- Algorithm কতটুকু fast run করে তা নির্ভর করে machine কতটুকু fast ~~কাজ~~ কাজ করে।

$O(n) \rightarrow$ Big-O-n

↳ (input)

- input size এর উপর operation নির্ভর না করলে complexity হবে $O(1)$.

⊛ $y=5; \longrightarrow O(1)$

$x=1+y; \longrightarrow O(1)$

$y=x+2; \longrightarrow O(1)$

$z=x+y; \longrightarrow O(1)$

- Big-O তে ধ্রুবক কে calculate করি না।

⊛ $7n+2 \rightarrow \cancel{7}\cdot O(n)+2 \longrightarrow O(n)+O(1)$

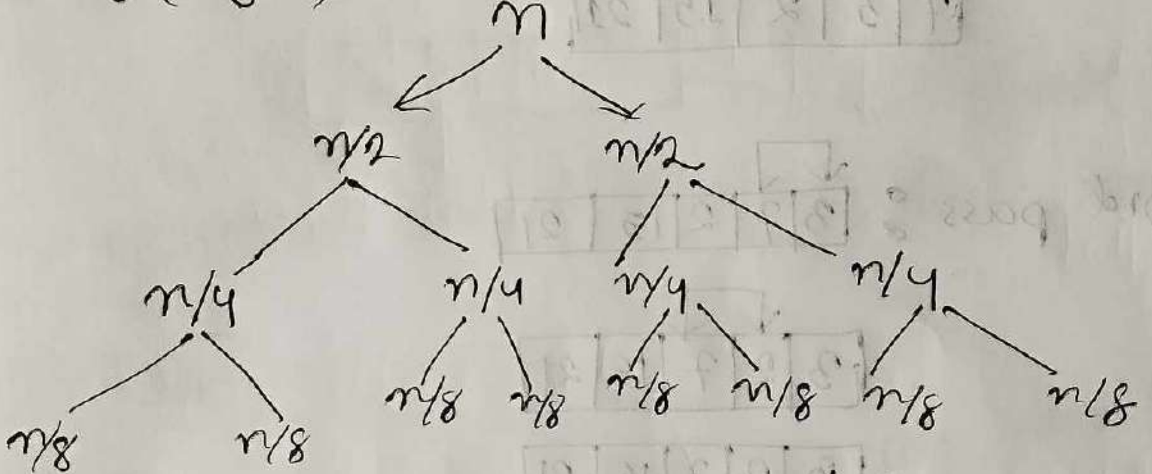
```
x=5 ; y=x+1 ; → O(1)
for(i=1 ; i<=7 ; i++){
    for(j=1 ; j<=n ; j++){

    }
}
```

- $3n^3 + 20n^2 + 5$

$= O(n^3) + O(n^2) + O(1) = O(n^3)$ → অবশেষে বড়টাই ~~পু~~ পুরো algorithm এর complexity হবে

- $3\log n + 5$

$= O(\log n)$ which is smaller than $O(n)$.



Binary Search
Recursion } $O(\log n)$

Bubble Sort :

- প্রত্যেকবার pass complete হচ্ছে এবং last থেকে একটা করে element sort হচ্ছে।

```
for(i=0 ; i<n ; i++){
    for(j=0 ; j<n-1 ; i++){
        if (a[j] < a[j+1])
        {

        }
```

Complexity : $O(n^2)$

7	15	3	2	21

1st pass:

7	15	3	2	21

7	3	15	2	21

7	3	2	15	21

7	3	2	15	21

2nd pass:

3	7	2	15	21

3	2	7	15	21

3	2	7	15	21

3	2	7	15	21

3nd pass:

2	3	7	15	21

2	3	7	15	21

2	3	7	15	21

2	3	7	15	21

Insertion Sort :

31	35	42	5	17

→ Worst Case $O(n^2)$

- 1টা 1টা করে আগের element গুলোর সাথে check করবে বড় না ছোট then replace করবে।

1	2	3	4	5

→ Best case $O(n)$

- আগের প্রত্যেকটা element বার বার check করা লাগে।

Selection Sort :

0 index নিয়ে কাজ 1st iteration:

0	1	2	3	4
2	7	1	6	8

→ 1 এর index 2 ছিল

1st element পুরো array check করে দেখে কোনটা ছোট then replace করবে direct.

