✏️

Data Structures

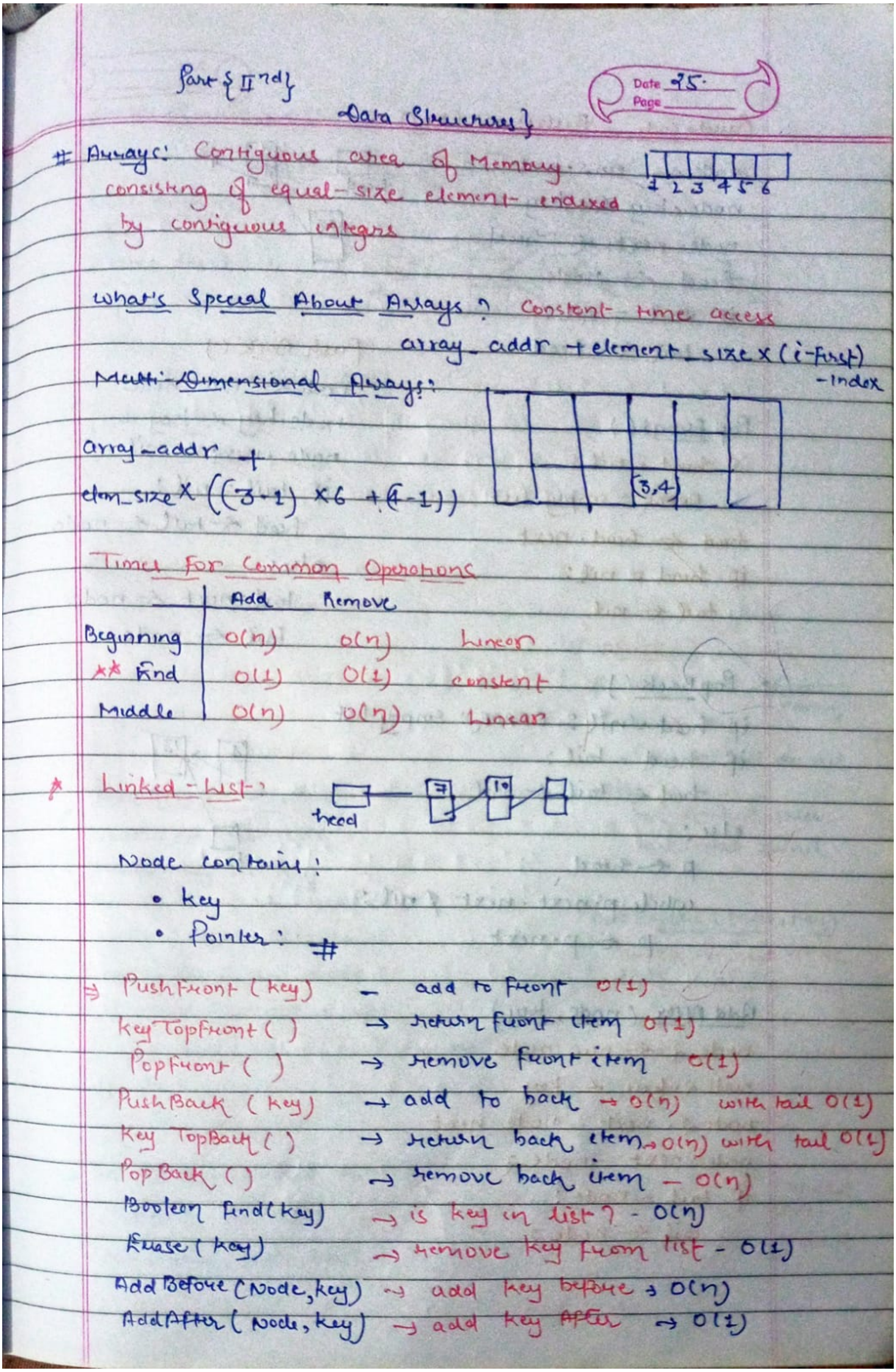
Topics Covered:

Arrays

Linked-List

Double-Linked list

Stack



Part {II nd}

Date 25.

