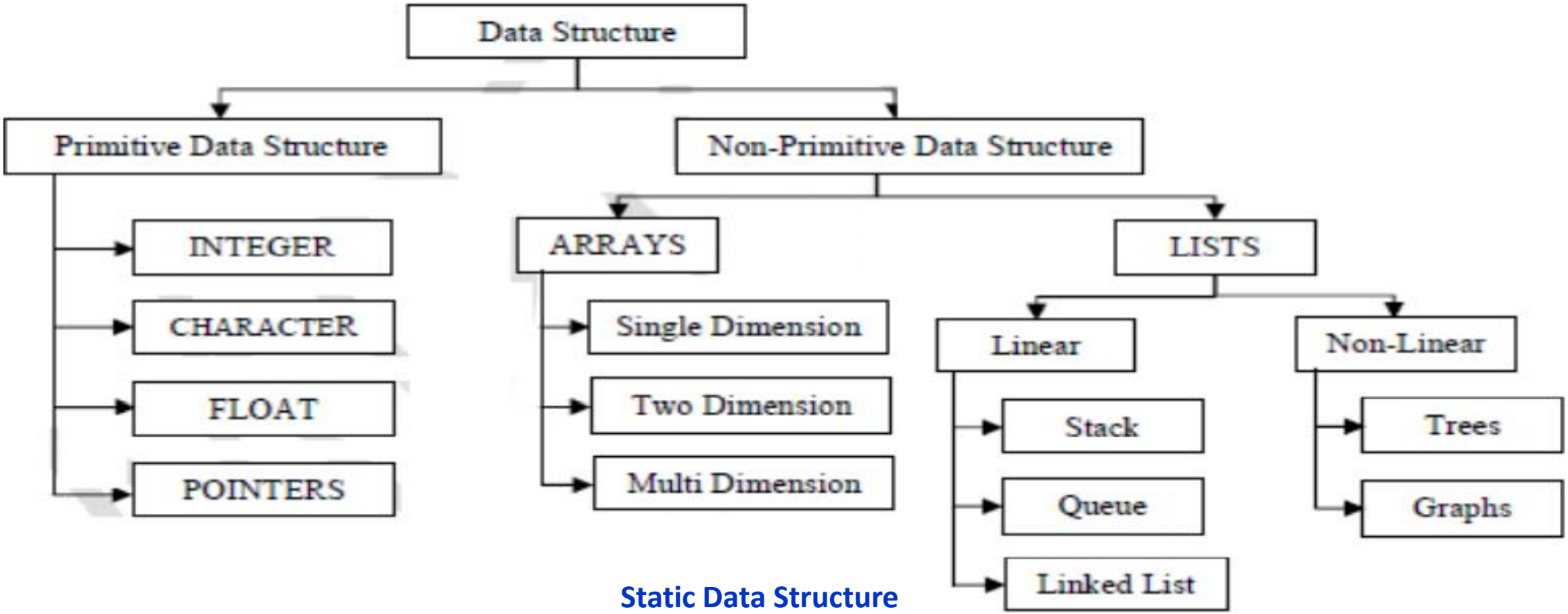
Algorithms & Data Structure

Kiran Waghmare

Classification of Data Structure



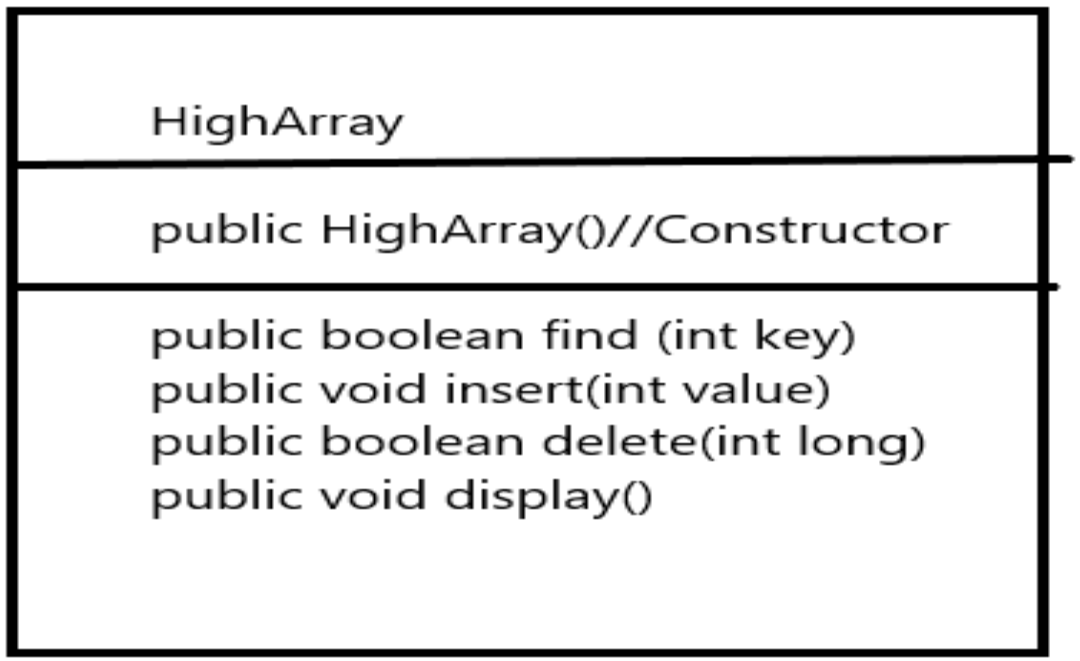
Static Data Structure

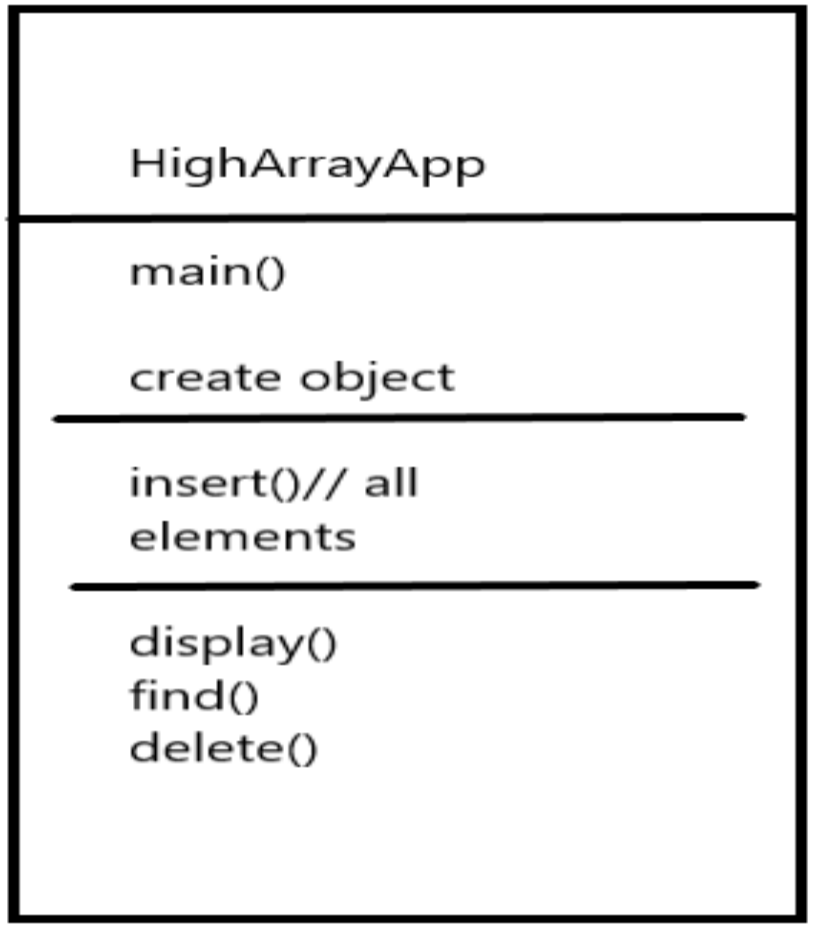
Dynamic data structure

Non-Primitive Data Structure

- The most commonly used operation on data structure are broadly categorized into following types:
  - Traversal
  - Insertion
  - Selection
  - Searching
  - Sorting
  - Merging
  - Destroy or Delete

Home work


HighArray
public HighArray()//Constructor
public boolean find (int key)
public void insert(int value)
public boolean delete(int long)
public void display()


HighArrayApp
main()
create object
insert()// all elements
display()
find()
delete()

Algorithm Complexity:-

Two main factors, which decide the efficiency of X.

- **Time Factor** – Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.
- **Space Factor** – Space is measured by counting the maximum memory space required by the algorithm.

Asymptotic Notations

Asymptotic analysis of an algorithm refers to defining the mathematical boundation of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Usually, the time required by an algorithm falls under three types –

- **Best Case** – Minimum time required for program execution.
- **Average Case** – Average time required for program execution.
- **Worst Case** – Maximum time required for program execution.

