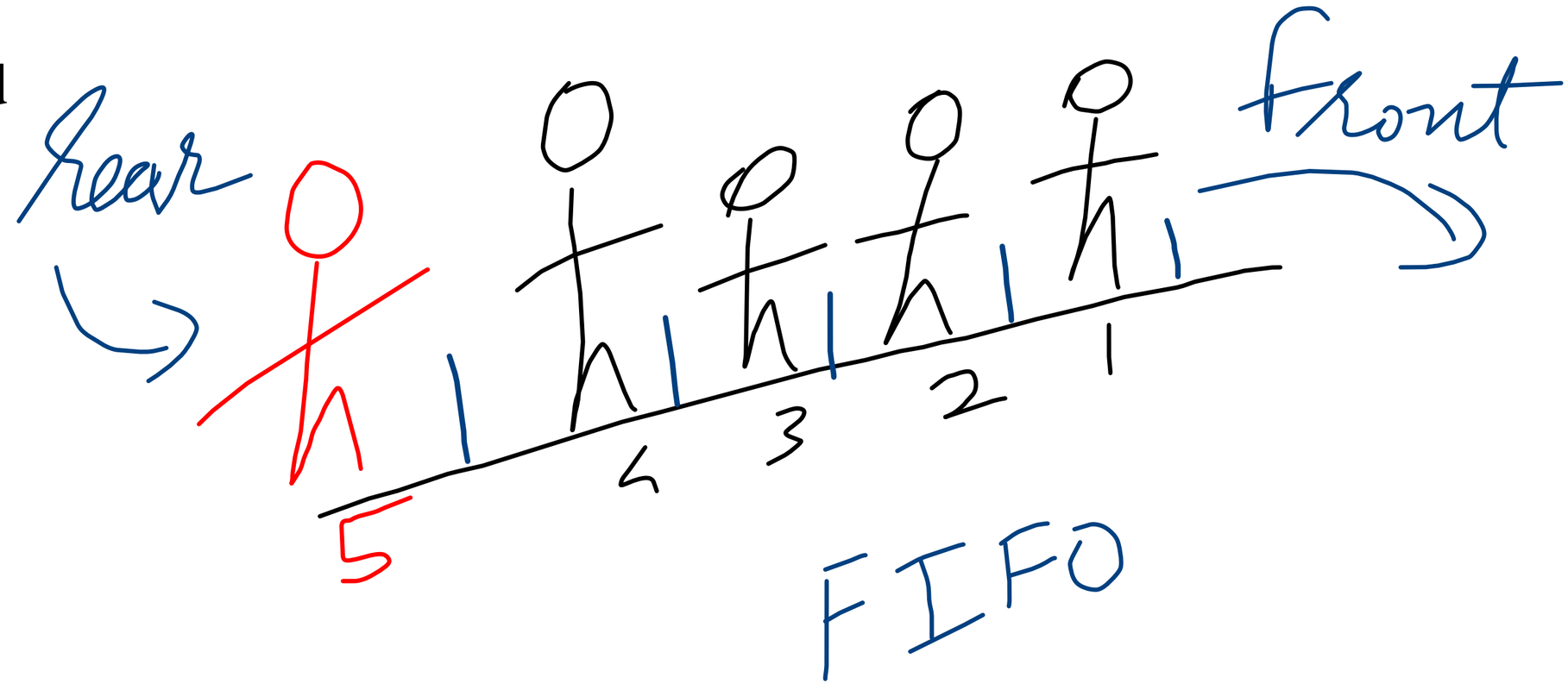
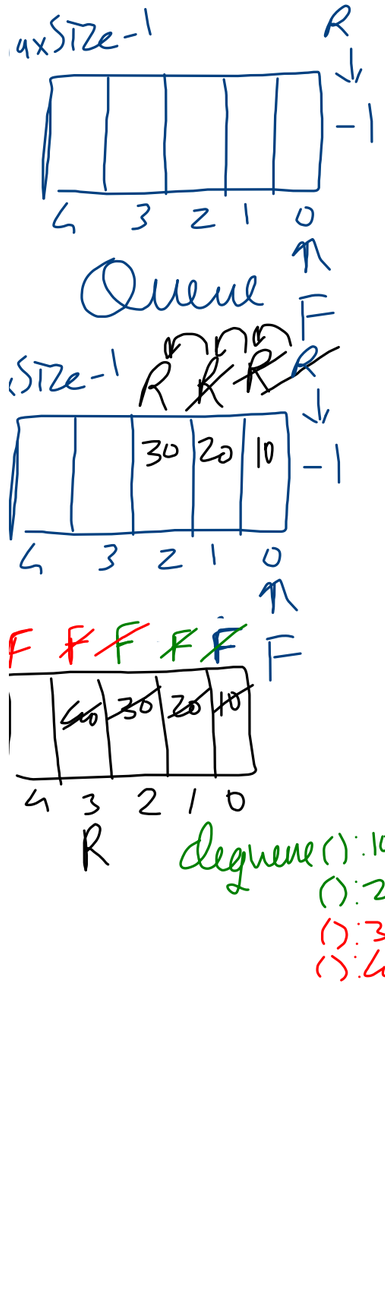
Queue:
It is a double-ended linear structure(FIFO)

