Lecture-1 Abstract data types

What is a Data Structure?

A data structure (DS) is a way of organizing data so that it can be used effectively.

Why Data Structures?

They are essential ingredients in creating fast and powerful algorithms.

They help to manage and organize data.

They make code cleaner and easier to understand.

Abstract Data Types vs. Data Structures

An abstract data type (ADT) is an abstraction of a data structure which provides only the interface

to which a data structure must adhere to.

The interface does not give any specific details about how something should be implemented or in what programming language.

Examples

Abstraction (ADT)	Implementation (DS)
List	Dynamic Array Linked List
Queue	Linked List based Queue, Array based Queue, Stack based Queue
Map	Tree Map Hash Map / Hash Table
Vehicle	Golf cart, Bicycle, Smart car

Lecture-2

Introduction to Big-O

Complexity Analysis

As programmers, we often find ourselves asking the same two questions over and over again:

How much time does this algorithm need to finish?

How much space does this algorithm need for its computation?

Big-O Notation

Big-O Notation gives an upper bound of the complexity in the worst case, helping to quantify performance as the input size becomes arbitrarily large.

Big-O Notation

n - The size of the input

Complexities ordered in from smallest to largest.

Constant Time : $O(1)$
Logarithmic Time : $O(\log(n))$
Linear Time : $O(n)$
Linearithmic Time : $O(n\log(n))$
Quadric Time : $O(n^2)$
Cubic Time : $O(n^3)$
Exponential Time : $O(b^n), b>1$
Factorial Time : $O(n!)$

Big-O Properties

$$O(n+c) = O(n)$$
$$O(cn) = O(n), c>0$$

Let f be a function that describes the running time of a particular algorithm for an input of size n:

$$f(n) = 7\log(n)^3 + 15n^2 + 2n^3 + 8$$

$$O(f(n)) = O(n^3)$$

Practical examples coming up don't worry :)

Big-O Examples

The following run in constant time : $O(1)$

```
a := 1
b := 2
c := a + 5 * b
```

```
i := 0
while i < 11 Do
    i = i + 1
```

The following run in linear time : $O(n)$

```
i := 0
while i < n Do
    i = i + 1
```

$f(n) = n$

$O(f(n)) = O(n)$

```
i := 0
While i < n Do
    i = i + 3
```

$f(n) = n/3$

$O(f(n)) = O(n)$

Big-O Examples

Both of following run in quadratic time. The first may be obvious since n work done n times is $n*n = O(n^2)$, but what about the second one?

```
For (i := 0 ; i < n ; i = i + 1)
    For (j := 0 ; j < n ; j = j + 1)
```

$f(n) = n*n = n^2$, $O(f(n)) = O(n^2)$

```
For (i := 0 ; i < n ; i = i + 1)
    For (j := i ; j < n ; j = j + 1)
                ^ replaced 0 with i
```

For a moment just focus on the second loop. Since i goes from [0, n) the amount of looping done is directly determined

by what i is. Remark that if $i = 0$, we do n work, if $i = 1$, we do $n-1$ work, if $i = 2$, we do $n-2$ work, etc...

So the question then becomes what is:

$$(n) + (n-1) + (n-2) + (n-3) + \cdots + 3 + 2 + 1 \; ?$$

Remarkably this turns out to be $n(n+1)/2$, so

$$O(n(n+1)/2) = O(n^2/2 + n/2) = O(n^2)$$

```
For (i := 0; i < n; i = i + 1)
    For (j := i; j < n; j = j + 1)
```

Big-O Examples

Suppose we have a sorted array and we want to find the index of a particular value in the array, if it exists. What is the time complexity of the following algorithm?

Ans: $O(\log_2(n))$
$= O(\log(n))$

```
low := 0
high := n - 1
While low <= high Do

    mid := (low + high)/2

    If array[mid] == value : return mid

    Else If array[mid] < value : lo = mid + 1

    Else If array[mid] > value : hi = mid - 1

return -1 // value not found
```

Big-O Examples

```
i := 0
While i < n Do
    j = 0
    While j < 3 * n Do
        j = j + 1
    j = 0
    While j < 2 * n Do
        j = j + 1
    i = i + 1
```

$f(n) = n * (3n + 2n) = 5n^2$

$O(f(n)) = O(n^2)$

Big-O Examples

```
i := 0
While i < 3*n Do
    j := 10
    While j <= 50 Do
        j = j+1
    j = 0
    While j < n*n*n Do
        j = j+2
    i = i+1
```

$$f(n) = 3n * (40 + n^3/2)$$

$$= 3n/40 + 3n^4/2$$

$$O(f(n)) = O(n^4)$$

Big-O Examples

Finding all subsets of a set — $O(2^n)$

Finding all permutations of a string - $O(n!)$

Sorting using mergesort - $O(n\log(n))$

Iterating over all the cells in a matrix of size n by m - $O(nm)$

Lecture - 3

Static and Dynamic Arrays

Part 1/2

Outline

- Discussion and examples about Arrays
  - What is an array?
  - When and Where is a Array used?
  - Complexity
  - Static array wage example
- Dynamic Array implementation details
- Code Implementation

Discussion and examples

What is a Static Array?

A static array is a fixed length container containing n elements indexable from the range [0, n-1].

Q: What is meant by being 'indexable'?

A: This means that each slot/index in the array can be referenced with a number.

When and where is a static Array used?

