Marks

Attn 5%

Quiz 15%

Lab 25%

Mid 20% (30 marks)

~~Di~~final 35% (50 marks)

MAT
NZU

Linear Array

Python → List append() ×
copy() ×
len() ✓

* List ≠ capacity
~~arr~~

initialization

array = [None] * capacity

* Array ≠ property:
contigous memory Allocation
↑ সারিবদ্ধভাবে থাকতে হবে, মাঝে ফাঁকা থাকতে না
* Random access memory

* def iteration(arr):
    for i in range(len(arr)):

~~card~~

```
def revIteration(arr):
    for i in range(len(arr)-1, -1, -1):
```

step → last index
step → index

array ⇒ random access

```
arr = [1, 3, 0, 2]
print(arr[[[1]]])
```

, ভিতরের step আগে

= arr[[3]]

= arr[2]

= 0

```
def copy(source):
    new_array = [None] * len(source)
```

× new_array = source } shallow copy করা হয়েছে location নিয়ে যাবে

```
    for i in range(len(source)):
        new_array[i] = source[i]
    return new_array
```

0, 6   1+1 = 2   2−1 = 1

* new_array = [None] * len(source)

capacity (len) change করে না তাহলে initialize করার সময় len declare করে দিবে

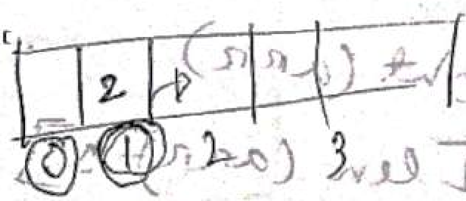
```
def resize(old_array, new_capacity):
    new_array = [None] * new_capacity
    for i in range(len(old_array)):
        new_array[i] = old_array[i]
    return new_array
```

যদি K বার shift করতেছি

```
def left_shift(arr):
    for j in range(k):
        for i in range(1, len(arr)):
            arr[i-1] = arr[i]
        arr[len(arr)-1] = None
    return arr
```

2, 3, 10, 12

3, 10, 12, None



2,3,10,12
3,10,12,2

* def rotateLeft(arr):

```
    temp = arr[0]
    for i in range(1, len(arr)):
        arr[i-1] = arr[i]

    arr[len(arr)-1] = temp
```

* ~~def ro~~

* def rightShift(arr):

```
    for i in range(len(arr)-1, 0, -1):
        arr[i] = arr[i-1]

    arr[0] = None
```

[1, 2, 3, 4]

1 পর্যন্ত যাবে লুপটা দিয়ে দেখবে

1, 2, 3, 4, None

* size	len(arr)
valid element	capacity
4	5

* def rotateRight(arr)

```
    temp = arr[len(arr)-1]
    for i in range(len(arr)-1, 0, -1):
        arr[i] = arr[i-1]
    arr[0] = temp
```


```
def insert(arr, size, elem, index):
    if size >= len(arr):
        print(error)
    else:
        for i in range(size, index, -1):
            arr[i] = arr[i-1]

        arr[index] = elem
        return arr

def remove(arr, index, size):
    for i in range(index+1, size):
        arr[i-1] = arr[i]

    return arr
```

geek for geeks
programiz

Circular Array

start idx
0

[1, 5, 7, 10, N, N]

start idx = 2

[7, 10, N, N, 1, 5]

```
def forward_iteration(cir_arr, start, size):
    k = start
    for i in range(size):
        print(cir_arr[k])
        k = (k+1) % len(cir_arr)
```

```
def backward_iteration(cir_arr, start, size):
    last_idx = (start + size - 1) % len(cir_arr)
    k = last_idx
    for i in range(size):
        print(cir_arr[k])
        k = k - 1
        if k < 0:
            k = len(cir_arr) - 1
```

[2, 5, 6, 1]
st = 3

[2, 5, 6, N, N, 1]

```
def linearize(cir_arr, start, size):
    lin_arr = [None] * size
    k = start
    for i in range(size):
        lin_arr[i] = cir_arr[k]
        k += 1
        k = (k+1) % len(cir_arr)
    return lin_arr

def circularize(lin_arr, start, size):
    cir_arr = [None] * size
    k = start
    for i in range(size):
        cir_arr[i] = lin_arr[k]
        k = (k+1) % len(cir_arr)
    return cir_arr
```

lin_arr = [1, 2, 5, 7, 8]

st = 2

cir_arr = [5, 7, 8, 1, 2]

* def resize (cir_arr, start, size, new_capacity):

    new_arr = [None] * (new_capacity)

    k = start

    for i in range (size):

        new_arr[i] = cir_arr[k]

        k = (k+1) % len(cir_arr)

input [2, 5, 7, 8]

st = 2, cap = 7

output [7, 8, 25, N, N, N]

front index — inserted elem

* def insert (cir_arr, start, size, elem, pos):

    if size == len(cir_arr):

        cir_arr = resize (cir_arr, ...) (start idx

    no of shifts = size - pos

    from = (start + size - 1) % len(cir_arr)

    to = (from+1) % len(cir_arr)

offset position

[2, 3, N, N, 10, 12]

[17, 2, 3, N, 10, 12]

```
for i in range (no of shifts):
    cir-arr[to] = cir-arr[from]
    to = from
    from = from - 1
    if from < 0
        from = len(cir-arr) - 1
idx = (start + pos) % len(cir-arr)
cir_arr[idx] = elem
size += 1
```

def remove

left shift

row col [5, 5]
col

import numpy as np

numpy array ⟶ [instead of list in all assignment]

arr = [0] * 5 → list

arr = np.array([0]*5) → np array

arr = np.zeros(5)

def fun(arr):

[function unchanged]

return arr ⟵ np array return হবে

Dimension

0 1

[[1,2,3], [4,5,6]] = 1 2 3
4 5 6

Len = 2

1-D 2D array (2 x 3)

arr = np.zeros((2,3))

[[0 0 0]
[0 0 0]]

Rahul Das

↗ row

```
for x in arr:
    for y in x:
        print(y)
```

00	01	02
1	2	3
4	5	6
10	11	12

output:

1
2
3
4
5
6

0, 1

```
for i in range(len(arr)):
    for j in range(len(arr[0])):
        print(arr[i][j])
```

* [10, 2, 5, 2, 7, 2, 8]

[10, 5, 7, 8, N, N, N]

remove-all(arr, num):

remove -

while ...

remove

left shift

[2, 5, 7, 10]

mean =

for i in range(len(arr)):

sum += arr[i]

mean = sum / len(arr)

$$SD = \sqrt{\frac{\Sigma(x-\mu)^2}{N-1}}$$

math.sqrt

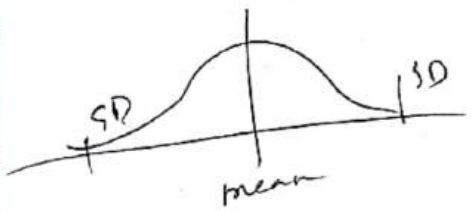
sum / (len(arr) - 1)

$\frac{\Sigma}{N} - 1$

* for i in range(len(arr)):

sumt = (arr[i] - mean(arr)) **2

SD = math.sqrt(sum / (len(arr) - 1))



high = mean + 2 SD

low = mean - 2 SD

n%2
2:

Input: [10, 20, 30, 40, 50, 60]

output: [high এর চেয়ে বড়, low এর চেয়ে ছোট]

- ~~দুইবার~~ check. ১মবার count

output = [0] * count

output: [. . . .]

2nd বার,

j = 0
output[j] = arr[i]
j += 1

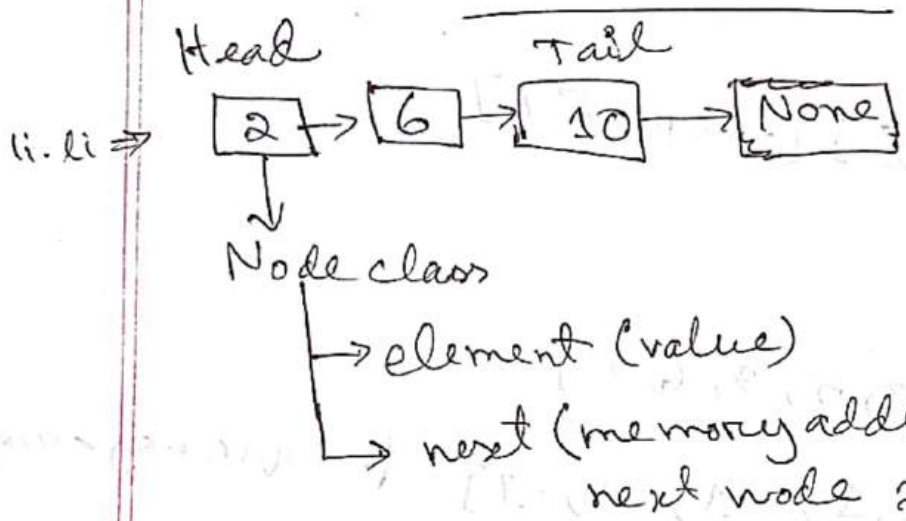
* input [3, 6, 3, 5, 3, 6, 3]

output [4, 2, -1, 1, -1, -1, -1] frequency array

```
def fun(arr):
    for i in range(len(arr)):
        frequency[i]=0

    for i in range(len(arr)):
        ~~count=0~~
        for j in range(i+1, len(arr)-1):
        count=0
        if frequency[i]==0:
```

```
if arr[i] == arr[j]:
    count += 1
    frequency[j] -= 1
frequency[i] = count
```

Linked List



array ⇒ [2, 6, 10]

Charact. of L.L.:

1. randomly element বসাতে পারে
2. Length বলে কিছু নাই
3. random access করা যায় না sequential access
4. Insert করা যায় easily

12

2 , 6 , 10 → 15 ⇒ 2 , 12 , 6 , 10

disturb the memory slot

linked list ১ disturb করে for insert.

```
class Node:
    def __init__(self, element, next = None):
        self.element = element
        self.next = next
```

