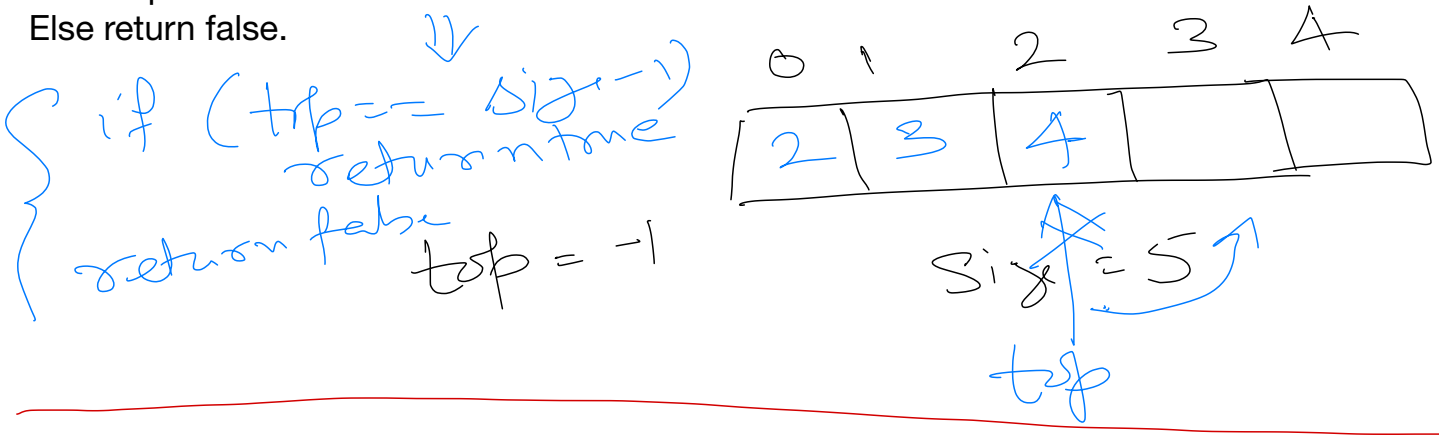
Push(element)
- If stack is full then stop.
- Make space at top for new element.
- Store new element and make it topmost element.

Pop()
- If stack is empty then stop.
- Set topmost element as result.
- Remove topmost element and make element below top, the topmost element.
- Return the result.

IsEmpty()
- If no element stored at top then return true.
Else return false.

IsFull()
- If no space left for new element to be stored then return true.
Else return false.

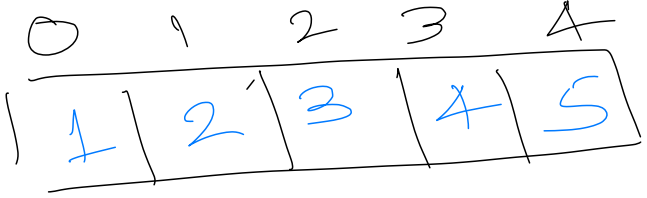


AddQ(element)
- If queue is full then stop.
- Make space at rear for new element.
- Store new element and make it the rear element.

DeleteQ()
- If queue is empty then stop.
- Move the front towards rear.
- Return the front element as result.

IsEmpty()
- If no elements stored in queue then return true.
Else return false.

IsFull()
- If no space left for new element to be stored then return true.
Else return false.



Size = 5

front = 4
rear = 4

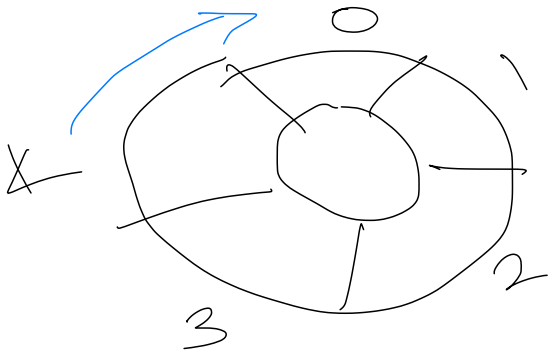
Add 5 times

Delete 5 times

Queue is full as well as empty.

Problem with linear Queue

Circular Queue: After last element comes first element.



Size = 5

i = 0 1 2 3 4 5 6 7 ...