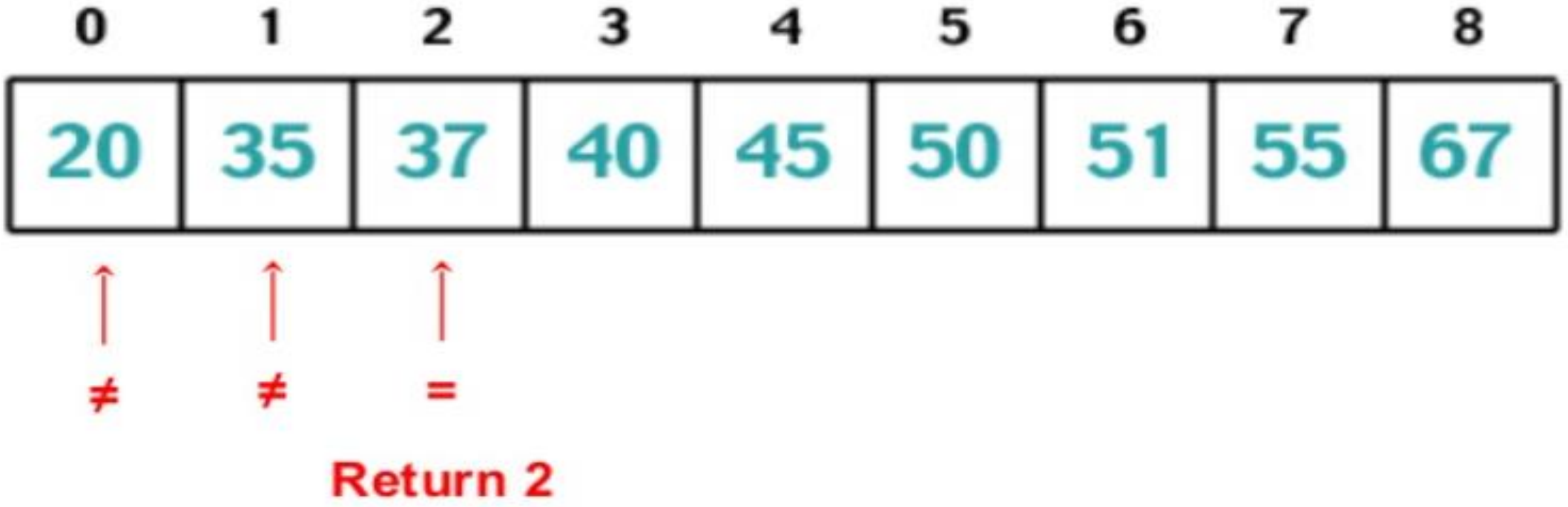
Asymptotic Notations

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

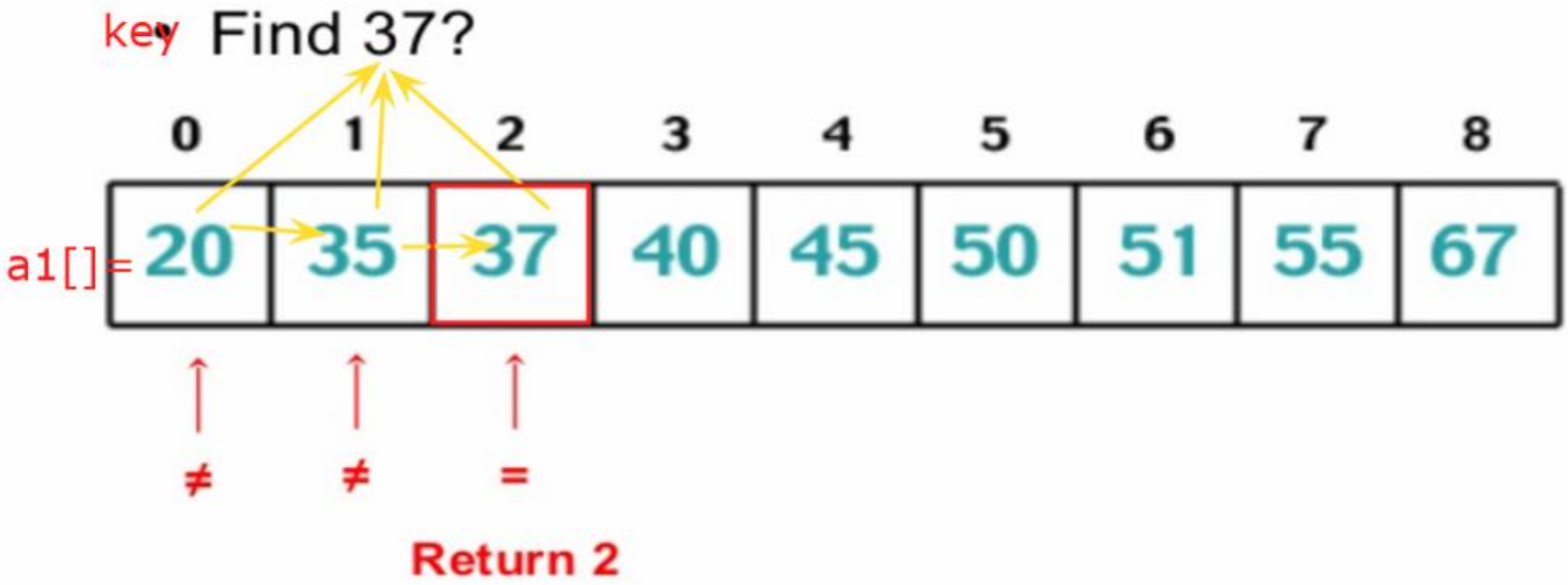
- **O Notation**
- **Ω Notation**
- **θ Notation**