Creation of linked list:

```
def createList(arr):
    head = Node(arr[0])
    tail = head
    for i in range(1, len(arr)):
        new = Node(arr[i])
        tail.next = new
        tail = new
    return head
```

```
# def iteration(head):
    temp = head
    while temp != None:
        print(temp.element)
        temp = temp.next

# def count(head):
    count = 0
    temp = head
    while temp != None:
        count += 1
        temp = temp.next
    return count

# def nodeAt(head, idx):
    count = 0
    temp = head
    while temp != None:
        if count == idx:
            return temp
        temp = temp.next
        count += 1
```

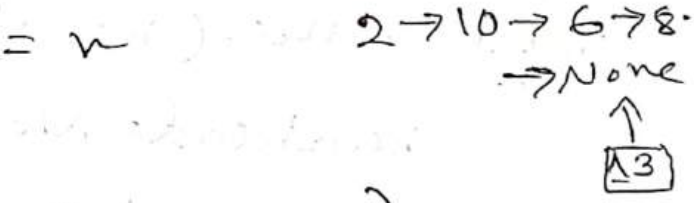
```
# def elementinsert(head, idx, value):
    count=0
    temp= head
    while temp!= None:
        if count == idx:
            temp.element = value
        temp = temp.next
        count += 1
# def sanity check:
    l= count(head)
    if idx < 0 or idx >= l
# def search(head, value):
    temp=head
    while temp! = None:
        if temp.element == value:
            return true
        temp= temp.next
    return False
```

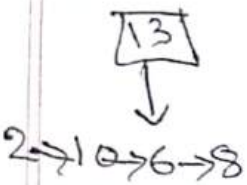
```
def search(head, value):
    temp = head
    count = 0
    while temp != None:
        if temp.element == value
            return count

        temp = temp.next
        count += 1

    return(-1)
```

Quiz/ Linear তাহলে 2টা, 2D তাহলে 3টা

~~Insert~~

```
def insert(head, element, idx):
    length = countNodes(head)
    if idx<0 or idx > length:
        print("Invalid index")
    elif idx==0:
        n = Node(elem, head)
        head = n
    elif idx==length:
        n = Node(elem, None)
        last = NodeAt(head, length-1)
        last.next = n
    else:
        n = Node(elem, None)
        n1 = NodeAt(head, idx-1)
        n2 = NodeAt(head, idx)
```



```
n.next = n2
n1.next = n
```

```
def remove(head, idx):
    if idx == 0:
        head = head.next
    elif idx >= 1 and idx < count(head):
        n1 = nodeAt(head, idx-1)
        removed_node = n1.next
        n1.next = removed_node.next
    else:
        print("Invalid")
```

```
def reverse(head):
    newhead = Node(head.element, None)
    temp = head.next
    while temp != None:
        n = Node(temp.element, newhead)
        new_head = n
        temp = temp.next
```

return new_head

```
def rotate_left(head):
    new_head = head.next
    temp = new_head
    while temp.next != None:
        temp = temp.next
    temp.next = head
    head.next = None
    head = new_head
    return head
```

tail determine

highest +1

freq = [0] * 7

~~arr.shape~~

np.(shape)(arr)

arr = np.ndarray(shape=(c,r), dtype=int)

.mod.abs

[(2,1), (2,-1), (-2,+1.)(-2,-1)

(1,2) (1,-2) (-1,2) (-1,-2)

$0 \leq i \leq 8$ $0 \leq j \leq 8$

sum of last k nodes of linked list:

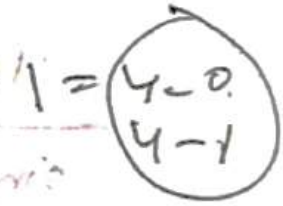
```
def sum_nodes(head, K):
    length = count_nodes(head)
    sum = 0
    for i in range(K):
        temp = nodeAt(head, length-1-i)
        sum = temp.element
    return sum
```

2 → 3 → 4 → 5

sum of first k nodes:

K = 2

```
def sum_first(head, K):
    sum = 0
    temp = head
    for i in range(K):
        sum = temp.element
        temp = temp.next
    return sum
```



(Alternative merge)

2 5 4
3 6 9

2 → 3 → 5 → 6 → 4 → 9

```
def merge(head1, head2):
    p_curr = head1
    q_curr = head2
    while p_curr != None and q_curr != None:
        p.next = p_curr.next
        q_next = q_curr.next
        p_curr.next = q_curr
        q_curr.next = p_next
        p_curr = p.next
        q_curr = q_next
        head2 = q_curr
    return head1, head2
```

Types of Linked List

Singly vs Doubly

S: 2 → 3 → 4 → N

D: N ← 2 ⇄ 3 ⇄ 4 → N

Linear vs circular

L: 2 → 3 → 4 → N

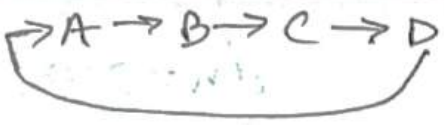
C: 2 → 3 → 4 (back to 2)

non-Dummy vs Dummyheaded

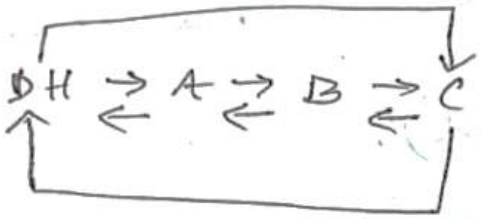
2 → 3 → 4 → N

DH → 10 → 2 → 3 → 4 → N

Change होता ना



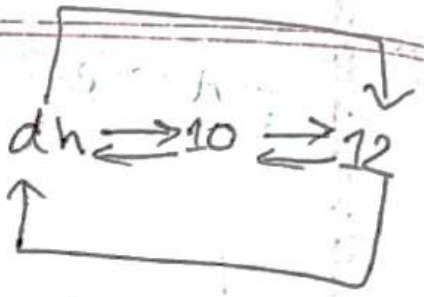
⇒ non-Dummy headed singly circular linked list



DH D C

```
class Node:
    def __init__(self, elem, next, prev):
        self.element = elem
        self.next = next
        self.prev = prev

def creation(arr):
    dh = Node(None, None, None)
    dh.next = dh
    dh.previous = dh
    tail = dh
    for i in range(0, len(arr)):
        n = Node(arr[i], dh, tail)
        tail = n
```


```
tail.next = n
tail = n
dh.prev = tail
```




```
def iteration(dhead):
    temp = dhead.next
    while temp != dhead:
        print(temp.element)
        temp = temp.next
```

```
def back_iteration(dhead):
    temp = dhead.prev
    while temp != dhead:
        print(temp.element)
        temp = temp.prev
```

Q. Non dummy head ~~linear~~ circular doubly

~~temp~~ কিভাবে handle করবে?

1st এর আগের print করব, আবার শুরুতে loop

Stack (ढेर)

LIFO data structure

↓

Last In First Out

3 function of stack

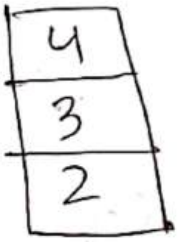
- push (elem) → insert
- pop(→ removes the top value
- peek(→ just returns the topvalue

5 ← top arr = [3, 4, 5]

4

3

arr = [3, 4, 5] → arr[len(arr)-1] (top)



push()
stack overflow

pop()
stack underflow

Linked List ⇒ 5 → 4 → 3 → N

↓

head = top

```
class &&stack:
    def __init__(self):
        self.top=None
    def push(self,elem):
        if self.top== None:
            self.top = Node(elem,None)
        else:
            n=Node(elem,None)
            n.next= self.top
            self.top =n
    def pop( ):
        if self.top==None:
            return None
        else:
            popped = self.top
            self.top=self.top.next
            popped.next = None
            return popped.element
```

```
def peek( ):
    if self.top == None: return None
else: return self.top.element
```

Postfix Expression Evaluation:

2 + 5 Infix

+ 2 5 prefix

2 5 + postfix

2 3 + 7 - 3 4 + *

5

-2

↳ - 2 3 4 + *

- 2 7 *

-14

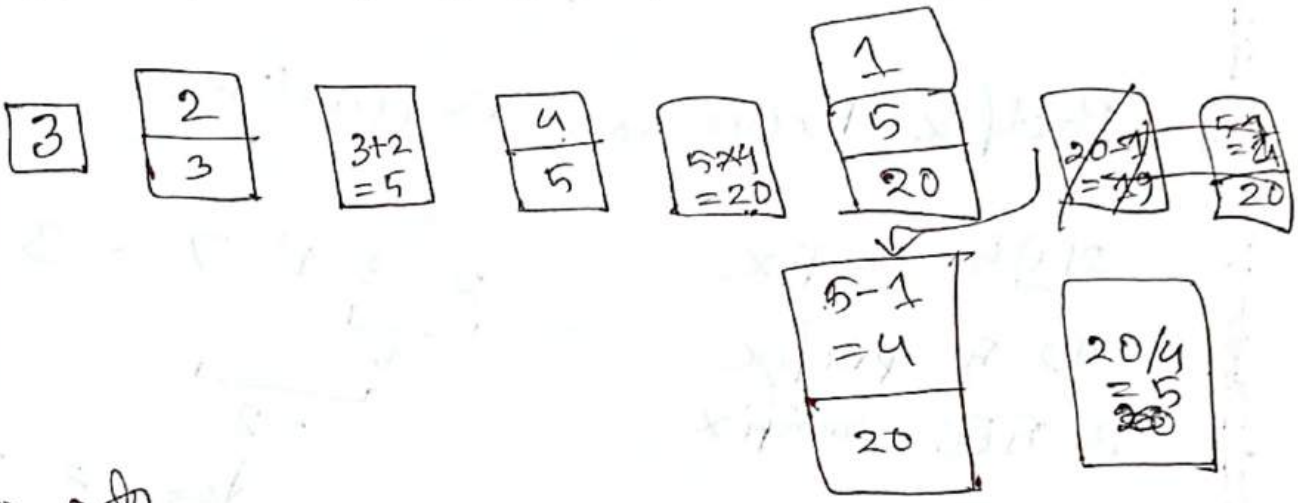
operator না পেলে push করতে থাকবে

operator পেলে top value pop করতে হবে দুইবার

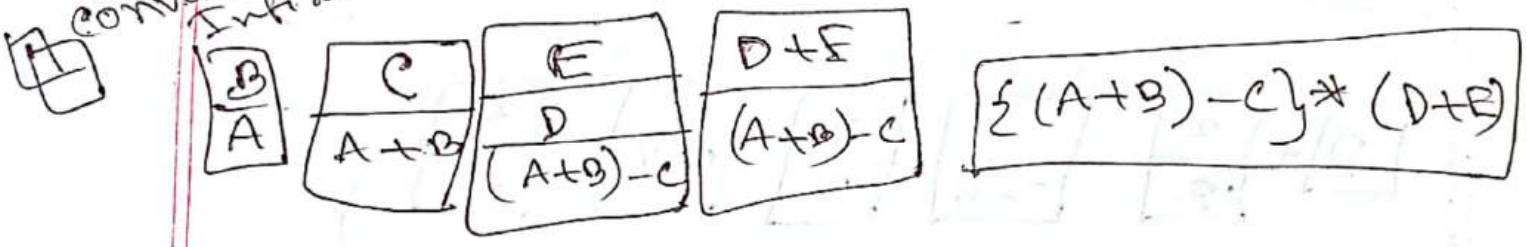
তারপর নিচেরটা operate করে উপরটা

9. Evaluation OR Infix এ convert করা

3 2 + 4 * 5 1 − /



converting to Infix



Paranthesis matching

[(a+b) − c }

{ [{a+b} − c] + d } ✓ correct

[{ (a+b) − c } − d]

[{ (a+b) − c } + d)

Loop চালিয়ে, প্রথমে parentheses গুলোর iterate
opening
first এ push করতে হবে, ~~যদি~~ closed পাওয়ার
পর ~~check pop~~ top এর POPP করে check
করতে হবে।

```
if st.POP() != None
   POP.element
def count(st, array):
c = 0
for exp in array:
    if exp == '<':
        st.push(exp)
    elif exp == ">":
        val = st.pop()
        if val != None:
            c += 1
```

><<a><bc>><<
c d>"
ef

<

len(--[i])

Queue

Opposite of stack

~~Queue~~ & ~~LIFO~~ Last in first out | ~~Stack~~ FIFO First In first out

Stack vs Queue

1 attribute (top)	2 attributes (front, rear/back)
push(elem), pop(), peek()	enqueue(elem), dequeue(), peek() → 3 basic operations
	enqueue(elem) → last or insert value
	dequeue() → front or remove front
	peek() → return front
Linear Array based, Linked list based	Circular Array based, Linked list based
	enq → back+1
	deq → front+1

```
def rotate(queue):
    for i in range(3):
        val = queue.dequeue()
        queue.enqueue(val)
```

inp: 2, 4, 8, 10, 12

out: 10, 12, 2, 4, 8

```
def ~~rotate~~evenen(queue):
    for i in range():
        val = queue.dequeue()
        if val % 2 == 0:
            queue.enqueue(val)
```

capacity of circular Q = 4

front/start = 3

	0	1	2	3
enq(E)				E f,b
enq(4)	4 b			E f
enq(L)	4	L b		E f
peek()	4	L b		E → return E f
deq()	4 f	L b		

LI $ AD # $ $ G $ A #

alphabet → enQ (alphabet)

$ → deq ( )

→ peek ( )

capacity = 4

start / front = 2

	0	1	2	3	
enQ(L)			L f b		
enQ(I)			L f	I b	
deQ()				I f b	
enQ(A)	A b			I f	
enQ(D)	A b	D b		I f	
peek()	A	D b		I f	return I
deq()	A f	D b			
deq()		D f b			
enQ(G)		D f	G b		
deq()			G f b		
enQ(A)			G f	A b	
peek()					→ return G

Linked list based

enQ(2) $2 \to N$ (f, b)

enQ(4) $2 \to 4 \to N$ (f, b)

enQ(6) $2 \to 4 \to 6 \to N$ (f, b)

deQ() $4 \to 6 \to N$

return element

এখানে সবচেয়ে আগে add করা element remv হচ্ছে

stack

$4 \to 2 \to N$

$6 \to 4 \to 2 \to N$

pop()

6 = top (last added)

এখানে last এর element remv হচ্ছে

Queue

* Linked list based implementation

stack

enq(2)	2 → N f, b	
enq(4)	2 → 4 → N f b	4 → 2 → N
enq(7)	2 → 4 → 7 → N f b	
deQ( )	4 → 7 → N f b	
peek( )	4 → 7 → N f b returns 4	

~~def enqueue(elem):~~

```
class Queue:
    def __init__(self):
        self.front = None
        self.back = None

    def enqueue(elem):
        if self.front == None:
            self.front = Node(elem, None)
            self.back = self.front
```

```
        else:
            self.back =
            n = Node(elem, None)
            self.back.next = n
            self.back = self.back.next

    def dequeue( ):
        if self.front == None:
            print("Underflow")

        else:
            dequeued_value = self.front.element
            self.front = self.front.next
            return dequed_value

def peek( ):
    if self.front == None:
        print ("Underflow")
    else:
        peek_value = self.front.element
        return peek_value
```

* Circular array based Queue

```
Class Queue:
    def __init__(self):
        self.front = 0     --> (start index যেকোনো জায়গা থেকে হতে পারে)
        self.back = 0      -->
        self.queue = np.zeros(15)
        self.size = 0

    def enqueue(elem):
        if self.size == len(self.queue):
            print("Overflow")

        else:
            self.queue[self.back] = elem
            self.back = (self.back + 1) %
                              len(self.queue)
            self.size += 1

    def dequeue(self):
        if self.size == 0:
            print("Underflow")
```

```
        else:
            dequeued_value = self.queue[self.front]
            self.queue[self.front] = None
            self.front = (self.front + 1) % len(self.queue)
            self.size -= 1
            return dequeued_value
    def peek(self):
        if self.size == 0:
            print("Underflow")
        else:
            peek_value = self.queue[self.front]
            return peek_value
```

input: 2 7 16 8 5
output: 7 16 8 5 2

solve by queue

⇒ value = Q.dequeue()
Q.enqueue(value)

output: 8 5 2 7 16

```
for k in range(3):
    value = Q.dequeue()
    Q.enqueue(value)
```

output: 7 5

```
for k in range(~~size~~self.size):
    value = Q.dequeue()
    if value % 2 != 0:
        Q.enqueue(value)
```

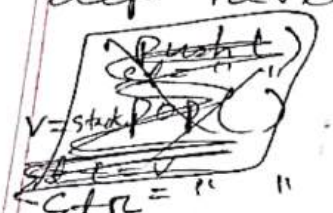
Linked list example

input: 2 → 4 → 6 → 10 → 6 → 8

output: 2 → 4 → 10 → 8

```
value = Q.dequeue()
if value != given_value:
    Q.enqueue(value)
```

* Input: Rafiq using stack (push, pop & func)
output: qifar
def reverse(string, stack):



for i in string:
    stack.push(i)
while stack.isEmpty() == False
    v = stack.pop()
    str += v
return str

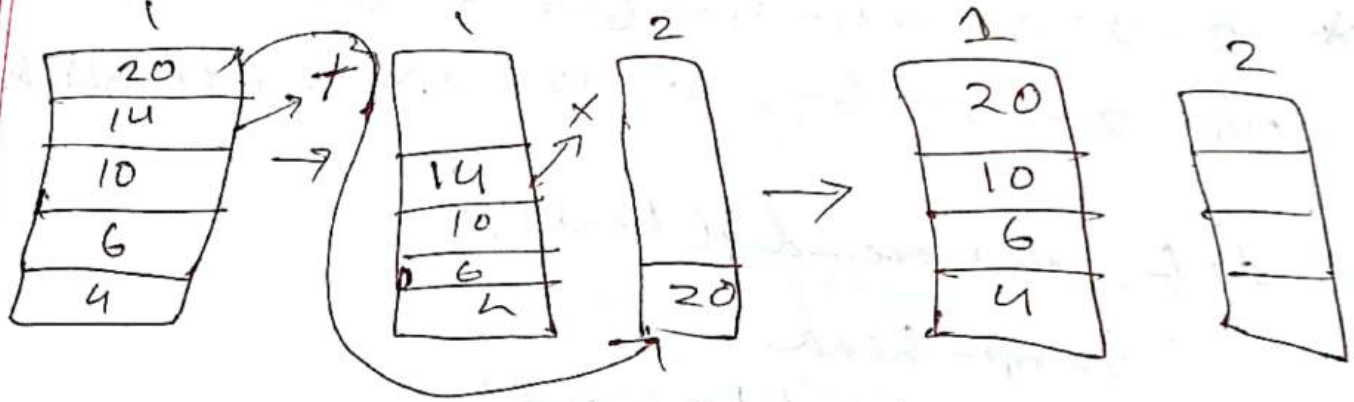
[def isEmpty(self):
    if self.top == None:
        return True
    else:
        return False

* Input : 4 6 10 14 20 using stacks

n = 2

output : 4 6 10 20

pop (1st) → push in another stack → remove the 2nd elem from end → do not push it → pop from that stack → push in given stack




```
def removeN(st,n, arr):
    temp = st
    i = 0
    for elem in arr:
        st.push(elem)
    while i < n:
        temp.push(st.pop())
        i += 1
    st.pop()
    while temp.isEmpty!=True: st.push(temp.p
```

* input : 3 , 10, 13, 17, 18 ⊠ using (enq(), deq())
output : 10, 18

#deq করে enq না করা

```
v = queue.deQ()
if V % 2 == 0:
    queue.enqueue(v)
```

* 2 → 2 → 2 → 5 → 5 → 6 → 10 → 10 → 10 → 12 → N

out: 2 → 5 → 6 → 10 → 12 → N (using Linked list)

```
def remove_dup(head):
    temp = head
    while temp != None:
        if temp.next.elem == temp.elem:
            val = temp.elem
            temp.next = temp.next.next
            while val == temp.next.elem:
                temp.next = temp.next.next
        temp = temp.next
    return head
```

```
def removeAll(head, n):
    idx = 0
    temp = head
    length = countNodes(head)
    tail = nodeAt(head, length-1)
    while temp != None:
        Next = temp.next
        if temp.elem == n:
            if temp == head:
                head = head.next
            elif temp == tail:
                n1 = nodeAt(head, countNodes(head)-2)
                n1.next = None
            else:
                n1 = nodeAt(head, idx-1)
                n1.next = temp.next
        else:
            idx += 1
        temp = Next
    return head
```

arrays, linked list থেকে code

Stack
Queue
> code + simulation

For Basic short questions at lecture notes.

*linked list এর merits:
*fixed size নয়, insert করলে shift করা লাগে না

(a) ~~to~~ Why Not success

(b)

Lab

- **reverse**

10 → 20 → 30 → 40 → None

↑
Head

new linked list ⇒ None → 40 → 30 → 20 → 10

new_node.next = head
head = new_node

array stack

10	20	30	40

size হিসাবে রাখতে হবে push() করার জন্য

pop() করার জন্য size = 0 কিনা দেখতে হবে

Queue 7
Stack 8
DHL 5
LinL 12
Array 15
CiAr 7

```
for char in string:
    if char=='<':
        stack.push(char)
    elif char=='>':
        if ...:
            pass
        else:
            pop=stack.pop()
            c+=1
```

AFTER MID

⑧ Recursion

```
def func1(param):
    - - - -
    - - - -
    func1(param1)
```

বারবার চলছে এমন function call হচ্ছে এর ভিতর

```
def recurr(param):
    if base case
    যে condition এর
    কাজ বন্ধ হবে
    else:
        recursive case
        যে condition এর জন্য
        চলতে থাকবে
```

* বড় Problem গুলোকে solve না করে, ছোট ছোট করে solve
  Overlapping subproblem

*
```
def factorial(n):
    if n==1 or n==0:
        return 1
    else:
        n*factorial(n-1)
```

Recursion tree:

4! => fact(4)

4 * fact(3)

↓

3 * fact(2)

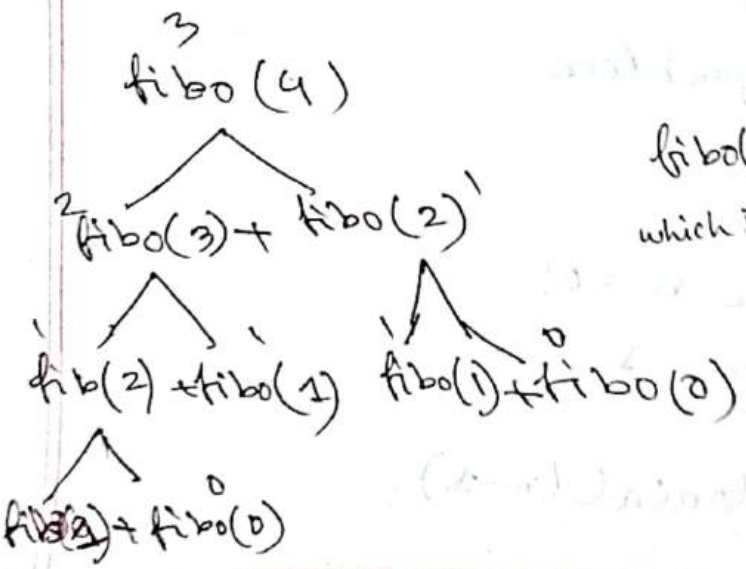
↓

2 * fact(1)

Fibonacci series

0 1 1 2 3 5 8 13

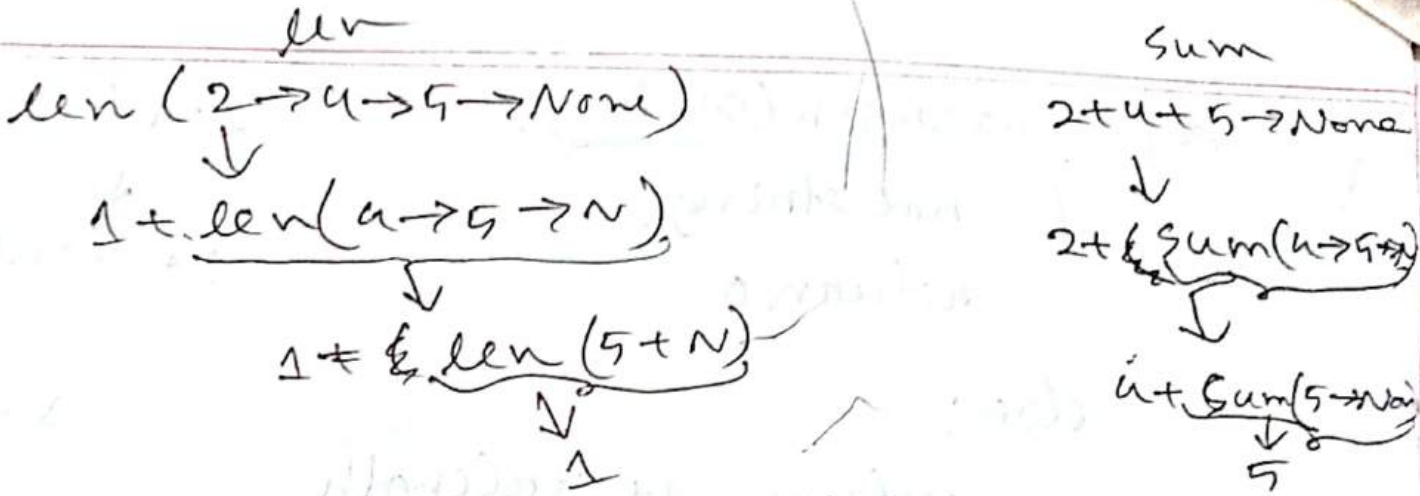
```
def fibo(n):
    if n==0 or n==1:
        return n
    else:
        return fibo(n-1) + fibo(n-2)
```



fibo(2) 2বার call করা হচ্ছে which is redundant.

So, fibo(2) এর value store করে রাখলে can be used later.

This is called memoization




```
def length(head):
    if head.next == None:
        return 1
    else:
        return 1 + length(head.next)

def sum(head):
    if head.next == None:
        return head.elem
    else:
        return head.elem + sum(head.next)
```

```
def strlength(string):
    if not string:
        return 0
    else:
        return 1+strlength
                (string[1:])
```

len(ABC)
↓
1+ len(BC)
↓
1+ len(C)
↓
1+ len("")

```
def printN(n):
    if n>5:
        return
    else:
        print(n) → python ke print
        printN(n+1) → recursive func
        return
```

Input: PrintN(1)
output: 1 2 3 4 5

5
4

```
def printN(n):
    if n>5:
        return
    else: printN(n+1)
          print(n)
          return
```

input same
output: 5 4 3 2 1

```
def printList(head):
    if head == None:
        return
    else:
        print(head.elem)
        printList(head.next)
```

উল্টা print করতে এরকম

Sequential search (LinkedList)

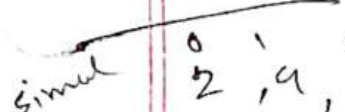
```
def seq_search(head, key):
    if head == None:
        return False
    elif head.elem == key:
        return True
    else:
        return seq_search(head.next, key)
```

Arrays version search

```
def seq_search(arr, key, left):
    if left >= len(arr):
        return False
    elif arr[left] == key:
        return True
    else: return seq_search(arr, key, left+1)
```

→ sorted array

```
def binary_search(arr, key, left, right):
    if left > right:
        return False
    mid = (left + right) // 2
    if arr[mid] == key:
        return True
    elif key > arr[mid]:
        return binary_search(arr, key, mid+1, right)
    else:
        return binary_search(arr, key, left, mid-1)
```



```
0  1  2  3  4  5  6  7
2, 4, 5, 7, 9, 10, 12, 14      Key = 12
l=0                    r=7

mid = (l+r)//2
    = 3
arr[mid] = 7
arr[mid] < key
l = mid+1 = 4
r = 7
mid = (4+7)//2 = 5
arr[mid] = 10
l = mid+1 = 6
r = 7
mid = 6
arr[mid] = 12
```

1. Find Maximum in a Linked list

```
def findMax(head):
    if head.next == None:
        return head.elem
    elif
    else:
        maxRest = findMax(head.next)
        return max(head.elem, maxRest)
```

2. MAX value in arr

```
def findMax(arr, left, right):
    if left == right:
        return arr[left]
    else:
        mid = (left + right)//2
        maxLeft = find_max(arr, left, mid)
        maxRight = find_max(arr, mid+1, right)
        return max(maxLeft, maxRight)
```

$3^4 = 3 \times 3^3$

$2^{100} \; 2^{99}$

$2^{50} \times 2^{50}$

$2^{25} \times 2^{25}$

```
def exp(a, n):
    if n == 0:
        return 1
    else:
        ~~product = 1~~
        ~~for~~
        ~~return a * exp(a, n-1)~~
        return a * exp(a, n-1)
```

```
def exp(a, n):                    [efficient for larger power]
    if n == 0:
        return 1
    elif n % 2 == 0:
        temp = exp(a, n/2)
        return temp * temp
    else:
        temp = exp(a, n/2)
        return a * temp * temp
```

$2^{25} = 2 \times 2 \times 2 \times 2 \times 2$

```
def sum(n):
    if n==1:
        return 1
    else:
        return n+ sum(n-1)
```

```
def harm(n):
    if n==1:
        return 1
    else:
        return 1/n+harm(n-1)
```

harm(3) = 1 + $\frac{1}{2}$ + $\frac{1}{3}$

↓

$\frac{1}{3}$ + harm(2)

↓

$\frac{1}{2}$ + harm(1)

Some recursion problem

sum of digits

2 4 1 → 7

2 4 1 % 10 = 1

2 4 1 // 10 = 2 4

2 4 % 10 = 4

2 4 // 10 = 2

```
def sum(num):
    if num < 10:
        return num
    else:
        return (num % 10) + sum(num // 10)
```

sum(241) 7

1 + sum(24) 6

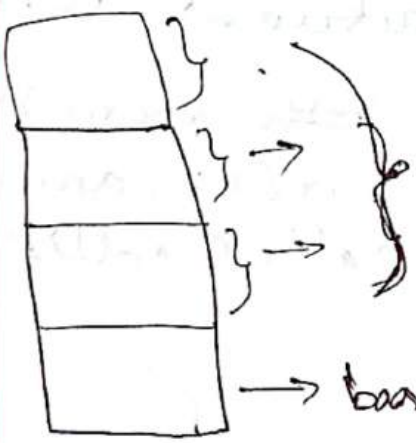
sum(2) 2

in: 4 → 3 → 7 → 10 → 2 → None

out: 4 + 10 + 2 = 16

```
def even_sum(head):
    if head == None:
        return 0
    else:
        if head.elem % 2 == 0:
            return head.elem + even_sum(head.next
```

else:
    return even_sum(head.next)



each level এগোতে ৫টা করে extra brick লাগে

base বানাতে ৮টা brick লাগে

def brick (~~level~~):
    if level == 1:
        return 8
    else:
        return 5 + brick(level-1)

~~8 + brick(3)~~ brick(9)

5 + brick(3)

5 + brick(2)

5 + brick(1)

Hashing

(key, value) → Pair

(Name, "ABC")

(Name, "DEF")

(Name, "BCF")

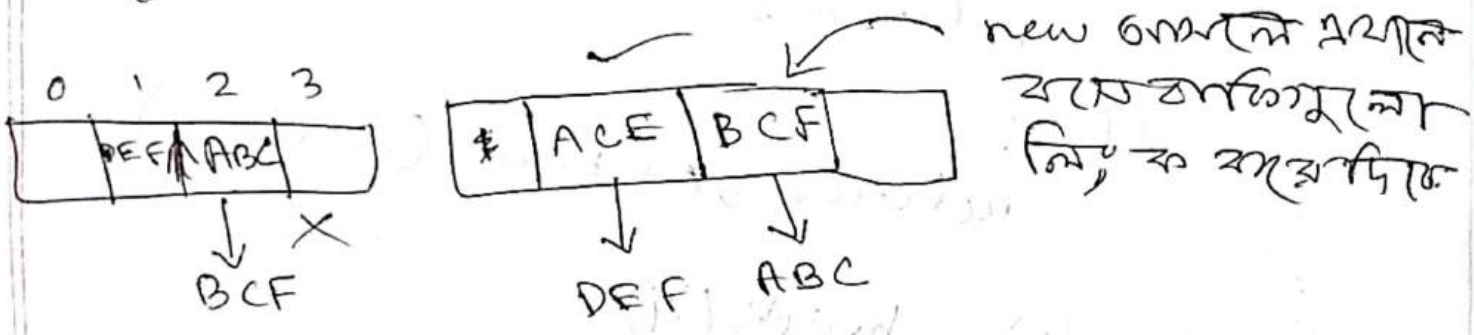
Hash function returns index

when more than one index are same its called collision.

Soln for collision:

forward chaining

linear probing ⟹ The next available index ১ যাওয়া



key, value both insert করতে হবে

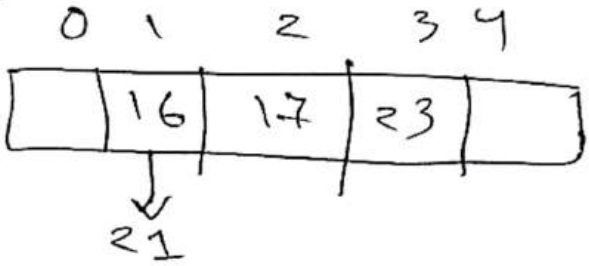
hash function

$h(k) = k \% 5$ ← 5 is length / size

[21, 16, 23, 17]

↓ ↓ ↓ ↓

1 1 3 2



h(K) = ASCII sum of character % 4

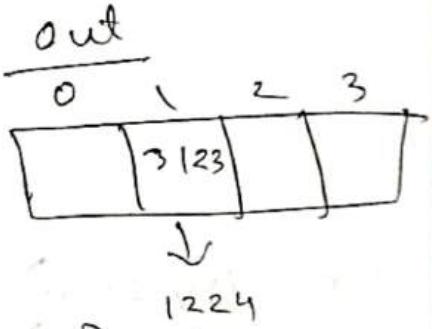
*When K is a string

** hash func use করে hash table তৈরি করা।

if K is odd:
    h(K) = count of even digits

else:
    h(K) = count of odd digits

[2137, 1224, 3123]
 ↓   ↓   ↓
 1   1   1



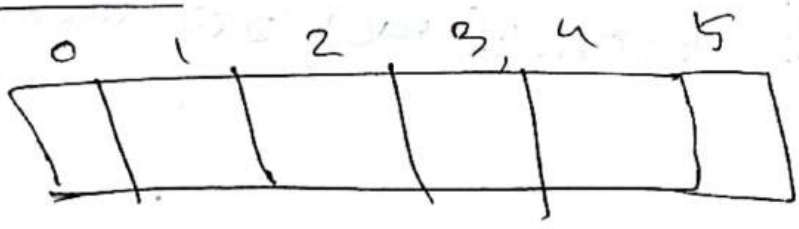
*Hash table len বলা লাগে, mod দ্বারা করে

এক্ষেত্রে len(arr) হবে minimum length

```
def hash(key):
    return key%5

def forward_chaining(arr):
    hashTable = [None]*5
    for elem in arr:
        idx = hash(elem[1])
        if hashTable[idx] == None:
            hashTable[idx] = Node(elem, None)
        else:
            prev = hashTable[idx]
            hashTable[idx] = Node(elem, prev)
```

arr = [(ID, 2130)
(ID, 1245)
(ID, 2100)
(ID, 1267)]

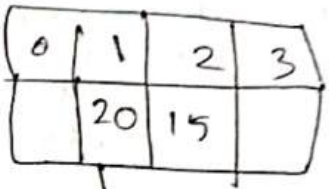
output:



⋮

```
def search(elem):
    idx = hash(elem[1])
    node = hashTable[idx]
    while node != None:
        if node.elem == elem[1]:
            return node.elem, True
        node = node.next
    else
```

17 given
returned
1



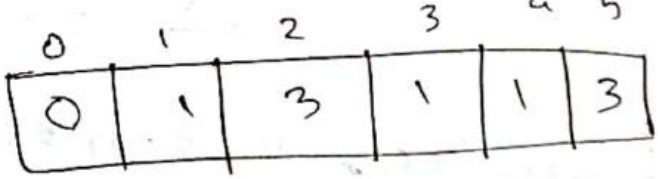
10 → 17 → 21 → None

find (17)

Key indexing

[5, 2, 5, 3, 1, 5, 2, 2, 4]

Auxiliary Array:



SQ: Key index & Hashing pros, cons

↓

memory ~~error~~

↓

easy to search/sort

search: arr[4] != None:

sort: 2 loops, elem এর পর index print.

1, 2, 2, 2, 3, 4, 5, 5, 5

[5, -2, 5, -3, 1, 5, 2, 2, 4]
8 1 8 0 4 8 5 5 7

5 + 1 - (-3) = 9
max+1 - min → size এর array

0	1	2	3	4	5	6	7	8
1	1	0	0	1	2	0	1	3

সবগুলো 3 ঘর right shift করে

search: arr(elem+3) != None:

sort: -3, -2, 1, 2, 2, 4, 5, 5, 5 [print(elem - shift)

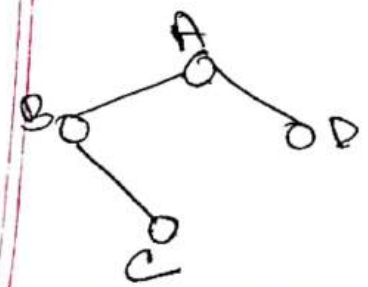
shift = min * (-1) = -3 * (-1) = 3

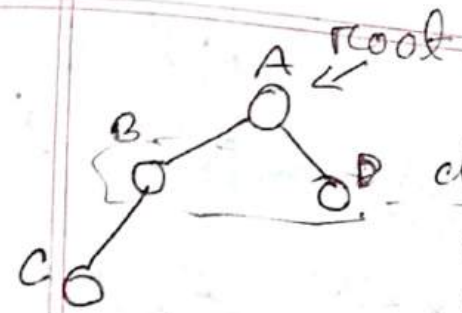
Tree

vertices, V = {A, B, C, D} → Nodes

Edges, E = {(A, B), (B, C), (A, D)}

Tree => 1. connected
2. Acyclic





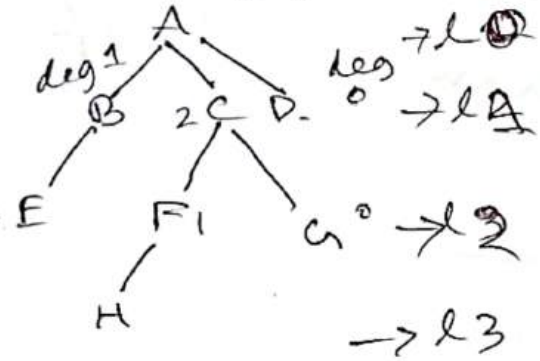
children
B, D
Siblings

C → leaf
(child নাই)

non-terminal
internal nodes

উপর থেকে নিচে নেমে
Directed.
A → B

Degree → নিচের দিকে কয়টা Node directly connected

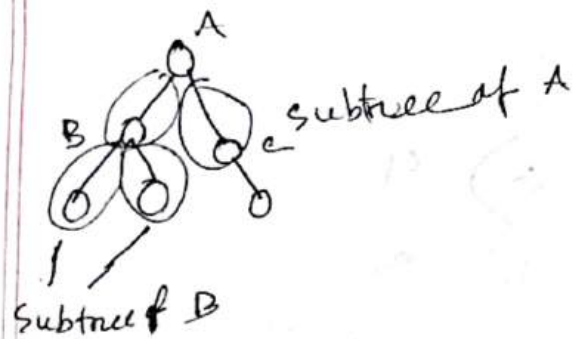


Depth / level : Node থেকে root পর্যন্ত যতটা edge পাবে তত level

E → level 2

H → 3

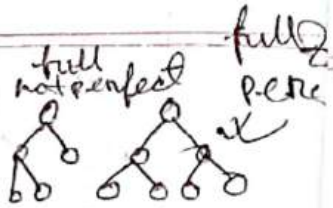
Weight : Node থেকে leaf পর্যন্ত



height of the tree
= height of root
= max depth / level of leaf node

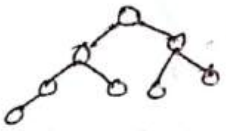
Binary tree ⇒ M-array tree ⇒ M = 2
কোনো node এর highest 2 টা ছাড়া বেশি সন্তান থাকতে পারবে না

4 types of BT:



Full/strict ⇒ প্রত্যেক node এর 2টি children অথবা 0 থাকবে

complete ⇒ left থেকে fillup হবে



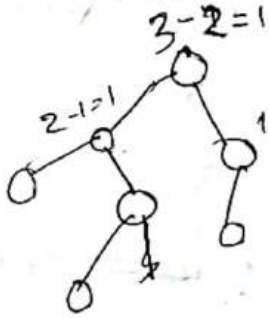
perfect ⇒ সব same level এ থাকবে

Balanced

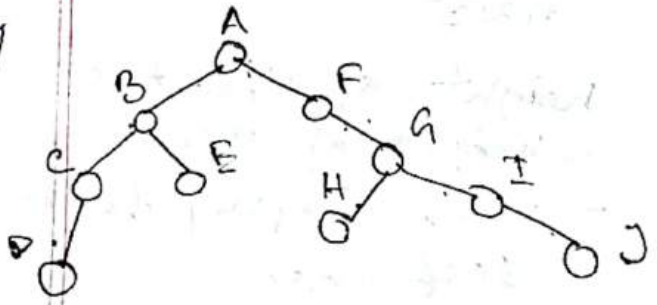
↓

$df = |$height of left child − height of right child$|$

not more than 1 [$df <= 1$]



⇒ balanced



a) height of tree

b) depth of (H)

c) Full, perfect, complete balanced?

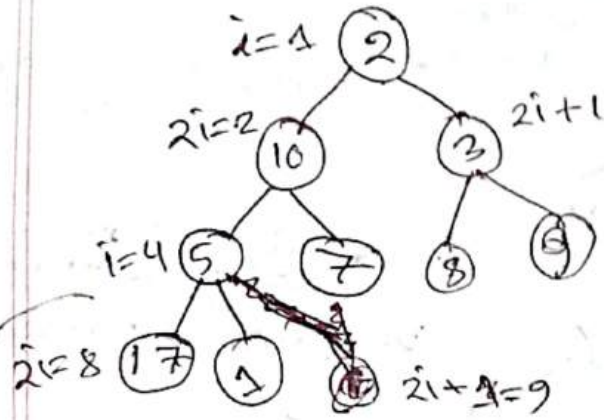
a) 4

b) 3

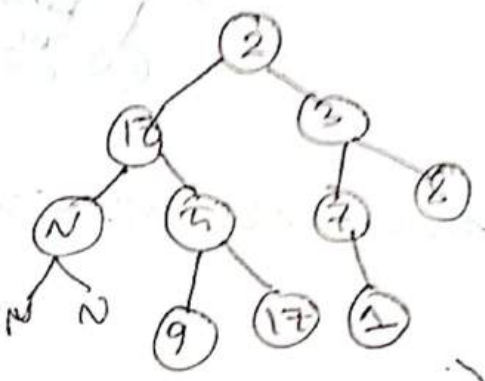
c) Not full, not perfect, not complete, not balanced

Array থেকে Binary Tree creation

N 2, 10, 3, 5, 7, 8, 9, 17, 1



2, 10, 3, None, 5, 7, 8, N, N, 17, 1



Tree traversal:

- In-order → Left → Root → Right
- Pre-order → Root → left → Right
- Post-order → Left → Right → Root

Pre order 2, 10, 5, 9, 17, 3, 7, 1, 8

In order 10, 9, 5, 17, 2, 1, 7, 3, 8

Post-order 9, 17, 5, 10, 1, 7, 8, 3, 2

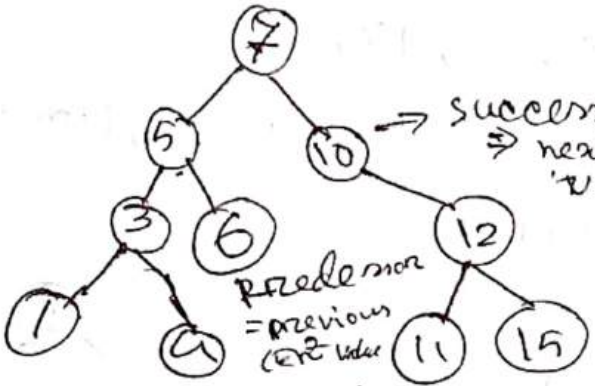
In order 17, 5, 1, 10, 7, 2, 8, 3, 9

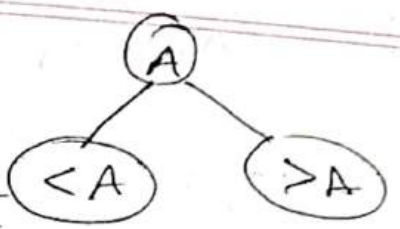
Post order: 17, 1, 5, 7, 10, 8, 9, 3, 2

Binary search Tree (BST)

7, 5, 10, 12, 3, 4, 11, 15, 1, 6

7 always root



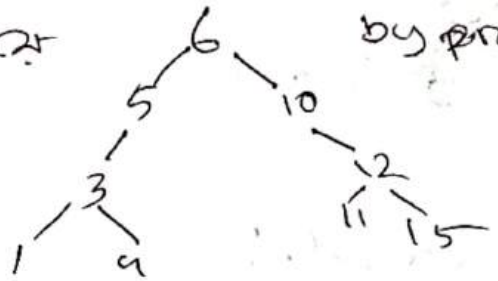


→ successor ⇒ next to value

→ lowest value at right subtree

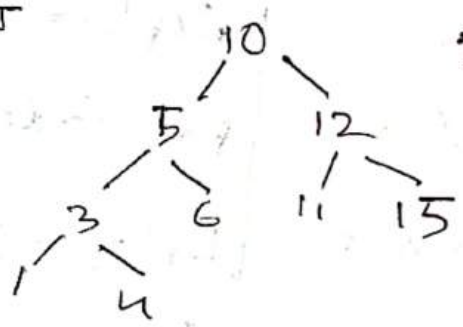
→ highest value at left subtree

যদি 7 ডিলিট করতে হয়



by predssor

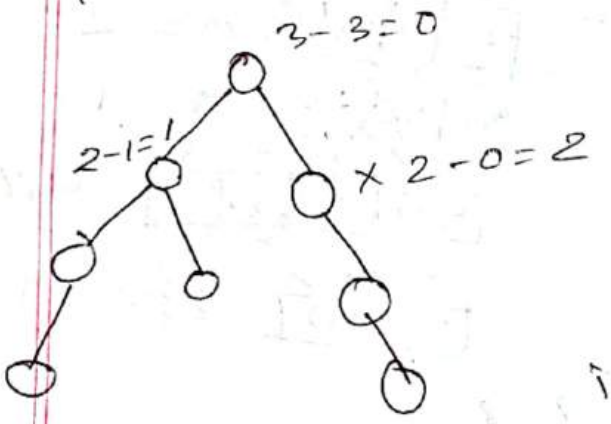
যদি 7 ডিলিট করতেই হয়

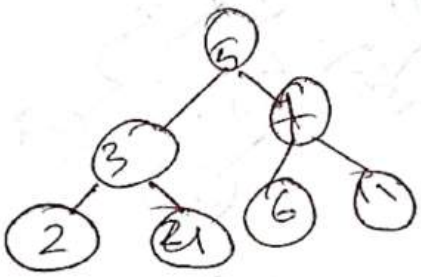


by successor

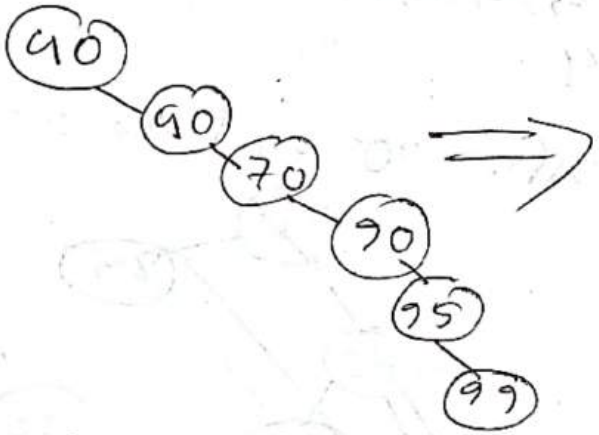
Tree Balancing

[height of difference of left and right child = 0/1]

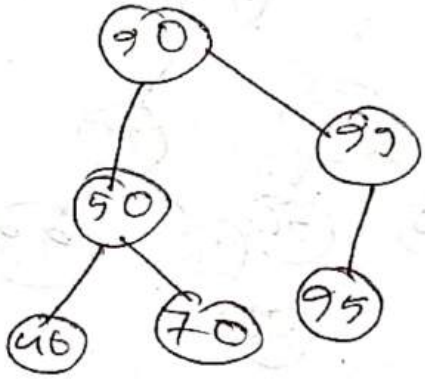




inorder 2, 3, 4, 5, 6, 7, 11



Final ans tree

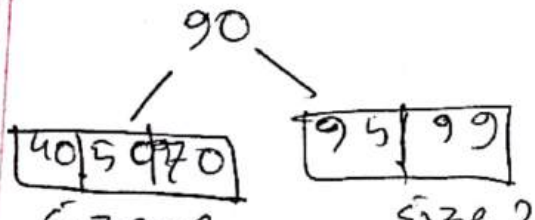


inorder traversal:

40,	50,	70,	90,	95,	99

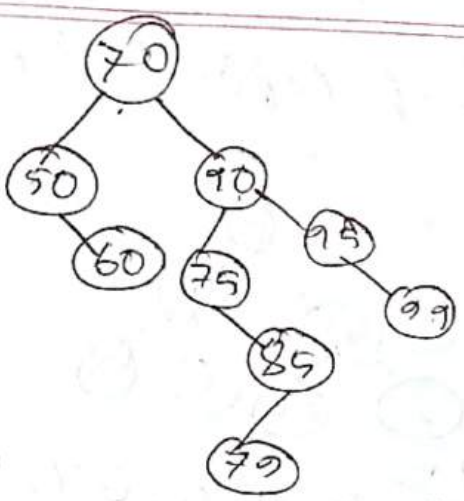
This will be sorted always

size = 6 ; root position = 6/2 = 3



size = 3 size 2

root = 3/2 = 1 root = 2/2 = 1



left

0	1	2	3
50	60	70	75

4/2 = 2

0	1
50	60

2/2 = 1

0	1	2	3
85	90	95	99

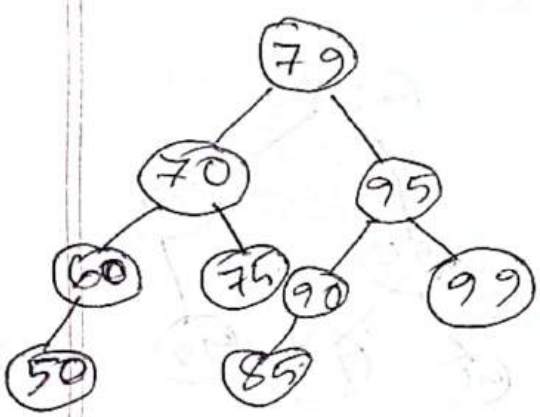
4/2 = 2

0	1
85	70

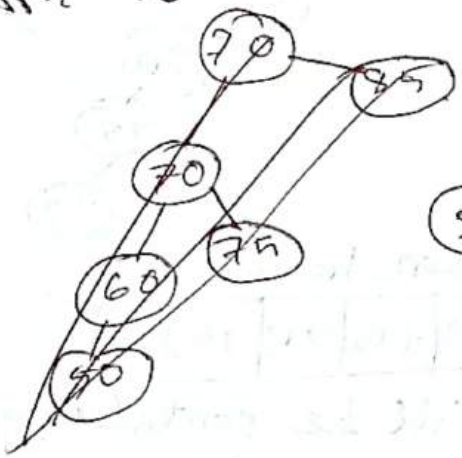
2/2 = 1

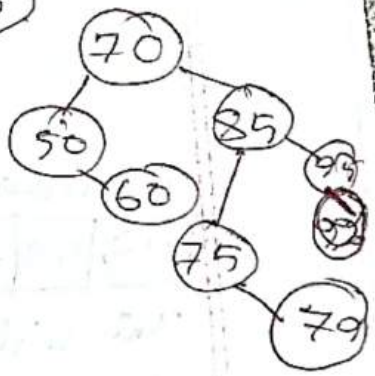
50, 60, 70, 75, 79, 85, 90, 95, 99

size=9 ; root = 9/2 = 4



যদি 90 বাম দিকে যায় predecessor





```
class Node:
    def __init__(self, elem):
        self.elem = elem
        self.left = self.right = None

def tree_construction(arr, i):
    root = None
    n = len(arr)
    if i < n:
        if arr[i] != None:
            root = Node(arr[i])
            root.left = tree_construction
                                (arr, 2i)
            root.right = tree_construction
                                (arr, 2i+1)
    return root
```

```
def in_order(root):
    if root != None:
        in_order(root.left)
        print(root.elem, end=" ")
        in_order(root.right)
def pre_order(root):
    if root != None:
        print(root.elem, end=" ")
        pre_order(root.left)
        pre_order(root.right)
def post_order(root):
    if
```

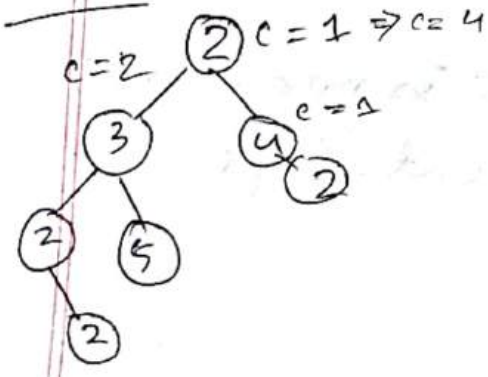
left
right
root.

~~def~~ get number of nodes

```
def nodes(root):
    if root == None:
        return 0
    else:
        return 1 + nodes(root.left) + nodes(root.right)
```

```
def height(root):
    if root == None:
        return 0
    else:
        return 1+max(height(root.left), height(root.right))

def key_count(root, key):
    if root == None:
        return 0
    else:
        count=0
        if root.elem == key:
            count+=1
        count+= key_count(root.left, key)
        count+= key_count(root.right, key)
```

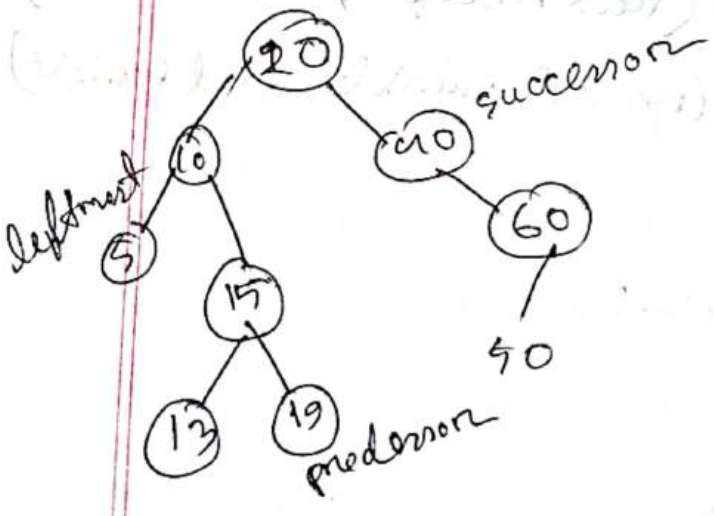
key=2



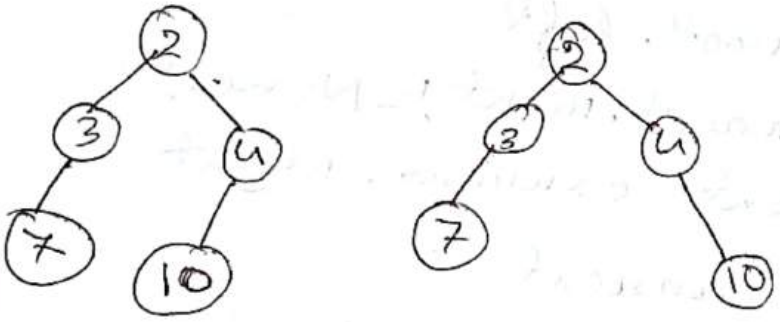
BST Insertion

```
def add(root, elem):
    if elem < root.elem and root.left == None:
        n = Node(elem)
        root.left = n
    elif elem > root.elem and root.right == None:
        n = Node(elem)
        root.right = n
    if elem < root.elem and root.left != None:
        add(root.left, elem)
    elif elem > root.elem and root.right != None:
        add(root.right, elem)

def leftmost_leaf(root):
    current = root
    while current.left is not None:
        current = current.left
    return current
```

```
def predesor (root):
    current = root.left
    while current.right != None:
        current = current.right
    return current
def successor (root):
    current = root.right
    while current.left != None:
        current = current.left
    return current
```

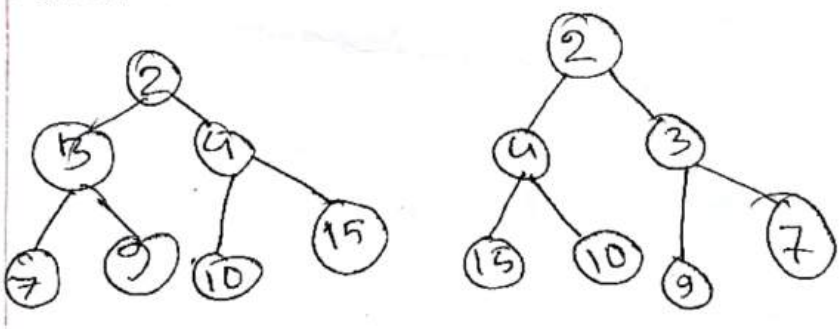


10. Check tree are Identical or not:




```
def identical (rootA , rootB):
    if rootA ==None and rootB ==None:
        return True
    elif rootA != None and rootB != None:
        return (rootA.elem == rootB.elem) and
                identical (rootA.left , rootB.left)
               and identical (rootA.right , rootB.right)

    return False
```

