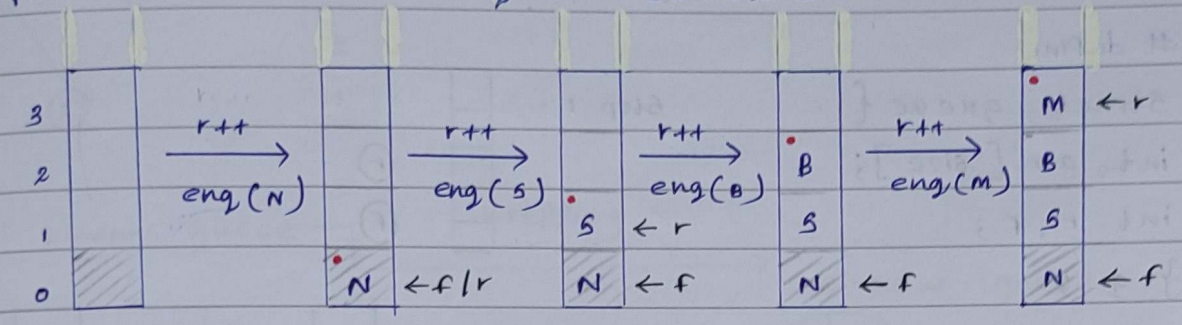
* In the queue data structure, elements are fixed and not moving when the previous element goes out. what is happenning here is the 'Front' position is changing according to the relavant element.

Rear → To 'add' elements to the data structure.
Front → To 'remove' elements from the data structure.

2022.10.02

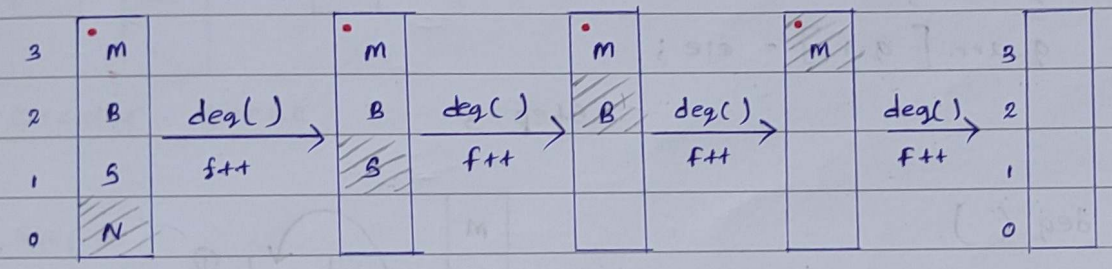
> queue

Put 'NSBM' into a queue structure.



front = 0

• rear = -1 enque → Insert deque → Delete.

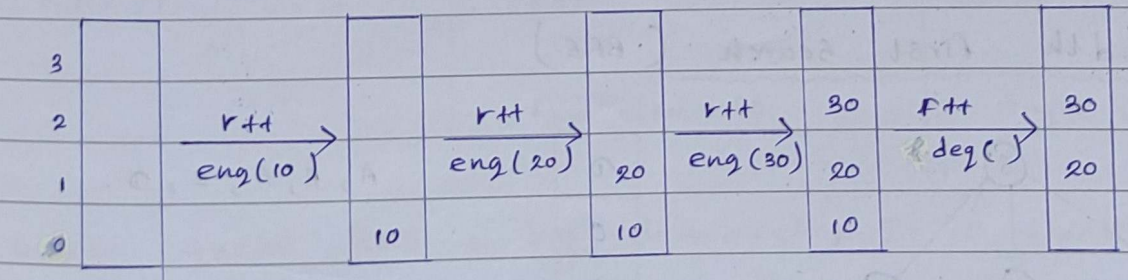


front = 0	f = 1	f = 2	f = 3	f = -1
• rear = 3	R = 3	R = 3	R = 3	R = -1

} empty

Q) perform these operation and get the final queue.

Enq(10), Enq(20), Enq(30), deq.



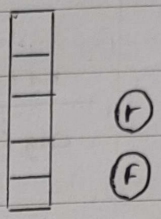
front = 0	f = 0	f = 0	f = 0	f = 1
rear = -1	R = 0	R = 1	R = 2	R = 2

* 'Front' operation won't make any changes to the data structure.

Q) 1) write a code for a queue of 5 elements 2) engue operation.
3) degue operation.

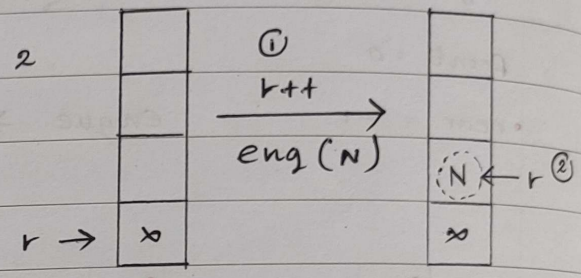
1)
```
#define
Struct queue {
int arr [size];
int r, f;
} q ;
```

Step 1 :



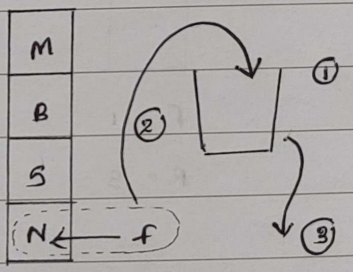
2)
```
void eng (int ele)
{
    q.r ++;
    q.arr [q.r] = ele;
}
```

Step 2

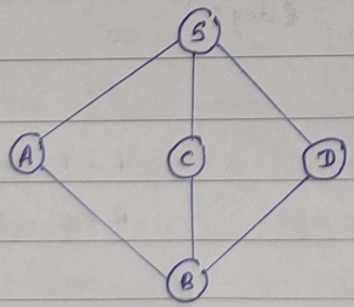


Step 3

3)
```
int deq ( )
{
    int out ;
    out = q. arr [q.f];
    q.f ++;
    return out ;
```



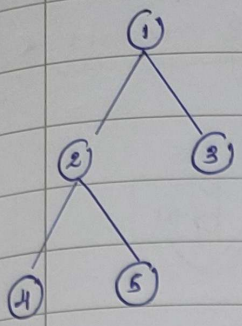
➢ Breadth first search (BFS)



①
C
S
B
A

A, B, S, C, D.

go with an 'And operation'.



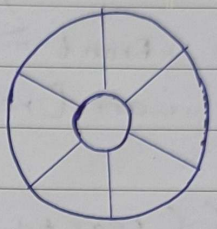
5
4
3
2
1

1, 2, 3, 4, 5

2022.11.09.

Linear queue → wrapping → circular queue.

50 ← Rear
40
30
20
10



circular queue.

Step:-

① declare all the elements
- array
- f = -1
- r = -1

② function to perform engue.
- Empty
- Full
- few elements

③ function to perform deque.
- Full / few
- Empty
- Single.

* Linear queue has a big drawback, It is wasting memory.

> function to perform engue.

```
#include <stdio.h>   ← empty
int arr[5];
Int front = -1;
int Rear = -1;
// function to perform engue.
void engue (int x){

    If ( front == -1  &&  rear == -1 ) {
            Front = rear = 0;
            arr [rear] = x; }

    else if ((rear+1) %. N == front ) {
        printf ( " queue if full " ); }

    else {  rear = (rear + 1) %. N;
            arr [rear] = x; }
```

> Function to perform deque;

```
// function to perform deque
Void  deque ( ){
    if ( front == -1  &&  rear == -1 ){
            printf ( " No elements " ); }

    else if ( rear == front ){
            printf ( " %.d ", arr [front])
```

* note

```
            Front = -1;
            rear = -1;
    }
```

2022.11.16

> Linear Search.

