Data Structures and Algorithms

Data structure

- orgnising data inside memory for efficient processing along with some operations which we can perform on orgnised datas.

to achieve

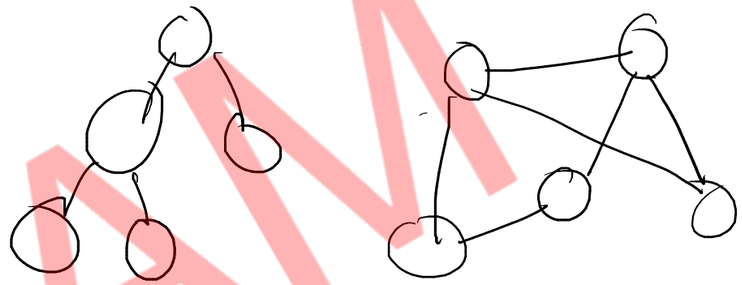
- **efficiency**
  - can be measured in two terms
    1) time
    2) space
- **Abstraction**
  - all DSs are ADTs (Abstract Data Type)
- **Reusability**
  - can be reused as per appln need
  - can be reused to implement another DSs
  - can be used in few algorithms (traversal)

Types of Data structrure

Linear Data structures

- Data is orgnised one after another
- data can be accessed linearly/sequentially
- Basic data structures
  1) Array
  2) struct/class
  3) stack
  4) Queue
  5) Linked List

Non-Linear Data structures



- Data is orgnised into multiple levels (Hierachy)
- data can not be accessed sequentially.
- Advanced data structures
  1) Tree
  2) Graph

Algorithm

program — set of instructions to the machine (CPU)

Algorithm - set of instructions to the human/developers/programmer

- step by step solution of given problem statement
- can be written in any human understandable language

e.g. find sum of array elements

step1 : create sum variable & initialise to 0
step2 : traverse array from 0 to N-1 index
step3 : add each element of array into sum
step4 : print/return sum variable

- algorithms are programming language independent
- algorithms are treated as templates/blue prints
- searching - linear & binary search
- sorting - selection, bubble, insertion, merge, quick

Searching Algorithms

- finding data(key) into collection of multiple data
- there are two types of searching algorithms
    1. Linear search (works on random data)
    2. Binary search (works on sorted data)

Linear search

```
//1.decide key to be searched
//2. start traversing from one end of collection
//3. compare key with each element of collection
//4. if key is found then stop searching and print/return the result
//5. if key is not found compare with next element of collection till
// last element
```

Binary search

```
//1. decide key to be searched
//2. find middle element of the array
//3. compare middle element with key
//3.1 if key is matching, then return index of it
//3.2 if key is less than middle element, then search it inside left partition
//3.3 if key is greater tha middle element, then search it inside right partition
//4. if key is not found, then return -1
```

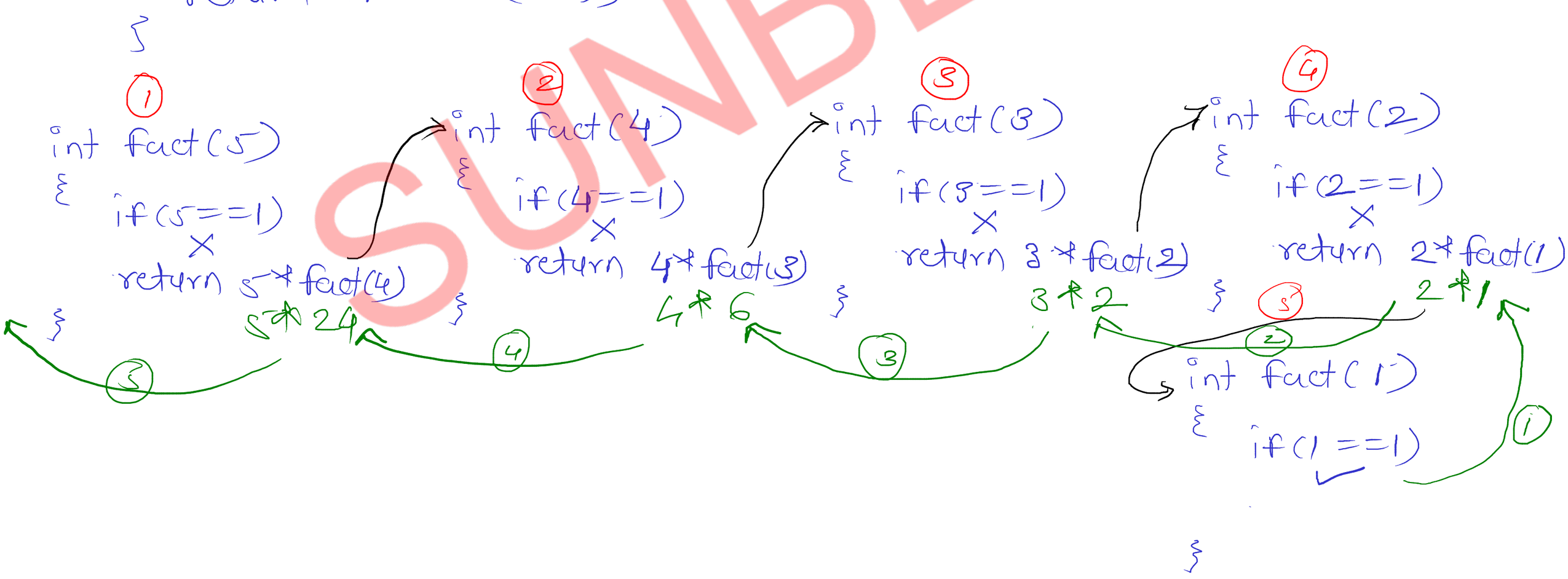
Recursion

- function calling itself
- we can use recursion if
    1. we know the process/formula in term of itself
    2. we know the terminating condition

$n! = n * (n-1)!$

$1! = 1$

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



Algorithm Implementation Approches

Any algorithm can be implemented using two approches

1. Iterative approach
- loops are used

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

2. Recursive approach
- recursion is used

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

Sorting Algorithms

- arrangement of data either in ascending or descending order of their values
- Basic sorting algorithms
    1. Selection sort
    2. Bubble sort
    3. Insertion sort
- Advanced sorting algorithms
    1. Merge sort
    2. Quick sort
    3. Heap sort

Selection sort

```
//1. select one position from array (0 to N-1)
//2. compare selected position element with all other elements one by one
//3. if selected position element is greater than other element, then swap both
//4. repeat above steps untill array is sorted
```

Bubble sort

//1. compare all consecutive elements of array one by one
//2. if left element is greater than right element, then swap both
//3. repeat above two steps untill array is sorted