Introduction to Data structure.

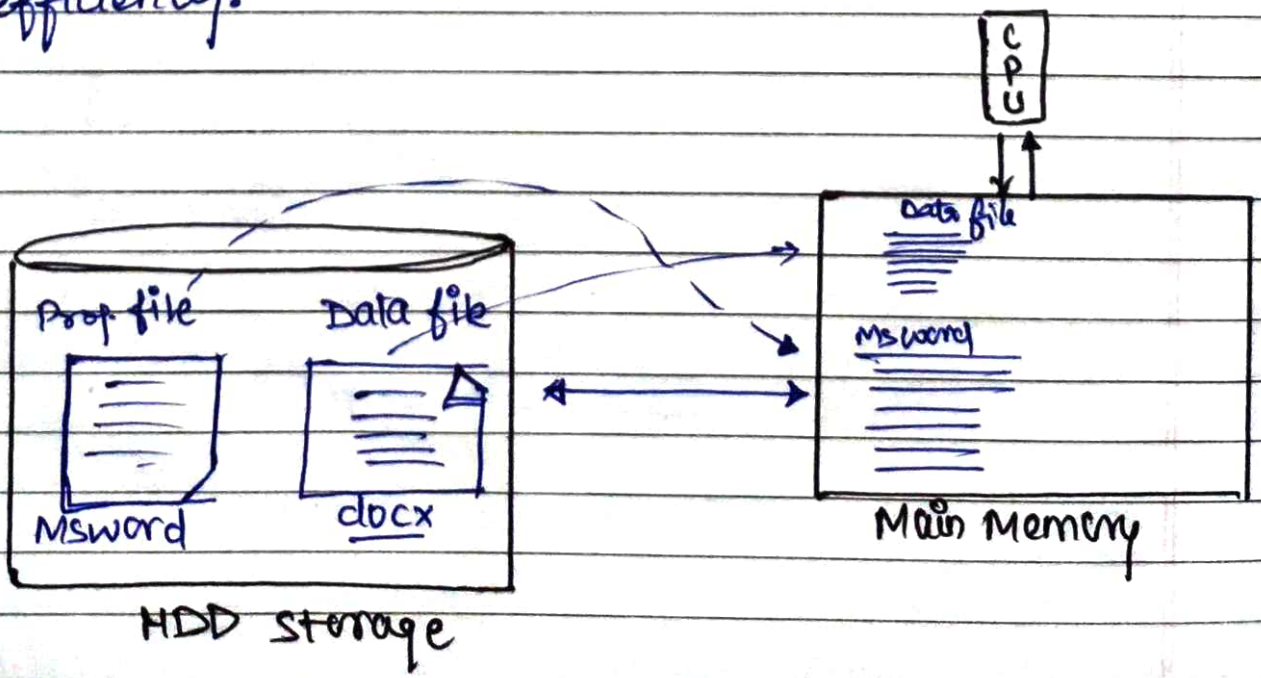
Data is an integral part of our application or programs.

A program is nothing, but set of instructions which perform operations on data to get some results.

(without Data means ~~to~~ no instruction)