0	1	2	3	4	5
10	20	30	40	50	60

Search key = 40

A[0] == SK ? ✗
A[1] == SK ? ✗
A[2] == SK ? ✗
A[3] == SK ? ✓

Algorithems

- Search
  - Linear S.
  - binary S.
- Sort
- Recursive.

* when number of elements are increasing The search time increases.

$t \propto N \uparrow$ t - time | N - no. of elements

* we use linear search for smaller data sets. It's not good for larger data sets.

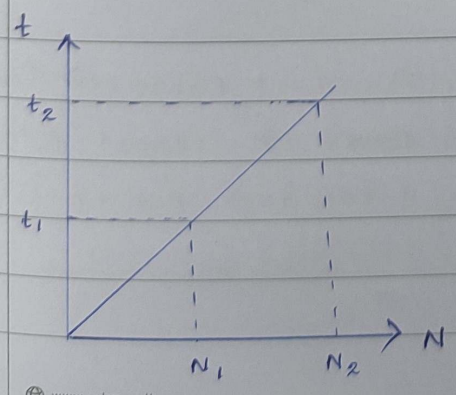
ex:-

case	10	100	1000
Best	0th, 1st	0th, 1st	0th, 1st
avg	4th, middle	49th, mid	499th, mid
Worst	9th, E/NF	99th, E/NF	999th, E/NF
	t_1	t_2	t_3

N

0th → index position
1st → search
E → End
NF → Not found.

$t_1 < t_2 < t_3$



- **Algorithem for linear Search** $2^{10} = 1024$

Set found to false; Set position to -1; Set index to 0

while index < number of elements. and found is false.

    if list [Index] is equal to search value
        found = true.
        position = index
    end if
        add 1 to index
    end while

return position.

Linear Search

```
int lsearch ( int arr [ ], int SK , int N ){
    bool found = false;
    int position = -1;
    int index = 0;

    while ( index < N && ! found ){
        if ( arr [ index ] == SK ){
            position = index;
            found = true;
        }
        index ++;
    }
    return position;
}
```

Index search → I searc

found == false → True
found → returns 'false'
! found → not false (True)

equal ලෙස ද ?

for larger data sets / organized sorted in order.

exam:- unsorted data set → sorted Data set.

> Binary Search

* using for large data sets.
* we input organized, sorted data set in order.

0	1	2	3	4	5
8	10	15	25	35	40

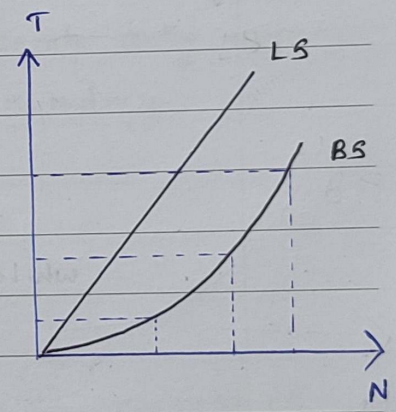
• Linear Search example.

2022. 11. 23.

10	2	15	25	18	30

SK = 18

Index	List[Index]	SK	Found	position.
0	10	18	F	-1
1	2	18	F	-1
2	15	18	F	-1
3	25	18	F	-1
4	18	18	T	4
5	30	-	-	-



• Algorithem of bimary Search.

Set first index to 0. Set last index to the last Subscript in the array. Set found to false. Set position to -1.

while found is not true and and first is less than or equal to last set middle to the subscript half-way between array [first] and array [last].

If array [middle] equals the desired value.
    Set found to true.
    Set position to middle.

Else if array [middle] is greater than the desired value.
    Set last to middle -1.

Else
    Set first to middle +1.

End If.

```
Int bSearch ( int A[], int SK, int N ){
    int first = 0;
    int Last = N-1;        6-1 = 5
    bool found = false;
    int position = -1;

    while ( !found && First <= last ){
        int midI = first + last / 2;

        if ( A [ midI ] == SK ){
            found = true;
            position = midI;
        }
        else if ( A [ midI ] > SK ){
            last = midI - 1;
        }
        else {
            first = mid + 1
        }
    }
return position;
}
```

binary Search

To apply binary Search, array Should be Sorted.

6	70
5	60
4	50
(3)	40 → mid I
2	30
1	20
0	10

i) $midI = \frac{f+l}{2} = \frac{0+6}{2} = 3$

A [midI] == SK ?

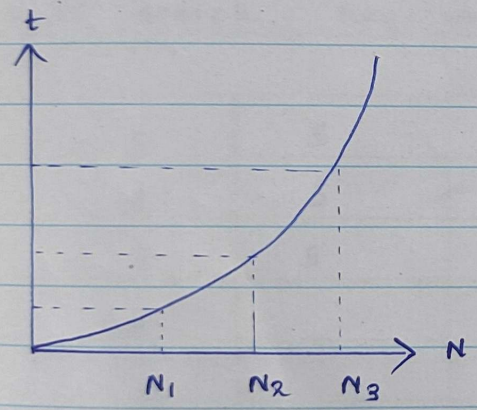
40 == 60 ×

ii) A [midI] > SK → True.

l = midI - 1 → search only lower boundary.

[else] iii) Search at the up

f = mid + 1 → Search only upper boundary.



$t \propto \log x$

i) $1024 = 2^{10}$

ii) 100, $2^6 = 64$, $2^7 = 128$ } closest is 128

So, $100 = 2^7$

ex:-

0	1	2	3	4	5
2	10	15	18	25	30

i)

first	last	midI	A[midI]	SK	found	position
0	5	2	15	18	F	-1
3	5	4	25	18	F	-1
3	3	3	18	18	T	3

ii) If search key was = 12,

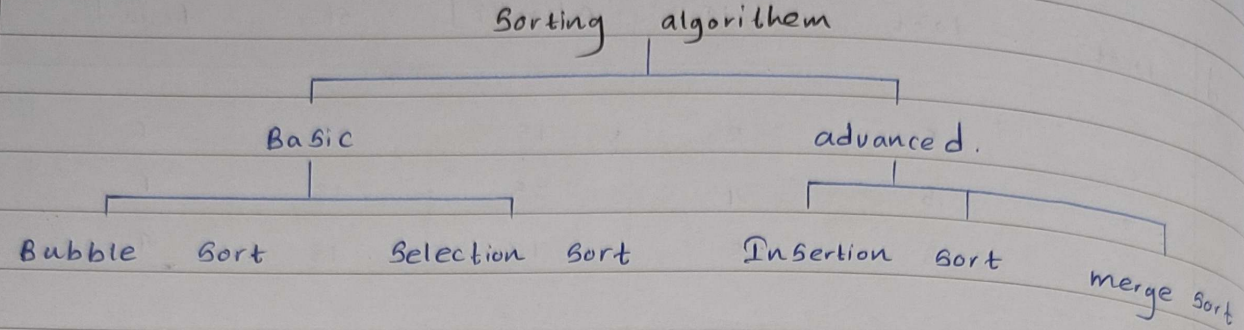
0	5	2	15	12	F	-1
0	1	0	2	12	F	-1
1	1	1	10	12	F	-1
2						

iii) If search key was = 35,

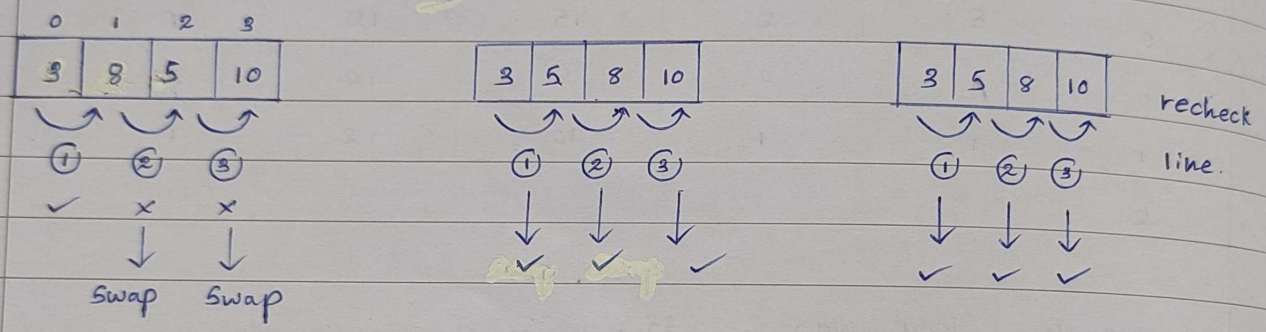
0	5	2	15	35	F	-1
3	5	4	25	35	F	-1
5	5	5	30	35	F	-1
6						

2022. 11. 30

➤ Sorting algorithem.