Page

Data Structures}

Arrays: Contiguous area of Memory consisting of equal-size element indexed by contiguous integers

What's Special About Arrays? Constant time access

array_addr + element_size x (i-first) -index

Multi-Dimensional Arrays:

array_addr + elem_size x ((3-1) x6 + (4-1))

Times For Common Operations

	Add	Remove	
Beginning	O(n)	O(n)	Linear
** End	O(1)	O(1)	constant
Middle	O(n)	O(n)	Linear

* Linked-List:

Node contains:

- key
- Pointer: #

⇒ PushFront (key) – add to front O(1)

key TopFront ( ) → return front item O(1)

PopFront ( ) → remove front item O(1)

PushBack (key) → add to back → O(n) with tail O(1)

Key TopBack ( ) → return back item → O(n) with tail O(1)

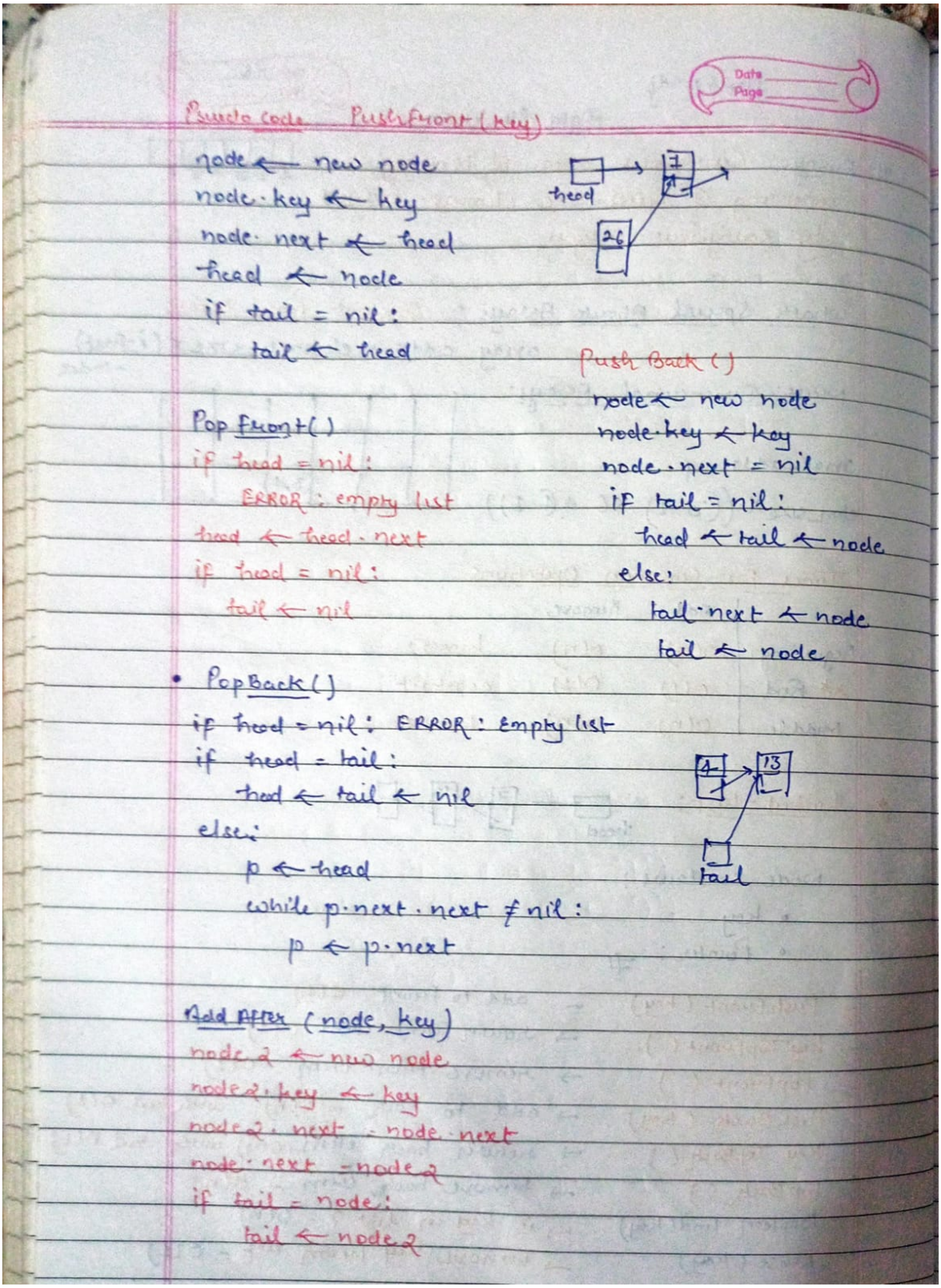
PopBack ( ) → remove back item – O(n)

