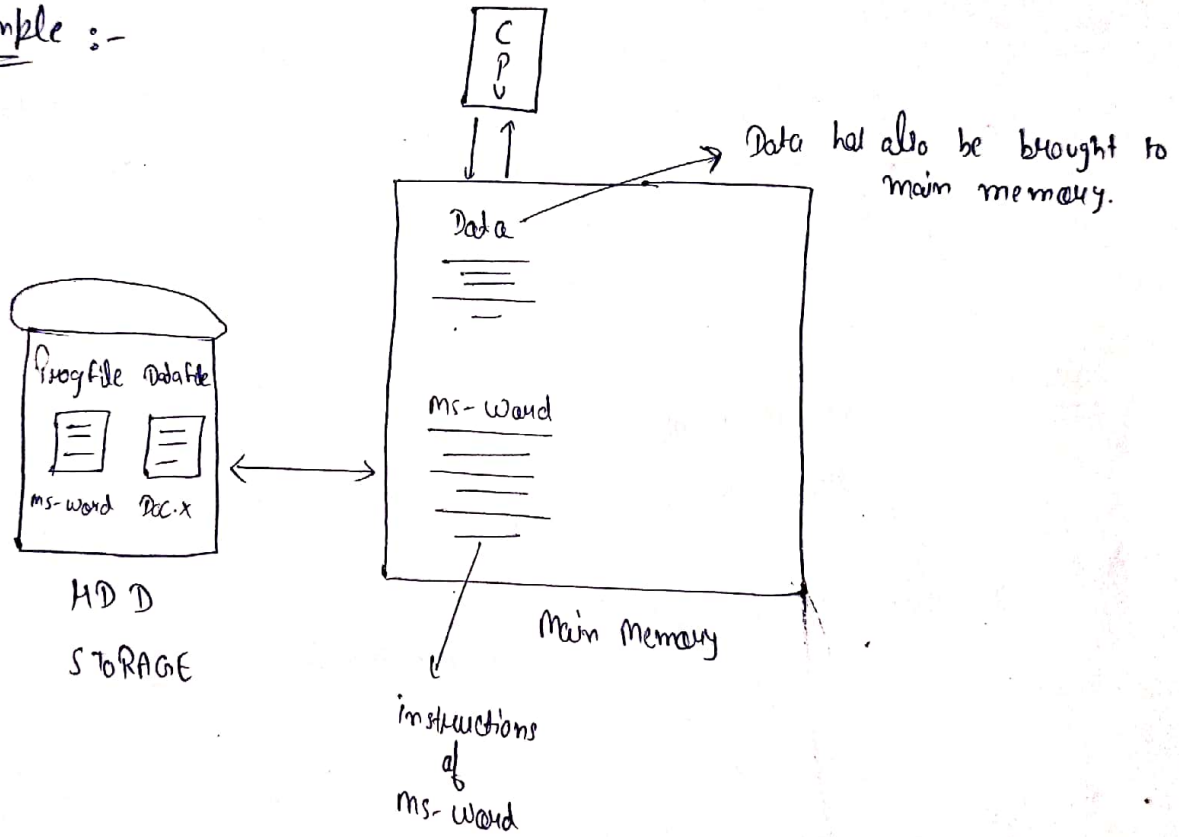
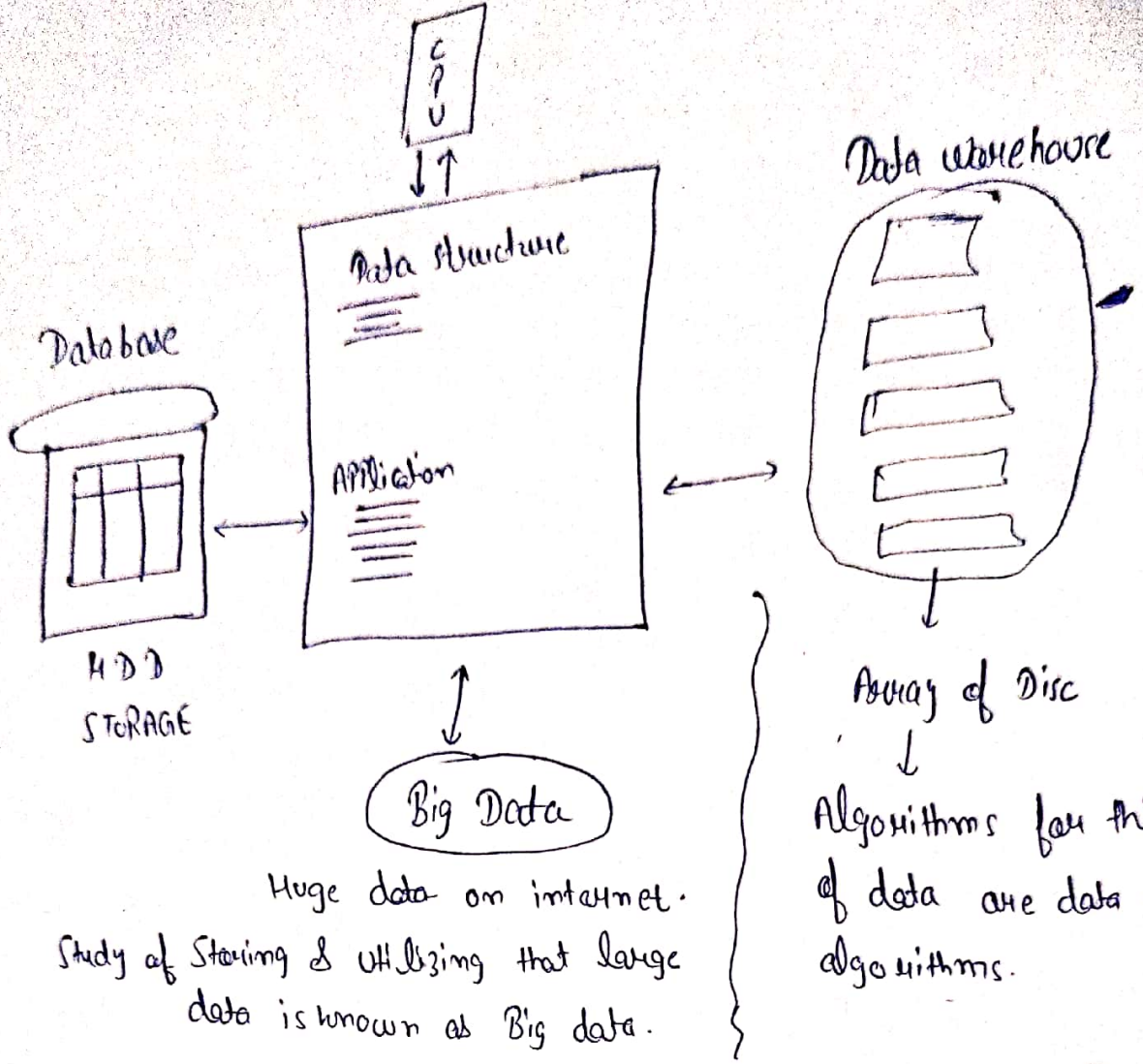
☆ Data Structures And Algorithms :- These are two Different things.

