Sunbeam Institute of Information Technology

Pune and Karad

Module – Data Structures and Algorithms

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Data Structure

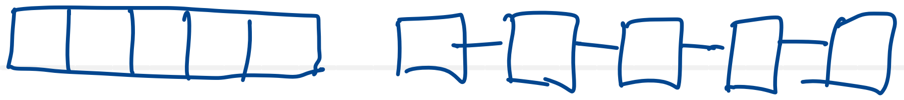
- organising data inside memory for efficient processing along with operations like add, delete, search, etc which can be performed on data.
- eg stack - push/pop/peek

- data structures are used to achieve
  - Abstraction
    (Abstract Data Types) (ADT)
  - Reusability
  - Efficiency
    - time : required to execute
    - space : required to execute

Types of data structures

Linear data structures
(Basic)

- data is organised sequentially/ linearly

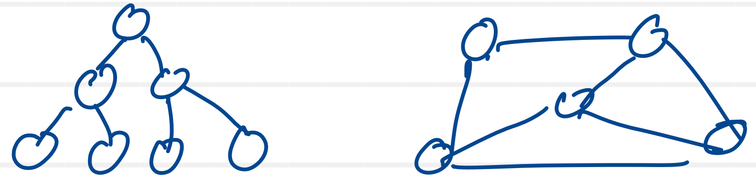


- data can be accessed sequentially

e.g. Array, structure/ class
stack, queue, Linked List

Non linear data structures
(Advanced)

- data is organised in multiple levels (hierarchy)



- data can not be accessed sequentially

e.g. Tree, Heap, Graph

Hash table

Algorithm

Program : set of rules / instructions to processor / CPU
Algorithm : Set of instructions to human (programmer)

- step by step solution of given problem

- Algorithms are programming language independent.
- Algorithms can be written in any human understandable language.
- Algorithms can be used as a templates

Algorithm ⟶ Program
(Template) (Implementation)

e.g. find sum of array elements
1. define sum & initialise to 0
2. traverse array from 0 to N-1 index
3. add each element into sum
4. print / return sum

e.g. searching , sorting
searching ↓ linear / binary
sorting ↓ selection / bubble / insertion

Linear search

1. decide/take key from user
2. traverse collection of data from one end to another
3. compare key with data of collection
   3.1 if key is matching
   return index/true
   3.2 if key is not matching
   return -1/false

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

key == arr[i]

77
Key

i = 0, 1, 2, 3, 4, 5
(Key is found)

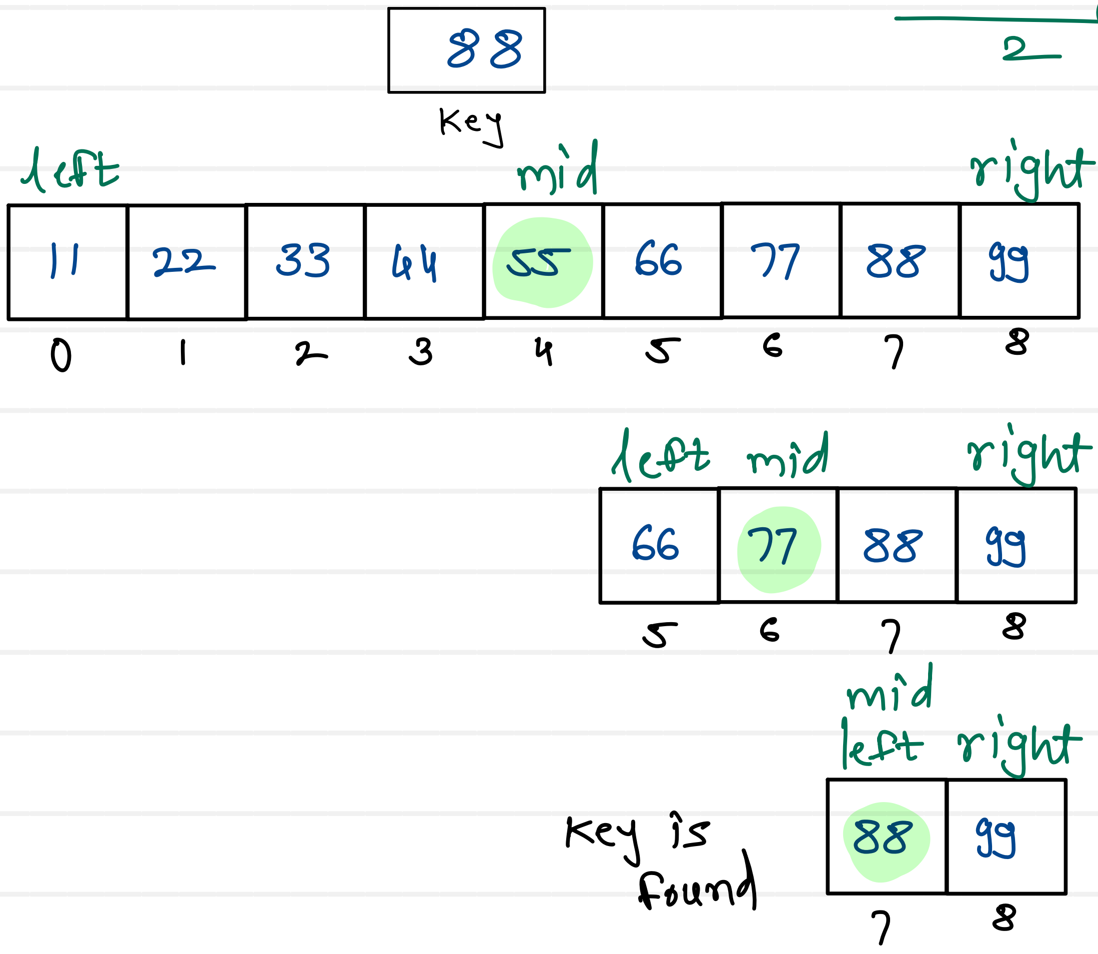
89
Key

i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
(Key is not found)

