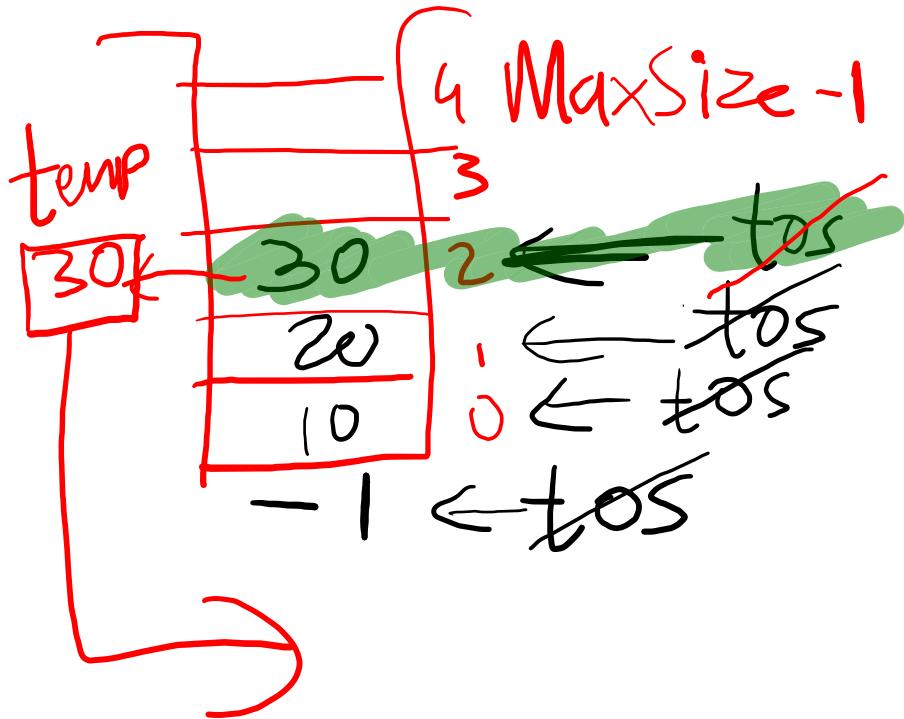


if  $tos == -1$   
 empty  
 if  $tos == MaxSize - 1$   
 Full

- Linear
- single ended
- mostly static

### ADT

↳ createStack(size)  
 ↳ push(e)  
 ↳ tos + 1  
 ↳ pop():e  
 ↳ peek():e  
 ↳ ~~tos~~

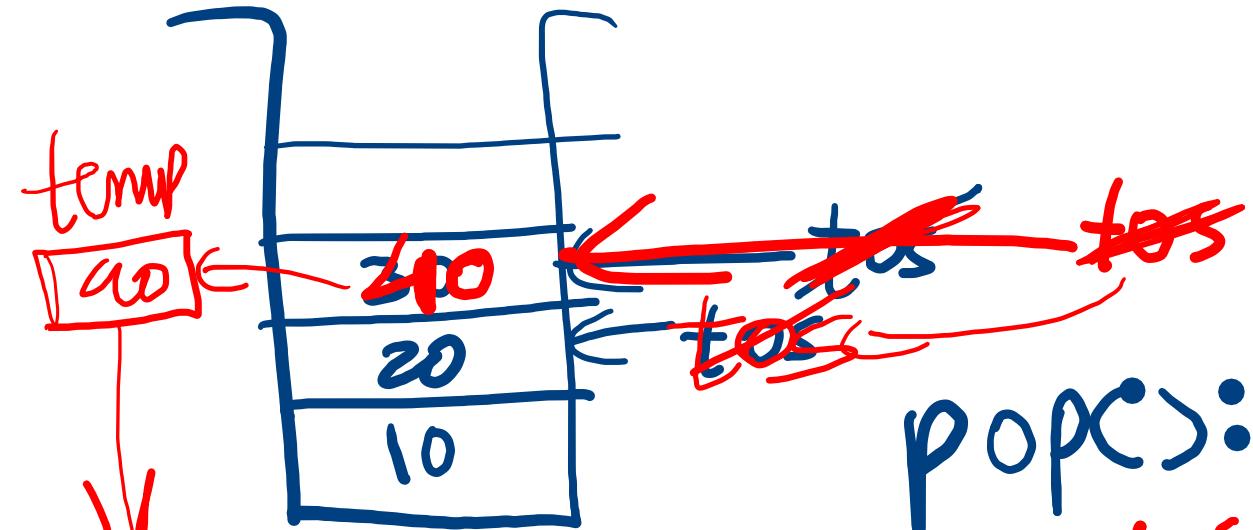


push(10)

push(20)

push(30)

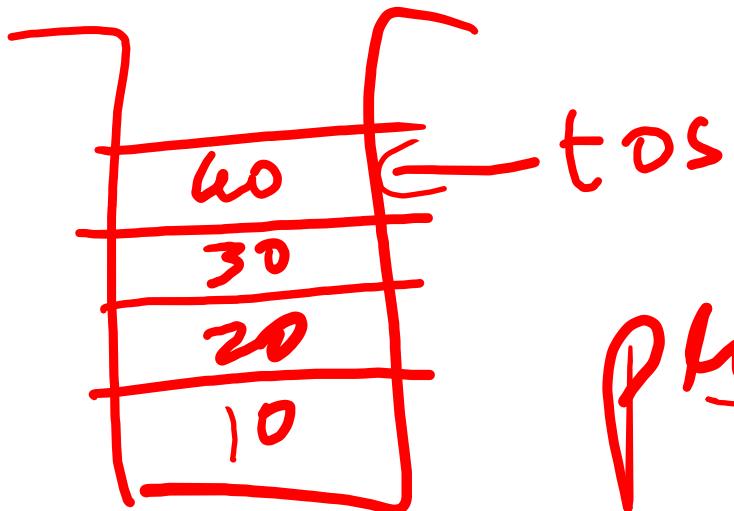
peek(): 30



pop(): 30

push(20)