Linear Search

- Find 37?



Linear Search



Linear Search

Algorithm

Consider **LA** is a linear array with **N** elements and **K** is a positive integer such that **K<=N**. Following is the algorithm to find an element with a value of ITEM using sequential search.

```
1. Start
2. Set J = 0
3. Repeat steps 4 and 5 while J < N
4. IF LA[J] is equal ITEM THEN GOTO STEP 6
5. Set J = J +1
6. PRINT J, ITEM
7. Stop
```

Home work

Problem: Given an array arr[] of n elements, write a function to search a given element x in arr[].

Examples :

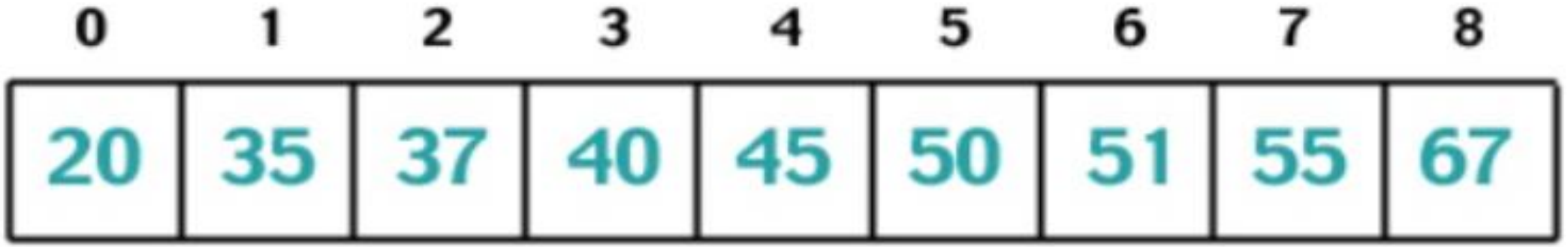
Input : arr[] = {10, 20, 80, 30, 60, 50, 110, 100, 130, 170}
x = 110;
Output : 6
Element x is present at index 6

Input : arr[] = {10, 20, 80, 30, 60, 50, 110, 100, 130, 170}
x = 175;
Output : -1

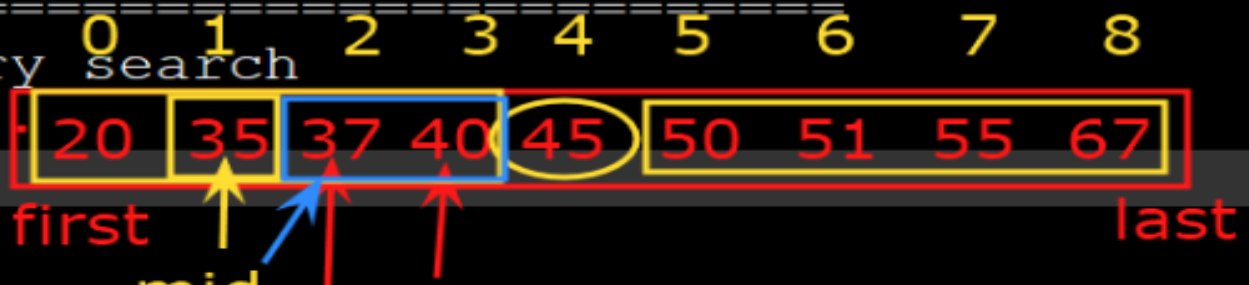
Element x is not present in arr[].

Binary Search

- Find 37?
  1. Sort Array.