1. Storing and accessing sequential data.
2. Temporarily storing objects.
3. Used by IO routines as buffers

4. Lookup tables and inverse lookup tables.

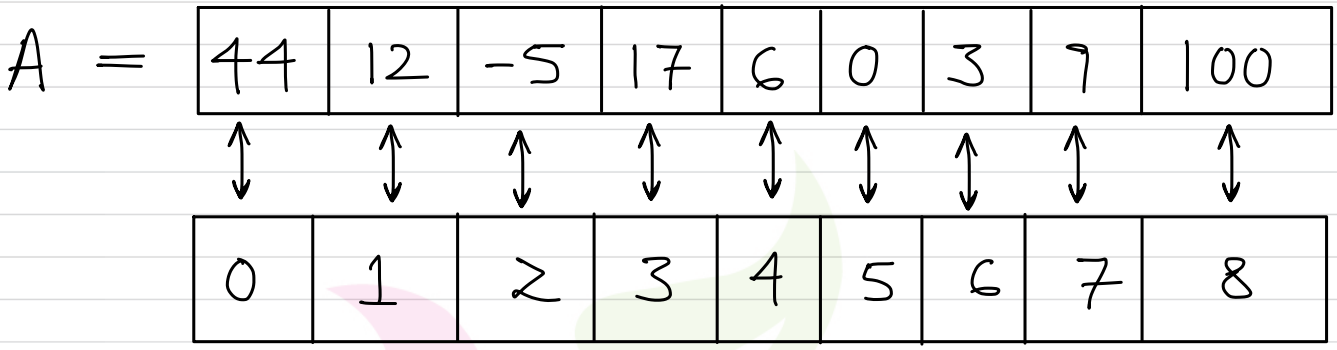
5. Can be used to return multiple values from a function.

6. Used in dynamic programming to cache answers to subproblems.

Complexity

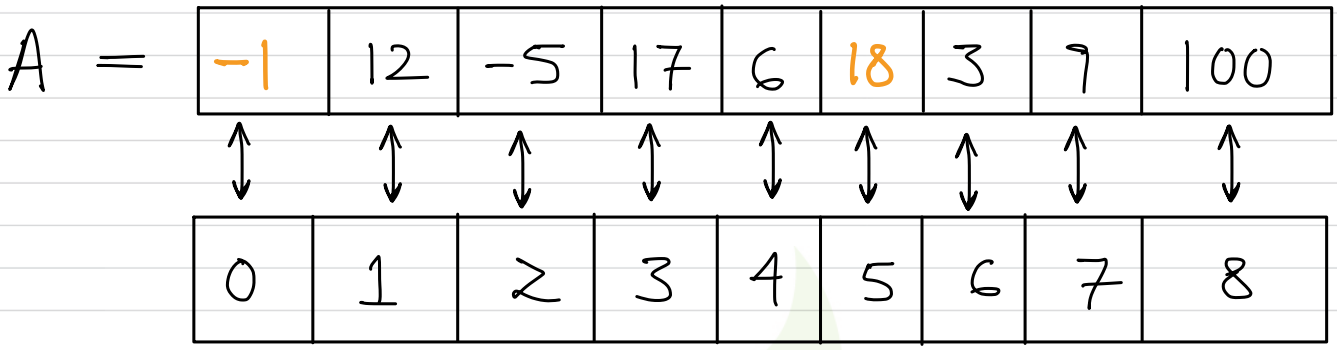
	Static Array	Dynamic Array
Access	O(1)	O(1)
Search	O(n)	O(n)
Insertion	N/A	O(n)
Appending	N/A	O(1)
Deletion	N/A	O(n)

Static Array



Elements in A are referenced by their index. There is no other way to access elements in an array. Array indexing is zero-based, meaning the first element is found in position zero.

A[0] = 44
A[1] = 12
A[4] = 6
A[7] = 9
A[9] ⇒ index out of bounds!



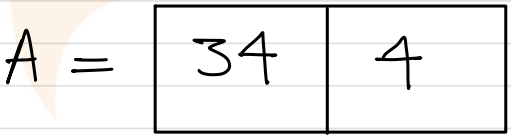
A[0] := -1

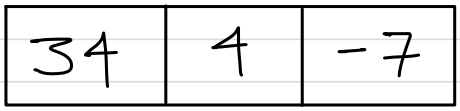
A[5] := 18

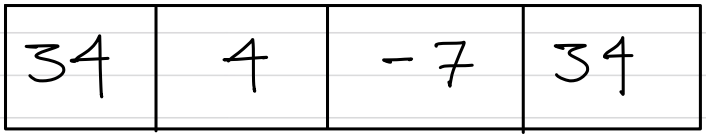
Operations on Dynamic Arrays

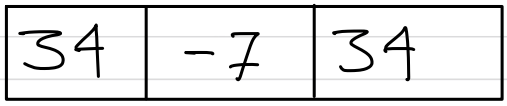
Dynamic Array

The dynamic array can grow and shrink in size.



A.add(-7) 

A.add(34) 

A.remove(4) 