pop(): 40

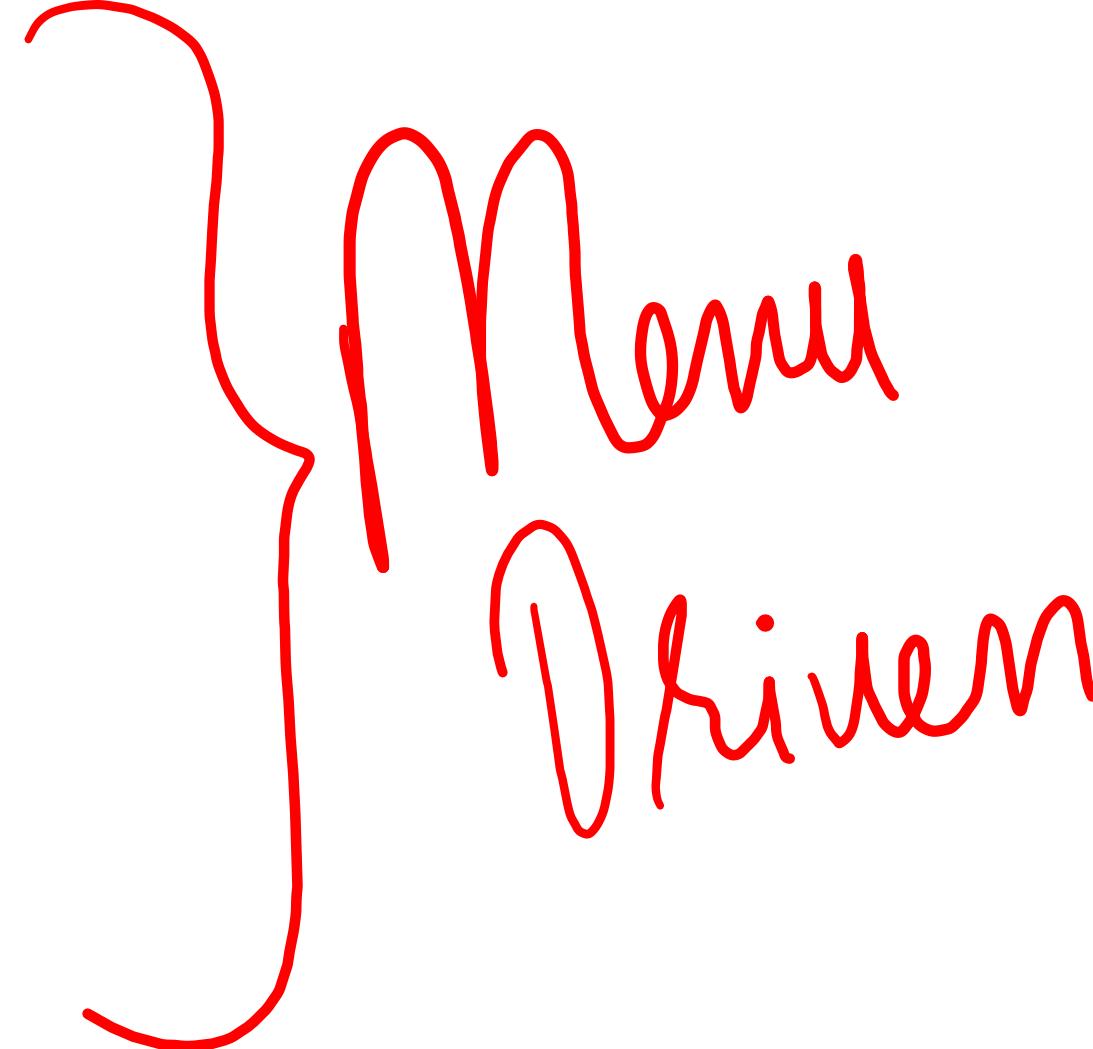


PrintStack()  
LIFO



40  
30  
20  
10  
↓ LIFO  
↓ order

```
do  
{  
    switch()  
    {  
    }  
} while
```



# App Stack

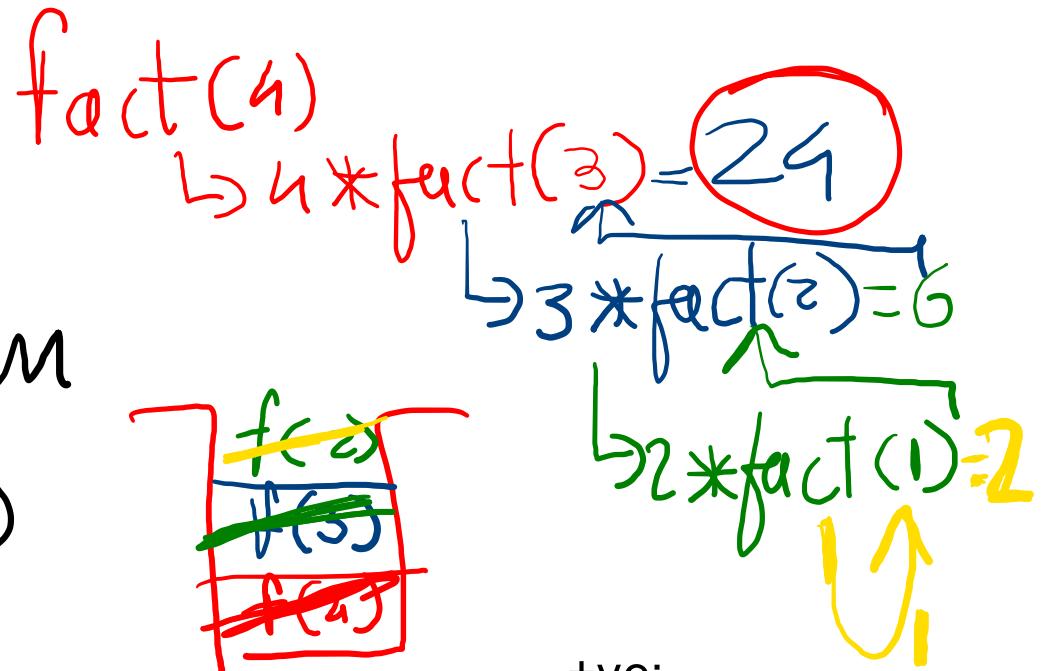
①  $\rightarrow$  Recursion

```
int fact(int no)
```

if(~~no == 1~~)  
return 1;

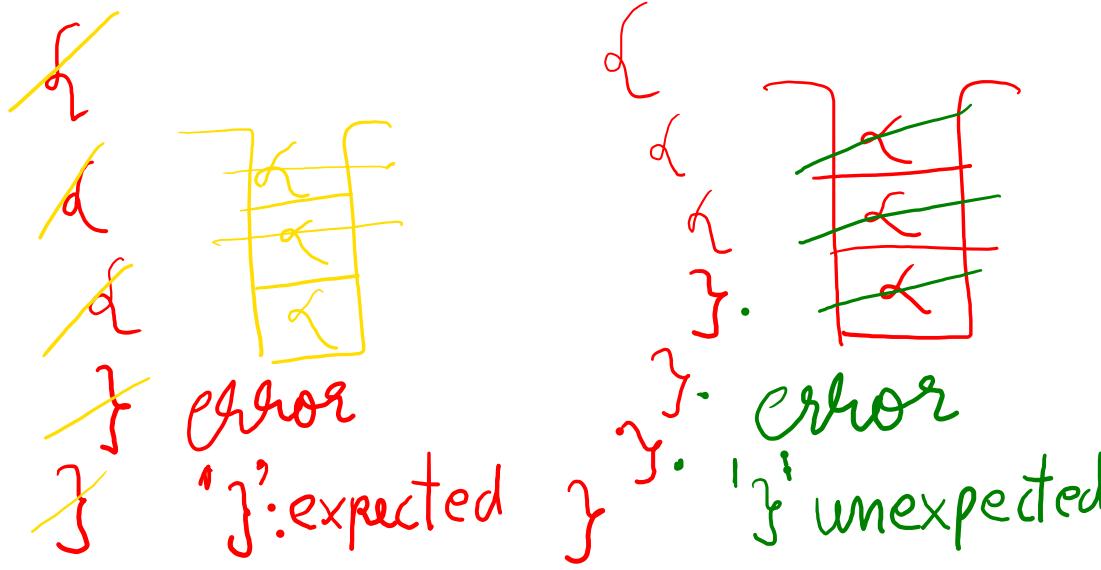
else return no \* fact(no - 1);