11. Convert mirror of the tree:



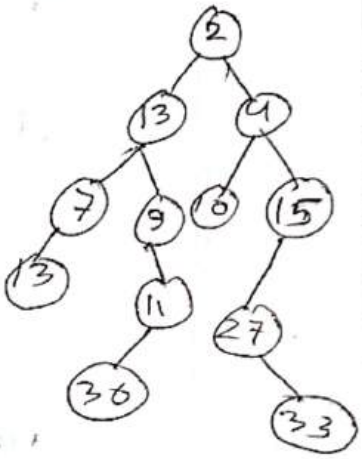
```
def mirror(root):
    if root == None:
        return
    else:
        new_node = Node(root.elem
        new_node.left = mirror(root.right)
        new_node.right = mirror(root.left)
    return new_node
```

4. Check the tree is balanced or not

```
def height(root):
    . . . . .
    . . . . .

def isBalanced(root):
    if root == None:
        return True
    lh = height(root.left)
    rh = height(root.right)
    if abs(lh - rh) <= 1 and isBalanced(root.left)
        == True and isBalanced(root.right) == True:
        return True
    return False
```

Find the largest value in each level:

```
def largest_level(root):
    return helper(root, d={}, l=0)

def helper(root, d={}, l=0):
    if root == None:
        return None
    if l not in d or root.elem > d[l]:
        d[l] = root.elem
    helper(root.left, d, l+1)
    helper(root.right, d, l+1)
    return d
```



output:

d = {0:2, 1:13, 2:15, 3:27, 4:36}

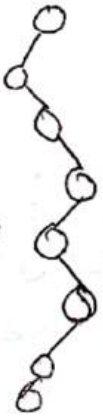
Check Duplicate value:

```
def duplicate(root):
    s = set()
    return helper(root, s)

def helper(root, s):
    if root == None:
        return False
    if root.elem in s:
        return True
    s.add(root.elem)
    return helper(root.left, s) and helper(root.right, s)
```

3. Print all nodes that are at k distance from root

```
def printNodes(root, k):
    if root == None or k<0:
        return
    if k==0:
        print(root.elem)
        return
    printNodes(root.left, k-1)
    printNodes(root.right, k-1)
```

4. Skewed Binary Tree:

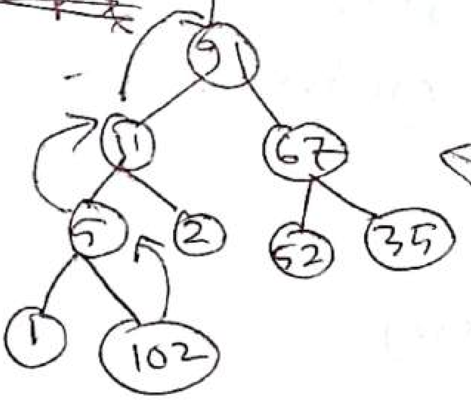


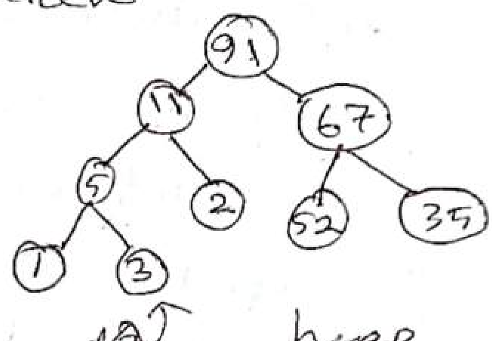
```
def skew(root):
    if root== None or (root.left==None
        and root.right==None):
        return True
    if root.left and root.right:
        return False
    If root.left:
        return skew(root.left)
    return skew(root.right)
```

Heap → Complete binary tree

Max heap (Mostly used) → value of parent must be greater than value of children

~~min heap~~



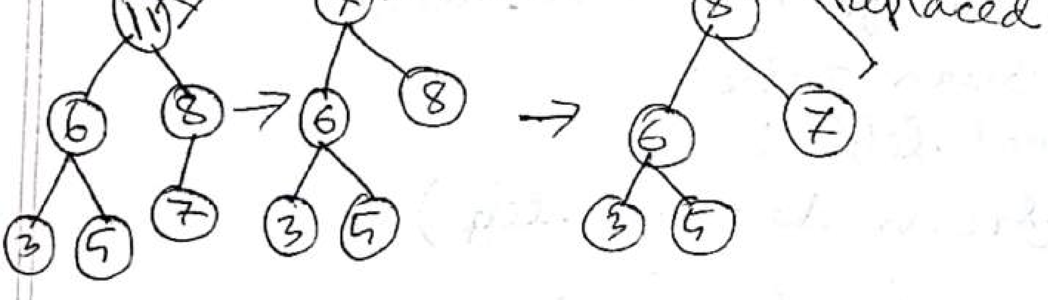


⇒ heap property satisfied

But 102 insert করলে heap property violate করবে তখন heap increasekey() or swim() method use করবো

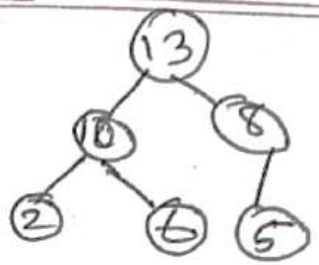
হবে 102 Node টা swim করে করে tree এর একদম শিরোভাগে যাবে, since max heap

Delete: In heap randomly node delete করা যায় না root delete করতে হয়। Replace the last node with root, then sink() method করে ঐ root টা tree এর last এ নিয়ে যেতে হয়. sink() ⇒ max heapify()



Heap sort (delete n times)

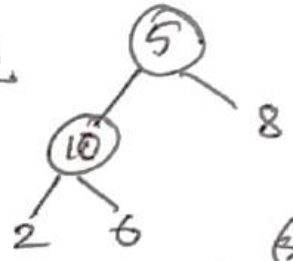
None, 13, 10, 8, 2, 6, 5



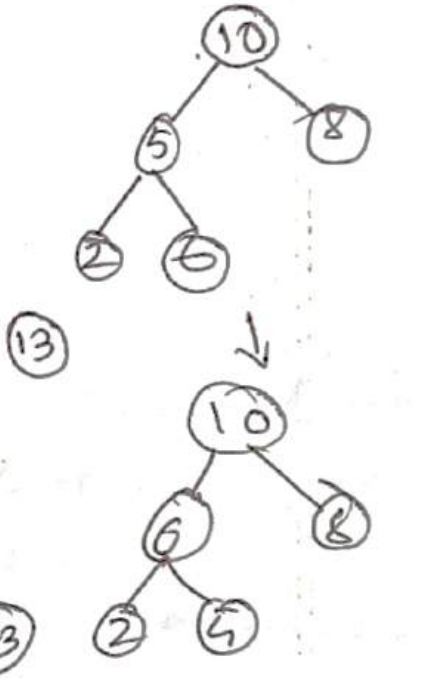
delete 10

None	8	6	5	2	10	13

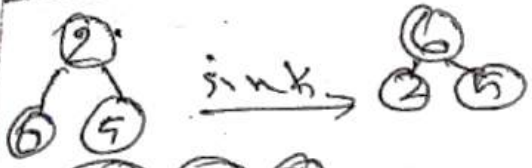
delete 13



sink

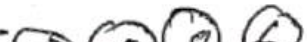


delete 8



None	6	2	5	8	10	13

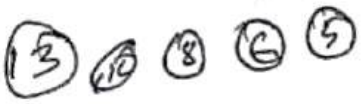
del 6



None	5	2	6	8	10	13

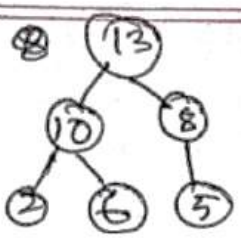
del 5

None	2	5	6	8	10	13

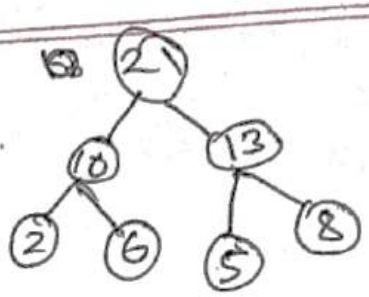


1ম iteration শেষ

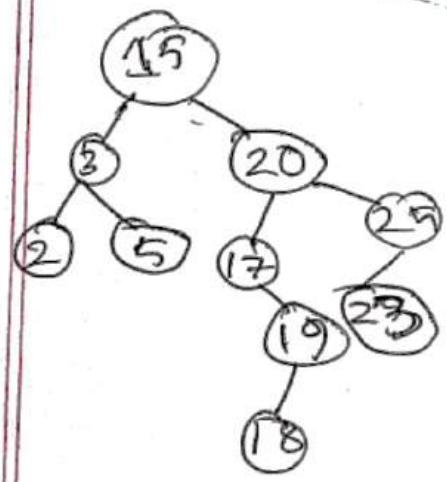
None	10	6	8	2	5	13



insert 21

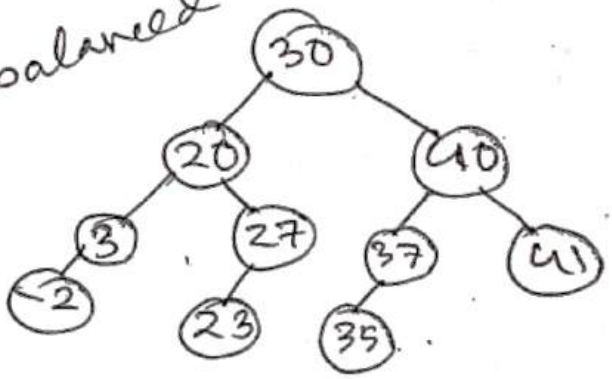



```
def fun(n)
    return helper(n, i=1)
def helper(n,i):
    base case
    helper(n, i+1)
```



$-2^{0}, 3^{1}, 20^{2}, 23^{3}, 27^{4}, 30^{5}, 35^{6}, 37^{7}, 40^{8}, 41^{9}$

balanced



normal BST