Boolean Find(key) → is key in list? - O(n)

Erase (key) → remove key from list - O(1)

AddBefore (Node, key) → add key before → O(n)

AddAfter (Node, key) → add key After → O(1)



Date ________
Page ________

Psuedo code PushFront (key)

```
node ← new node
node.key ← key
node.next ← head
head ← node
if tail = nil:
    tail ← head
```

Push Back ()

```
node ← new node
node.key ← key
node.next = nil
if tail = nil:
    head ← tail ← node
else:
    tail.next ← node
    tail ← node
```

Pop Front()

```
if head = nil:
    ERROR : empty list
head ← head.next
if head = nil:
    tail ← nil
```

- Pop Back()

```
if head = nil: ERROR : Empty list
if head = tail:
    head ← tail ← nil
else:
    p ← head
    while p.next.next ≠ nil:
        p ← p.next
```

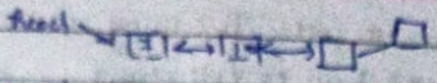
Add After (node, key)

```
node2 ← new node
node2.key ← key
node2.next = node.next
node.next = node2
if tail = node:
    tail ← node2
```

Date ________

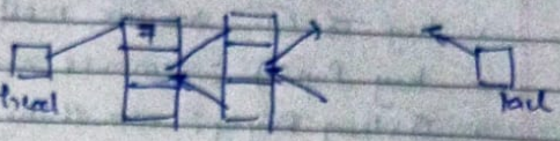
Page ________

Double-Linked List:



Node contains:

- key
- Next pointer
- prev pointer



popback → O(1)

```
PushBack(key)
node ← new node
node.key ← key;   node.next = nil
if tail = nil:
    head ← tail ← node
    node.prev ← nil
else:
    tail.next ← node
    node.prev ← tail
    tail ← node
```

```
* Add After(node, key)
node2 ← new node
node2.key ← key
node2.next ← node.next
node2.prev ← node
node.next ← node2
if node2.next ≠ nil:
    node2.next.prev ← node2
if tail = node:
    tail ← node2
```