0 index নিয়ে কাজ 1st pass :

0	1	2	3	4
1	7	2	6	8

→ 1 এর index 0 হয়ে গেল

1 index নিয়ে কাজ 2nd pass :

0	1	2	3	4
1	6	2	7	8

→ 7 এর index 1 থেকে 3 হয়ে গেল

2 index নিয়ে কাজ 3rd pass :

0	1	2	3	4
1	6	2	7	8

3 index নিয়ে কাজ 4th pass :

0	1	2	3	4
1	6	2	7	8

5 index নিয়ে কাজ 5th pass :

0	1	2	3	4
1	6	2	7	8

Class work

DSA Theory

11.02.2024

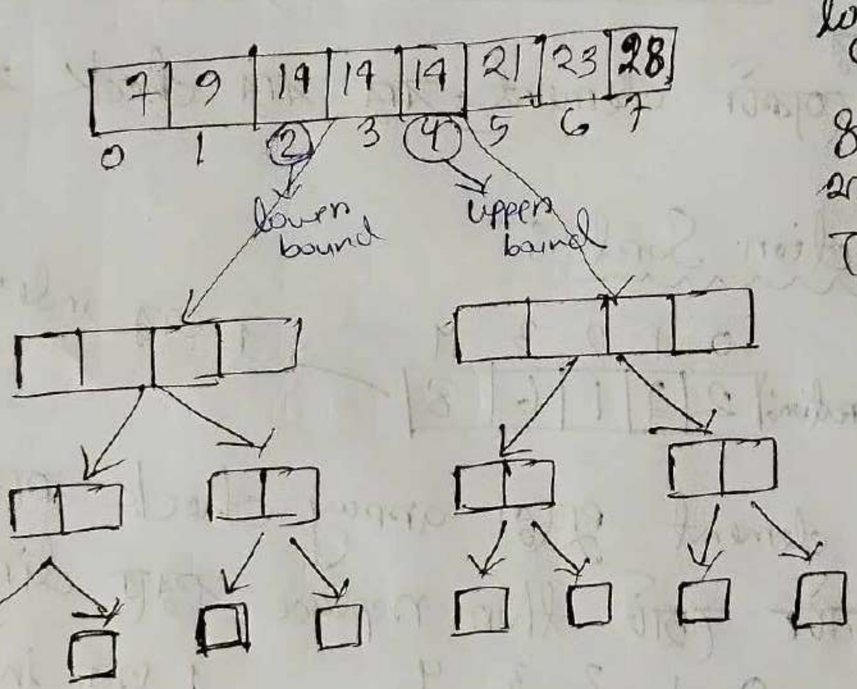
Searching:

Linear : ⊛ Complexity = O(n).

1 টা 1 টা করে check করে targeted element.

Binary: sorted থাকতে হবে। ⊛ Complexity = O(~~logn~~) $(\log_2(n))$ → log(2)

$$mid = \frac{high \; \text{~~first~~} + lower}{2}$$



$\log_2 8 = 3$

8 টি array থেকে 3 step-এ পেয়ে যাচ্ছি।

Interpolation :

$$1 \underbrace{\quad}_{1} 2 \underbrace{\quad}_{498} 500 \underbrace{\quad}_{2} 502 \underbrace{\quad}_{8} 510 \underbrace{\quad}_{490} 1000$$

Targeted value

$$mid = Lower + \frac{(High - Lower) * (X - A[Lower])}{A[High.] - A[Lower]}$$

⊛ Complexity : O(log(log(n)))

0	1	2	3	4	5	6
0	2	4	8	10	12	14

$$\frac{\overset{3}{\cancel{6}}-0}{\underset{7}{\cancel{14}}-0}=\frac{3}{7}$$

$$mid = 0+\frac{3}{7}\times(6-0)$$

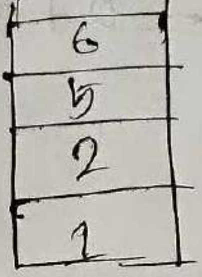
ss
ork

Stack

15.02.2024

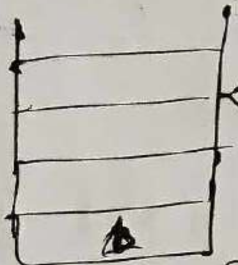
⊛ push() → number verify করে দিতে হয়

LIFO
↓
Last-এ যে element ঢুকবে সেটা first-এ out করবে



(stack-এ 4 টার বেশি element রাখবে না)

push(5)
push(6)
push(7) → যাবে না overflow হবে
pop()
pop()
pop()
pop()
peek() → null দেখাবে