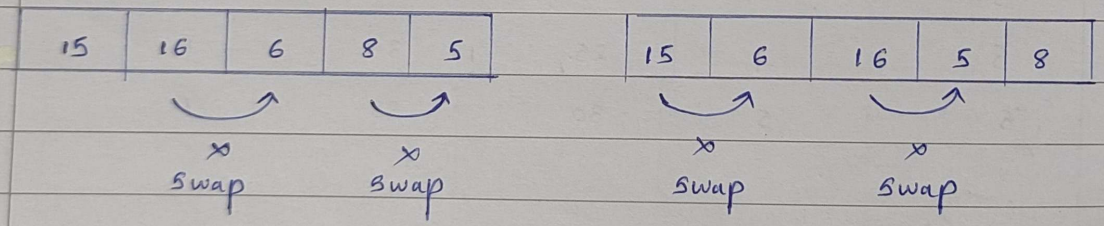


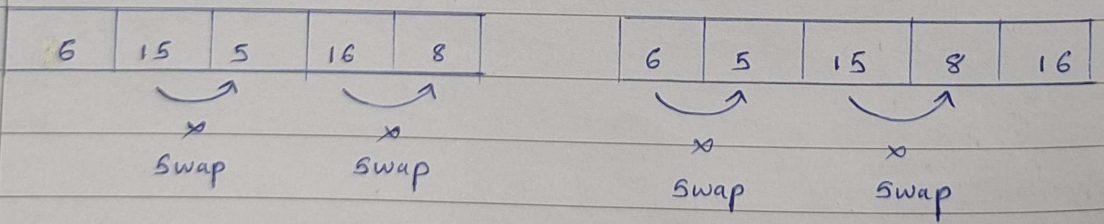
1) Bubble Sort

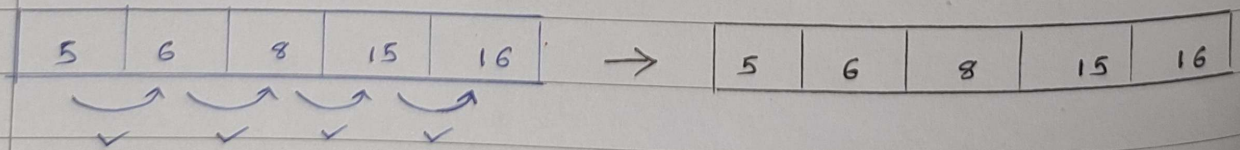


* when swap remains false it end upthe algorithem.

ex:-







No. of passes = 05

```
int i ; int A [size]; int temp
temp = A [i];
A [i] = A [i+1];
A [i+1] = temp;
do {
        swap = false;
        for ( i=0 ; i<n ; i++ ){
                if ( A [i] > A [i + 1 ]){
                        temp = A [i];
                        A [i] = A [i+1];
                        A [i + 1] = temp;
                        swap = true ; }
    }
} while ( swap );
```

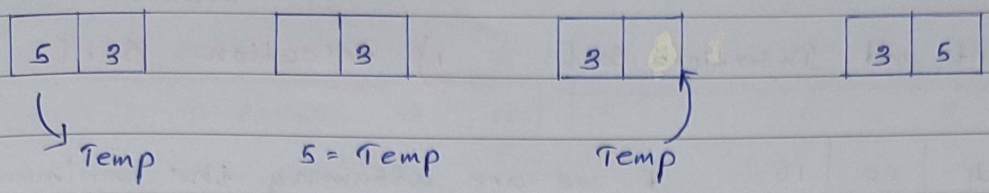
A = arr

- bubble Sort -

i Assign do temp

i+1 " " (overwrite)

Ex. -




```
int temp;
if ( arr[i] > arr[i+1])
{
   temp = arr[i];          arr[i] assign to temporary
   arr[i] = arr[i+1];      value.
   arr[i+1] = temp;        ( overwrite )
}
```

Ex. -

12	20	18	10

12 18 20 10 → 12 18 10 20

Iteration	i	i + 1	A[i]	A[i+1]	Swap	
					false	
①	0	1	12	20	false	
	1	2	20	18	True	
	2	3	20	10	True	
②	0	1	12	18	false	
	1	2	18	10	True	12 10 18 20
	2	3	18	12	false	12 10 18 20
✕	3	4	—	—		
③	0	1	12	10	T	10 12 18 20
	1	2	12			

3 8 12 15 16

1st Iteration

0	2	1
15	8	3

2nd iteration

1	2
16	8

11th c Iteratio

3	4
16	15

3rd Iteration

2	3
16	12

2022.12.08

> Selection Sort and Insertion Sort.

1) Selection Sort

1)

18	12	4	20	16
0	1	2	3	4

* we are assuming the minimum index have the minimum value in array.

1st Iteration → min Index → 0 1 2

min value → 18 12 4

4	12	18	20	16

2nd Iteration → min Index → 1

min value → 12

4	12	18	20	16

unchanged

3rd Iteration → min Index → 2 3 4

min value → 18 20 16

4	12	16	20	18

4th Iteration → min Index → 3 4

min value → 20 18

4	12	16	18	20

2)

0	1	2	3	4	5
15	16	6	8	12	5

1st, M. Index → 0 2 5

M. value → 15 6 5

5	16	6	8	12	15

2nd, M. Index → 1 2

M. value → 16 6

5	6	16	8	12	15

3rd, M. Index → 2 3

M. value → 16 8

5	6	8	16	12	15

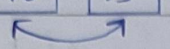
4th, M. Index → 3 4

M. value → 16 12

5	6	8	12	16	15

5th , M. Index → 4 5

M. value → 16 15



5	6	8	12	15	16

Code for the Selection Sort.

```
for ( i=0 ; i< n-1; i++ ){
        Int  min = i ;
        for ( j= i+1 ; J< n; i++ ){
              if ( arr [ i ] < arr [ min ] )
                    min = j;
        }

        if ( min ! = i )
              Swap ( arr [ i ], arr [ min ] );
}
```

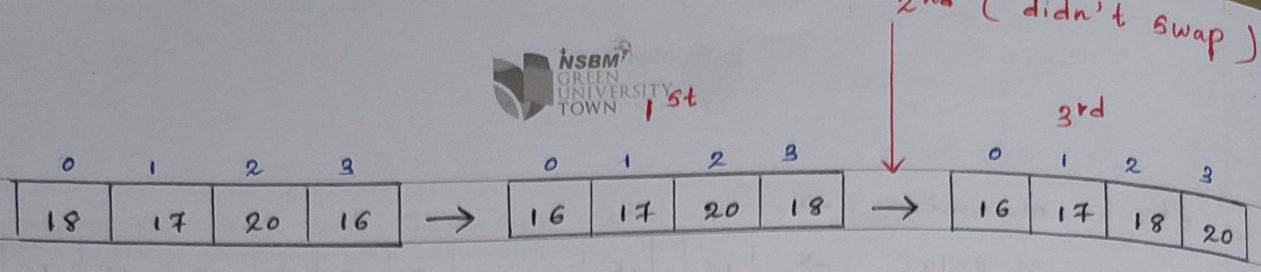
Desk check

18	19	20	25

outer for loop

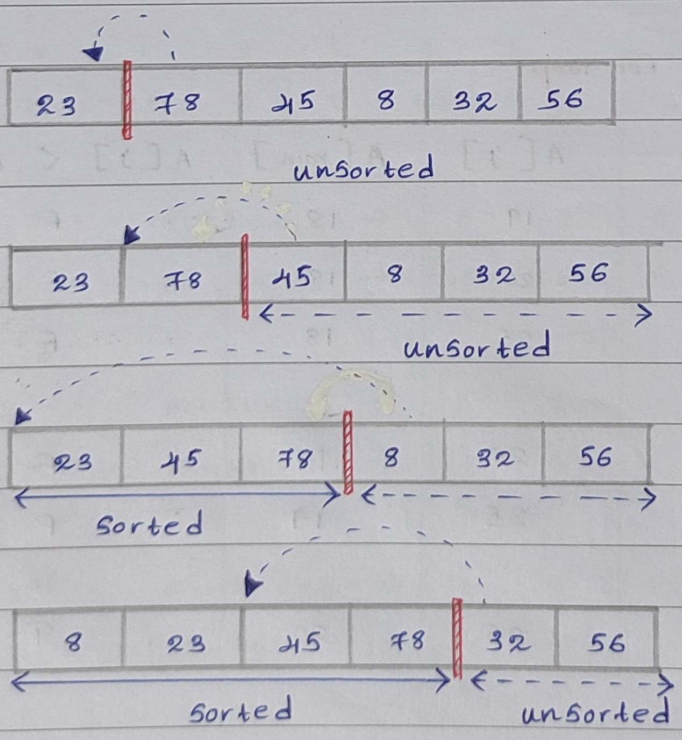
Inner for loop

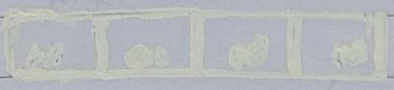
I	j	min D.	A [j]	A [min]	A [j] < A [min]
0	1	0	19	18	F
	2	0	20	18	F
	3	0	25	18	F
1	2	1	20	19	F
	3	1	25	19	F
2	3	2	25	20	F T

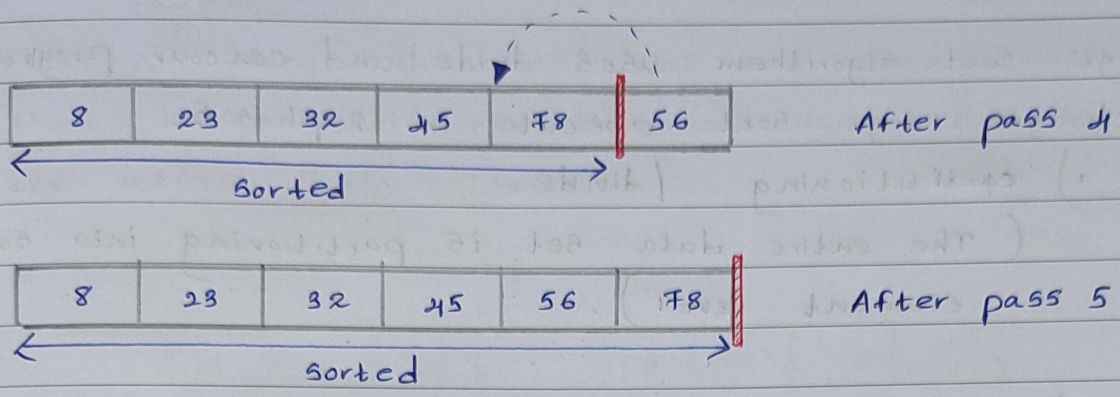


	i	j	min ID.	A[j]	A[min]	A[j] < A[min]
1st →	0	1	0 '1'	17	18	True
		2	'1'	20	17	False
		3	1 '3'	16	17	True.
2nd →	1	2	1	20	17	False
		3	1	18	17	False
3rd →	2	3	(2) 3	18	20	True
		4				

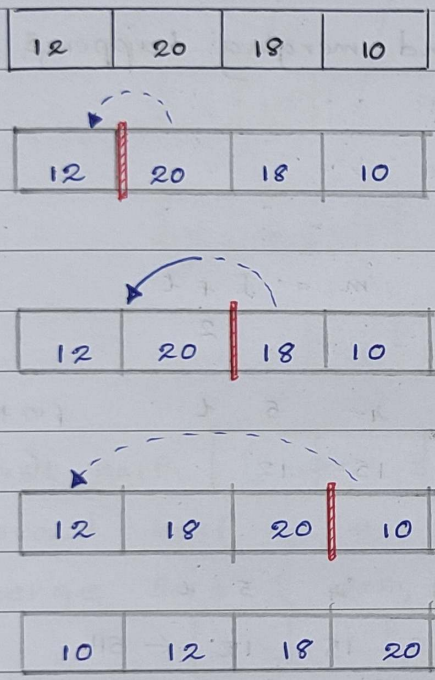
> Insertion Sorting algorithem.







a)



- Insertion Sort Implementation

1)
```
For (k=1 ; k < size; k++){
    temp = A[k];
    J= k-1;
    while (j>=0 && A[j] >temp)
      {
        A[j+1] = A[j];
        j--;
      }
      A[j+1] = temp;
}
```

2)
```
void insort (float d[ ], int size ){
    int i, k, t;
    Float temp;
    For ( k=1; k<size ; k++ ){
     For (i=0; i<k && d[i]<= d[k], i++);
        temp = d[k];
    For( t=k;  >i; t--) d[t] = d[t-1]
        d[i] = temp;
    }
}
```

2022.12.14

merge Sort

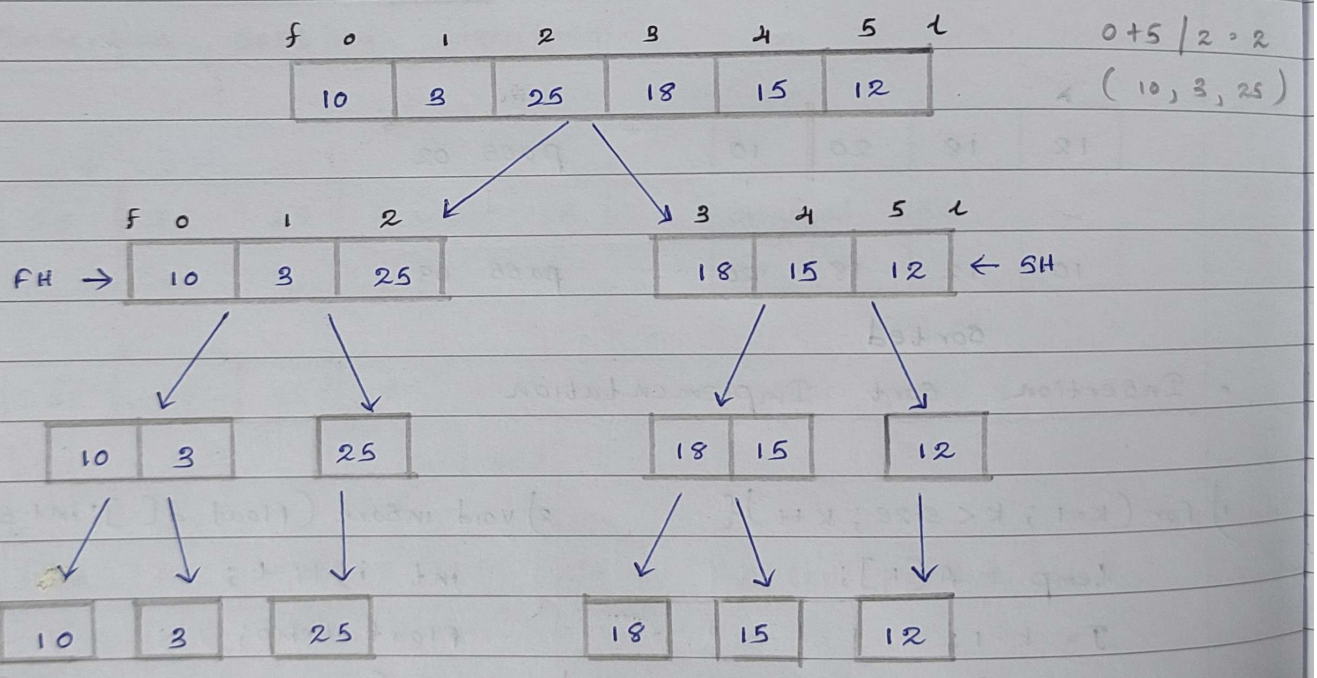
* merge Sort algorithem uses divide and conquer programming technigue merge Sort execute in 2 phases.

1) partitioning / divide
   (The entire data set is partitioning into single element level)

2) conquering / merge
   (The actual sorting and merging happens here)

1) partitioning / divide. phase.

n/2 , balanced part → $m = \frac{f + l}{2}$



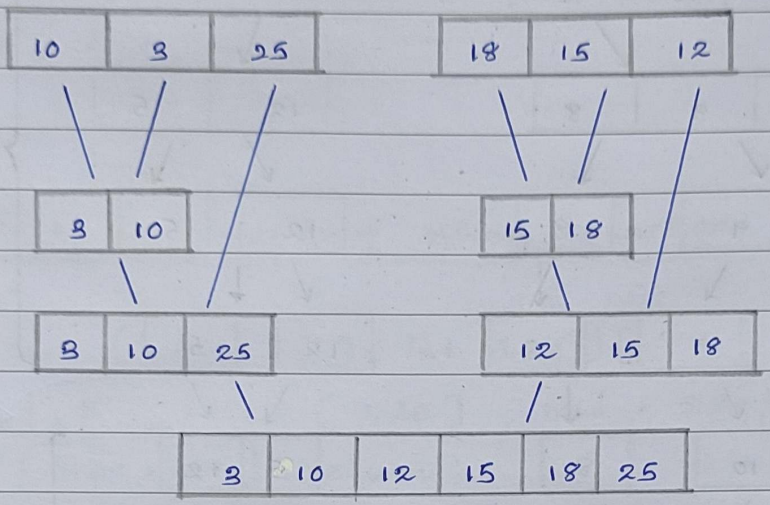
first half → FH (arr , f , m)

Second half → SH (arr , m+1 , l)

l > f (Stoping criteria)

2) conquer / merge phase.

* First we merge the first half and second half parrelally then merge the first half and second half to complete the merge sort.



first half (arr , f , m)
second half (arr , m+1 , l)
merge sort (arr, f , m , m+1 , l)

> steps for the merge sort.

mergesort (arr [ ], f , l)
if (l > f)

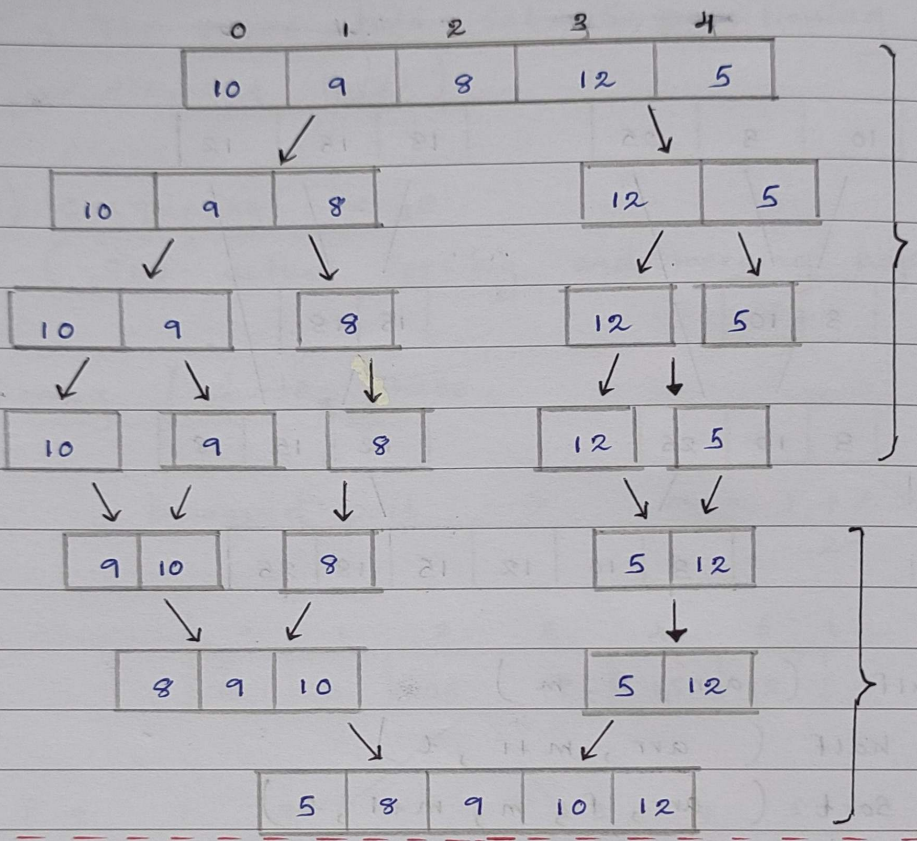
- find the middle point to divide the array into two halves:
  * middle m = (f+l) / 2

- call mergesort for first half:
  * call merge sort (arr , f , m)

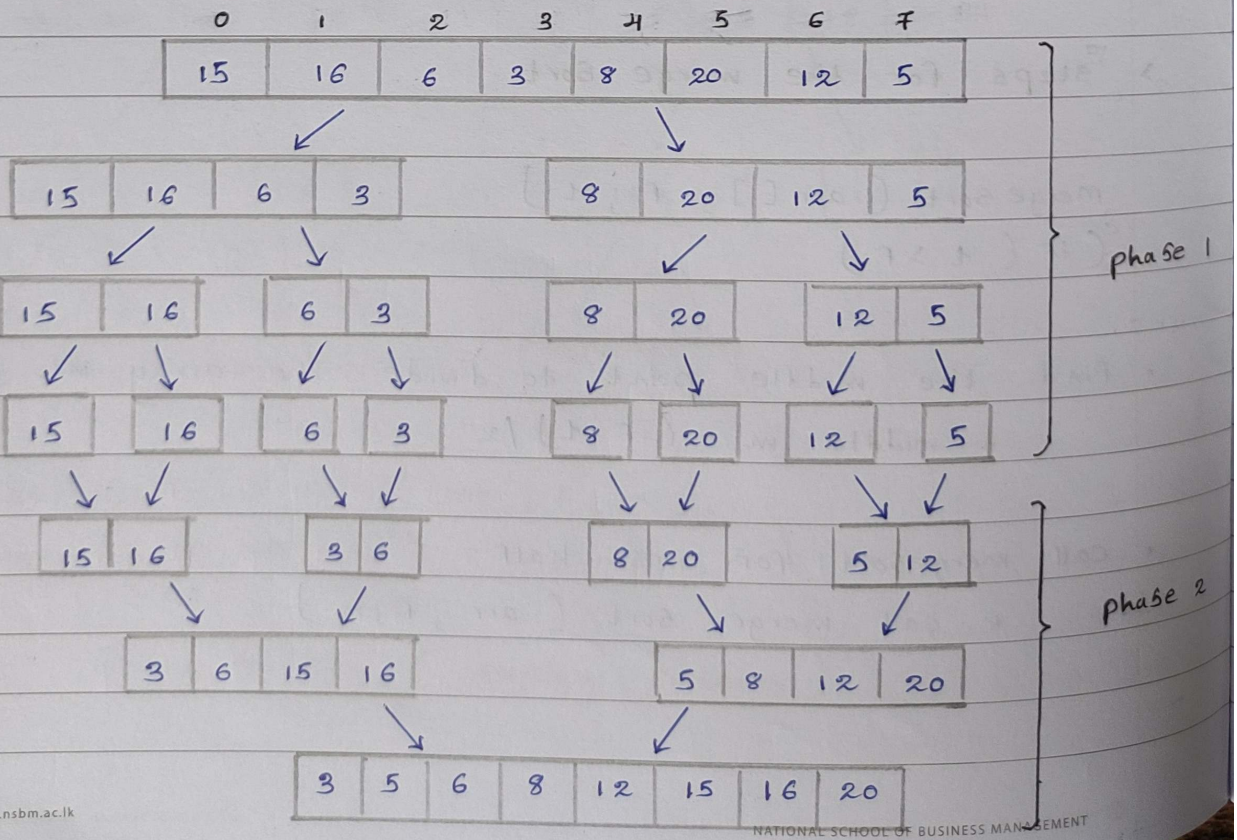
- Call merge Sort for Second half:
  - * Call merge Sort (arr , m+1 , r)

- merge the two halves Sorted in Steps 2 and 3 :
  - * Call merge (arr , f , m , r)

Q)



Q)



```
msort (int d[], int l, int r){
      if (l<r)
         mid = (l+r)/2
   msort (d, l, mid);
   msort (d, mid+1, r);
   merge (d, l, mid, r);
}
```

> merge sort Implementing using recursion:

```
void msort ( int d[], int size ){
int  i, j, k , result [size] , mid = size/2;
      if (size == 1) return;
      msort (d, mid);
      msort (&d [mid], size-mid);
         for ( k=0, i=0, j = mid, i<mid && j<size , k++ )
         result [k] = d [i] < d [j] ? d [i++] : d [j++];
         while ( i< mid ) result [k++] = d [i++];
         while ( j < size ) result [k++] = d [ j++];
         For ( k=0 ; k < size ; k++ ) d [k] = result [k];
   }
```

2022.12.21

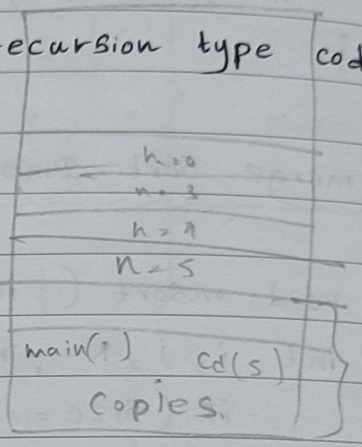
Recursion.

A function calling itself defining as Recursive function.

- Recursive functions : Base case / Recursive call.

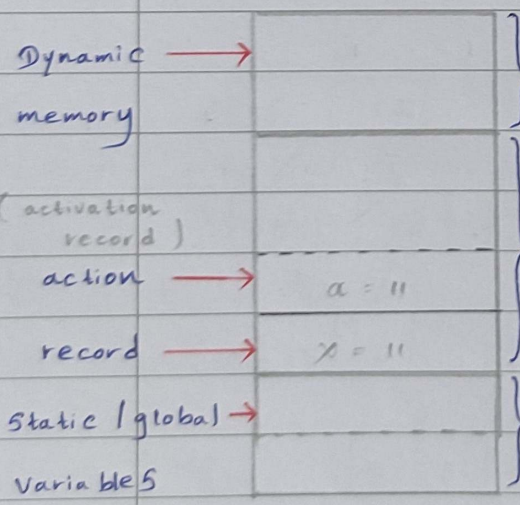
1) A recursive function for counting down to 0; (n = 5)

```
Void countdown ( int num )
{
    IF ( num == 0 )
        print Done ! ;
    else
    {
        print num ;
        Countdown ( num - 1 ) ;   // Recursive call
    }
}
```

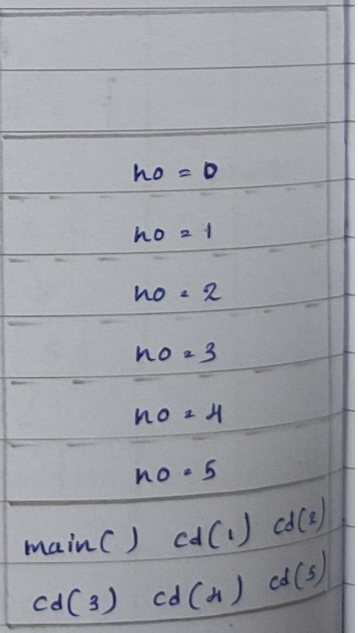


output → 5 4 3 2 1 done.

- memory alloction • while coding & runing



* finally delete the all code copies and the main() function copy.



Dynamic memory - All the things happen in runtime.

Code section - machine code is loaded in here.

Base case - where y going to

NSBM GREEN UNIVERSITY TOWN

03) void (head - recursion type code)

{

if . . . .

else

{

countdown (num-1);

print num);

}

In this code It's execute the last line (print num) first. The out put generate's while deleting the action record.

n = 1
n = 2
n = 3
main( ) cd(3)
cd(2) cd(1)

output : Done 1 2 3

* any given recursive function has two components.

1) Base - case / stopping condition / termination point.

2) The recursive call

(Recursive call runs towards the base case.)

Q) write a function to get the factorial value of a given number.

```
int fact ( int n ){

    if ( no == 0 );
        return 1 ;

    else
        return n * fac (n-1);

}
```

n! = (n-1) * n

5! = 4! * 5

↳ 3! * 4

↳ 2! * 3

↳ 1! * 2

↳ 0! * 1

↳ 1 return.

factorial of '1'

2022.12.22

n = 3

n * fac(n-1)

3 * (3-1) = 6

2 * (2-1) = 2

1 * (1-1) = 0

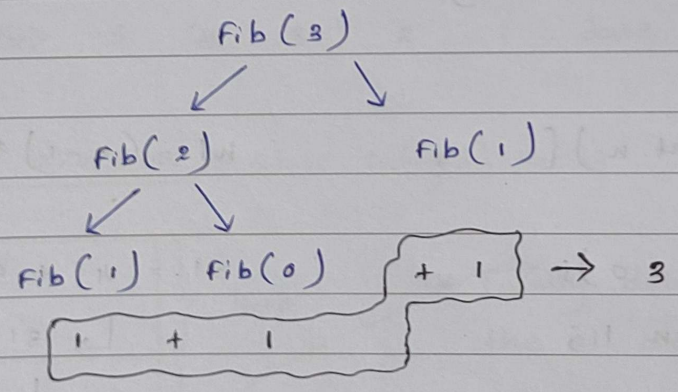
Fibonacci Sequence.

0 1 2 3 4 5 6 7 8 9

1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55

```
int fib (int n)
    if (n <= 1)
    return 1;
  else,
    return fib(n-1) + fib(n-2);
```

- Call tree for n = 3;



50

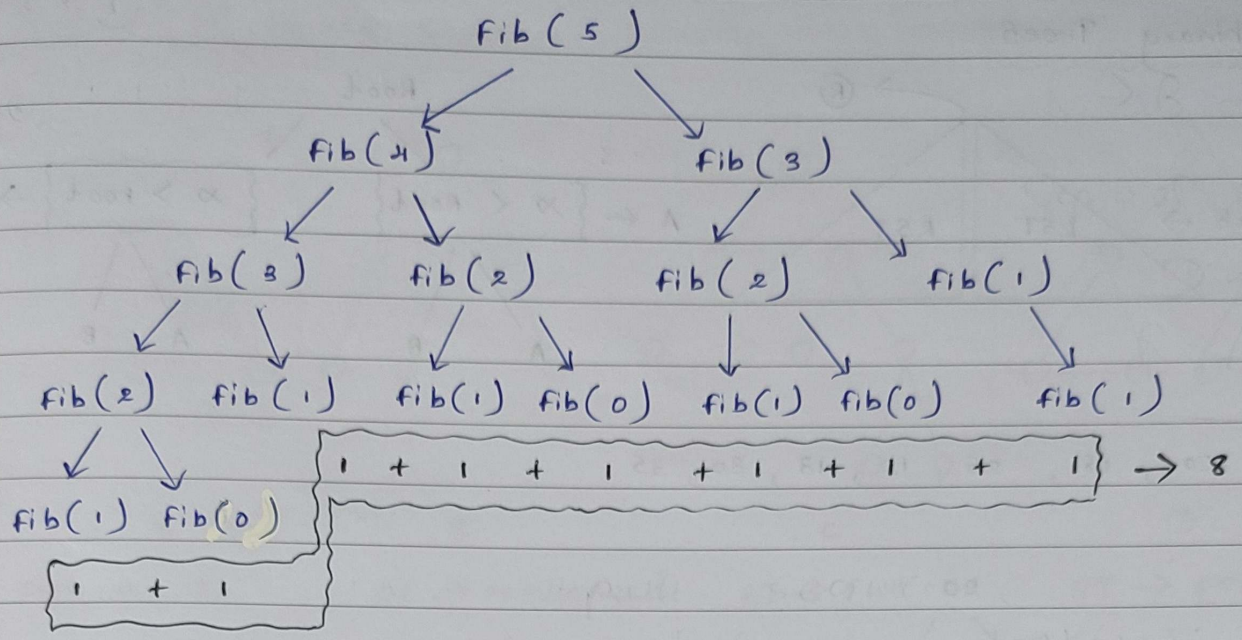
25

15 35 65

85

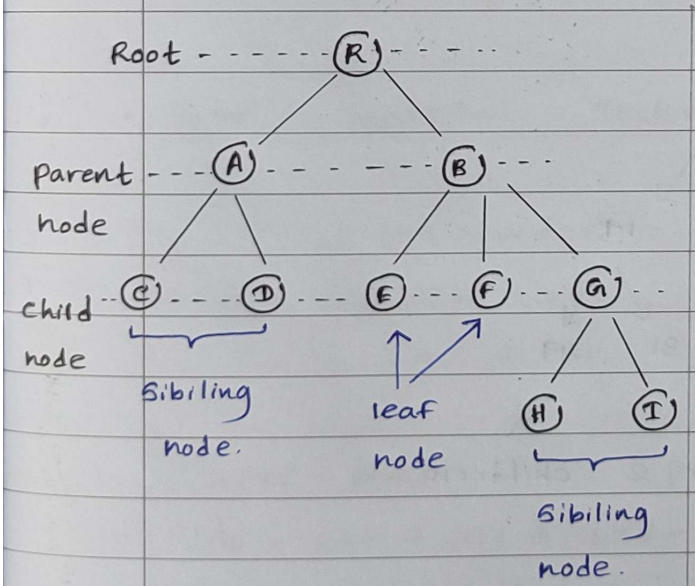
10

Root
parent
child
Leaf nd
leaves
sibling



> Tree data structure.

- A Tree is a collection of nodes and are used to interconnect nodes.
- A node with children called parent node.
- The top most node of a tree structure is 'Root' and there is only one root for any tree structure. (supergrandparent)
- also the top most node is the starting point. (0 or more)



path → Sequence (R → B → G)

hight → max depth of structure. (depth of all the nodes)

depth → number of nodes in the path.

binary tree → maximum of 2 childs min = 0.

multimay tree →

Leaf node → node with no children. normally the end point of Structure.

Sibiling node → nodes who have common parent nodes.

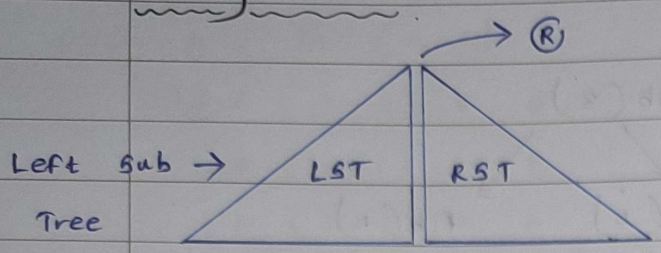
Root (R, B, G) path: R → B → G

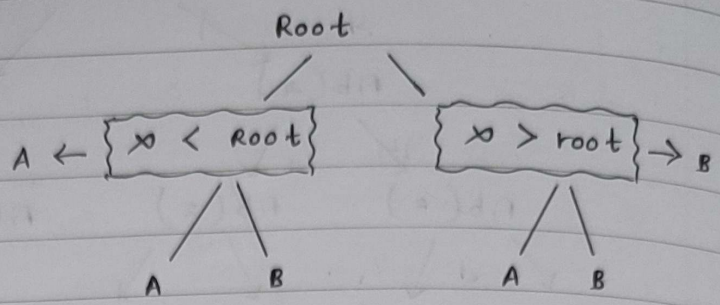
Depth of node = 3

↓

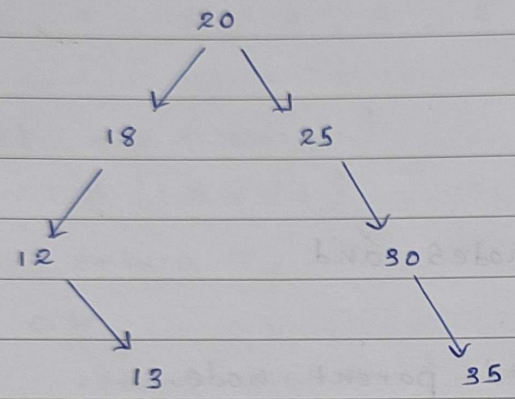
(Value)

- binary Trees.

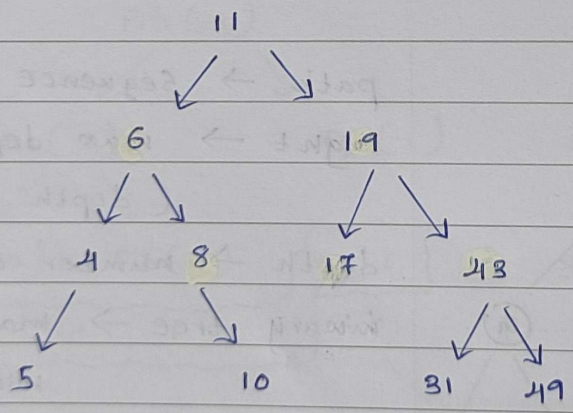




01) 20, 18, 25, 12, 13, 30, 35



02) 11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31.



- binary tree have only 0, 1 or 2 children.