++i in a loop

MOD N

0 .. (N-1)

```
if (i == size) i = 0;
```

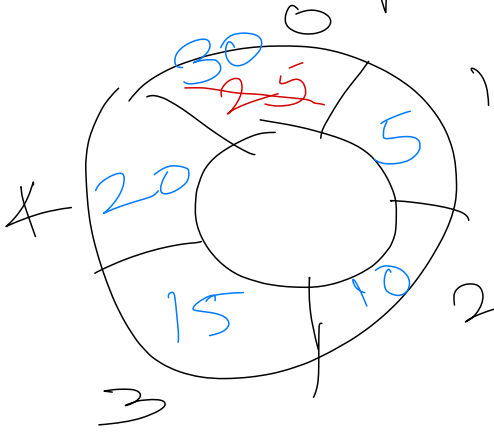
i = (i+1) MOD size

rear = (rear+1) % size

rear → 3 → 4 → 0

5

Initial value of front & rear for circular queue will be 0.



Size = 5

front = 0 1

rear = 0

0 ... 0 ~~1~~ ~~2~~ 3 ~~4~~ 0 1

if (rear + 1)% Size == front

If after rear comes front then queue is full fail

AddQ(5)

~~AddQ(10)~~

AddQ(15)

~~AddQ(20)~~

~~AddQ(25)~~

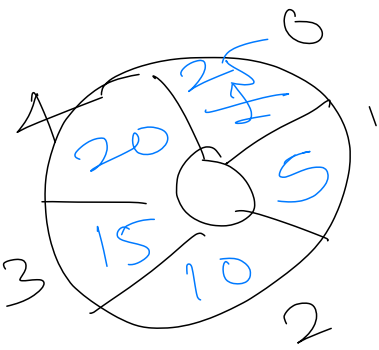
5 ← DeleteQ()

AddQ(30)

front = -1

rear = -1

Size = 5



Add 0 1 5 10

25 20 15

Assignment:

1. Implement Circular Queue using Arrays.

Recommended: Practice TDD while doing the implementation.

Practice Problems

Problem to solve involving Stack - Check for balanced parenthesis.

((([[ ]]))) ([ ]) - This is

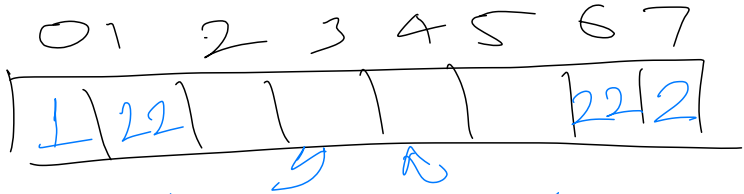
([ )] - Not balanced/matched

Implement Stack using two Queues for storing data.

Implement Queue using two Stacks for storing data.

Two Stack - Implement two stacks in a single array.