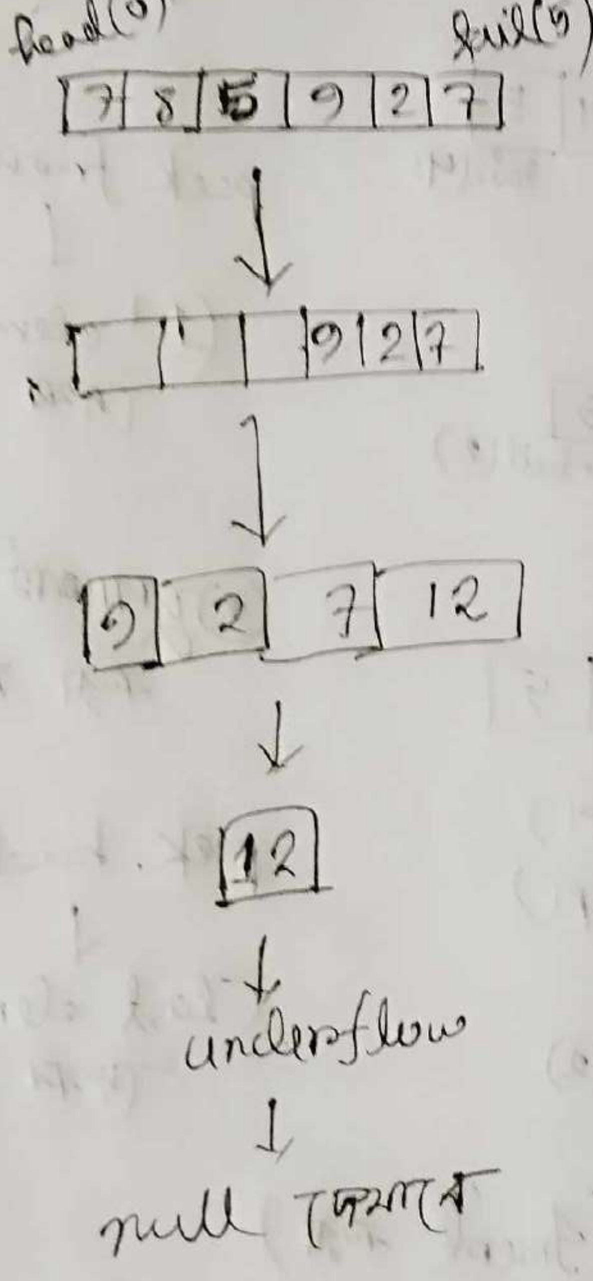
stack খালি (null)
↓
আবার pop() করলে underflow দেখাবে।

⊛ head() → ~~state~~ stack এর 1st element (0)
tail() → " " last " (n)

⊛ peek() করলে always head() এর element টা দেখাবে

Queue

(*) FIFO → First in first out



push(7)
push(8)
push(5)
push(9)
push(2)
push(7)
pop( )
pop( )
pop( )
push(12)
pop( )
pop( )
pop( )
pop( )
peek( )

(*) প্রতিবার pop() বা, push( ) করলে head() change হয়। ~~But~~ tail( )-এ change হবে

DQueue, Enqueue (Double Added Queue)

or, Delete করা

✱ pop() টাই হচ্ছে D.Queue.

front() :

~~front~~

1	2	3	4	5

head(0) tail(4)

front. pop()

front. pop()

3	4	5

head(0) ~~to~~ tail(2)

peek. front()

↓

(1st element টা দেখব।)

back() :

1	2	3	4	5

back. pop()

back. pop()

1	2	3

head(0) tail(2)

✱ দুপাশেই add করা যায়।

peek. back()

↓

(last element ~~o~~ টা দেখব]

Enqueue (Insert করা)

✱ দুপাশে add করা যায় না element.

✱ push() টাই হচ্ছে Enqueue.

10

Department of Computer Science & Engineering
University of Asia Pacific (UAP)

Class Test #2 Fall 2023 2nd Year 1st Semester

Course Code: CSE 203 Course Title: Data Structures and Algorithms I Credits: 3.0

Full Marks: 10 Duration: 20 minutes

Instructions:

Non-programmable calculators are allowed.

Name: Faiza Haque Shoily	ID: 22201183

1. Printers have much less memory. Even in this day and age, some only have a few megabytes of RAM available. By using printer spooling, we can send one or more large document files to a printer without having to wait for the current task to be completed. Consider it as a cache or buffer. It is a location where your papers gather one after one and prepare for printing when another printing activity has been finished. All print tasks are managed by a program known as a "spooler". [3 + 7 = 10]

a. What type of data structure will be better for managing this task? Give reasons behind your choice.

b. Now, simulate the following scenario using your preferred data structure:

Think you have a printer whose spooler can tend a maximum of 5 jobs at a time. Initially when the power of the printer is turned on, there is no task to do. What was the present condition of the data structure? Is it full or empty?

Then, 4 jobs came at the one by one. After finishing 2 jobs 4 more tasks appear gradually one by one. What is the current condition now? Is it full or empty?

Again, 1 more job is performed and removed. Now, what will be the current condition?

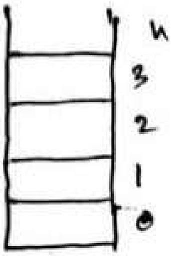
Draw the data structure along with the pointers (top/rear/front) **after every operation.**

ⓐ ~~Stack~~ Queue will be better for managing this task.

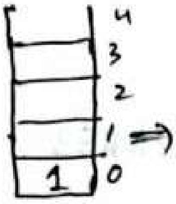
In, ~~Printer~~, Buffer, papers gather one after one and print the first paper first. In Queue, there is FIFO system, which is First in First Out. So, In Spooler, Entered first page prints first, so In spooler program, Queue will be better.