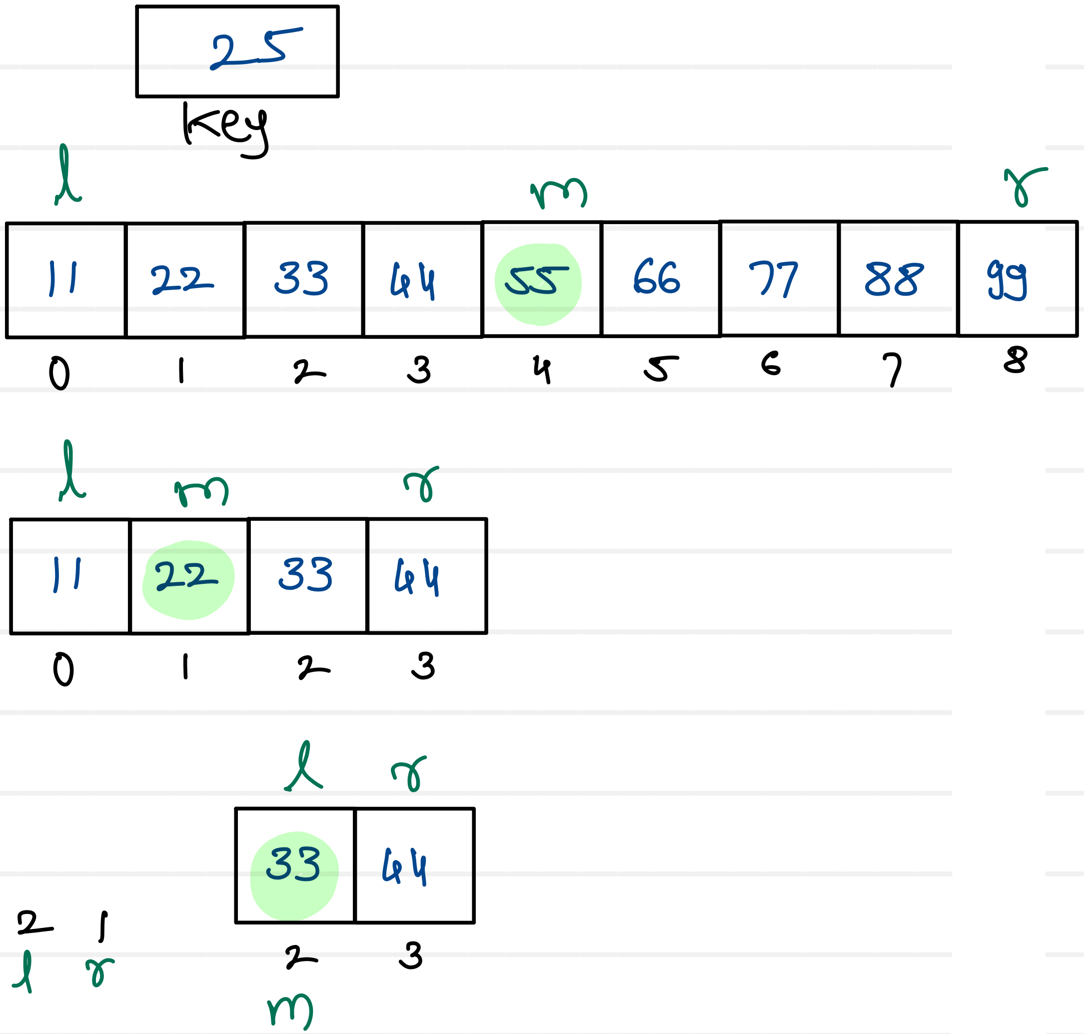
Binary search

1. take key from user
2. divide array into two parts
   (find middle element)
3. compare middle element with key
   3.1 if key is matching
       return index (mid)
   3.2 if key is less than middle element
       search key in left partition
   3.3 if key is greater than middle element
       search key in right partition
   3.4 if key is not matching
       return -1

$$mid = \frac{left + right}{2}$$



Binary search



left partition,
left = left
right = mid - 1

right partition,
left = mid + 1
right = right

valid partition : left <= right
invalid partition : left > right

```
l=0 , r=8 , m;
while(l <= r) {
    m=(l+r)/2;
    if( key == arr[m]
        return m;
    else if(key < arr[m])
        right= m-1;
    else
        left = m+1;
}
return -1;
```

11	22	33	44	55	66	77	88	99
0	1	2	3	4	5	6	7	8

key = 88

l	r	l<=r	m
0	8	T	4
5	8	T	6
7	8	T	7

key = 25

l	r	l<=r	m
0	8	T	4
0	3	T	1
2	3	T	2
2	1	F	

Recursion

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

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

$0! = 1! = 1$

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

24

① n=4

```
fact(4) {
    if(4==1) ✗
    return 4 * fact(3)
}
```

6

②

```
fact(3) {
    if( 3==1) ✗
    return 3 * fact(2)
```

2

③

```
fact(2) {
    if(2==1) ✗
    return 2 * fact(1)
}
```

④

```
fact(1)
    if(1==1) ✓
        return 1;
```

① ② ③ ④

Algorithm implementation approaches

Iterative

loops are used

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

Recursive

recursion is used

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

SUNBEAM

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com