Binary search



mid=(low+high)/2

=(0+8)/2=4

mid =35

mid=37

key = 37

Iteration 1:

if(a1[mid]==key)

return mid

else

if key>mid-->high

else

if key<mid-->low

-Array Applications
-Stack
-Stack Applications

Binary Search

```
Procedure binary_search
    A ← sorted array
    n ← size of array
    x ← value to be searched

    Set lowerBound = 1
    Set upperBound = n

    while x not found
        if upperBound < lowerBound
            EXIT: x does not exists.

        set midPoint = lowerBound + ( upperBound - lowerBound ) / 2

        if A[midPoint] < x
            set lowerBound = midPoint + 1

        if A[midPoint] > x
            set upperBound = midPoint - 1

        if A[midPoint] = x
            EXIT: x found at location midPoint
    end while

end procedure
```

Problem Statement 1: Find the Missing Number

You are given a list of n-1 integers and these integers are in the range of 1 to n. There are no duplicates in the list. One of the integers is missing in the list. Write an efficient code to find the missing integer.
Example:

Input: arr[] = {1, 2, 4, 6, 3, 7, 8}
Output: 5
Explanation: The missing number from 1 to 8 is 5

Input: arr[] = {1, 2, 3, 5}
Output: 4
Explanation: The missing number from 1 to 5 is 4

Problem statement: Search an element in a sorted and rotated array

Example:

Input : arr[] = {5, 6, 7, 8, 9, 10, 1, 2, 3};
key = 3
Output : Found at index 8

Input : arr[] = {5, 6, 7, 8, 9, 10, 1, 2, 3};
key = 30
Output : Not found

Input : arr[] = {30, 40, 50, 10, 20}
key = 10
Output : Found at index 3

Problem statement 3: Program for array rotation

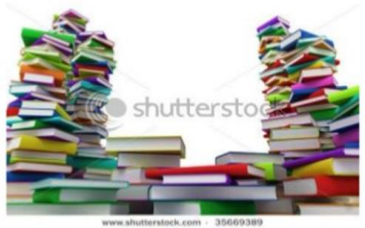
Write a function rotate(ar[], d, n) that rotates arr[] of size n by d elements.