}



-ve:  
takes more memory, slow  
down, complex

## ② Wellness of Control



### ③ String reversal



$\Rightarrow \text{pop}()$   $\rightarrow$

The diagram shows the result of popping the character 'h' from the stack. To the right of the stack, the characters 'a', 'l', 'p', and 'h' are stacked vertically, representing the reversed string "alph".

④ Dec to binary

$$(12)_{10} \Rightarrow (0011)_2$$

The diagram illustrates the conversion of the decimal number 12 to binary using two methods: division by 2 and division by 3.

**Method 1: Division by 2**

12 is divided by 2:

$$\begin{array}{r} 6 \\ 2 \overline{)12} \\ -12 \\ \hline 0 \end{array}$$

The quotient 6 is then divided by 2:

$$\begin{array}{r} 3 \\ 2 \overline{)6} \\ -6 \\ \hline 0 \end{array}$$

The quotient 3 is then divided by 2:

$$\begin{array}{r} 1 \\ 2 \overline{)3} \\ -2 \\ \hline 1 \end{array}$$

The quotient 1 is then divided by 2:

$$\begin{array}{r} 0 \\ 2 \overline{)1} \\ -1 \\ \hline 1 \end{array}$$

The remainders 0, 0, 1, and 1 are written vertically below the division lines, forming the binary representation 0011.

**Method 2: Division by 3**

12 is divided by 3:

$$\begin{array}{r} 4 \\ 3 \overline{)12} \\ -12 \\ \hline 0 \end{array}$$

The quotient 4 is then divided by 3:

$$\begin{array}{r} 1 \\ 3 \overline{)4} \\ -3 \\ \hline 1 \end{array}$$

The quotient 1 is then divided by 3:

$$\begin{array}{r} 0 \\ 3 \overline{)1} \\ -1 \\ \hline 1 \end{array}$$

The remainders 0, 1, 1, and 1 are written vertically below the division lines, forming the binary representation 0011.

# ⑤ Expression Conversion & Evaluation

Prefix:  $+ab(s/w)$   $\text{add}(a, b)$

infix:  $a + b$

Postfix:  $ab + (H/w)$

① Left to right if all equal

else 1<sup>st</sup> higher than lower

② (( )) then start from innermost

^ 3  
\* / % 2  
+ - 1

$$\begin{array}{r} a+b-c \\ \underline{+ab-c} \\ \hline A-c = -Ac \\ \underline{-tabc} \end{array}$$

$$\begin{array}{r} a+b-c \\ ab+ -c \\ \hline A-c = Ac- \\ \underline{ab+c-} \end{array}$$

$$\underline{(a+b)} * (c-d)$$

$$\begin{array}{r} \underline{+ab} * \underline{-cd} \\ \hline * +ab - cd \end{array}$$

$$(a+b\wedge c \cdot d - e) \quad | \quad (\underline{a \cdot b / c} - d \cdot e \% f)$$

$$\begin{array}{l} \cancel{(ab * /c - d * e \% f)} \\ \cancel{ab * c /} - \cancel{de * \% f} \end{array}$$

$$\cancel{ab * c /} - \cancel{de * f \%}$$

$$\cancel{ab * c /} \cancel{de * f \%} =$$

$$\begin{array}{l} \cancel{ab * /c - d * e \% f} \\ \cancel{(* abc} - d * e \% f \\ \cancel{(* abc} - \cancel{* de \% f} \\ \cancel{(* abc} - \cancel{\% * def} \\ = \cancel{(* abc \% * def} \end{array}$$

$$\frac{(a+b) * d - e}{\cancel{f * g / h}}$$

$$\frac{ab + \cancel{d} - e}{}$$

$$A * \cancel{d} = Ad *$$

$$\frac{ab + \cancel{d} * -e}{}$$

$$A - e = Ae -$$

$$\underline{ab + \cancel{d} * e - fg * h //}$$

$$\frac{fg * h}{}$$

$$\frac{A / h = Ah /}{}$$

$$\frac{fg * h /}{}$$

$$\underline{\underline{fg * h //}}$$

1.read infix from start to end

2.if read is

2.1 {[ push on stack

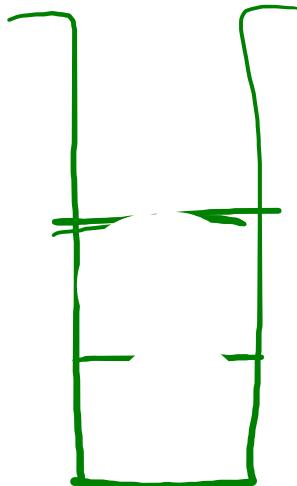
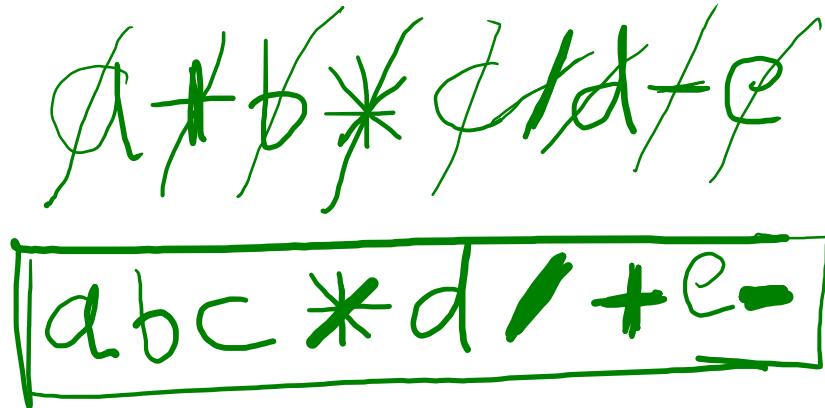
2.2 )]} then pop from stack till {[  
and copy popped to postfix