Dynamic Array

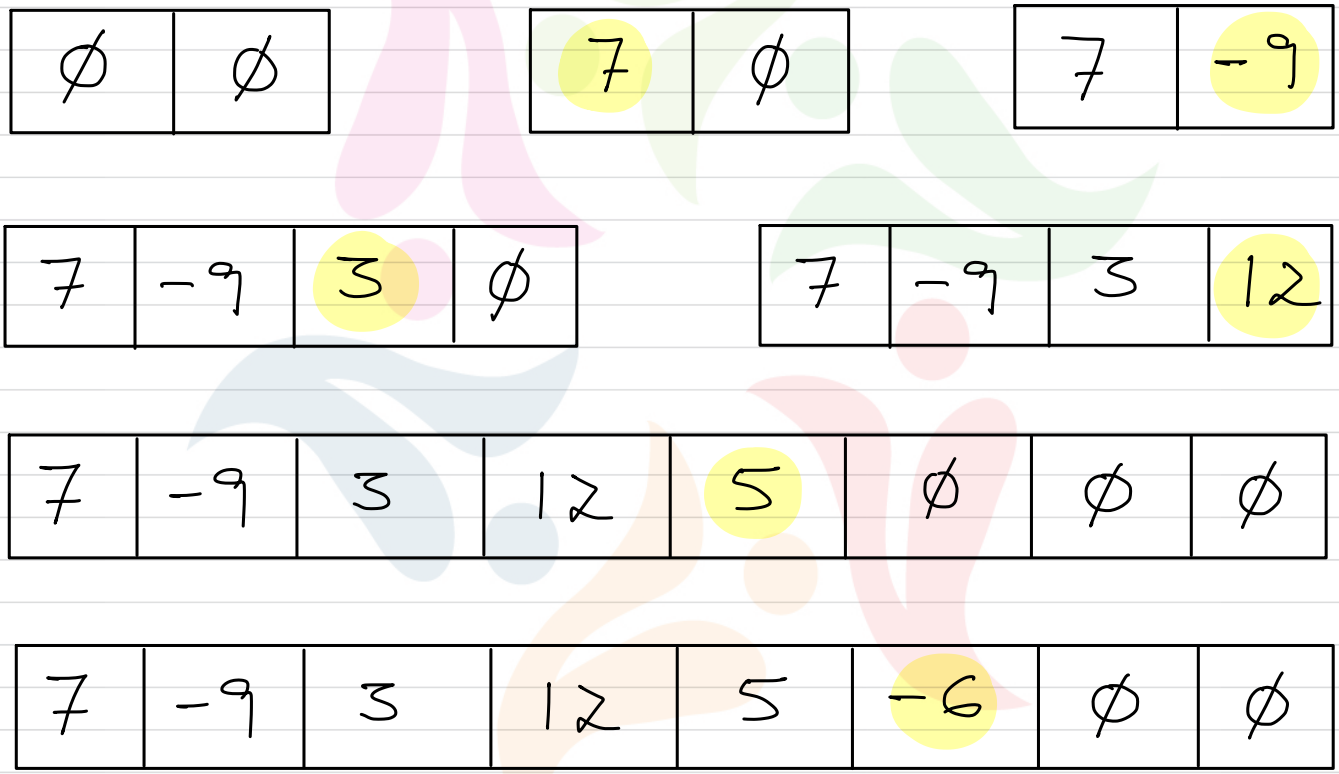
Q. How can we implement a dynamic array?

A: One way is to use a static array!

1. Create a static array with an initial capacity.
2. Add elements to the underlying static array, keeping track of the number of elements.
3. If adding another element will exceed the capacity, then create a new static array with twice the capacity and copy the original elements into it.

Dynamic Array

Suppose we create a dynamic array with an initial capacity of two and then begin adding elements to it.



Lecture - 4

Singly and Doubly Linked Lists !

Outline :

- Discussion about Singly & Doubly Linked Lists
  - What is a linked list ?
  - Where are linked lists used ?
  - Terminology
  - Singly Linked vs. Doubly Linked
- Implementation Details
  - How to insert new elements
  - How to remove elements
- Complexity Analysis
- Code Implementation (Doubly linked list)

What is a linked list?

A linked list is a sequential list of nodes that hold data which point to other nodes also containing data.

Where are linked lists used?

- Used in many List, Queue & Stack implementations.
- Great for creating circular lists.
- Can easily model real world objects such as trains.
- Used in separate chaining, which is present certain Hashtable

implementations to deal with hashing collisions.

- Often used in the implementation of adjacency lists for graphs.

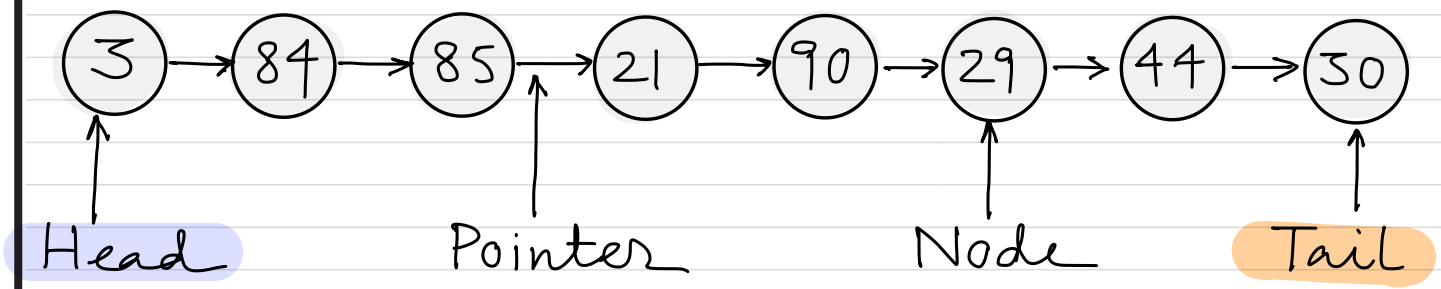
Terminology

Head: The first node in a linked list.