Siz=8

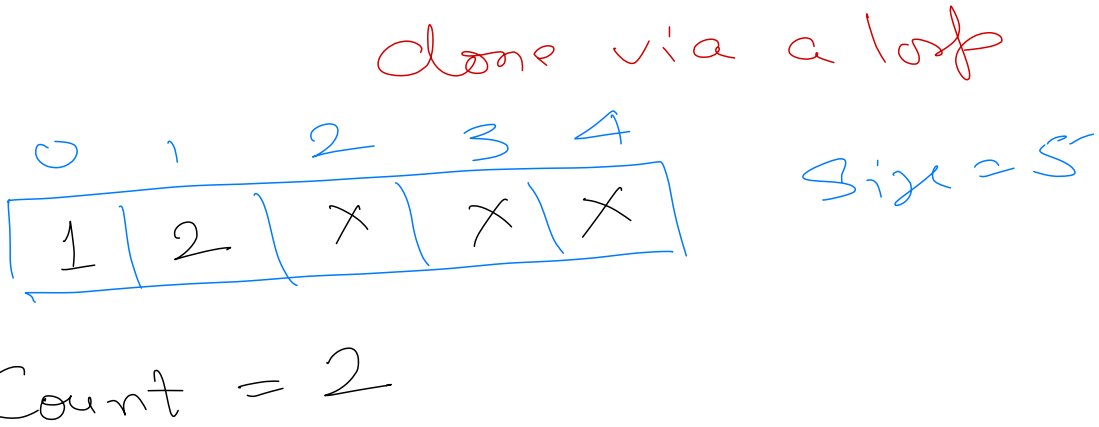


Linear Data Structures

Array

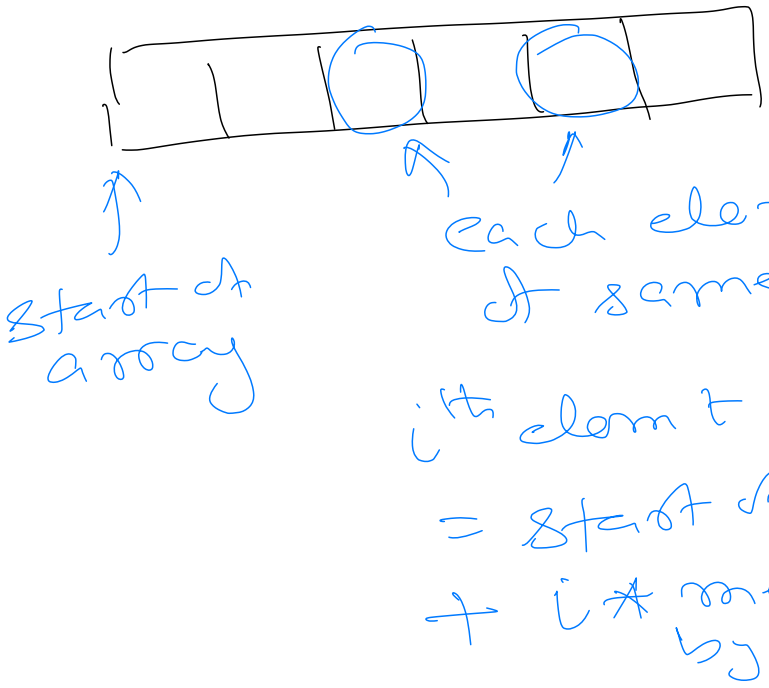
- Need for an array?

When we need to store multiple elements.
And do same processing on those elements.



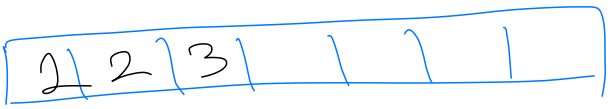
Properties of Array

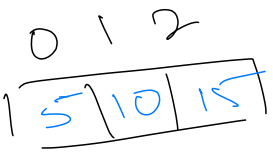
- Data Structure that stores multiple elements, all the same type.
- All elements of an array are stored sequentially in memory, one after another.



Pros and Cons of Array

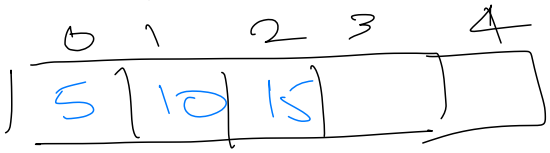
- Advantages
  - Efficient lookup OR Random access.
  - Efficient in adding and removing elements at the end of array
- Disadvantages
  - Fixed size. Resizing of array is inefficient.
  - Insertion and deletion of elements, in middle of array is inefficient.





Resize array to size 5

→ Allocate a new array of size 5
→ Copy elements from existing memory to new.

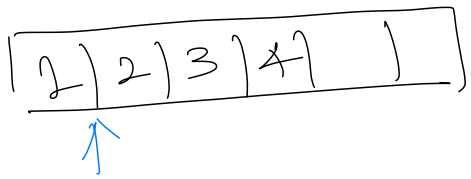


→ Release old memory block.