Data structure :-

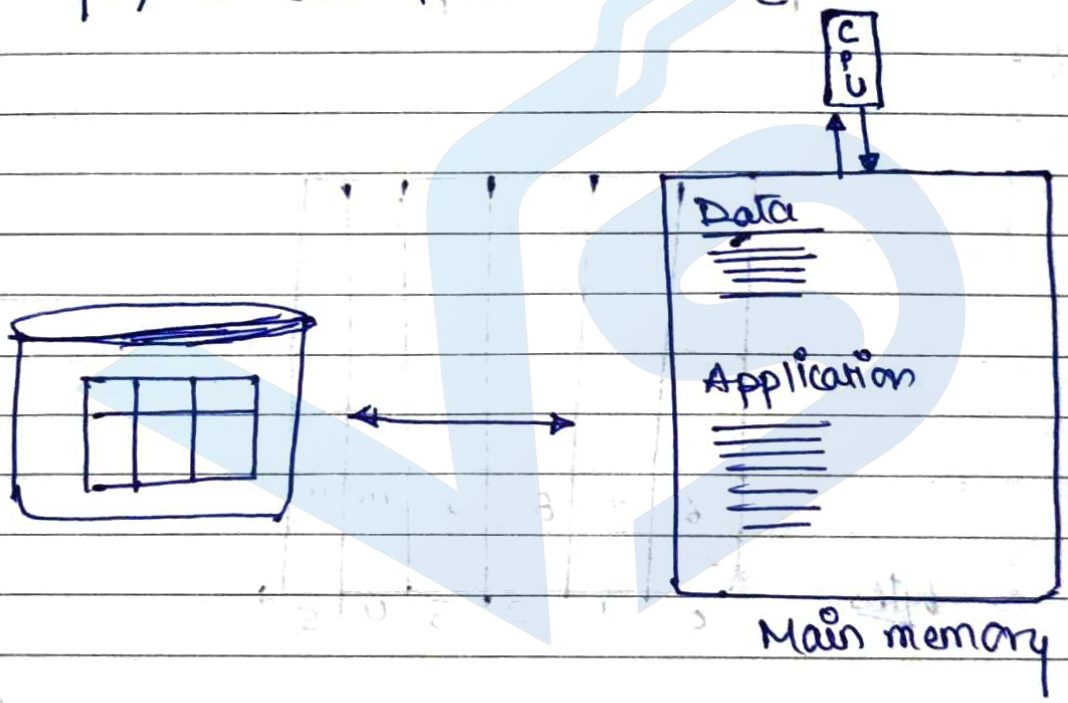
Data structures can be defined as arrangement of collection of data items, so that they can be utilized efficiently, operations on the data can be done efficiently.



Notebook
Pg. No.
Date / /

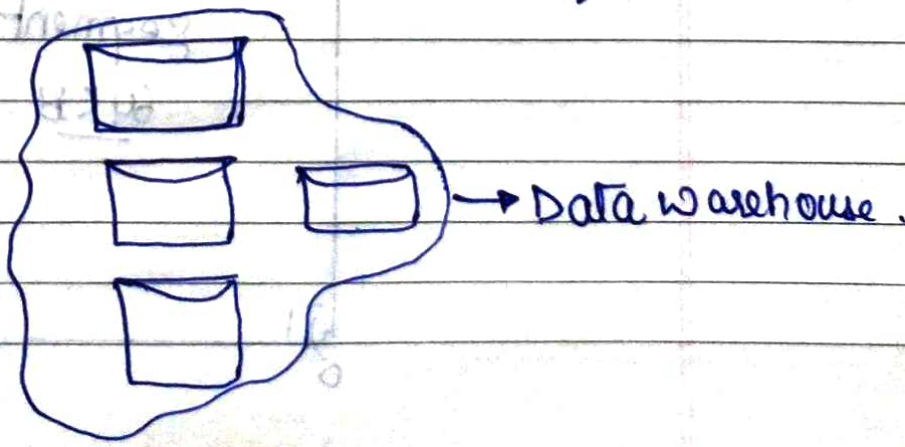
Database :-

A database means arranging the data in some model like ~~rational~~ relational model in the permanent storage, so that it can be retrieved or to accessed by application easily. that arrangement in the hard-disk, or in the permanent storage, it's called as database.



Data-warehouse :-

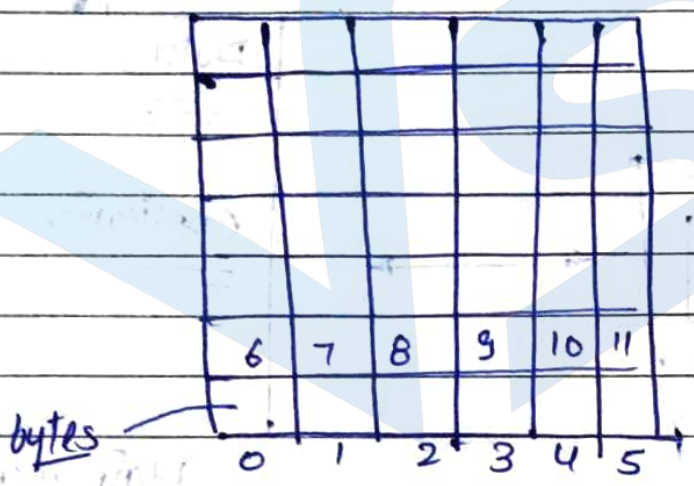
Data warehouse is a central repository of information that can be analysed to make more informed decisions.