3

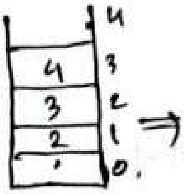
(b)



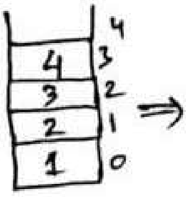
→ Rear = -1
Front = -1



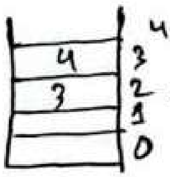
⇒ Rear = 0
Front = -1



⇒ Rear = 3
Front = 0

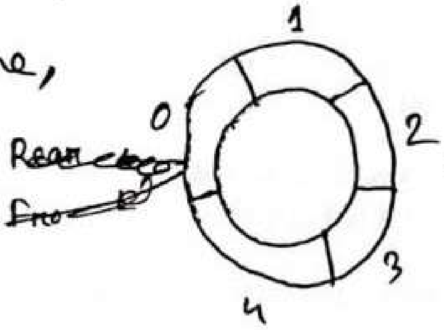


⇒ Rear = 3
Front = -1

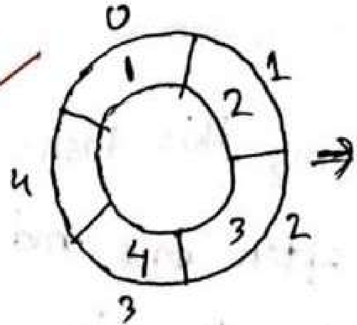


⇒ Rear = 3
Front = 1

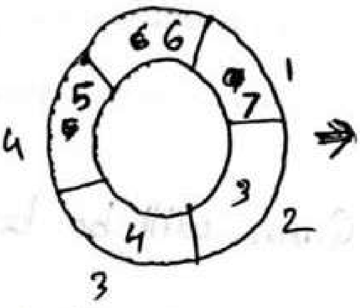
If Spooler can tend a maximum of 5 jobs at a time,



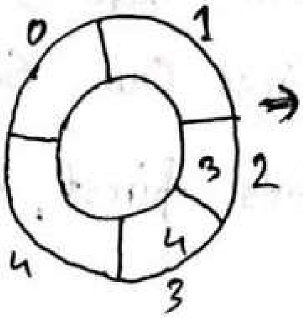
⇒ Rear = -1
Front = -1 , So the Queue is empty



⇒ Rear = 3
Front = -1



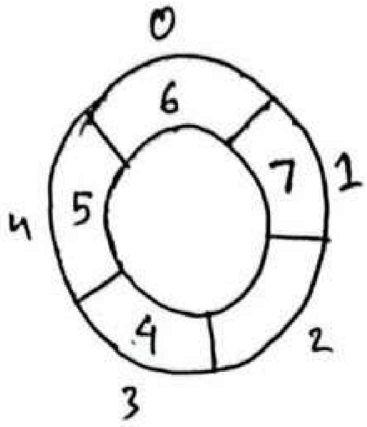
⇒ ~~Rear = 3~~
~~Front = 0~~,
Rear = 1
Front = 1



⇒ Rear = 3
Front = 1

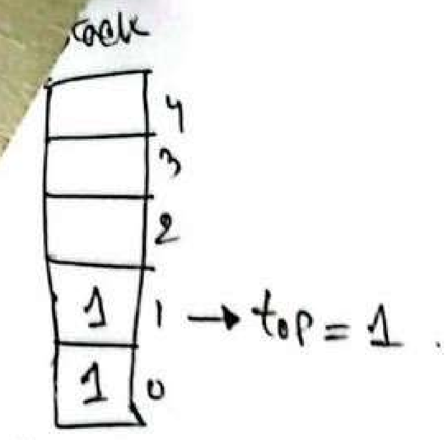
So, fourthworks can not be appeared gradually as the circular queue will be overflowed. So, the queue is full.

Faiza Haque Shoily
Id: 22201183

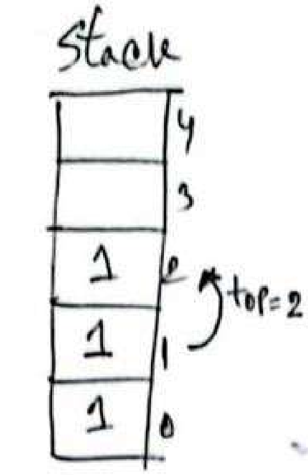


⇒ Rear = 1
Front = 2

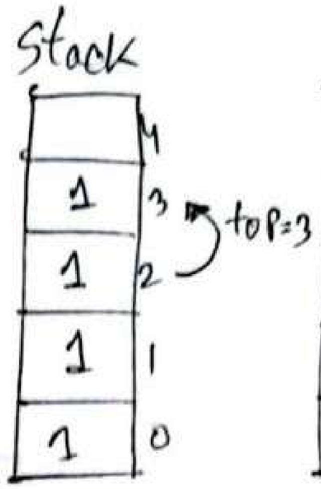
Now the circular queue is not full.