* In Boubly-Linked list

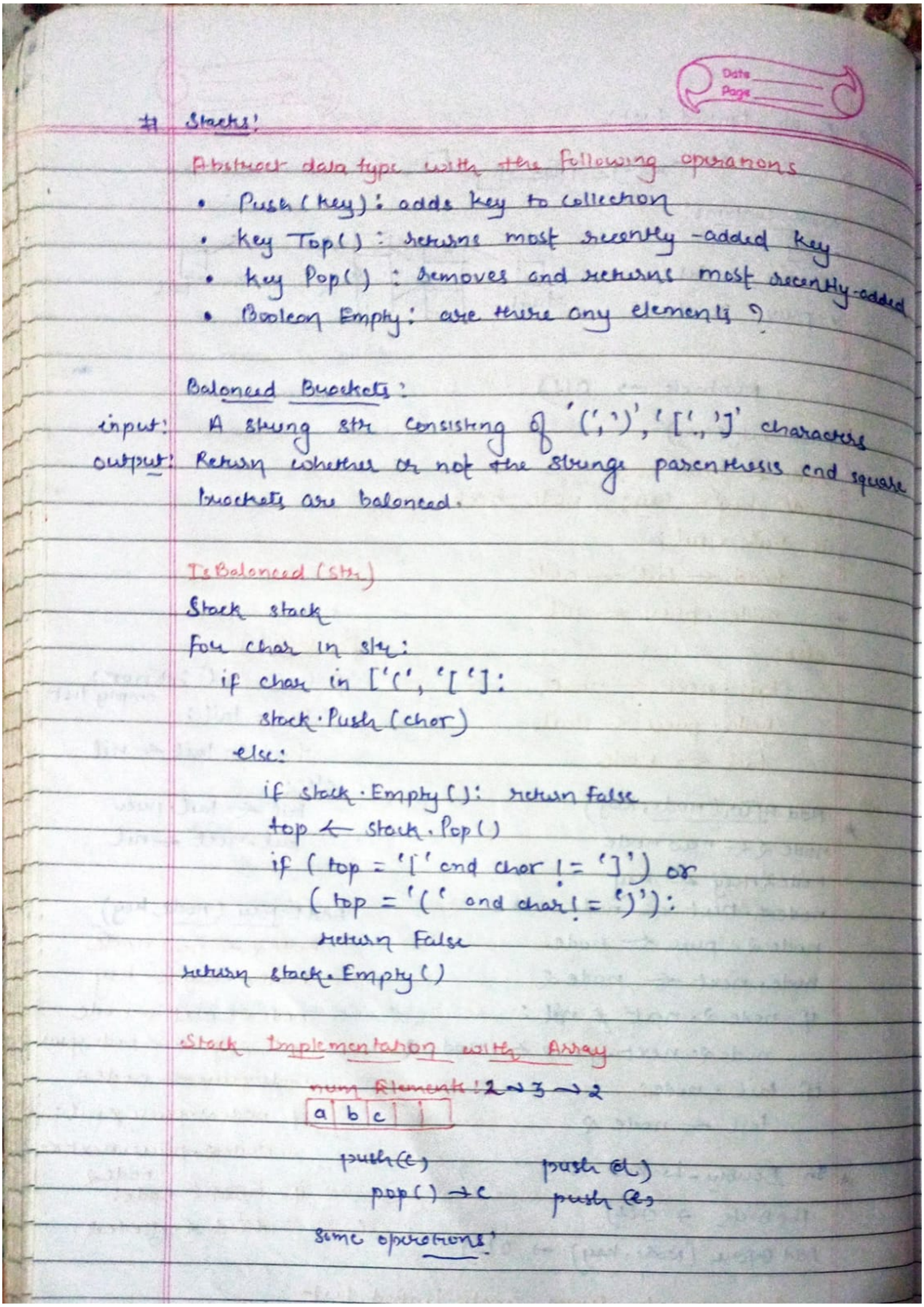
PopBack → O(1)

Add Before (Node, key) → O(1)

Improvement from single Linked List

```
PopBack()
if hed = nil : Error:
                empty list
if head = tail :
    head ← tail ← nil
else:
    tail ← tail.prev
    tail.next ← nil
```

```
Add Before (node, key)
    node2 ← new node
    node2.key ← key
    node2.next ← node
    node2.prev ← node.prev
    node.prev ← node2
    if node2.prev ≠ nil:
        node2.prev.next ←
                node2
    if head = node:
        head ← node2
```



Date ______
Page ______

Stacks:

Abstract data type with the following operations

- Push(key): adds key to collection
- key Top(): returns most recently-added key
- key Pop(): removes and returns most recently-added
- Boolean Empty: are there any elements?

Balanced Brackets:

input: A string str consisting of '(', ')', '[', ']' characters

output: Return whether or not the strings parenthesis and square brackets are balanced.

```
IsBalanced(str)
Stack stack
For char in str:
    if char in ['(', '[']:
        stack.Push(char)
    else:
        if stack.Empty(): return False
        top ← stack.Pop()
        if (top = '[' and char != ']') or
           (top = '(' and char != ')'):
            return False
return stack.Empty()
```

Stack Implementation with Array

~~num Elements~~: 2 → 3 → 2

a	b	c		

push(c)
pop() → c

push(d)
push(e)

Some operations: