# Sunbeam Institute of Information Technology

# Pune and Karad

## Module – Data Structures and Algorithms

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

# Data Structure

- organizing data inside memory for efficient processing along with operations like add, delete, search etc.

  e.g. stack - push/pop/peek.

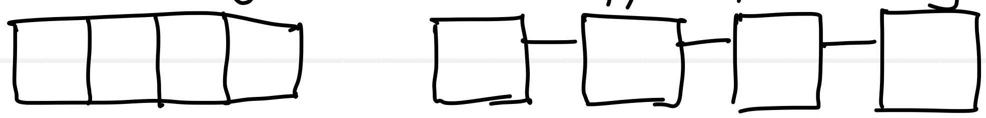
used to achieve :-

1) Abstraction
   - Abstract Data Types
2) Reusability
3) Efficiency
   - time (time required to execute)
   - space (space inside memory)

## Types

### Linear

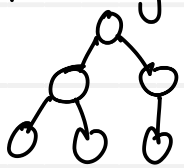
- data is organized linearly / sequentially.

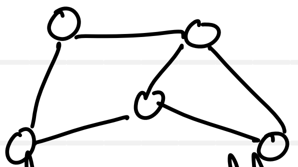


- data can be accessed sequentially.
- Basic data structures
  e.g. Array, structure/class, stack, Queue, Linked List

### Non Linear

- data is organized in multiple levels (heirachy)





- data can not be accessed sequentially.
- Advanced data structures
  e.g. Tree, Graph

# Algorithm

Program - set of instructions to machine (CPU)
Algorithm - set of instructions to human (developer)

- step by step solution of given problem statement

- written in human understandable languages.
- programming language independent
- templates / blue print.

Algorithm → Programs

e.g. searching, sorting

- Find sum of array elements.

1) Create sum & initialize to 0
2) traverse array from start to end
3) add every element of array in sum
4) print / return sum variable

# Searching Algorithms

## Linear search

- it works on random data

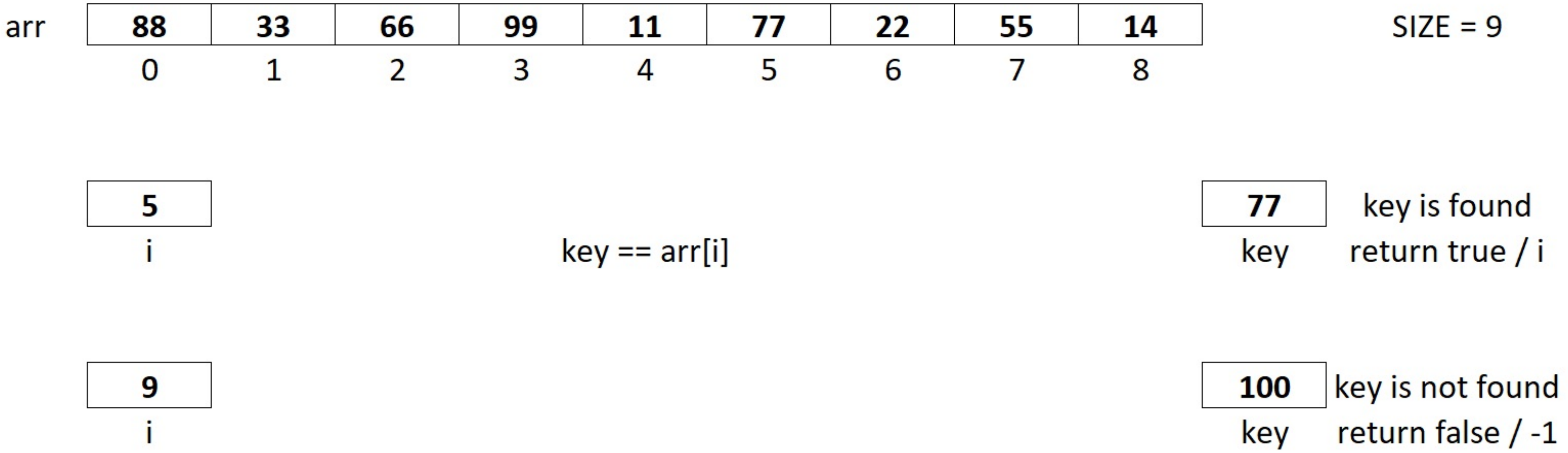
1) decide / take key from user
2) traverse collection (array) from one end to another.
3) compare key with each element of the array.
4) if key is matching, return true/index.
5) if key is not found, return false/-1.

## Binary Search

- it works on sorted data

1) Find middle element of array.
2) compare key with middle element.
3) if key is matching, return true/index.
4) if key is less than middle element, then search it in left side
5) if key is greater than middle element, then search it in right side
6) if key is not found, return false/-1

# Linear search



| arr | 88 | 33 | 66 | 99 | 11 | 77 | 22 | 55 | 14 |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

SIZE = 9

**5**
i

key == arr[i]

**77**
key

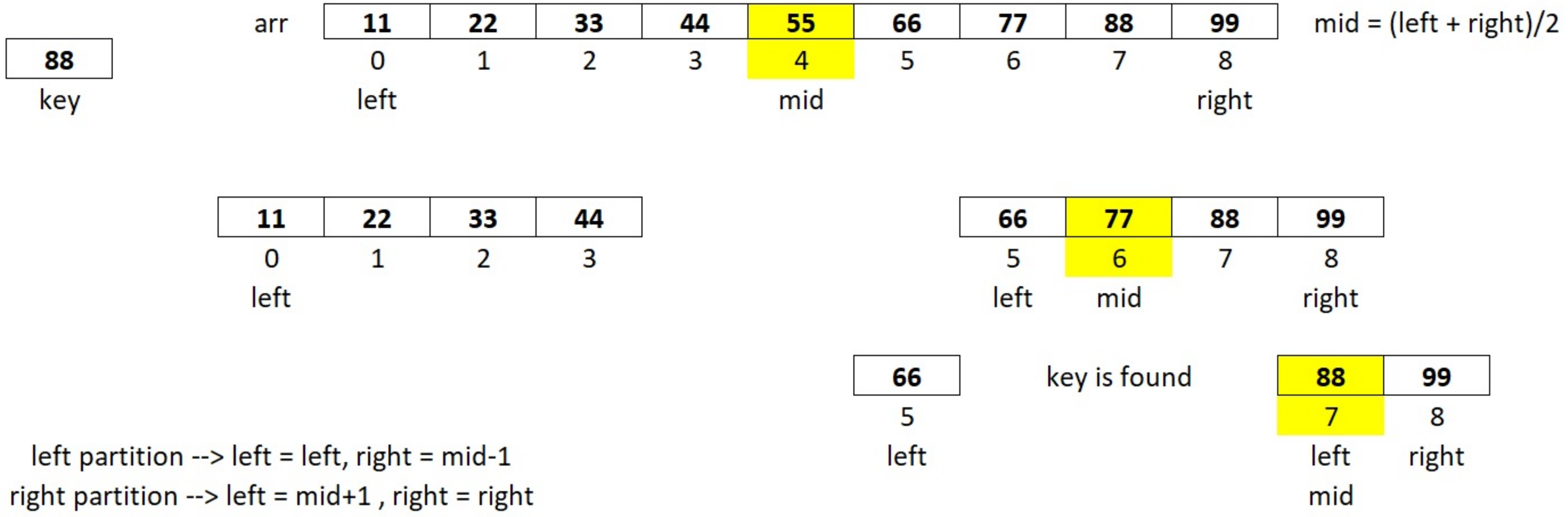
key is found
return true / i

**9**
i

**100**
key

key is not found
return false / -1

# Binary search



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | arr | 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99 |
| 88 | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| key | | left | | | | mid | | | | right |

mid = (left + right)/2

| | | | |
|---|---|---|---|
| 11 | 22 | 33 | 44 |
| 0 | 1 | 2 | 3 |
| left | | | |

| | | | |
|---|---|---|---|
| 66 | 77 | 88 | 99 |
| 5 | 6 | 7 | 8 |
| left | mid | | right |

| |
|---|
| 66 |
| 5 |
| left |

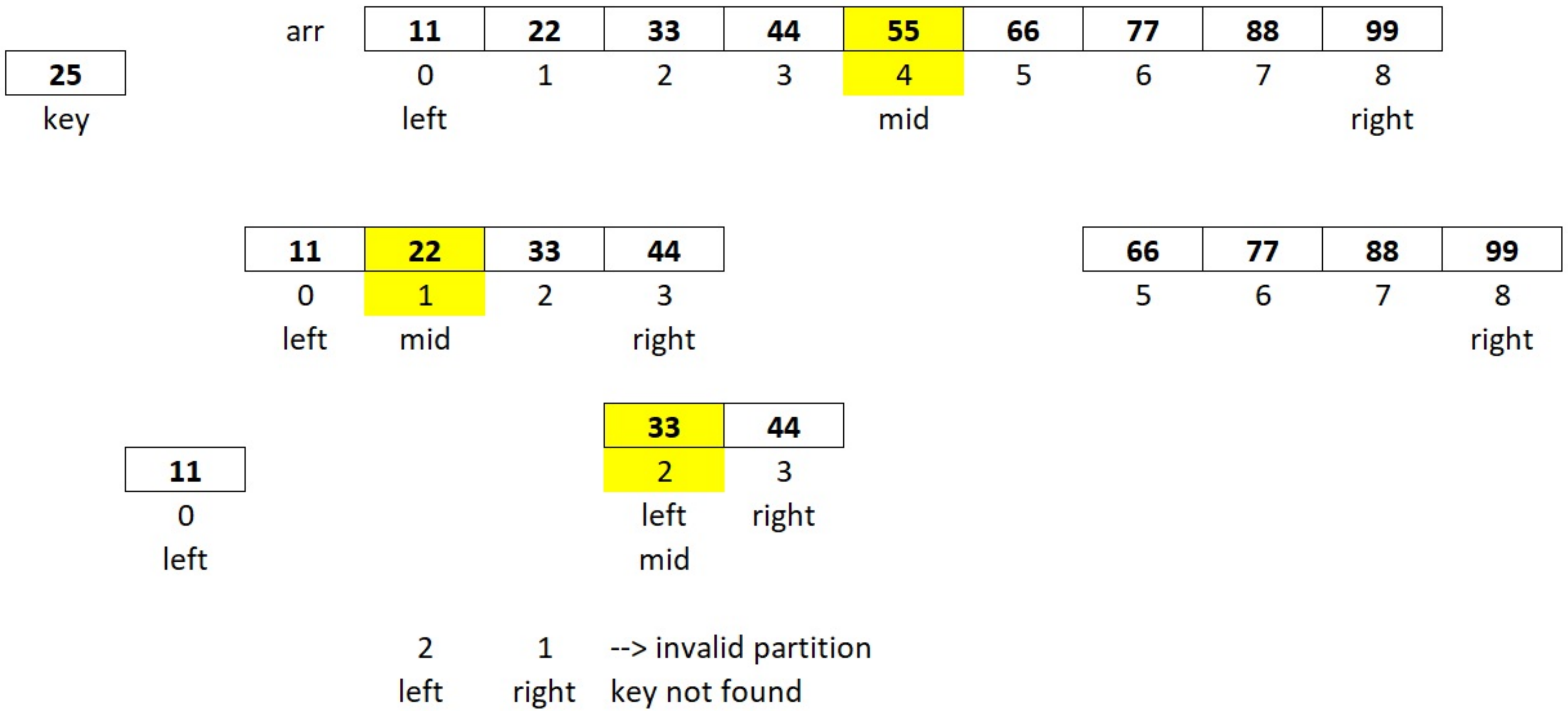
key is found

| | |
|---|---|
| 88 | 99 |
| 7 | 8 |
| left | right |
| mid | |

left partition --> left = left, right = mid-1

right partition --> left = mid+1 , right = right

# Binary search



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| arr | 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| | left | | | | mid | | | | right |

| |
|---|
| 25 |
| key |

| | | | |
|---|---|---|---|
| 11 | 22 | 33 | 44 |
| 0 | 1 | 2 | 3 |
| left | mid | | right |

| | | | |
|---|---|---|---|
| 66 | 77 | 88 | 99 |
| 5 | 6 | 7 | 8 |
| | | | right |

| |
|---|
| 11 |
| 0 |
| left |

| | |
|---|---|
| 33 | 44 |
| 2 | 3 |
| left<br>mid | right |

2 1 --> invalid partition

left right key not found

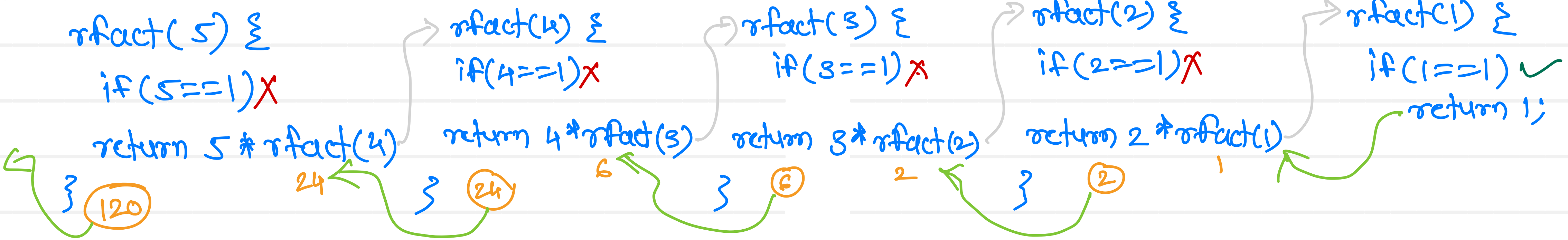
# Recursion

- calling function within itself
- we can use recursion
  - when we can define process/formula in terms of itself
  - when we know the terminating condition

```
int rfact(int num) {
  if(num==1)
     return 1;
 return num * rfact(num-1);
}
```

e.g. $n! = n*(n-1)!$

$0! = 1! = 1$



```
rfact(5) {            rfact(4) {            rfact(3) {            rfact(2) {            rfact(1) {
  if(5==1) ✗            if(4==1) ✗            if(3==1) ✗            if(2==1) ✗            if(1==1) ✓
  return 5*rfact(4)     return 4*rfact(3)     return 3*rfact(2)     return 2*rfact(1)      return 1;
}  120                }  24                 }  6                  }  2
```

# Algorithm implementation approaches

## Iterative

- loops are used

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

## Recursive

- recursion is used

```
int rfact(int num) {
  if(num ==1)
      return 1;
  return num * rfact(num-1);
}
```

SUNBEAM

# Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com