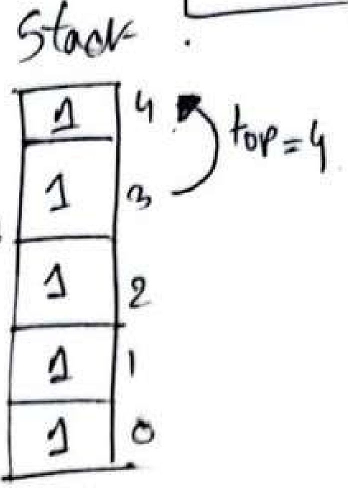
Present Condition



Undo Operation 1

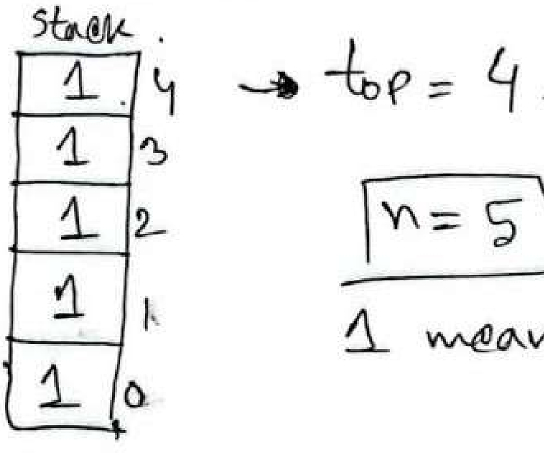


Undo Operation 2



Undo Operation 3.

After implementing 3 more undo operation. the Present stack is →.



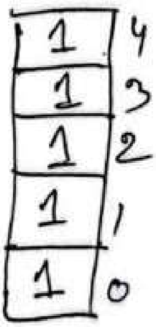
n-1 = 4
top = 4.

n= 5

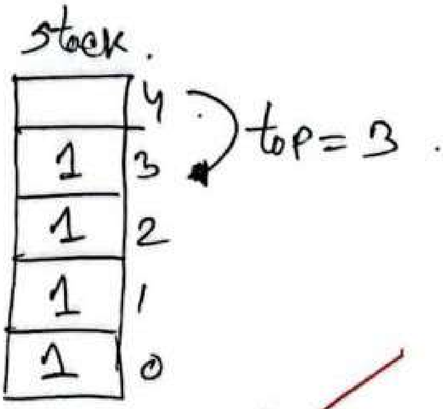
1 means Undo Operation

Here top = n-1, that's why stack is Full

If again 1 more redo is Performed then.



Present Condition

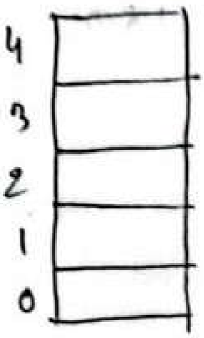


Current Condition.

to the stack. Because when undo performed it
to push in stack and when redo performed it w
to pop in stack.

That's why to implement this scenario stack data structure
will be better choice.

6



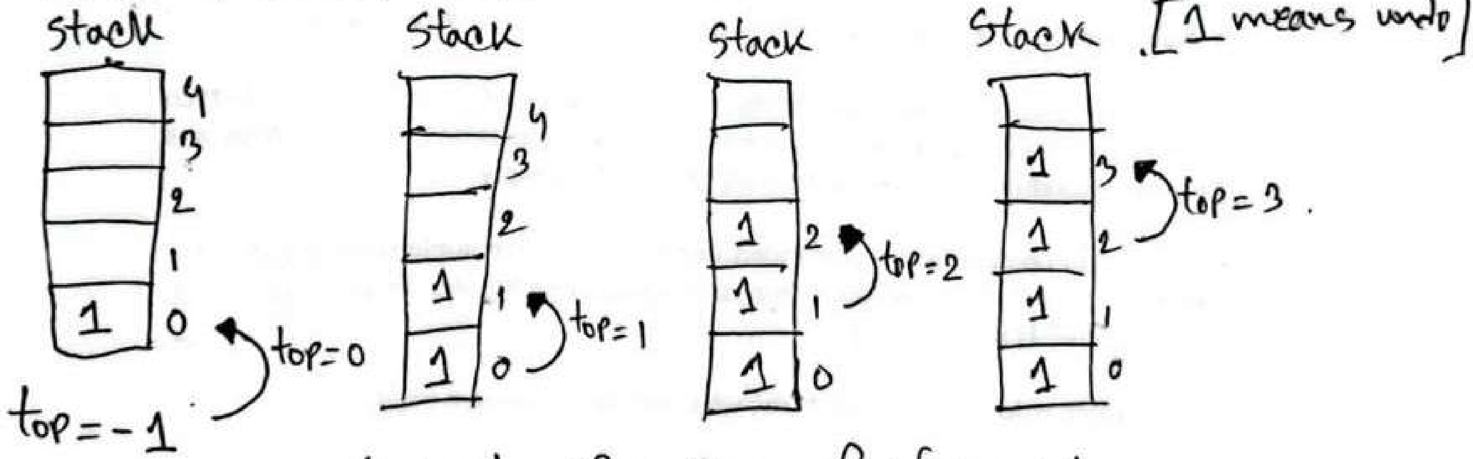
stack.