which operates from either side for different operations.



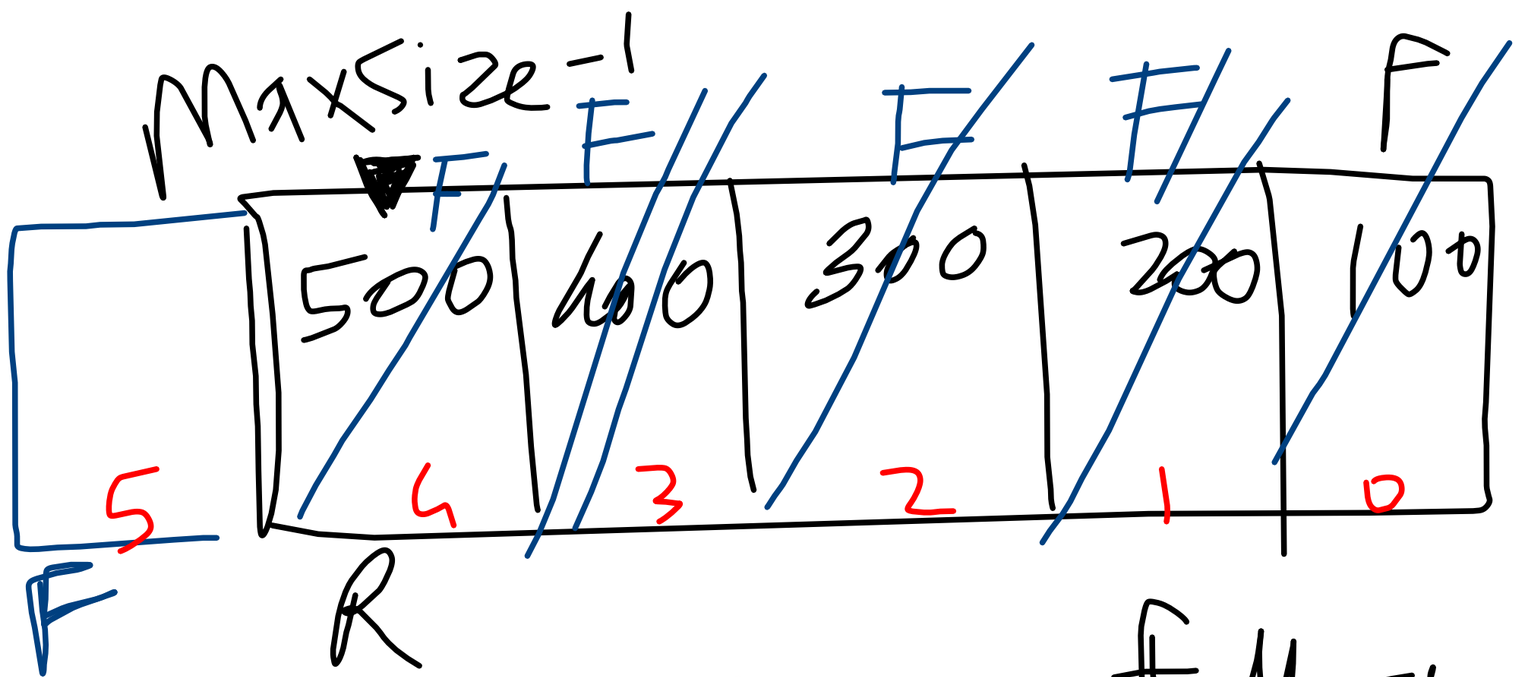
```
int queue[];
int front,rear,MaxSize;
void create_Queue(int size)
{
   rear=-1;
   front=0;
   MaxSize=size;
   queue=new int[MaxSize];
}
//enqueue:accepts an element in queue
//rear+1
void enqueue(int e)
{
   queue[++rear]=e;
}
boolean is_full()
{
   return(rear==MaxSize-1);
}
```




```
int dequeue()
{
   int temp=queue[front];
   front++;
   return(temp);
   //return(queue[front++);
}

boolean is_empty()
{
   if(front>rear)
      return true;
   else
      return false;
   //return(front>rear);
}

void print_queue()//in FIFO-->front to rear
{
   for(int i=front;i<=rear;i++)
      System.out.print(queue[i]+" - ");
}
```