Array

Rotation of the above array by 2 will make array

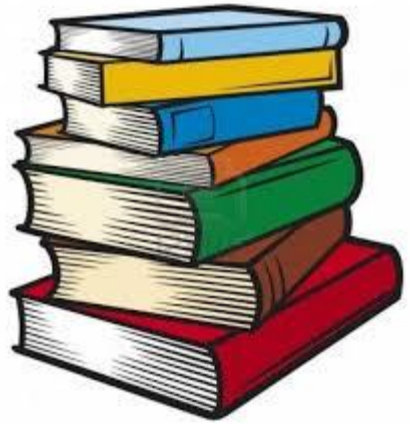
ArrayRotation1

Examples of stack



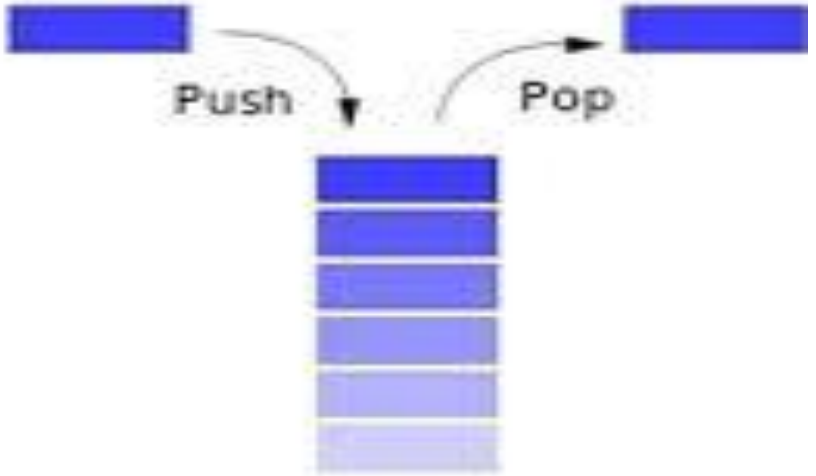
Stacks

Kiran Waghmare



Stack of books

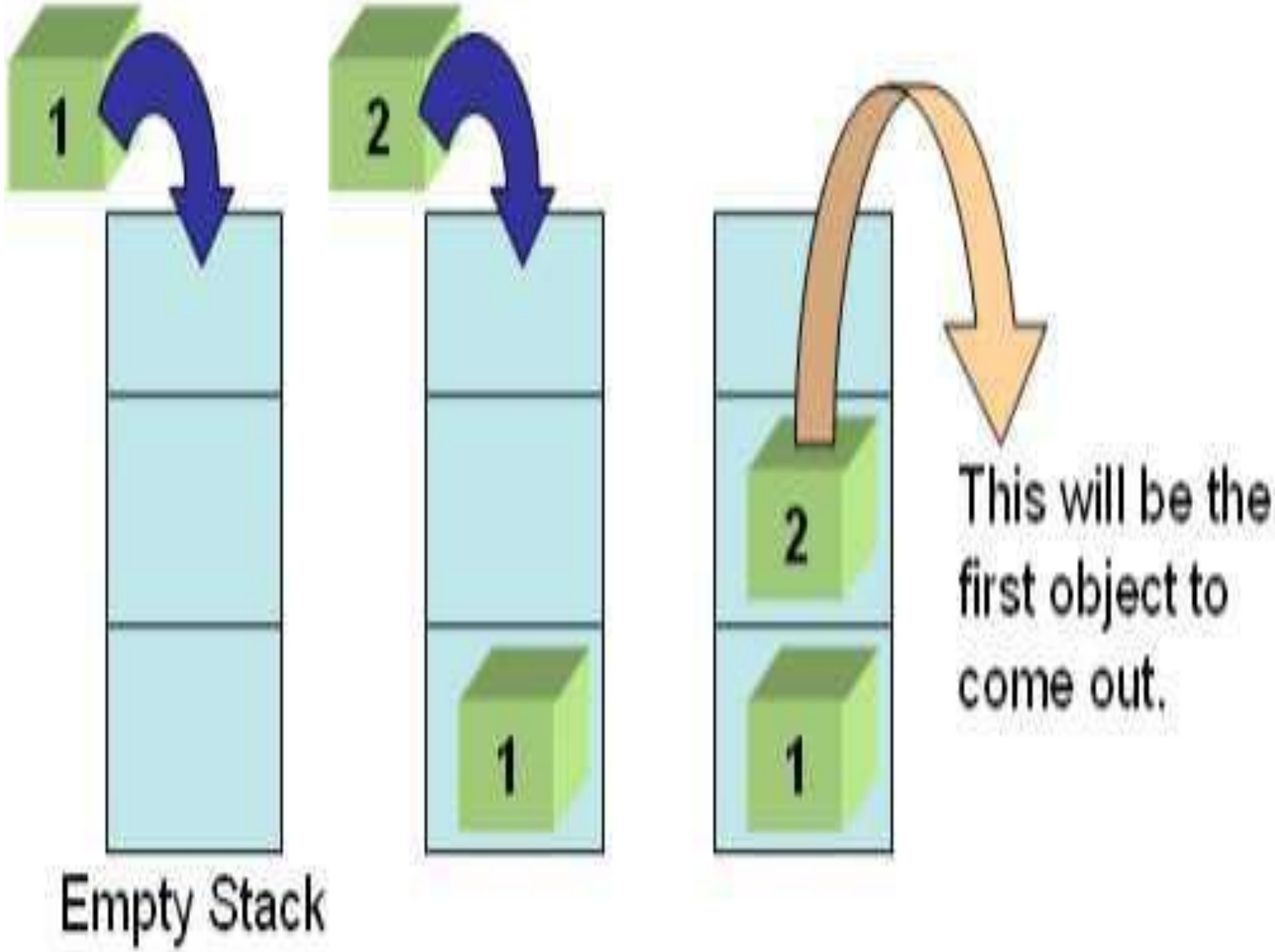
Stack of Coins

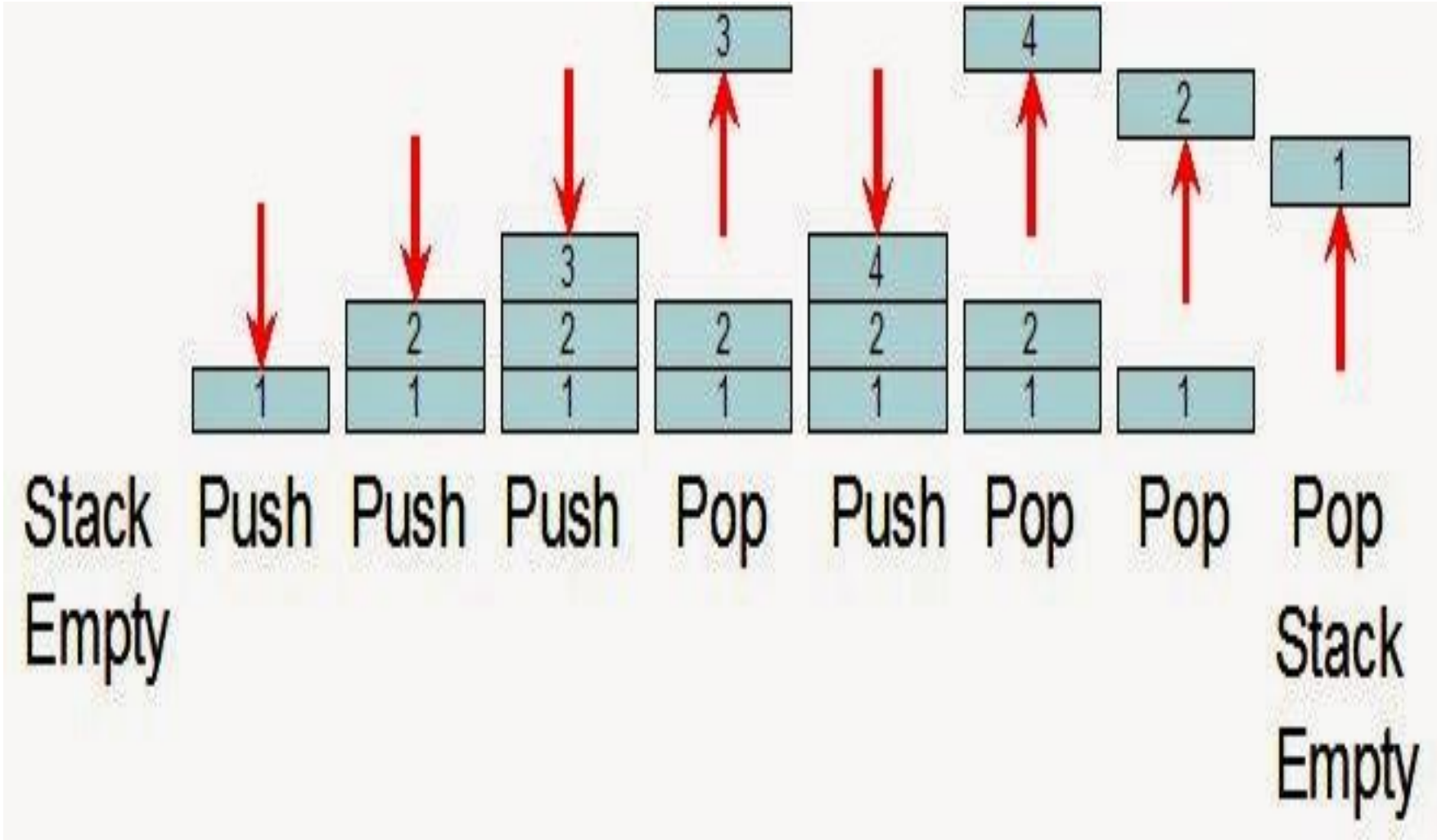


Memory stack

Standard Stack Operations

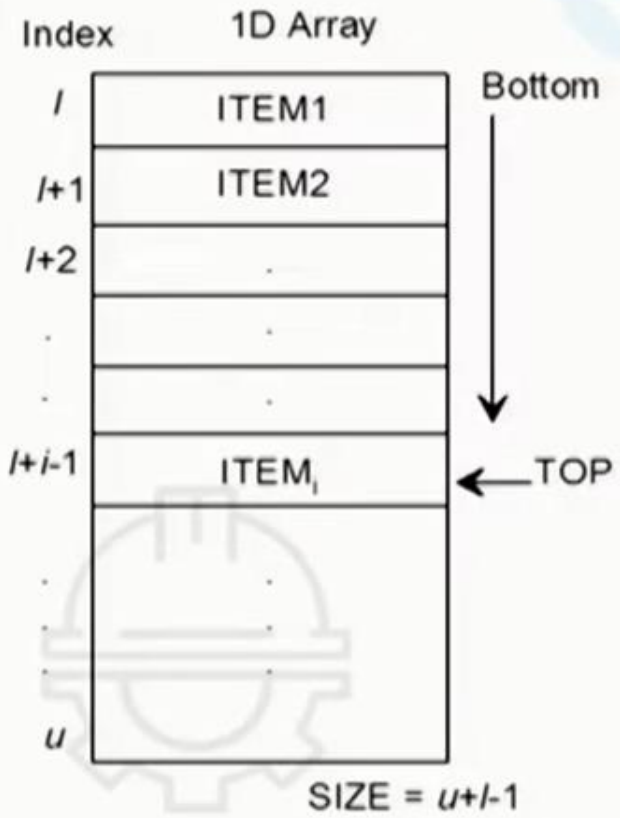
- **The following are some common operations implemented on the stack:**
- **push():**
  - When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.
- **pop():**
  - When we delete an element from the stack, the operation is known as a pop. If the stack is empty means that no element exists in the stack, this state is known as an underflow state.
- **isEmpty():**
  - It determines whether the stack is empty or not.
- **isFull():**
  - It determines whether the stack is full or not.'
- **peek():**
  - It returns the element at the given position.
- **count():**
  - It returns the total number of elements available in a stack.
- **change():**
  - It changes the element at the given position.
- **display():**
  - It prints all the elements available in the stack.