Size = 5.

let, top = -1,

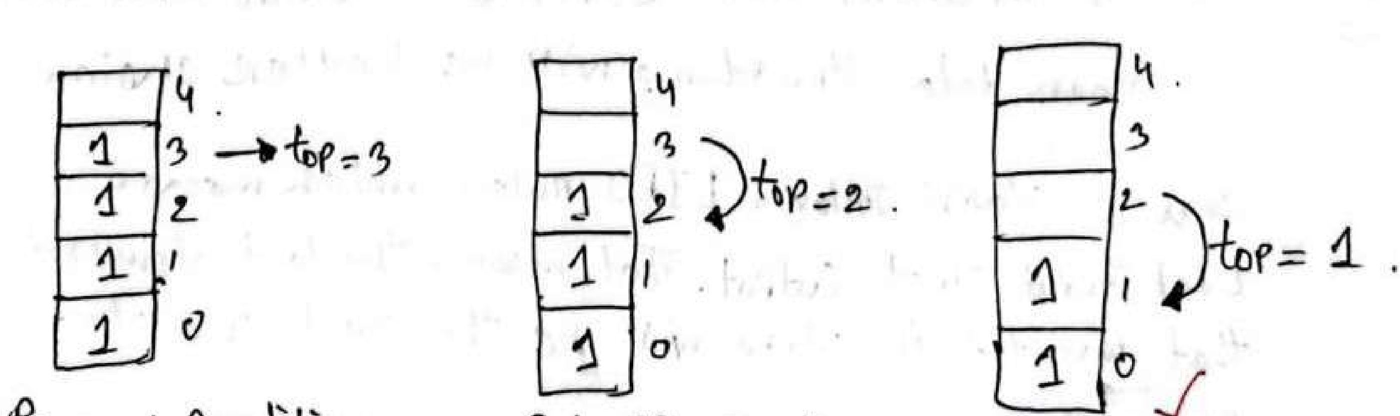
Initially there is no operation, no task to do then the value of top will be still -1 that means the stack is empty.

Then 4 undo operation performed one by one.



4 undo operation performed.

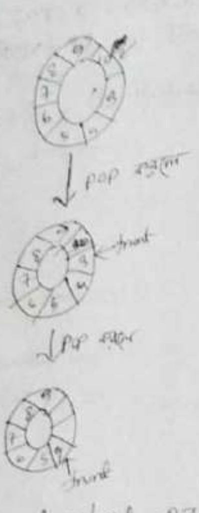
The implementing 2 redo operation.



Present Condition Redo Operation 1 Redo Operation 2.

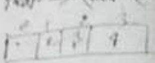
Class Work

Circular queue :



front → প্রথম লেখা থাকবে

rear → শেষের লেখা



4 no. element push করা যায় না n=4 থাকা সত্ত্বেও।

front = 1 rear = 3

তাই circular queue use হয়।

front = (front+1) % n

~~rear = (i++) % n~~

rear = (rear + 1) % n

প্রত্যেক opening braket-এর জন্য একটা করে pop হবে

(1) Stack-এ প্রবেশ করানো

(()((())))

(2) Operator Middle-এ থাকলে Infix 2+3

First-এ " Prefix +(2,3)

Last-এ " Postfix (1,2+

Computer এর ⇒ ——→ "

Infix can be converted in Postfix using Stack

1+3÷3−2×4

Expression (1) 1 3

(2) ÷ কে pop করা হয়েছে

কম Priority এর Sign টা নিচে থাকে

Higher Priority 2 DM

1st AS

(3) (−) এর priority (1)

তাই (÷) কে pop করে

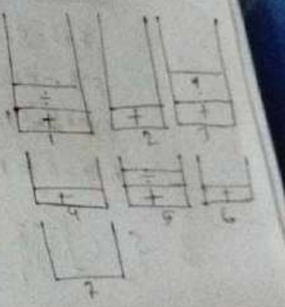
(+) কেও pop করা হলো

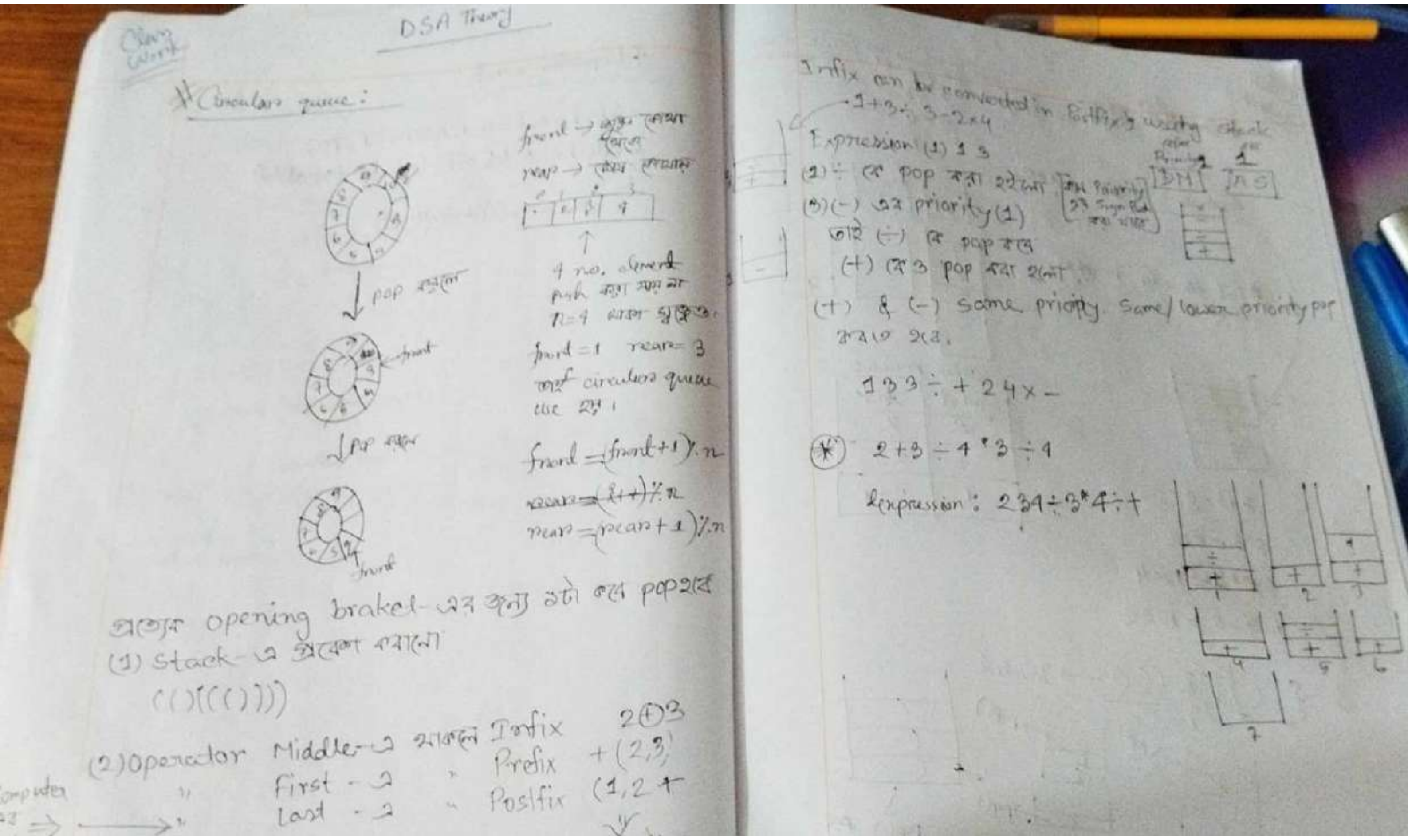
(+) & (−) same priority Same/lower priority pop করাতে হবে.

1 3 3 ÷ + 2 4 × −

(*) 2+3÷4*3÷4

Expression : 234÷3*4÷+





Class Work

DSA Theory

Circular queue :

front → প্রথম কোথায় থাকে
rear → শেষ কোথায়

4 no. element push করা যায় না n=4 থাকা সত্ত্বেও.

front=1 rear=3
তাই circular queue use হয়।

front = (front+1)%n

~~rear = (i++)%n~~

rear = (rear+1)%n

pop করলে
front
pop করলে
front

প্রত্যেক opening braket-এর জন্য একটা করে pop হবে

(1) Stack-এ প্রবেশ করানো

(()(())))

(2) Operator Middle-এ থাকলে Infix 2+3
" First-এ " Prefix +(2,3)
" Last-এ " Postfix (1,2 +

Infix can be converted in Postfix's using stack

1+3÷3-2×4

Expression (1) 1 3

(2) ÷ কে pop করা হচ্ছে না [কম priority এর sign কে pop করা যাবে]

(3) (-) এর priority (1)
তাই (÷) কে pop করে
(+) কেও pop করা হলো.

(+) & (-) same priority. Same/lower priority pop করাতে হবে.

1 3 3 ÷ + 2 4 × -

2+3÷4*3÷4

Expression : 234÷3*4÷+

Class Work

25.02.2024

Convert to postfix:

Infix
$2*3^\wedge(5+3)+2$

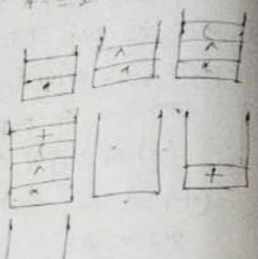
[illegible] pop() [illegible]

^ = 3

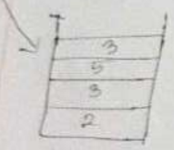
*/ = 2

+ - = 1

Expression: 2353+^*2+



(*) Post fix - এ bracket থাকে না।



5+3=8

5^3 = 512

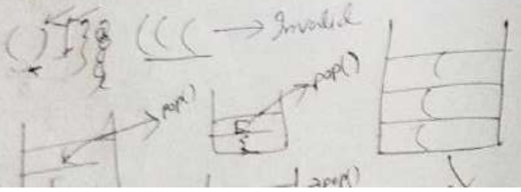
512*2 = 1024

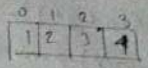
1024 - 2 = 1022

Other process:

Symbol	Stack
2	2
3	23
1	231

{[()]} (((→ Invalid



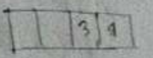
n=4 

front = 0
rear = -1

1 আসলে rear = 0
2 " " = 1
3 " " = 2
4 " " = 4

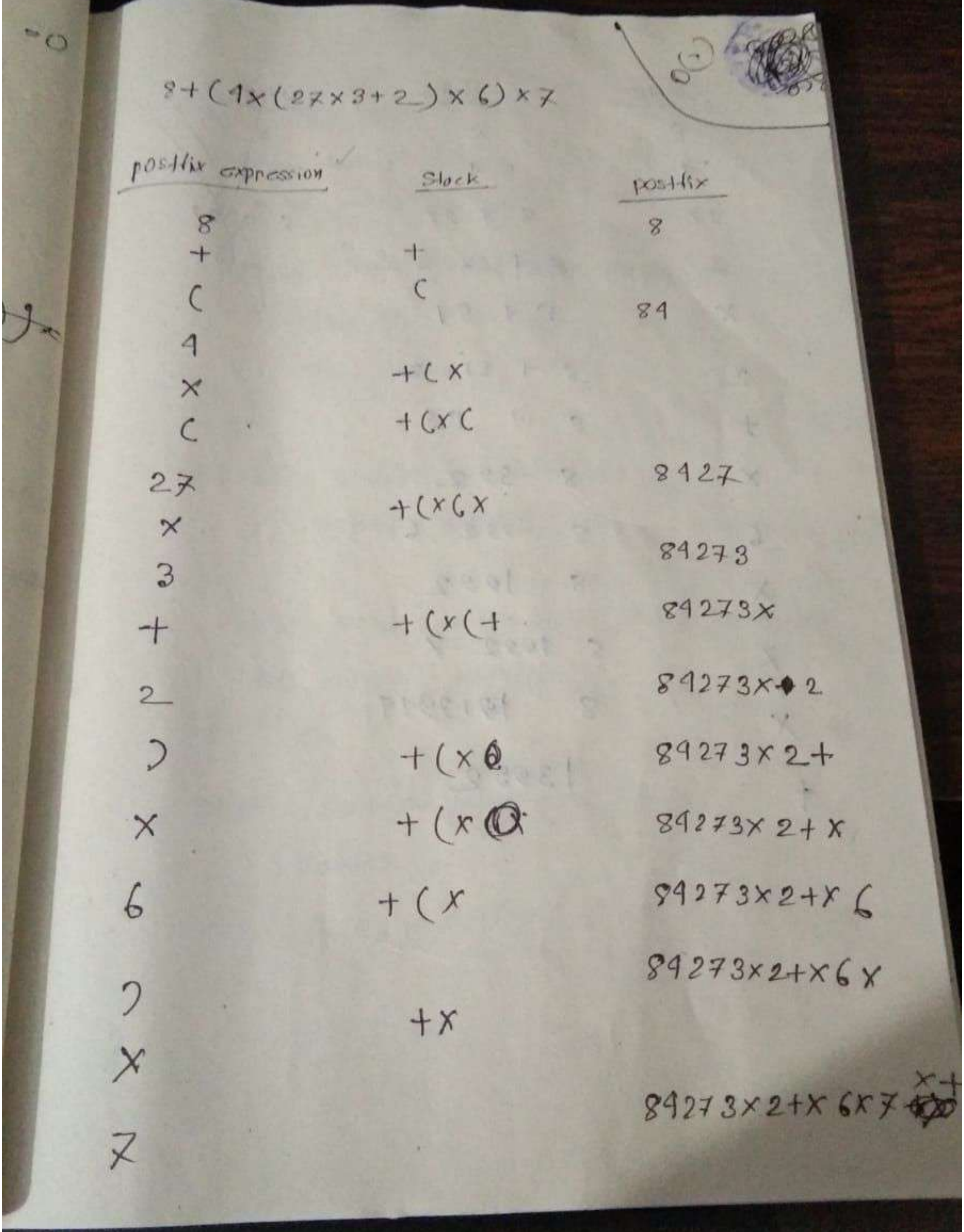
1 pop() → front = 1

2 pop() → front = 2



Circular queue এর জন্য rear = (rear+1) % n

এবং front = (front+1) % n



8+(4x(27x3+2)x6)x7

postfix expression	Stack	postfix
8		8
+	+	
(	(	
4		84
x	+(x	
(	+(x(	
27		8427
x	+(x(x	
3		84273
+	+(x(+	84273x
2		84273x 2
)	+(x	84273x2+
x	+(x	84273x2+x
6	+(x	84273x2+x6
		84273x2+x6x
)	+x	
x		
7		84273x2+x6x7x+

Symble	Stack
8	8
4	8 4
27	8 4 27
3	8 4 27 3
X	8 4 81
2	8 4 81 2
+	8 4 83
X	8 332
6	8 332 6
X	8 1992
7	8 1992 7
X	8 ~~19~~13944
+	13952

~~[illegible]~~

~~8~~

~~8 4~~

~~8 4 27~~