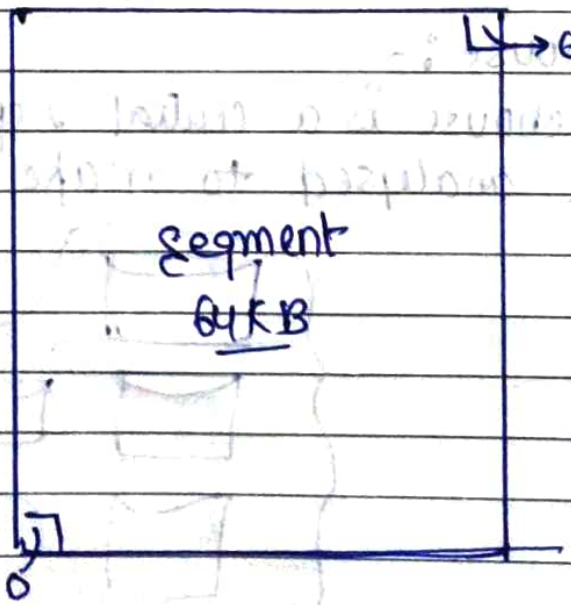
Notebook
Pg. No.
Date / /

Stack vs Heap Memory

Static vs Dynamic Memory Allocation



Address are linear



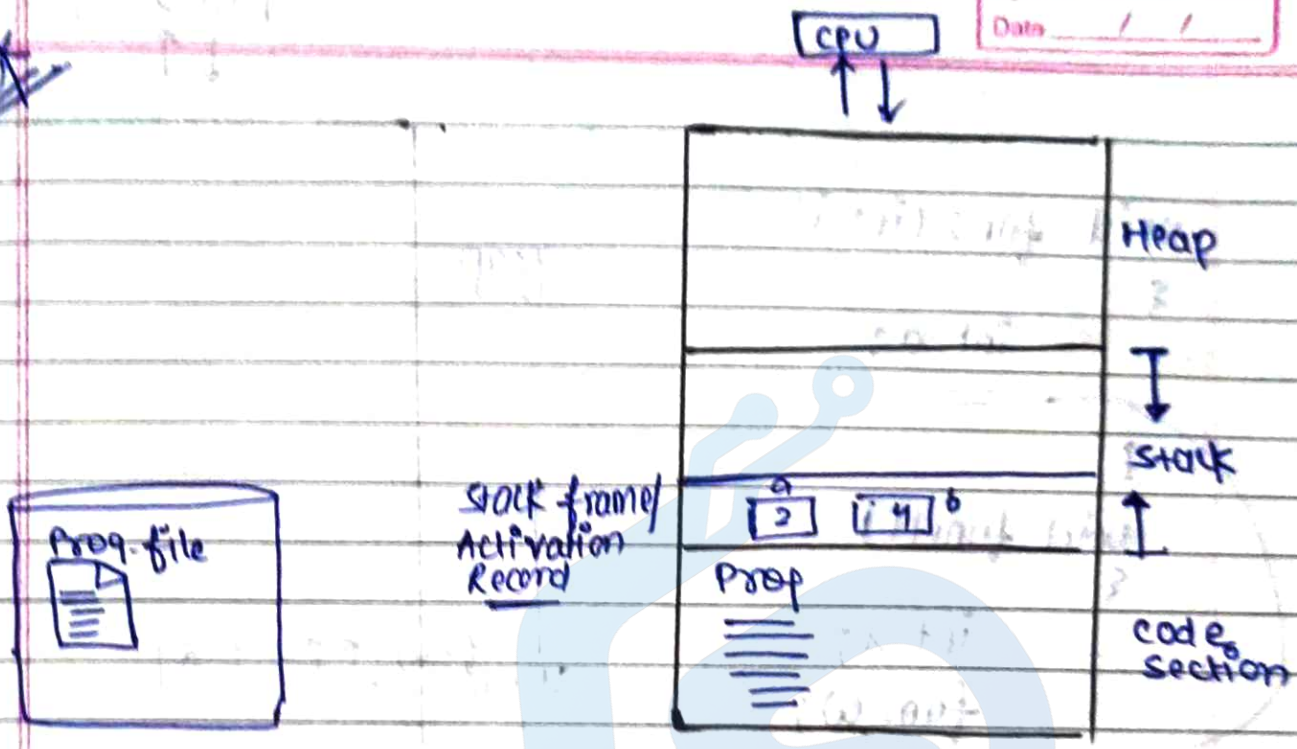
0 – 65535 ⇒ 65536

= 64 × 1024

= 64KB

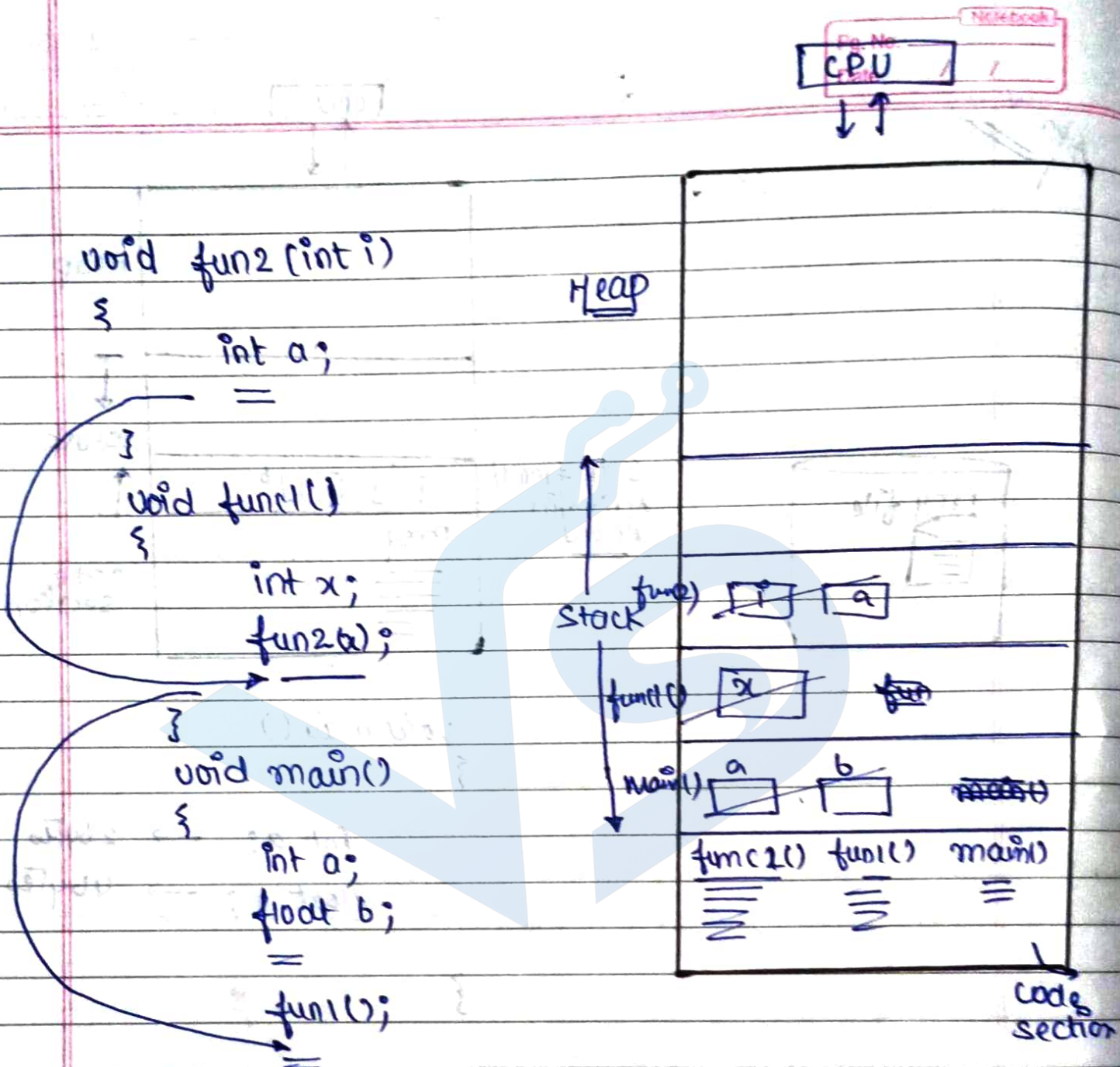
Notebook
Pg. No.
Date / /

Stack




```
void main()
{
    int a;   → 2 bytes
    float b; → 4 bytes
}
```

```
void fun2(int i)
{
    int a;
    =
}
void fun1()
{
    int x;
    fun2(x);
    =
}
void main()
{
    int a;
    float b;
    =
    fun1();
    =
}
```



Dynamic

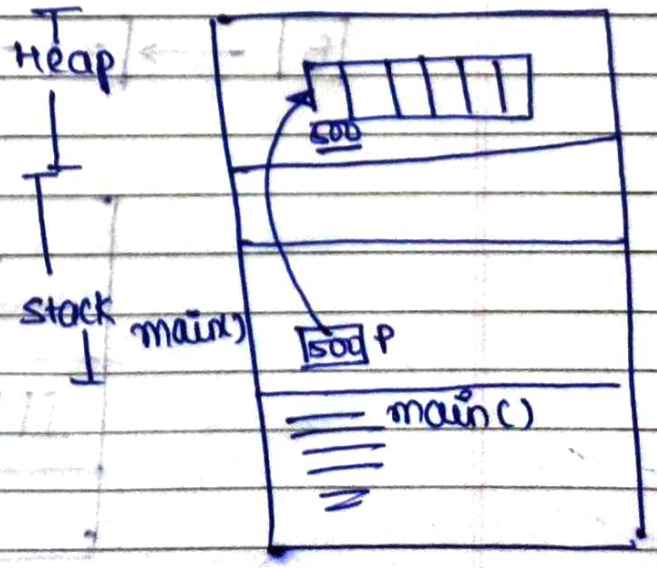
Word Heap is used for → Organised thing
→ Unorganised thing

→ It is treated like as resource.

Heap memory is unorganised.
Stack memory is organised.

Program cannot directly access the Heap memory.
→ they can access Heap memory using pointers.

```
void main()
{
    int *p; → 2bytes
    p = new int[5];
    p = (int*) malloc(2*5);
    delete[] p or free(p);
    p = null;
}
```

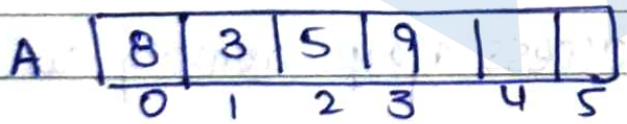


Physical vs Logical Data Structure

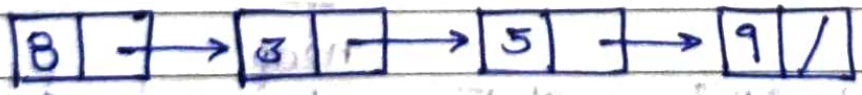
Types of Data Structure

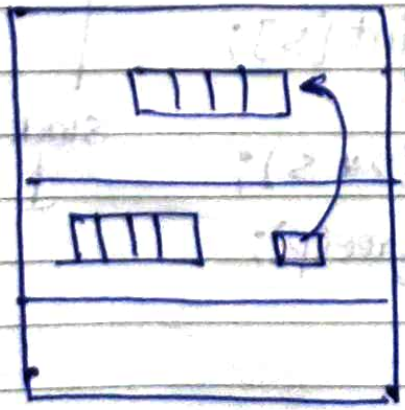
- Physical
  1. Array
  2. Linked list
- Logical → 1. stack, 2. Queues → linear
  3. Trees, 4. Graphs → non-linear
  5. Hash Table → Tabular

Array (size of the list is fixed)



Linked list (size of the list is not known)