- **Data Structure :-** Arrangement of data so that they can be used efficiently in main memory. During execution of Program.
- **Algorithms :-** Sequence of steps on data using efficient data structures to solve a given problem.

Example :-



How you organize the data so that it will be efficiently used by the application is known as data structure. Data structures are formed in memory during execution of Program.



Huge data on internet. Study of Storing & Utilizing that large data is known as Big data.

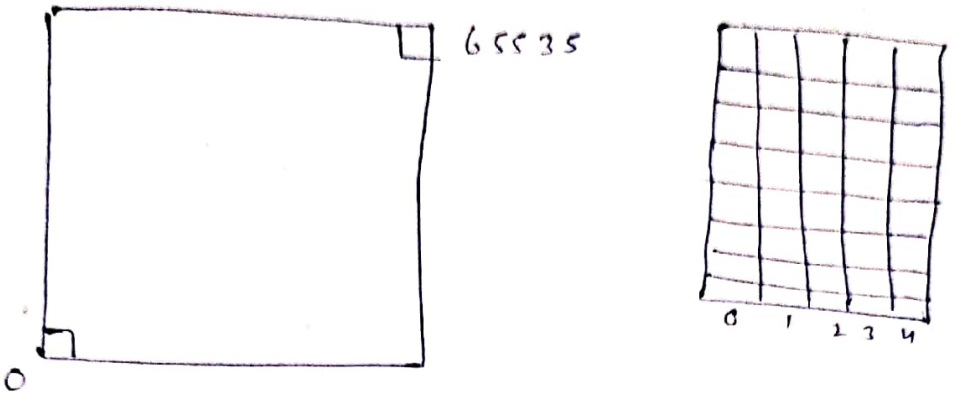
Algorithms for this legacy of data are data mining algorithms.

- **Data base** :- Collection of information in permanent storage for faster retreival and updation.

- **Data warehousing** :- Management of huge amount of legacy data for better Analysis.

- **Big Data** :- Analysis of too large or complex data which cannot be dealt with traditional Data processing application.

☆ Statics Vs. Dynamic Memory Allocation :-

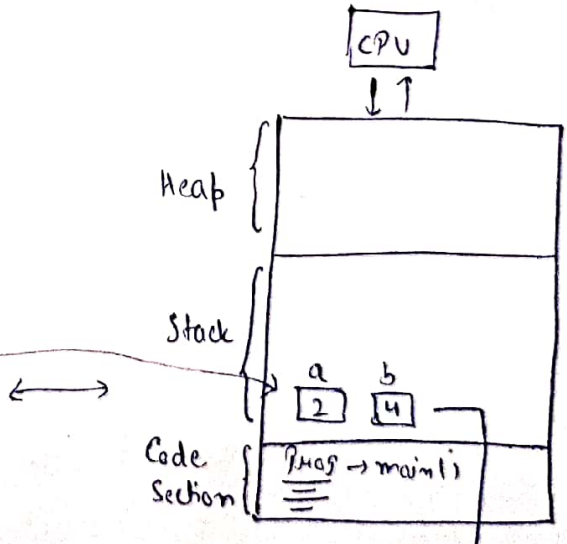


Assumption; Now, 0 to 65535 = 65536 Bytes = 64×1024 = 64 kB

Every byte is having its address, In our PC, memory is used as segments of having 64kB size usually.

• How a Program utilize main Memory :-

When the program starts, its code is copied to main memory. and program kept some area in memory known as Code Section.



Let us 'Assume'

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

Stack Frame / Activation Record of main function.

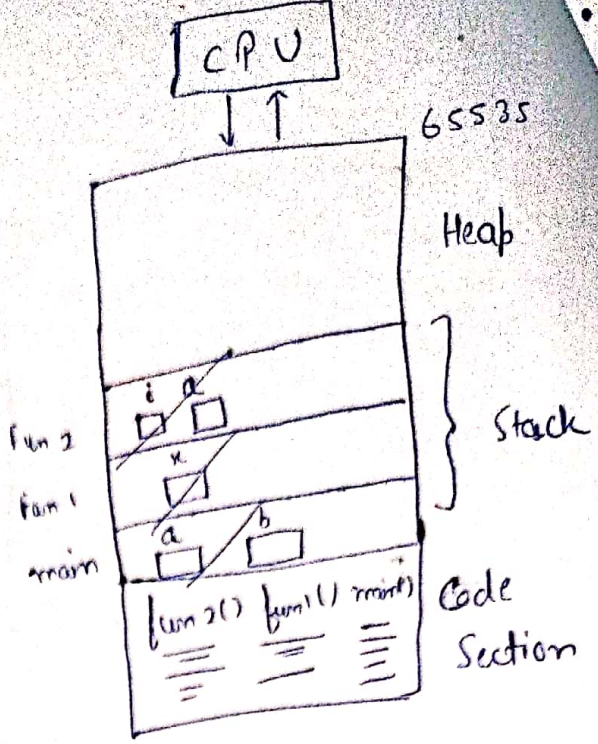
How many memory is reserved by this function was decided during compiling time. So it is static memory Allocation, As the size of memory is static.

If there are sequence of function calls then how the memory is allocated is shown.

```
Void fun2 (int i)
{
   int a;
   ==
}
```

```
Void fun1 ()
{ int x;
   fun2 (x);
}
```

```
Void main ()
{
   int a;
   float b;
   ==
   fun1 ();
}
```



This memory is automatically Allocated & destructed. Programmer just has to declare the variables or parameters in a function call.

- Stack holds the memory occupied by the functions.

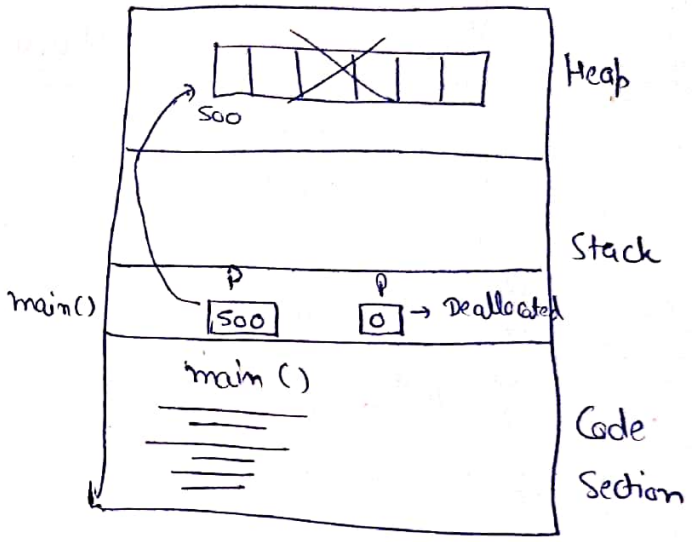
- Heap Contains the data which is requested by the program as dynamic memory. The Program does not Access heap directly but can be accesed using pointers.

```
Void main ( )
{
    int *p;
```

in C++ ⟶ `p = new int [5];`

in C ⟶ `p = (int *) malloc (2*5);`

```
    :
    :
    delete [ ] p;
    p = NULL;
}
```



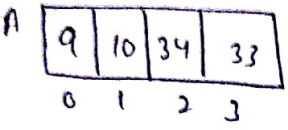
So this is how Heap memory Explicitly requested and Explicitly Disposed. if we don't release the memory then it can't be used again in the program and this will cause loss of memory known as Memory Leak.

★ Types of Data Structure

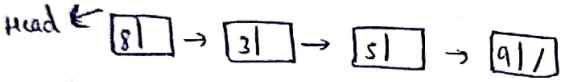
1. Physical Data Structure
2. Logical Data Structure

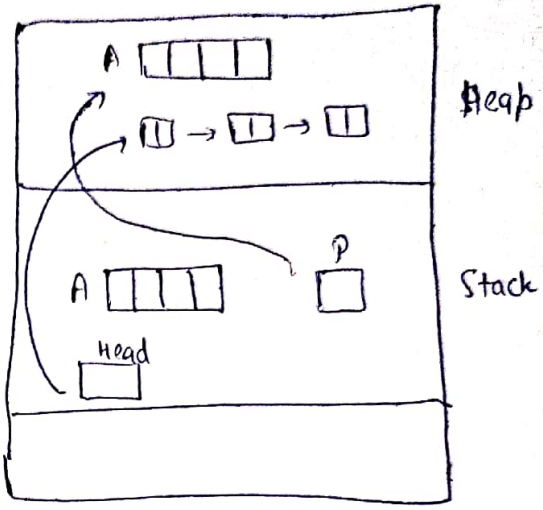
- **Physical Data Structures :-** They define how the memory should be organized for storing the data.

(i) Array



(ii) Linked List





Now, if we know the size of elements or length of elements then we can use Array. Array can be stored in stack memory as well as can be stored in Heap memory and can be accessed by using pointers.

Now, if we doesn't know about length of elements then we can use Linked list, where Linked List is a collection of nodes in which one node is connected to next. The length of this list can grow & reduce dynamically we can say it is having variable length. and this will always created in heap which can be accessed by pointer in stack.

Logical Data structures :-

These Logical Data structures are implemented using physical Data structures either Array or Linked List or Combination of Array and Linked List.

- Linear
  1. Stack
  2. Queues
- Non Linear
  3. Trees
  4. Graphs
- Tabular
  5. Hash Tables

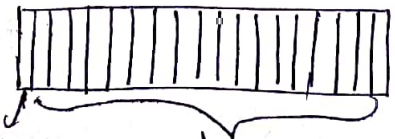
★ ADT (Abstract Data type) :-

Data type is defined as

(i) Representation of data

(ii) operation on data

Assume, int x; → 2 bytes



Now, operations can be done (i.e. arithmetic operations)

i.e. +, -, *, /, %, ++, --, etc.

Abstract data type is mainly used for C++ where we declare our own data types without knowing internal details we can use them thats why it is known as Abstract data type.

LIST ADT :-

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

Data :- 1) Space for storing element
2) Capacity
3) Size of list
→ for representing the list

Options for Representation :- 1) Array
2) Linked List

Operations :- add (x), remove ( ), Search ( ), etc.

Explanation :-

* add (element) / Append (element) :- Add something to the end of a list

* add (index, element) / Insert (element) :- Add at specific index by shifting.

* Remove (element) :- Removing from given index.

* Set (index, element) :- changing element at given index.
or
Replace (index, element)

* Get (index) :- Want to know Element at given index.

* Search (key) :- Searching element in List.
or
Contains (key)

* Sort ( ) :- To arrange in some order.

There are other more operations, we can perform on list.

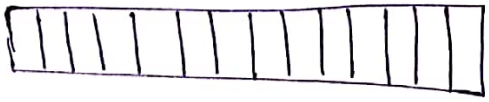
Time and Space Complexity :-

In a day to day routine, we do some tasks, we want to know how much time is reluired to do that task and we want now to do our tasks by machines / computers also, so we know how much time taken by machine.

"
Time Complexity is the study of efficiency of algorithms".
How much time taken to execute an algorithm grows with size of the input.

Now, Let us take some Examples :-

(i) Take an array



At most time that is taking, depends upon no. of elements, if there are n elements, so the time is 'n', and we represent as degree of n. $O(n)$. Now, if we want to access all elements. Code is :-

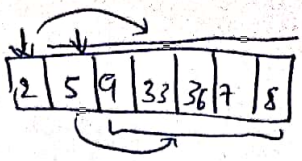
```
for (i=0 ; i<n; i++)
   {                    → O(n)
   } ==
```

- Now, for comparing or searching, than for each element n elements are processed. [i.e. n×n]

```
for (i=0 ; i<n ; i++)
 { for (j=0; j<n; j++)        → O(n²)
     {
 }   } ==
```

Time Complexity written in terms of degree of polynomial

- Now, suppose being on 1st, processing rest of element, it means for n-1 elements ---.

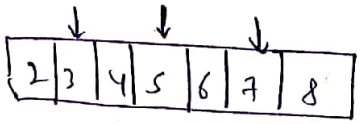


$1 + 2 + 3 + \cdots + n-3 + n-2 + n-1$

$= \frac{n(n-1)}{2} = \frac{n^2-n}{2} = n^2$

```
for (i=0 ; i<n ; i++)
  {  for(j=i+1 ; j<n; j++)
       {  ____                  → O(n²)
       }
  }
```

- Now, we suppose we are processing half of list then again on half & going on.



$\log_2 n$ → time Complexity

```
for (i= n; i>1 ; i= i/2)
  {  ____                  → log₂ n
  }
```

Note :- when something is divided successively until it reaches 1, that is represented by log

or

```
i=n;
while (i>1)
  {  ____
     i= i/2;
  }
```

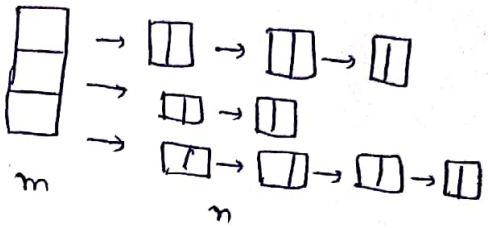
- For matrices $\begin{bmatrix} 8 & 3 & 5 & 9 \\ 7 & 6 & 4 & 2 \\ 6 & 5 & 3 & 9 \\ 10 & 4 & 2 & 6 \end{bmatrix} \rightarrow O(n^2)$

4×4

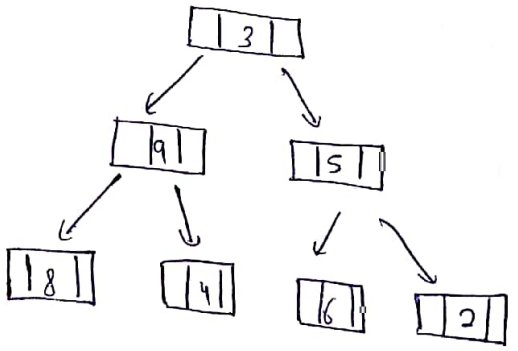
for row only / Column only $\rightarrow O(n)$

- For Array of Linked List

$O(m+n) \Leftarrow$



- For binary Tree



→ to process divide by 2 $\log_2(n)$

→ to process elements $O(n)$

→ Along height of tree $O(\log n)$

- Space Complexity :- It depends upon 'n', that is no. of elements; for an array it is $O(n)$.

For List :- Total elements $= 2n$, but $O(n)$ ← 

For matrices :- $O(n^2)$

For Array of Linked List :- $O(m+n)$

For binary tree :- $O(n)$, $O(\log n)$, $\log_2(n)$

☆ How to find Time Complexity from Code :-

Practice

(1)

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

$f(n) = 3$ [or $3n^0$]

$O(1)$

$O(n^0) = O(1)$

(2)

```
int Sum (int A[], int n)
{
    int s, i;
    s = 0;  — (1)
    for (i=0 ; i<n; i++)  — (n+1)
    {
        s = s + A[i];  — (n)
    }
    return s;  — (1)
}
```

$f(n) = 2n + 3$

$O(n)$

This runs for (n+1) bcz, it initializes from 0 & increment to n, and at last condition fails but it is going from 0 to n, ∴ (n+1) times

(3)

```
Void Add (int n)
{
  int i, j;
  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)
    }
  }
}
```

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

$\rightarrow O(n^2)$

$\rightarrow \Theta(n^2)$ $\rightarrow$ Big O

(4)

```
fun1 ()
{
  fun2 ();   — n
}
```

~~O(1)~~ O(n)

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