1
2
2
1
2
1
This will be the first object to come out.
Empty Stack

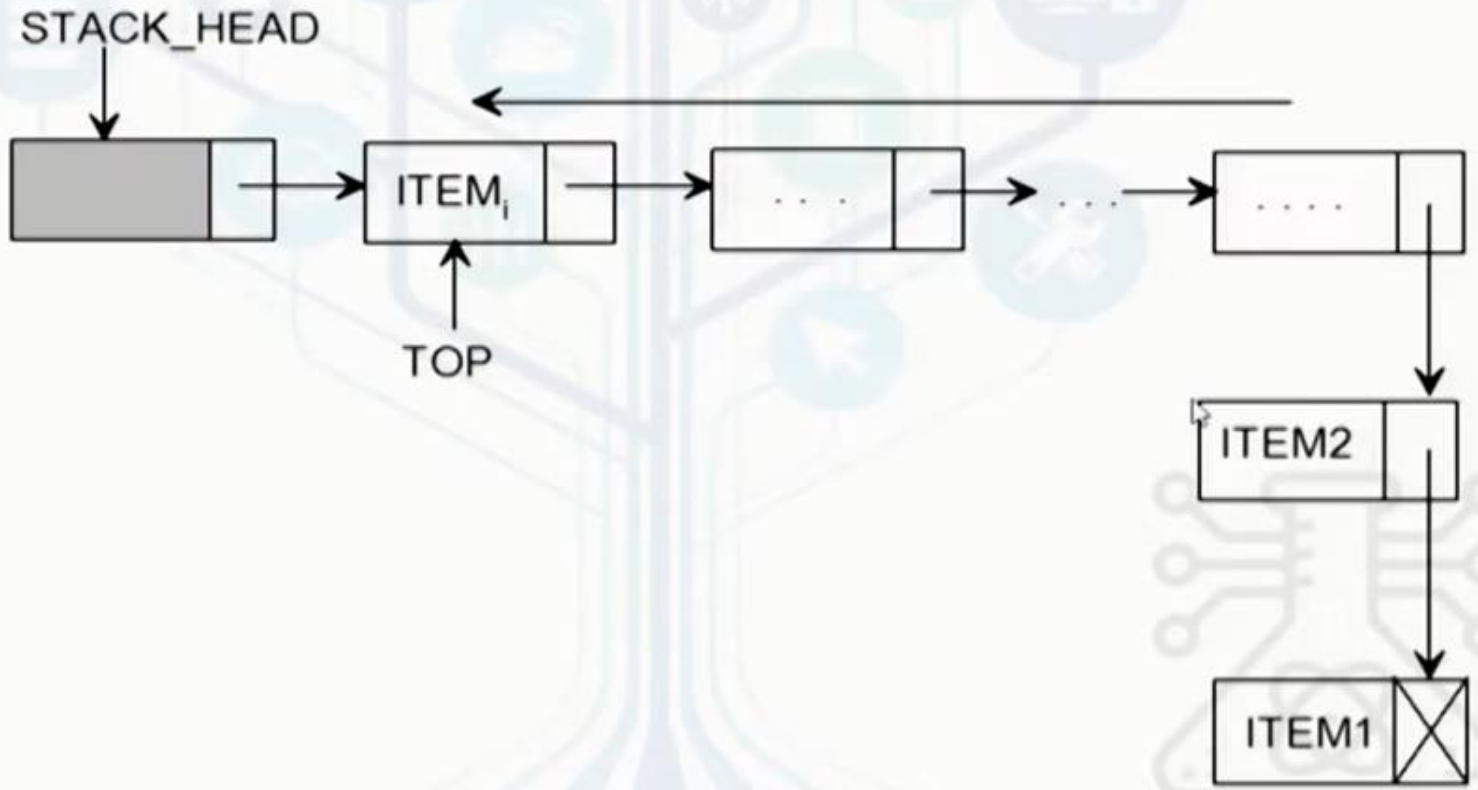

Stack Empty
Push
Push
Push
Pop
Push
Pop
Pop
Pop
Stack Empty
1
2
3
4

Memory representations

Array representation



Linked list representation



```
        S = new int[size];
        top =-1;
    }

    public void push(int j)
    {
        S[++top] = j;
    }

    public int pop()
    {
        return S[top--];
    }

    public int peek()
    {
        return S[top];
    }

    isEmpty()


}

class StackApp
```