2.3 operand copy to postfix

2.4 if operator then compare it with  
tos if operator > then tos  
operator push else pop till either  
condition satisfies or stack  
becomes empty

3 continue till infix not over

4 copy left over of stack to postfix



1.read infix from end to start

2.if read is

2.1 )}] push on stack

2.2 {[ then pop from stack till )}] and copy  
poped to prefix

2.3 operand copy to prefix

2.4 if operator then compare it with tos if  
operator $\geq$  then tos operator push else pop till  
either condition satisfies or stack becomes  
empty

3 continue till infix not over

4 copy left over of stack to prefix

5 reverse prefix and use

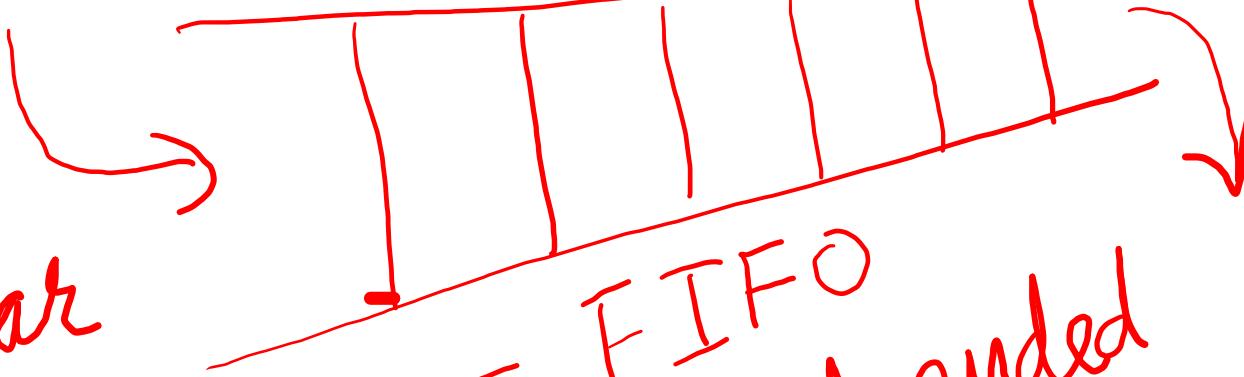
a f b \* c

- +

+ a \* b c

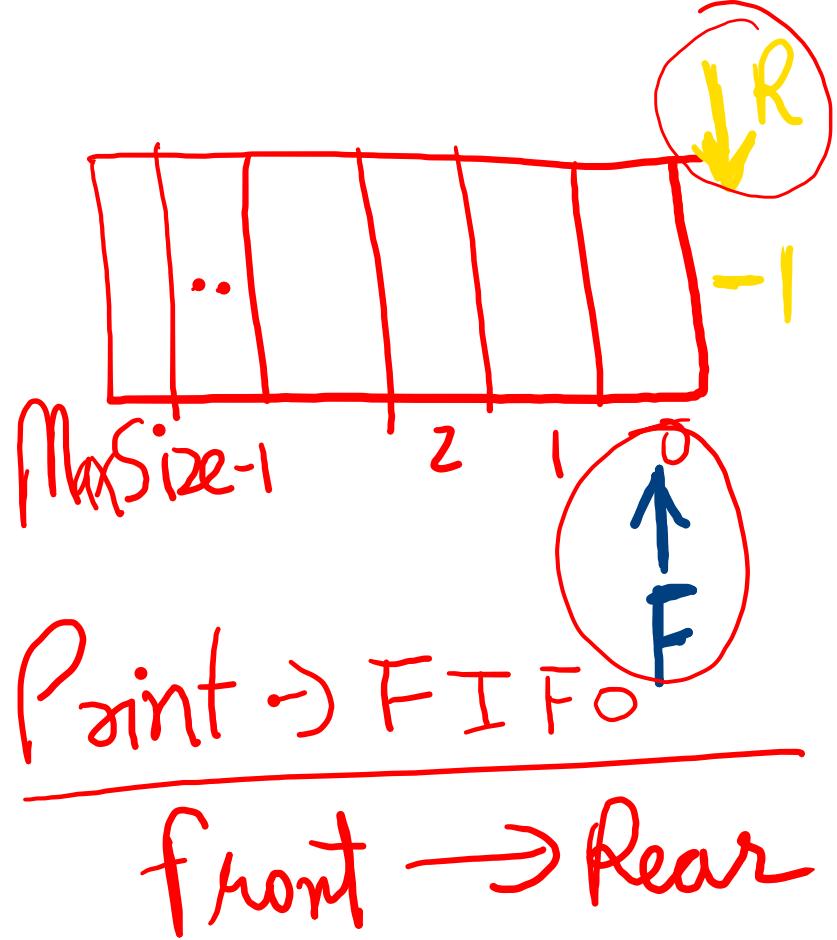
C b \* a +

Queue



rear

- FIFO
- Double ended
- Linear
- static



CreateQueue (size)

Enqueue(e): rear+1

Dequeue(): e front+1

if rear == MaxSize-1  
then full

if front > rear  
then empty

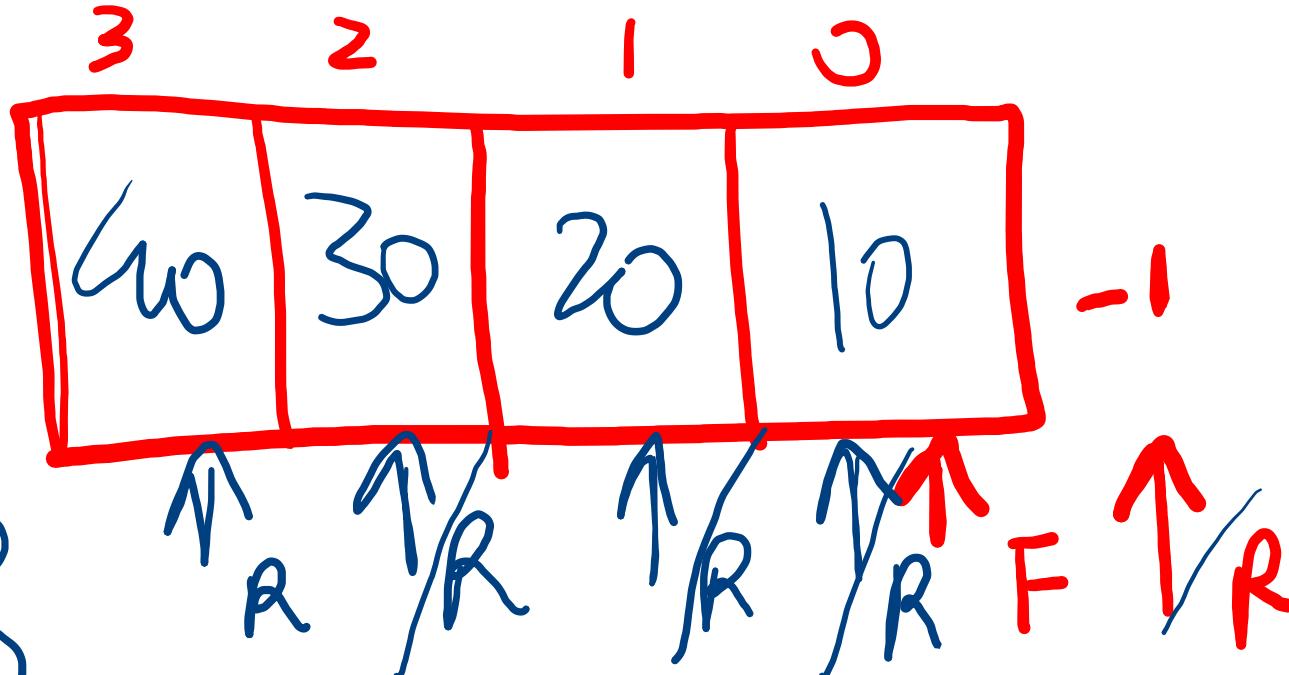


Enqueue(10)

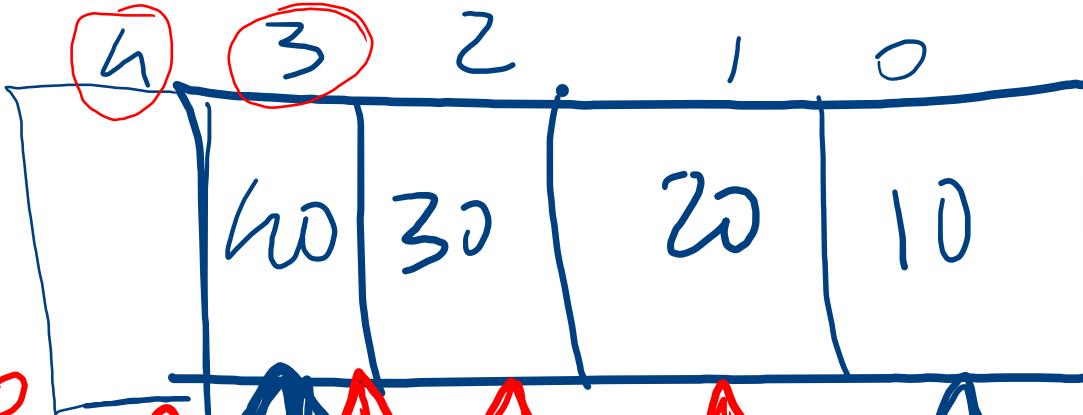
(20)

(30)

(40)



Dequeue():



() : 20

() : 30

() : 40

() : Empty

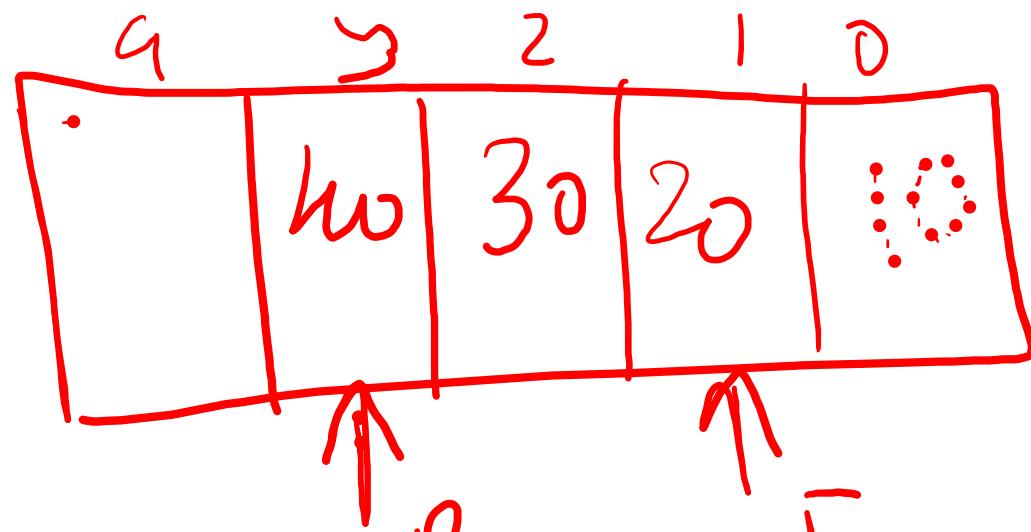
F R

F

/ F

/ F

/ F



R      F  
 Point from Front to hear first  
 FIFO



...va ThreadJoin.java × ThreadPriority.java × ThreadSleep.java × ThreadSync.java × SortingDemo.java × SearchDemo.java × Treemain.java × GraphDemo.java × StackDemo.java × QueueLinearDemo.java

Source

History

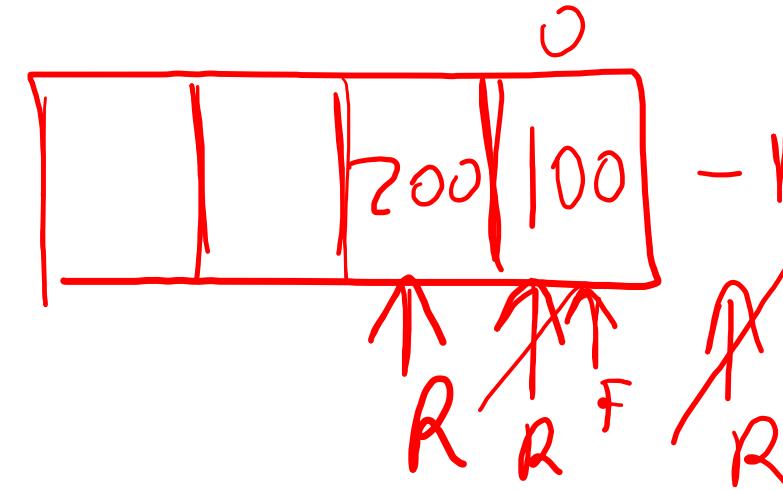
Projects

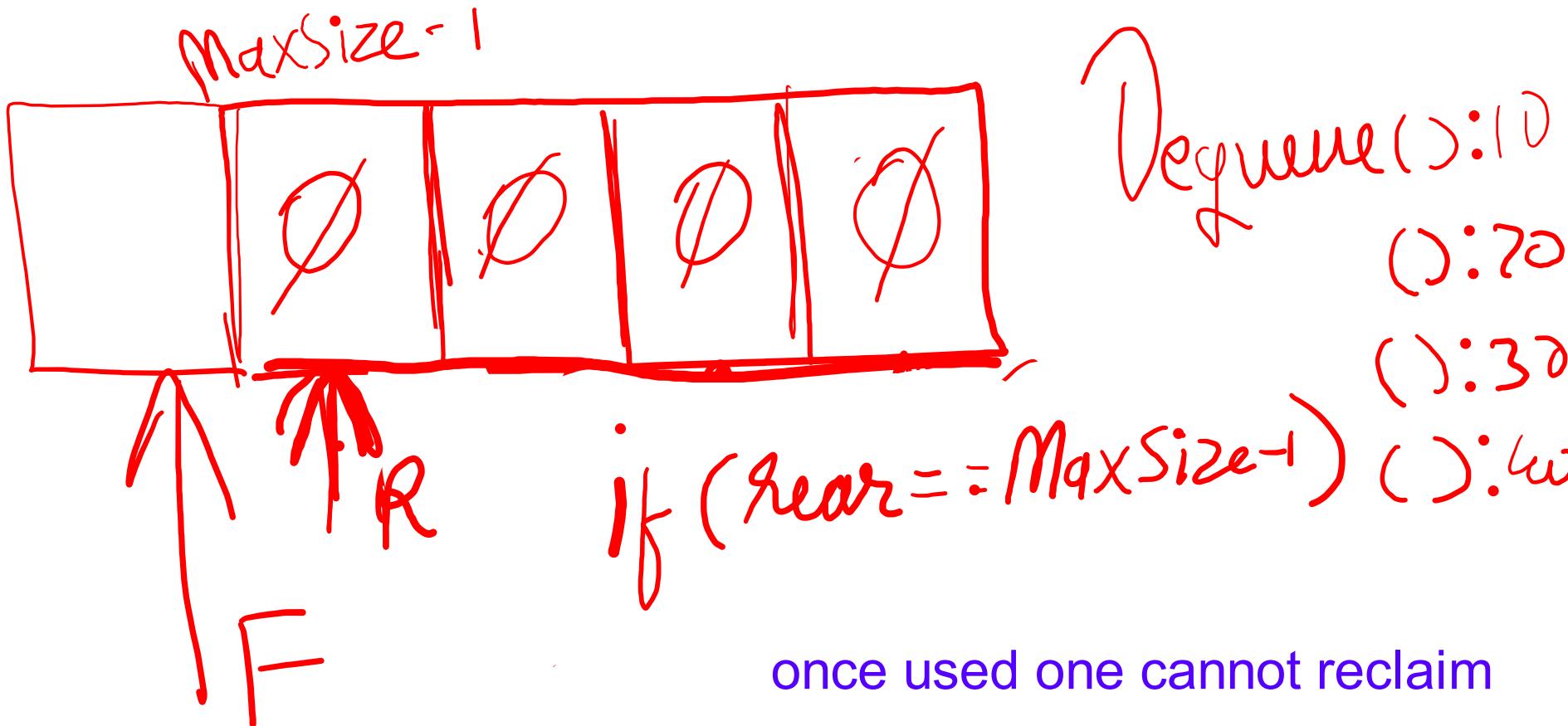
Files

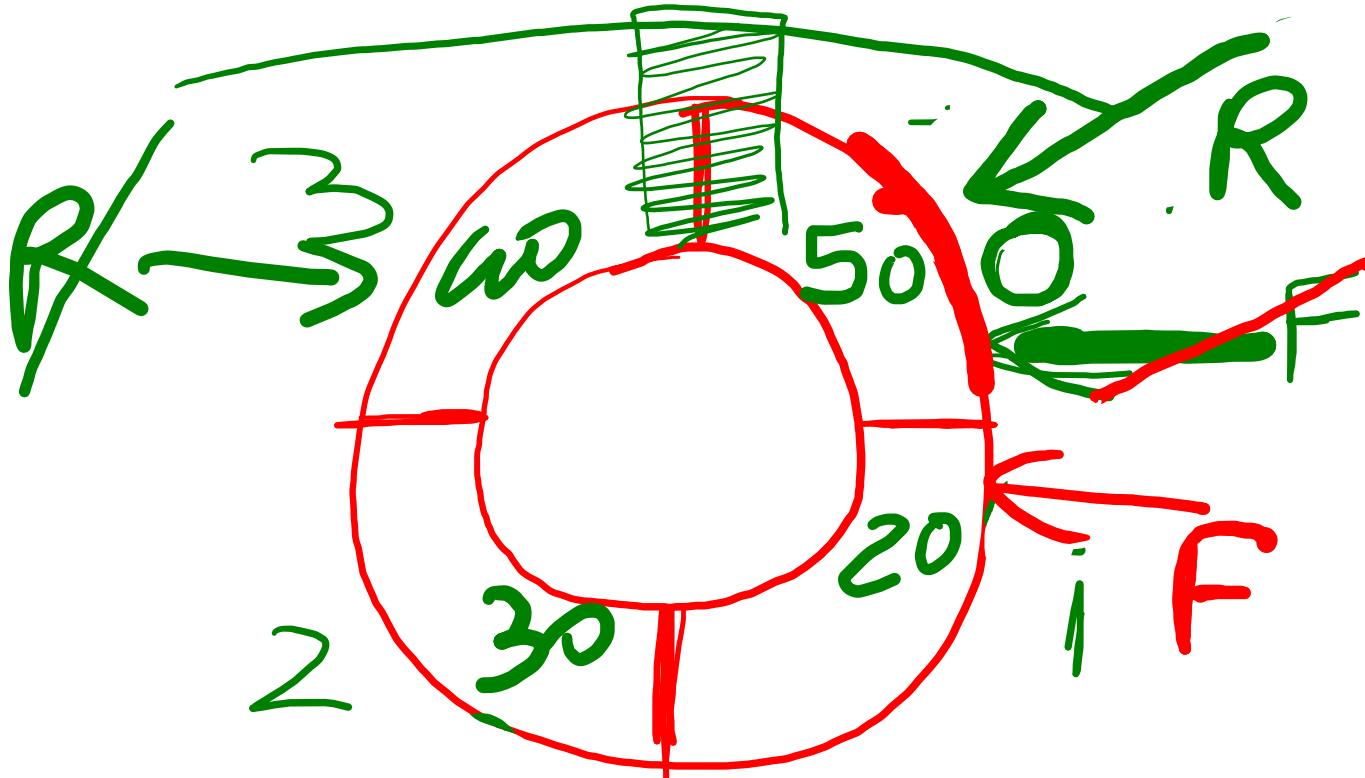
Services

Files

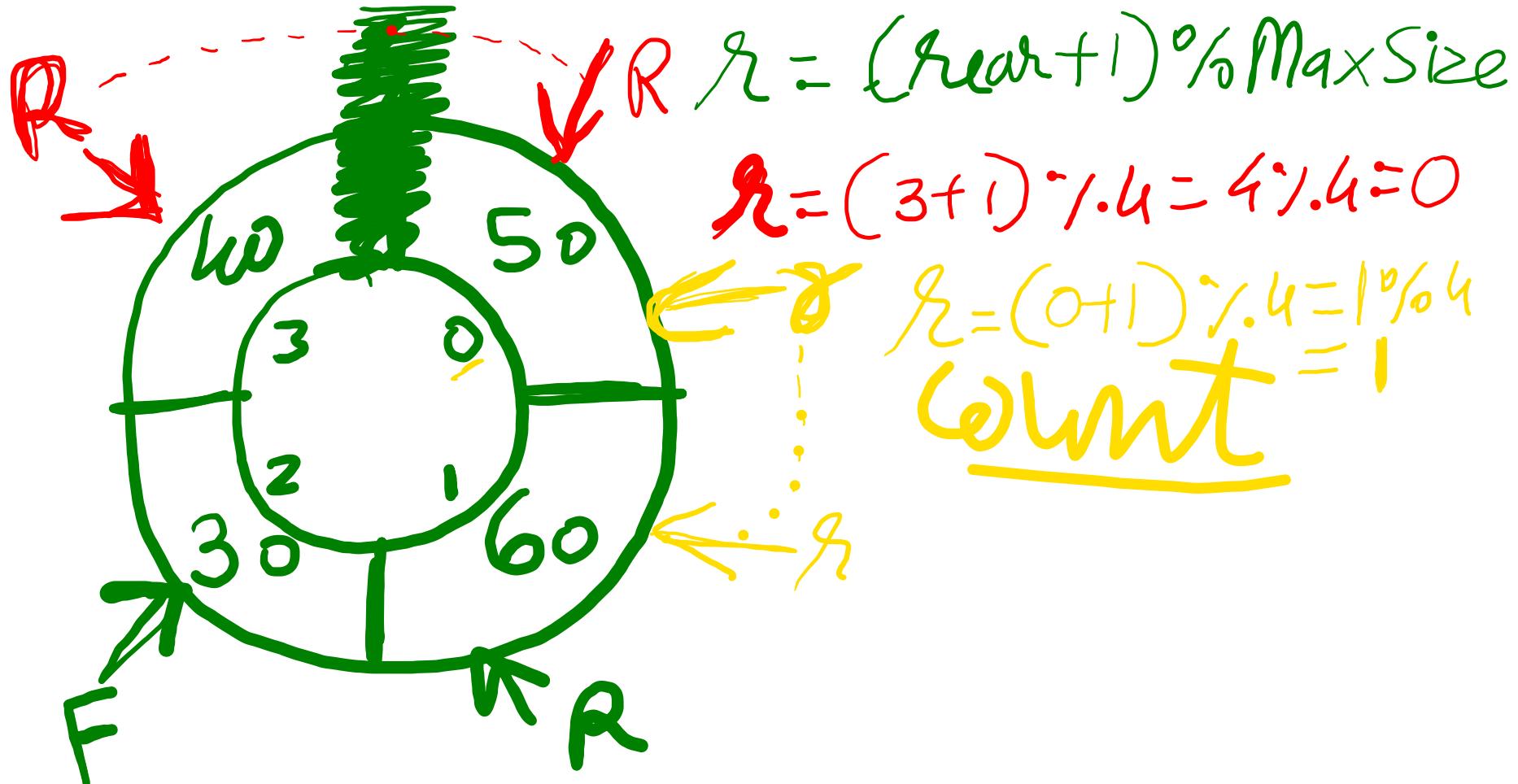
```
21 }  
22 void enqueue(int e)  
23 {  
24     rear++;  
25     q[rear]=e;  
26     //q[++rear]=e  
27 }  
28  
29 boolean isFull()  
30 {  
31     if(rear==MaxSize-1)  
32         return true;  
33     else  
34         return false;
```







$\ell(10)$   
 $\ell(20)$   
 $\ell(30)$   
 $\ell(40)$   
 $\ell(50)$   
 $D():10$



$$r = (\text{rear} + 1) \% \text{MaxSize}$$

$$r = (3 + 1) \% .4 = 4 \% .4 = 0$$

$$r = (0 + 1) \% .4 = 1 \% .4 = 1$$

Want



rear = -1  
Count = 0  
front = 0

if Count == 0  
Empty  
if Count == MaxSize  
Full

Enqueue

$r = (r+1) \% \text{MaxSize}$   
Count++

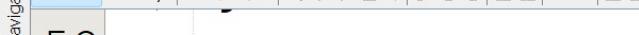
Dequeue

$f = (f+1) \% \text{MaxSize}$   
Count--



...va ThreadSleep.java x ThreadSync.java x SortingDemo.java x SearchDemo.java x Treemain.java x GraphDemo.java x StackDemo.java x QueueLinearDemo.java x CircularQueueDemo.java x

Source History



Services Projects Files



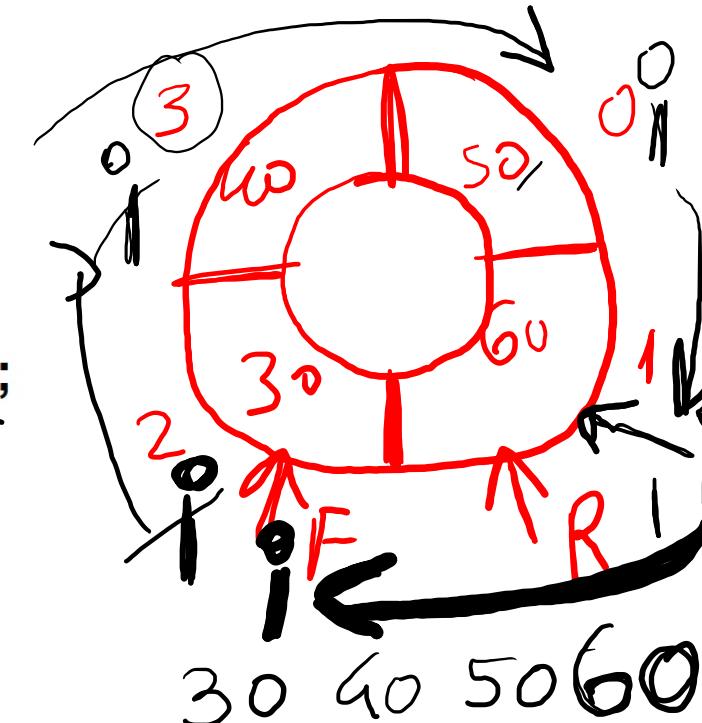
52  
53 void printQueue()  
54 {  
55 int c=0,i=front;  
56 while(c<count)  
57 {  
58 System.out.println(q[i]);  
59 i=(i+1)%MaxSize;  
60 c++;  
61 }  
62 }  
63 }  
64 }

**void printQueue()**

**int c=0,i=front;**  
**while(c<count)**

**System.out.println(q[i]);**  
**i=(i+1)%MaxSize;**

**c++;**



Count = 6

C = 0 < 4 ✓

C = 1 < 4 ✓

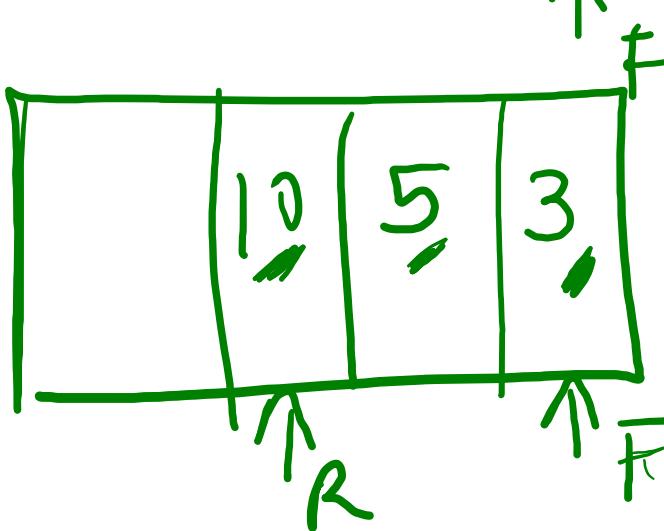
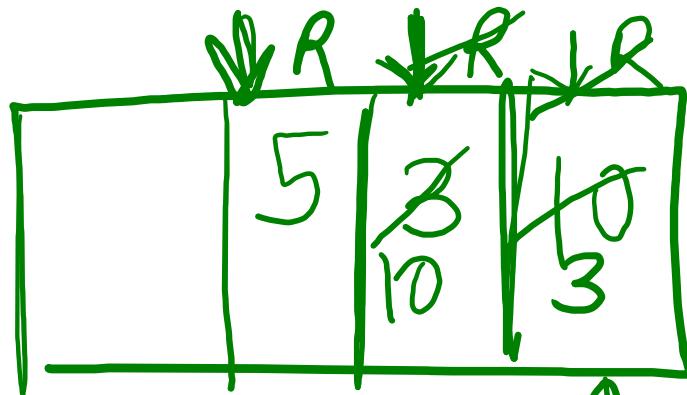
C = 2 < 4 ✓

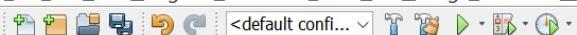
C = 3 < 4 ✓

C = 4 ↗

# Enqueue(c)

- 1. take element
  - 2. sort on ascending or descending as per need
- enqueue(10).  
enqueue(3)  
enqueue(5)



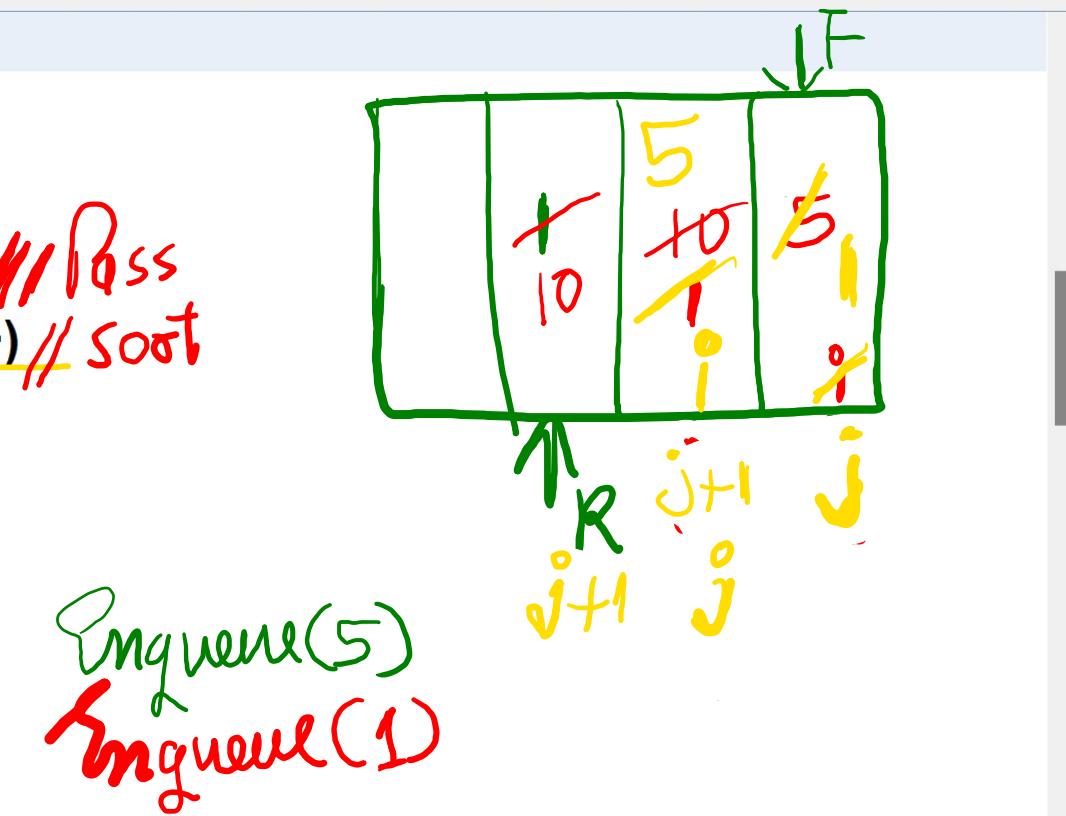


...va ThreadSync.java × SortingDemo.java × SearchDemo.java × Treemain.java × GraphDemo.java × StackDemo.java × QueueLinearDemo.java × CircularQueueDemo.java × PriorityQueueDemo.java

Source History

```

22 void enqueue(int e)
23 {
24     rear++;
25     q[rear]=e;
26     for(int i=front;i<rear;i++) // Pass
27     { for(int j=front,j<rear;j++)
28     { if(q[j]>q[j+1])
29     {
30         int t=q[j];
31         q[j]=q[j+1];
32         q[j+1]=t;
33     }
34 }
35 }
```



9821601163

Priority Queue: no LIFO no FIFO

↳ Dequeue():  $x \leftarrow \text{Max}$

↳ Dequeue(): 50      Dequeue(): 30      Priority