Tail: The last node in a linked list

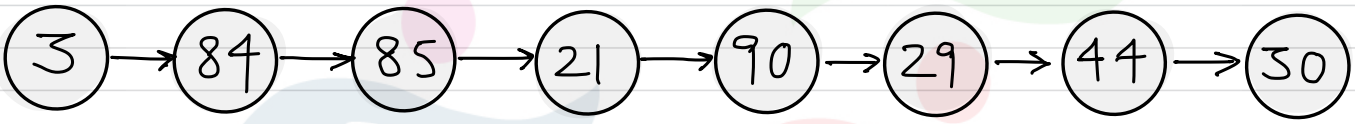
Pointer: Reference to another node

Node: An object containing data and pointer(s)

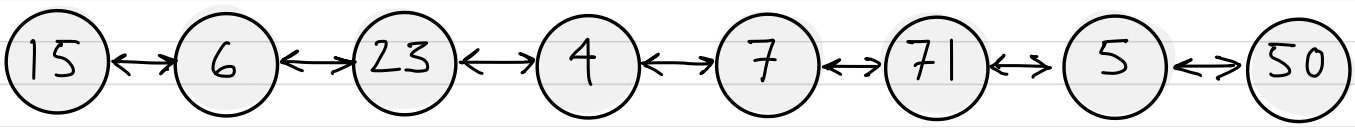


Singly vs Doubly Linked Lists

Singly linked lists only hold a reference to the next node. In the implementation you always maintain a reference to the head to the linked list and a reference to the tail node for quick additions/removals.



With a doubly linked list each node holds a reference to next and previous node. In the implementation you always maintain a reference to the head and the tail of the doubly linked list to do quick additions/removals from both ends of your list.



Singly vs Doubly Linked Lists

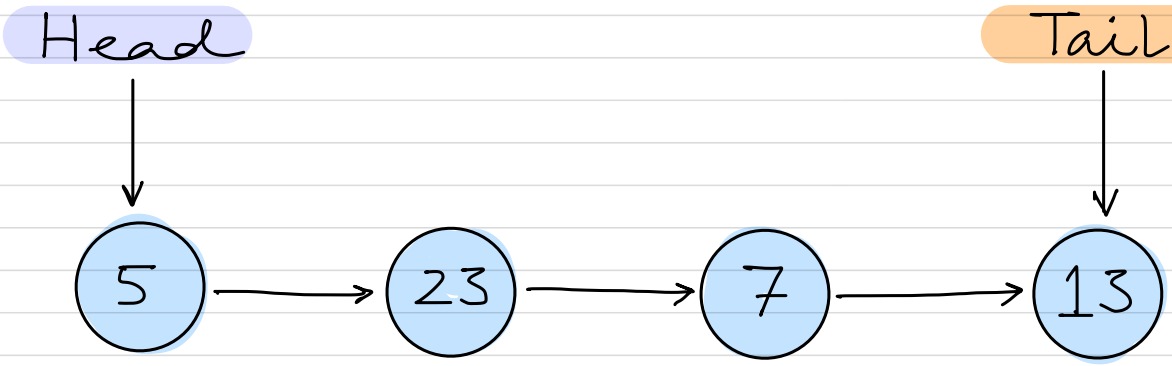
Pros and Cons

	Pros	Cons
Singly Linked	• Uses less memory • Simpler Implementation	cannot easily access previous elements
Doubly Linked	can be traversed backwards	Takes 2x memory