- 0, 2 children
full binary
all the
leaf nodes
are at
(LFL) the same
level

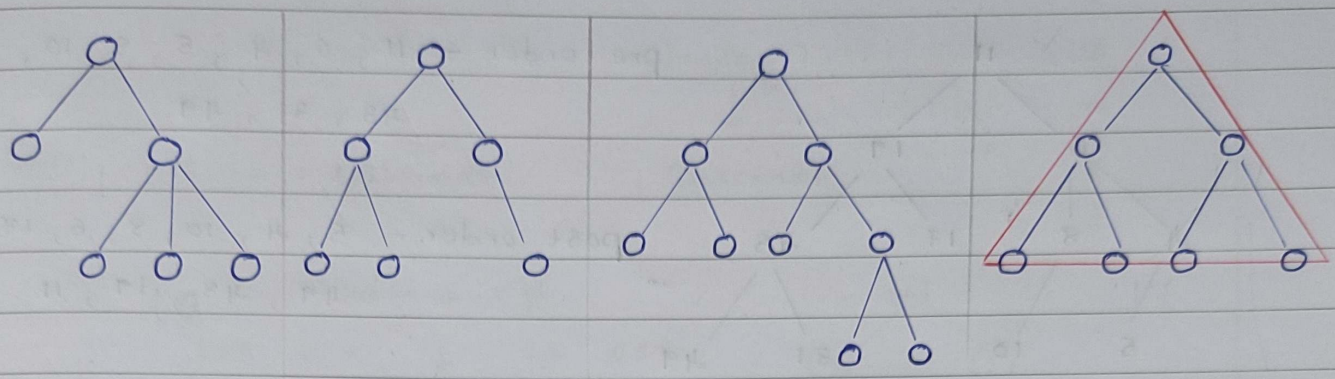
NSBM GREEN UNIVERSITY TOWN

2 - 3 publications

Research Publication

2023.01.04

- Types of binary trees



A | B | C | D

Not a BT | BT → unbalanced | BT → full | BT → perfect

(0, 1, 2) | (0, 2) | (0, 2) — full, LSL

* binary tree is recognized as a perfect binary tree when it is a full binary tree and all the leaf nodes at the same level.

- equation for a perfect binary tree.

$n = 2^h - 1$

$63 = 2^6 - 1$

n - nodes

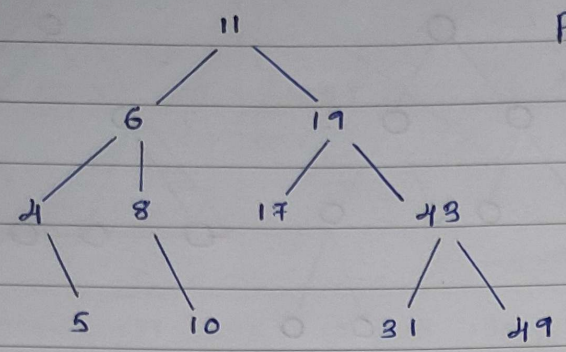
h - height of PBT

- Tree Traversal Techniques

1) pre - order
2) post - order
3) In - order.

pre - order	post - order	In - order
VLR - visit if left is null move to right if not stay left	LRV - If left is null, then left is null then visit the node.	LVR - This is called left visit right
parent first	child first	

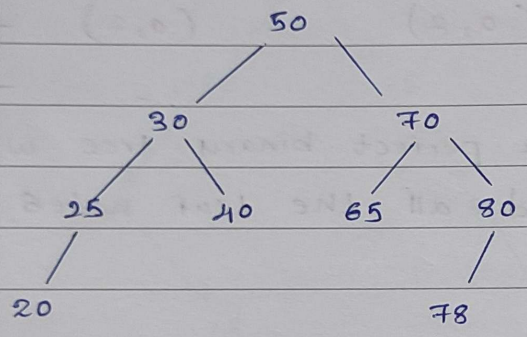
a) Find the pre-order



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

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

In order - , , , , , , ,



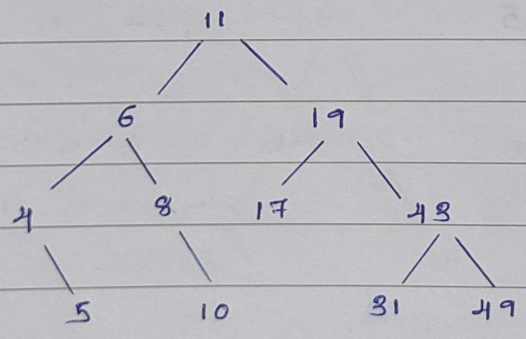
pre order - 50, 30, 25, 20, 40, 70, 65, 80, 78

post order - 20, 25, 40, 30, 65, 78, 80, 70, 50

In order - 20, 25, 30, 40, 50, 65, 70, 78, 80

2023.01.05.

ex:-



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

> Expression tree

B O D M A S

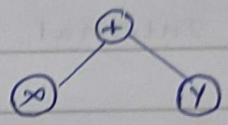
operand ↓ Leaf / External

operator ↓ Internal

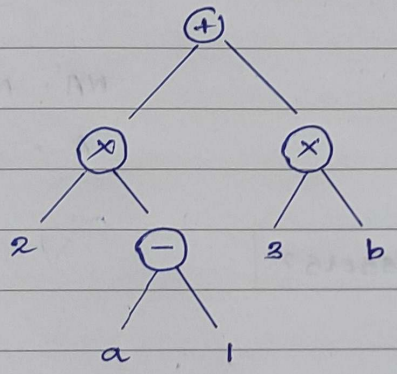
x + y

LST ← ↓ → RST

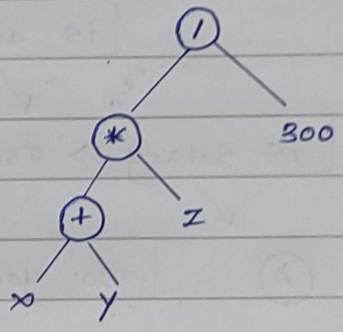
Root



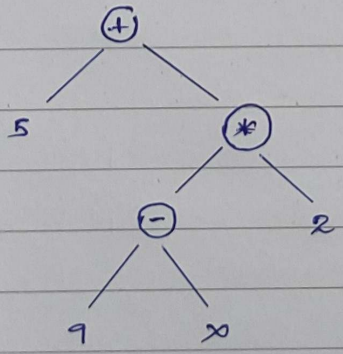
1) (2 * (a - 1) + (3 * b))



2) ((x + y) * z) / 300

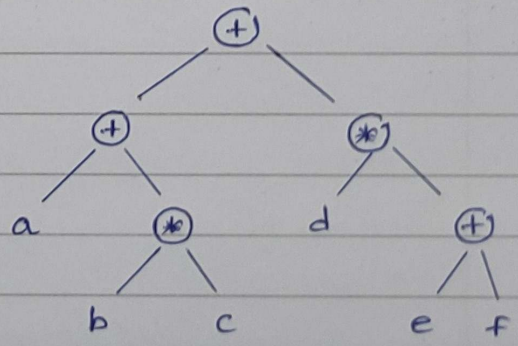


3) (5 + (9 - x) * 2)



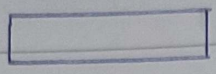
4) Test paper question

a + (b * c) + d * (e + f)



Decission Tree

Question with y / n



↓

Internal

Dession

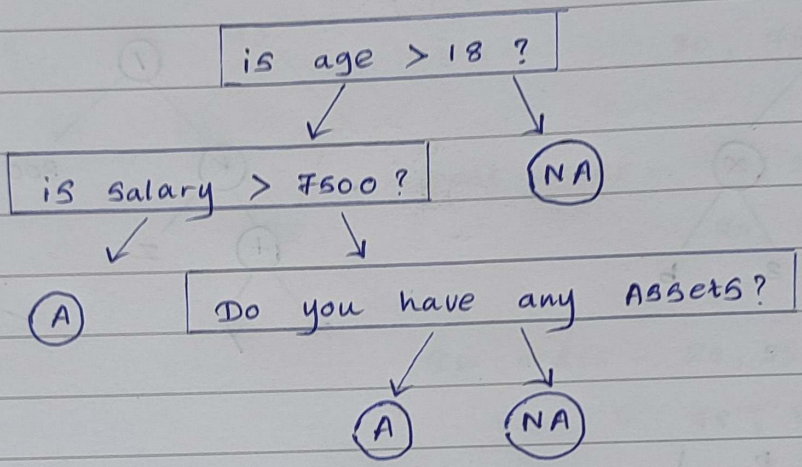
↓

External

1) Bank loan company ,

age > 18

Salary > ₹500 or Asset.



NA - Not Allowed

A - Approve