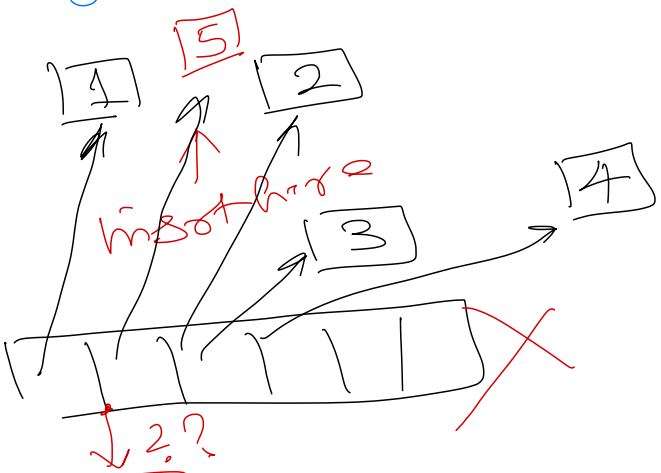
Linked List

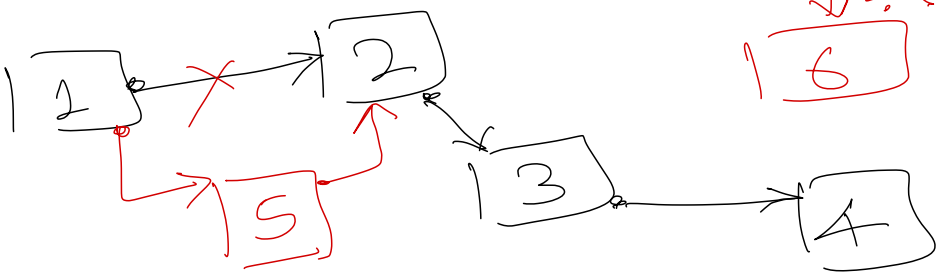
- Need for a linked list?

↳ Do not stores all element in one single memory block

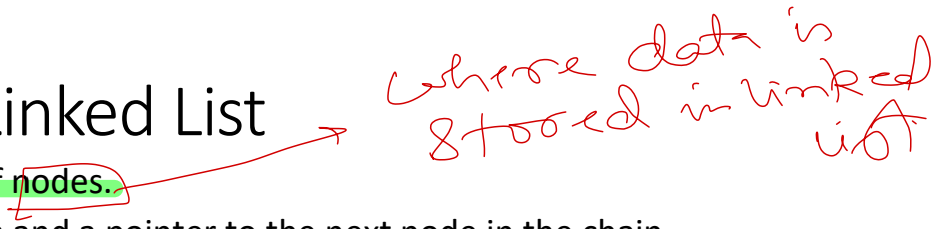


insert here





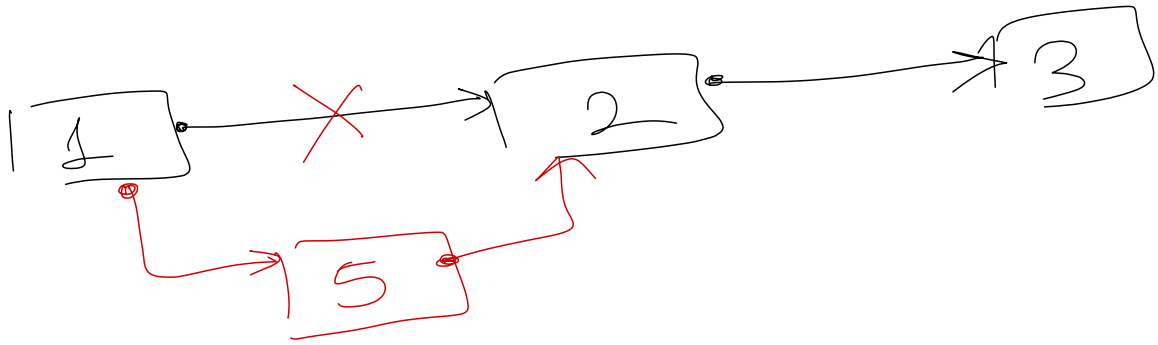
Properties of Linked List



- Stores data as a chain of nodes.
- Each node contains data and a pointer to the next node in the chain.
- First node of linked list is pointed by "head". When list is empty, head do not point to any node.
- Last node of list points to no node.

Pros and Cons of Linked List

- Advantages
  - Can dynamically grow or shrink is size.
  - Efficient in insertion and deletion of elements.
- Disadvantages
  - Lookup OR Random access is inefficient.



Types of Linked List

- Single linked list (Uni-directional).
- Doubly linked list (Bi-directional).
- Circular linked list.

One node keeps track of one neighbour node only.

Each node keeps track of two of its neighbours.