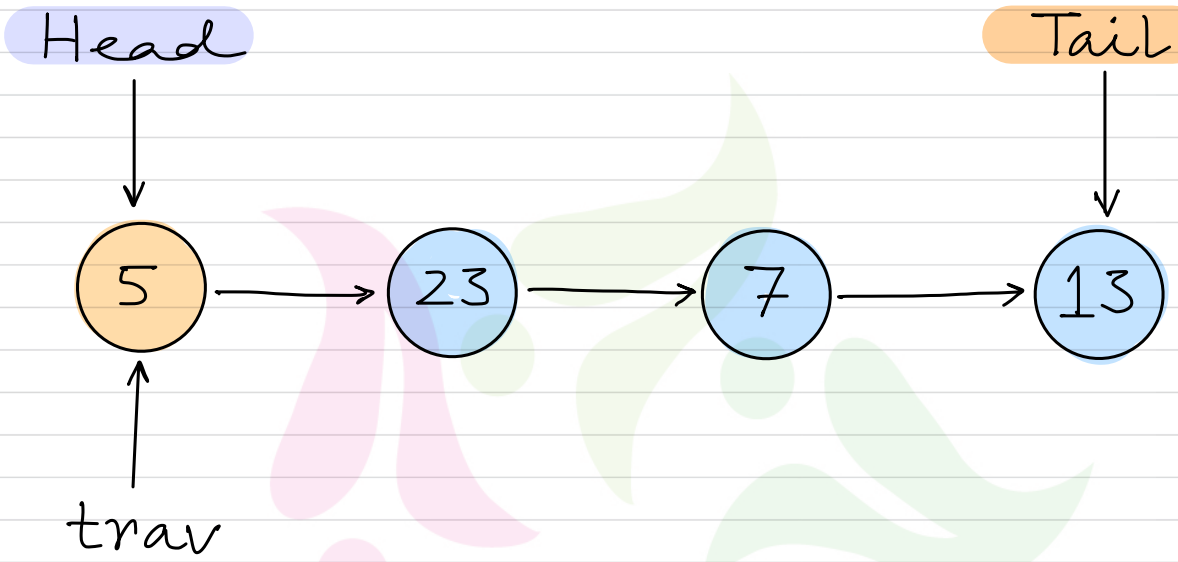
Implementation Details

Inserting Singly Linked List

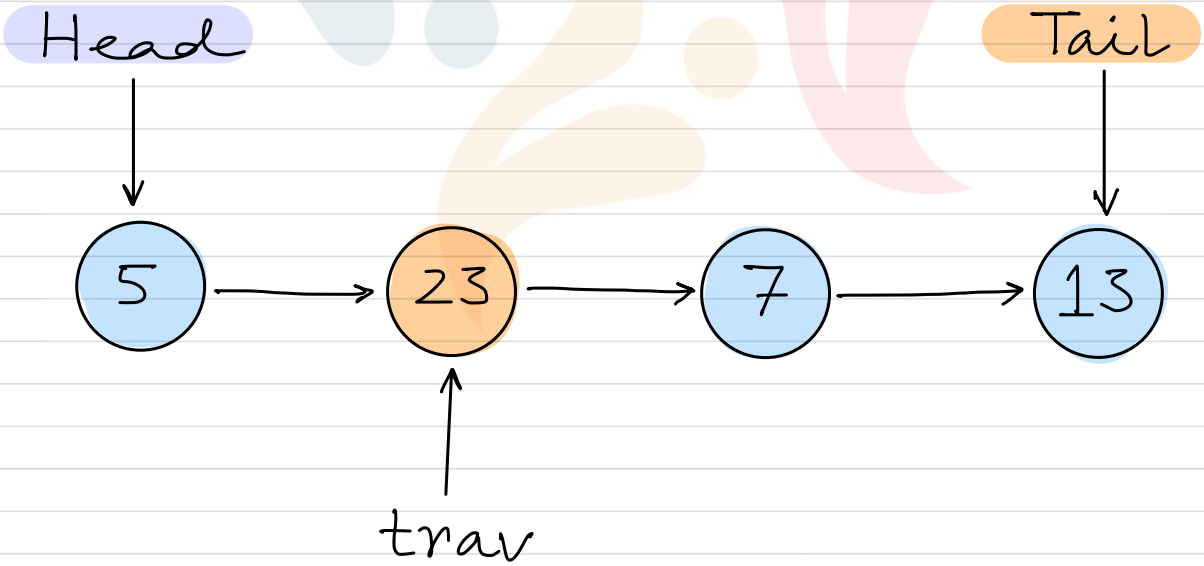
Insert 11 where the third node is.