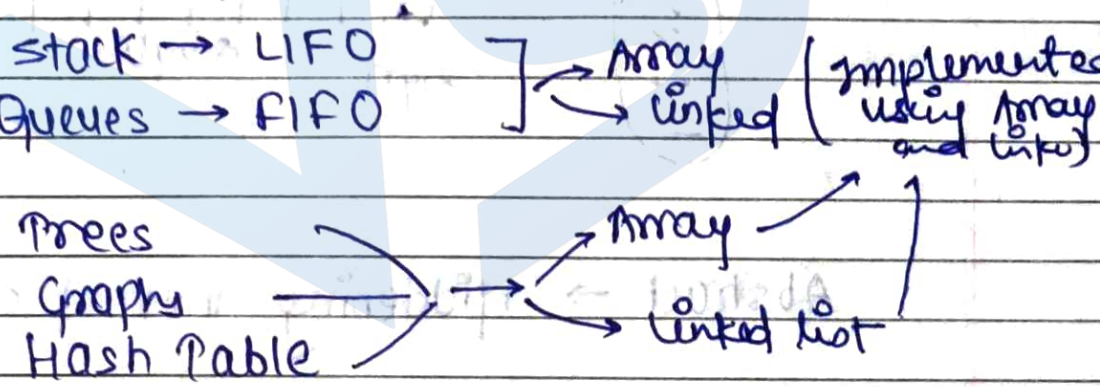


Notebook
Pg. No. _____
Date ___/___/___

linked is always created in Heap.

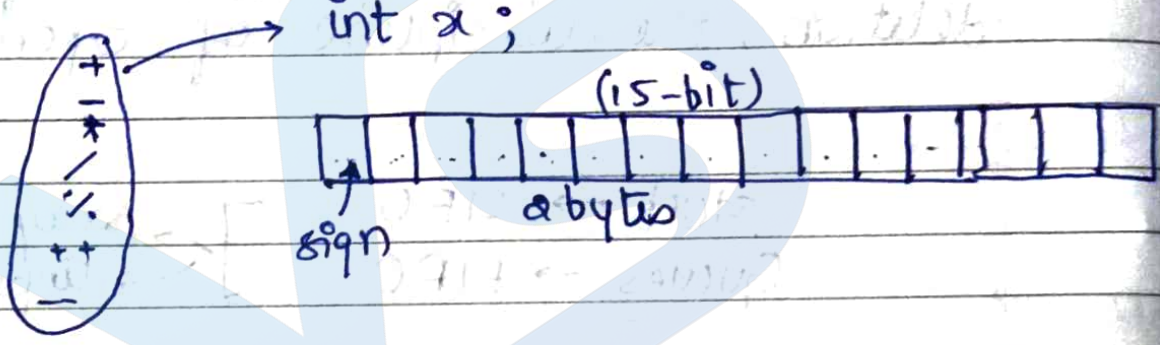
Physical data structure actually used to store the data in the memory.

Logical data structure are used for searching, deletion, the discipline of operation.



ADT (Abstract Data type)

Data Type
→ Representation of Data
→ Operation on Data



Abstract → Hiddening internal operation

Abstract Data-Type
→ Used in opps
→ Hiddening internal operation

List → 8, 3, 9, 4, 6, 10, 12
0 1 2 3 4 5 6

1. Array
2. Linked list

Data :
→ 1. space for storing element
2. Capacity
3. size

Operations:-

- add(x)
- remove(x)
- search(key)
- ⋮

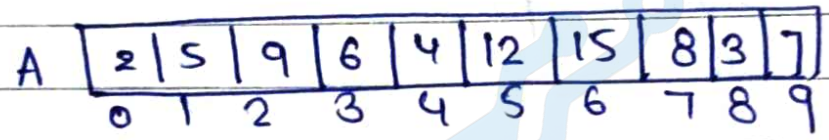
Abstract data-type is defined the data and operation on data, together is used as data-type, for hidding the internal details.

- add(element) / append(element)
- add(index, element) / Insert(index, element)
- remove(index)
- set(index, element) / Replace(index, element)
- get(index)
- search(key) / contains(key)
- sort()

Notebook
Pg. No.
Date / /

Time and space Complexity

Array :-



n → sum number of element

Time Complexity → O(n)

1) for(i=0; i<n; i++)
{
——
——
——
}

code is running n times.