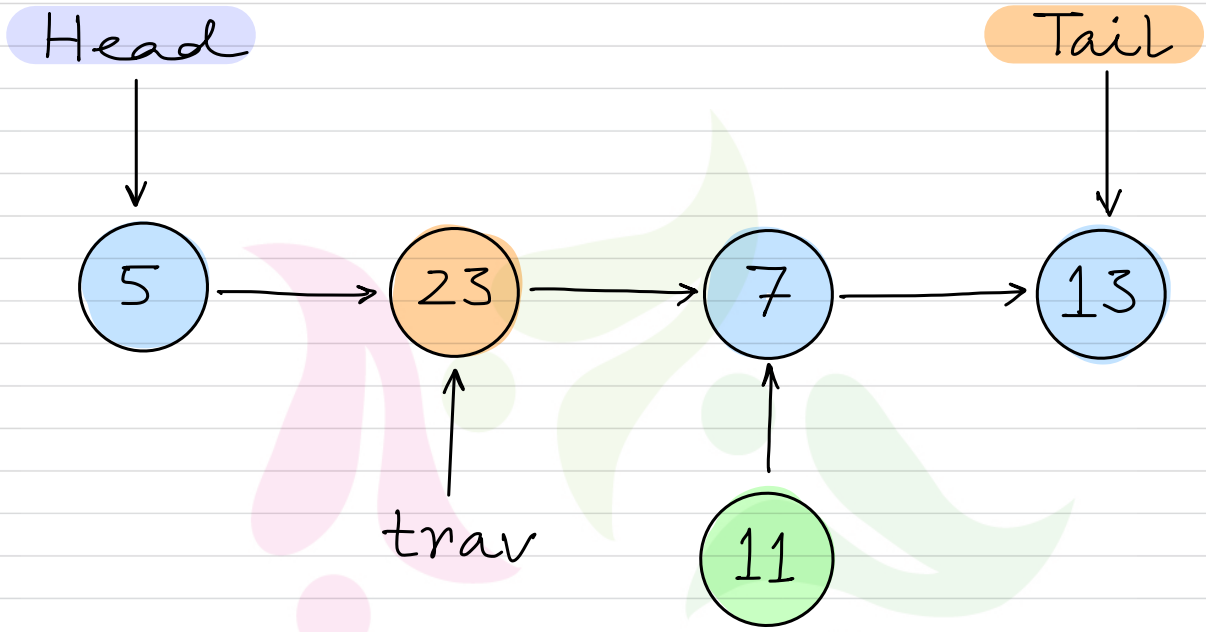
Step - 1



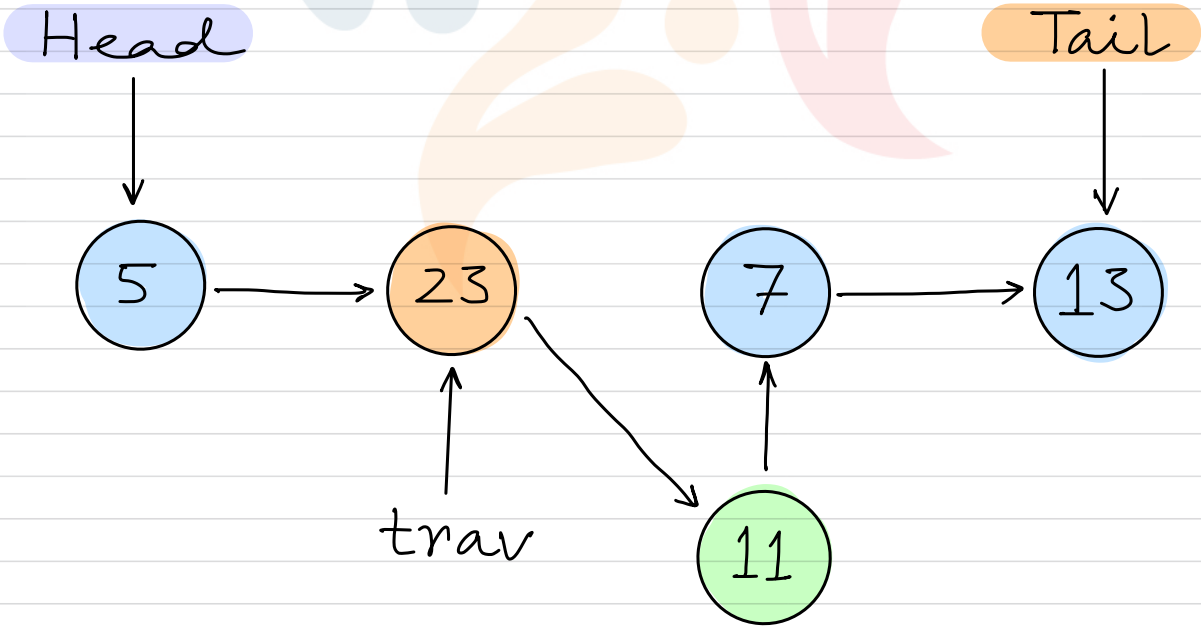
Step - 2



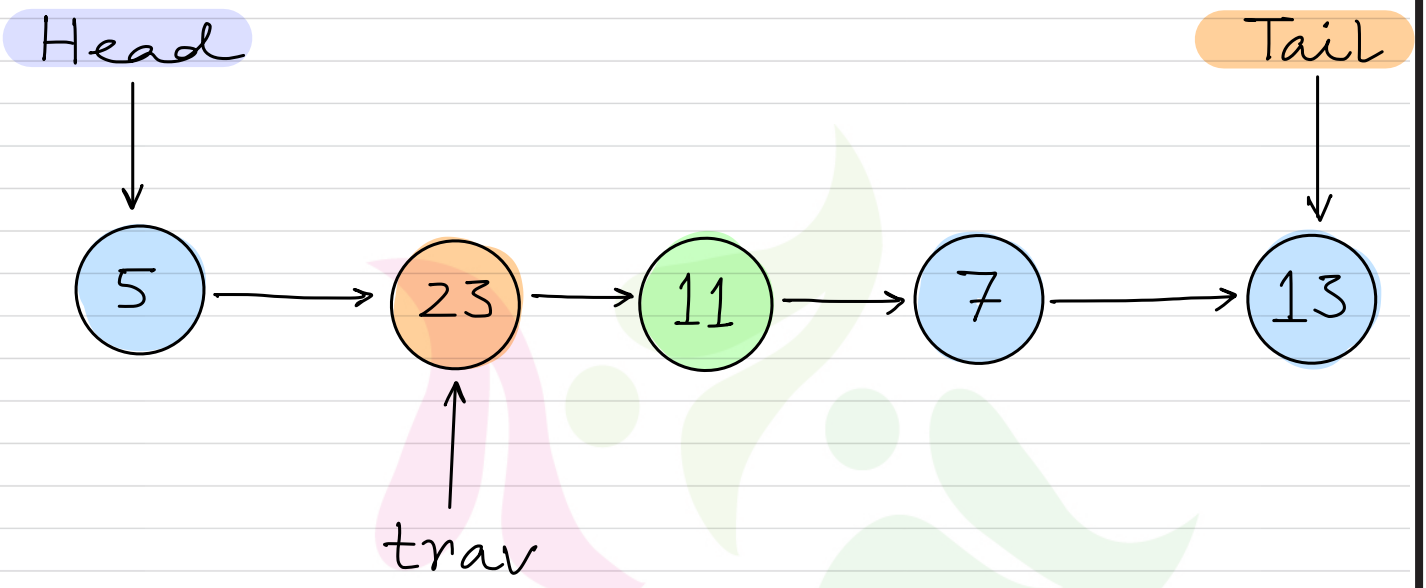
Step - 3



Step - 4



Step - 5



Lecture - 5

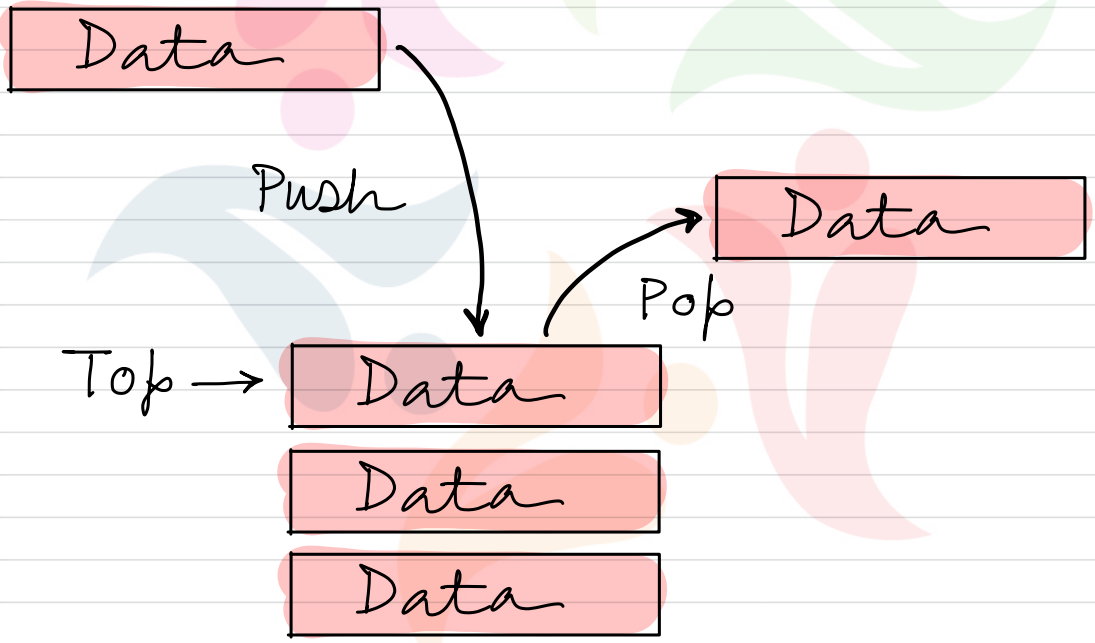
Stack Introduction

Outline :

- Discussion about Stacks
  - What is a stack?
  - When and where is a Stack used?
  - Complexity Analysis
  - Stack Usage examples
- Implementation details
  - Pushing elements on stack
  - Popping elements from stack
- Code Implementation

What is a Stack?

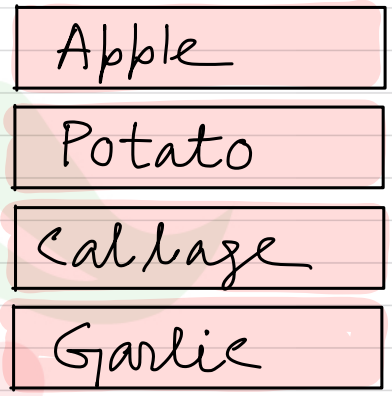
A Stack is a one-ended linear data structure which models a real world stack by having two primary operations, namely push and pop.



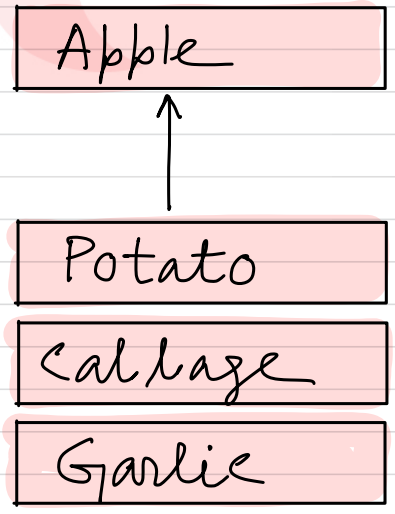
What is a Stack?

Instructions

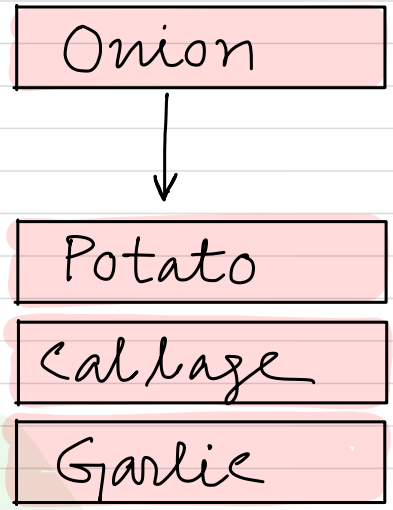
```
pop ( )
push ('onion')
push ('celery')
push ('Watermelon')
pop ( )
pop ( )
push ('Lettuce')
```



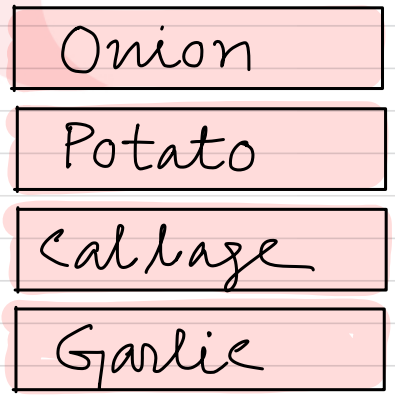

```
→pop ( )
 push ('onion')
 push ('celery')
 push ('Watermelon')
 pop ( )
 pop ( )
 push ('Lettuce')
```



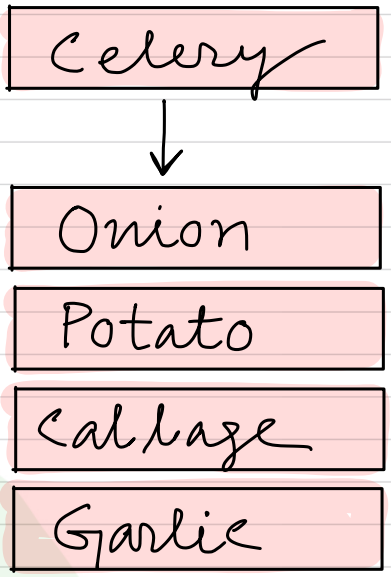
```
    pop ( )
→   push ('onion')
    push ('celery')
    push ('Watermelon')
    pop ( )
    pop ( )
    push ('Lettuce')
```



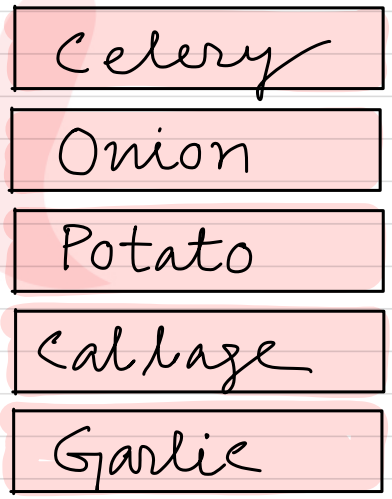

```
    pop ( )
→   push ('onion')
    push ('celery')
    push ('Watermelon')
    pop ( )
    pop ( )
    push ('Lettuce')
```