2) for(i=0; i<n; i++)
{

for(j=0; j<n; j++) O(n²)
{
——
}

}

Notebook
Pg. No.
Date / /

3)

$1+2+3+4 - - - - + n-3 + n-2 + n-1$

$\Rightarrow \frac{(n)(n-1)}{2}$

$= \frac{n^2-n}{2}$

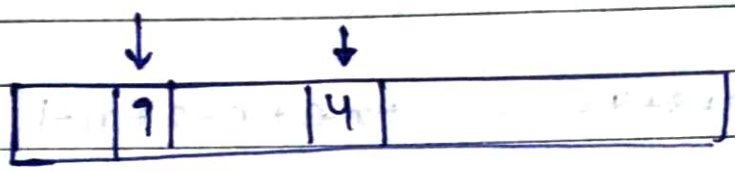
$= n^2$

$O(n^2)$

```
for(i=0; i<n; i++)
{
    for(j=i+1; j<n; j++)
    {
    }
}
```

$O(n^2)$

4)



Half of the list, then half of the list

$\log_2(n)$

—

For ($i = n$; $i > 1$; $i = i/2$)

{

==========

}

$\log_2(n)$ =

or

$i = n$

while ($i > 1$)

{

==========

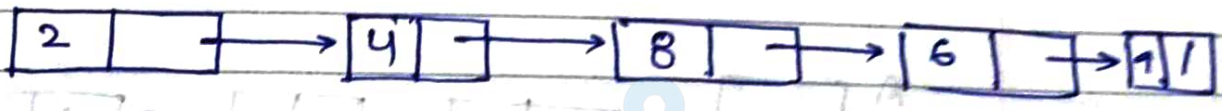
$i = i/2$;

}

→ $\log_2(n)$ =

Pg. No.
Date / /

Linked List :-



same as Array

Matrix :-

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

$O(n^2)$ → for all elements

$O(n)$ → For row (single)

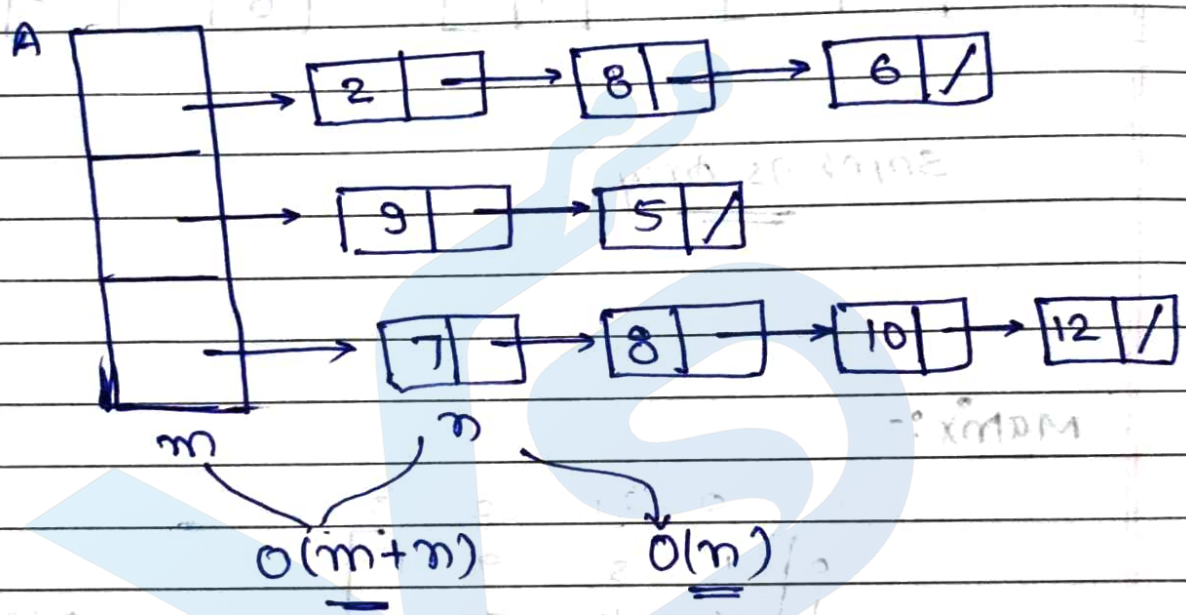
$O(n)$ → for column (single)