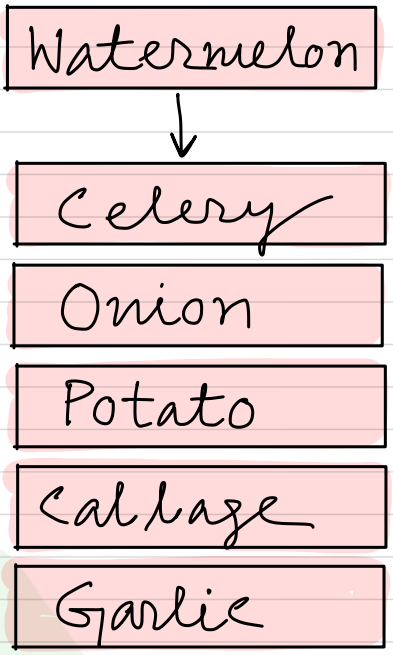
```
pop ( )
push ('Onion')
→ push ('celery')
push ('Watermelon')
pop ( )
pop ( )
push ('Lettuce')
```




```
pop ( )
push ('Onion')
→ push ('celery')
push ('Watermelon')
pop ( )
pop ( )
push ('Lettuce')
```

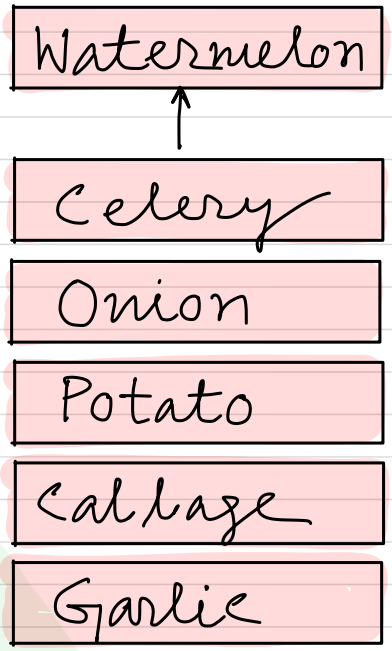


```
pop ( )
push ('onion')
push ('celery')
→ push ('Watermelon')
pop ( )
pop ( )
push ('Lettuce')
```

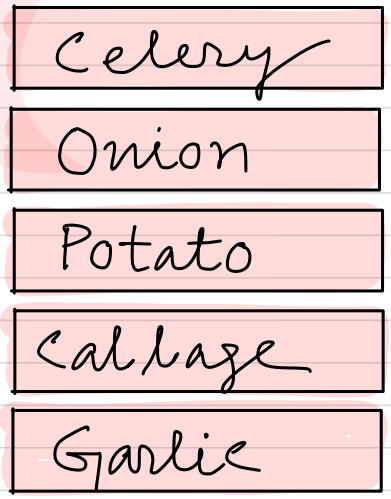



```
pop ( )
push ('onion')
push ('celery')
→ push ('Watermelon')
pop ( )
pop ( )
push ('Lettuce')
```

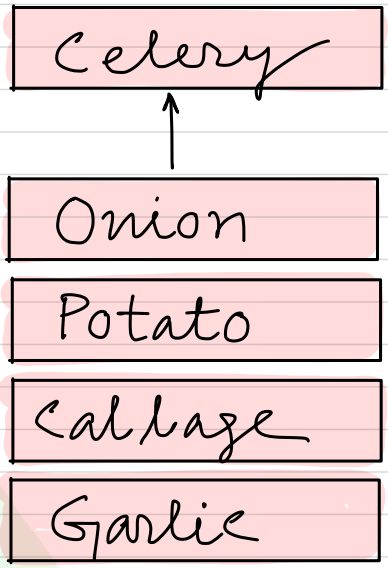
```
pop ( )
push ('onion')
push ('celery')
push ('watermelon')
→ pop ( )
pop ( )
push ('Lettuce')
```



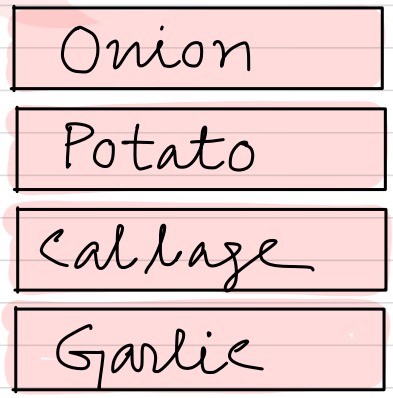

```
pop ( )
push ('onion')
push ('celery')
push ('watermelon')
→ pop ( )
pop ( )
push ('Lettuce')
```



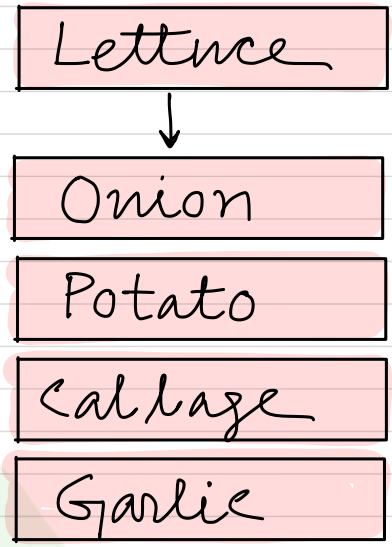
```
pop ( )
push ('onion')
push ('celery')
push ('Watermelon')
pop ( )
→ pop ( )
push ('Lettuce')
```




```
pop ( )
push ('onion')
push ('celery')
push ('Watermelon')
pop ( )
→ pop ( )
push ('Lettuce')
```



```
pop ( )
push ('onion')
push ('celery')
push ('Watermelon')
pop ( )
pop ( )
→ push ('Lettuce')
```




```
pop ( )
push ('onion')
push ('celery')
push ('Watermelon')
pop ( )
pop ( )
→ push ('Lettuce')
```