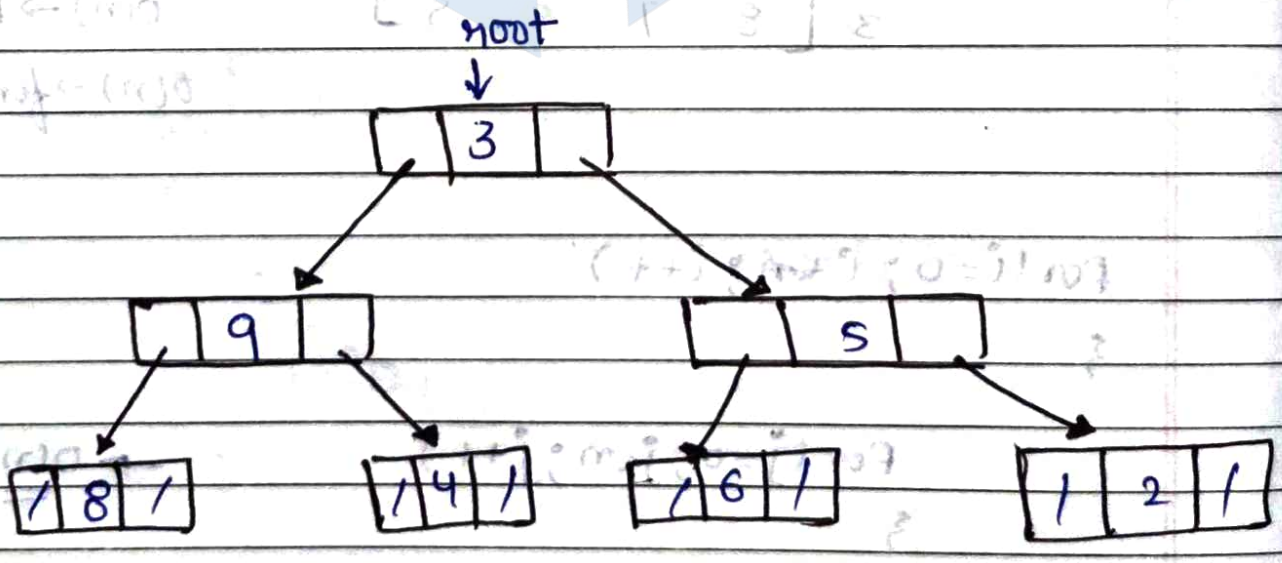
```
for(i=0; i<n; i++)
{
    for(j=0; j<n; j++)
    {

    }
}
```

→ $O(n^2)$

Notebook
Pg. No.
Date / /

Array of linked list :-





Along on path :- $O(\log_2 n)$
(Height)

~~Along~~ for all element → $O(n)$

Value of n more → space will be more

```
void swap(x, y)
{
    int t;      -
    t = x;      —1
    x = y;      —1
    y = t;      —1
}
```

$f(n) = 3n^0$

$\Rightarrow O(1)$

$\Rightarrow O(n^0)$

$= O(1)$

```
int sum(int A[], int n)
{
    int s, i;
    s = 0;                          —1
    for(i=0; i<n; i++)              → 2n+2 ⇒ 2(n+1)
                 1   n+1   n
    {                               → ~~O(n)~~
        s = s + A[i] - n
    }
    return s;                       —1
}
```

$\Rightarrow 1 + 2(n) + 1$

$= O(n)$

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

$= 2n + 3$

$O(n)$

```
void Add (int n)
{
    int i, j;                    ——— 1
    for (i=0; i<n; i++)          ——— n+1
    {
        For (j=0; j<n; j++)      ——— n(n+1)
        {
            c[i][j] = A[i][j] + B[i][j];   ——— n×n
        }
    }
}
```

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

$= 2n^2 + 2n + 1$

$O(n^2)$

Notebook
Pg. No.
Date / /

```
fun1()
{
    fun2();
}
fun2()
{
    for(i=0; i<n; i++)
    {
    }
}
```

fun2() → O(n)

for(i=0; i<n; i++) → n+1

for(i=0; i<n; i++) → O(n)