Empty as front > rear

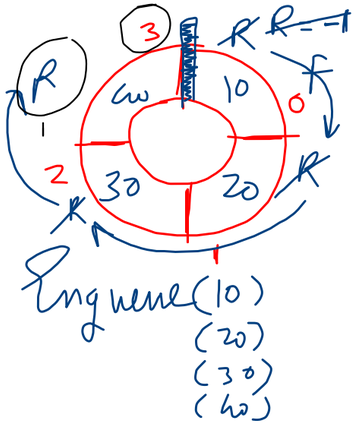
full as rear == MaxSize-1

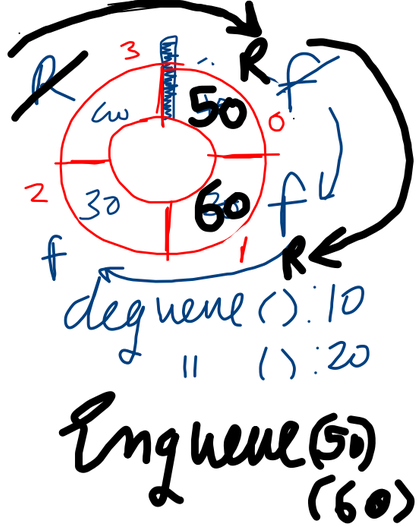
The problem with linear queue is once it moves ahead, it cannot reclaim free spaces that have been available due to dequeue: sol is Circular queue

front=(front+1)%MaxSize
rear=(rear+1)%MaxSize

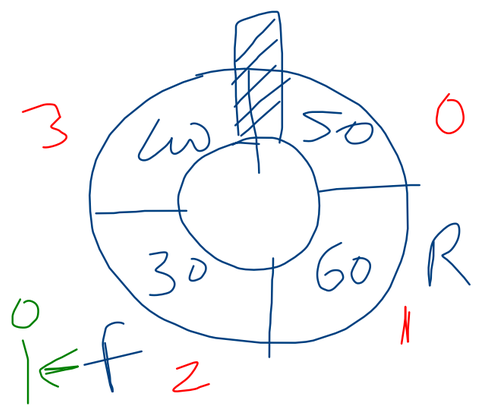
$R=(3+1)\%4$
$=4\%4=0$

$R=(0+1)\%4$
$=1\%4=1$





to print
count number of element
and print using i=(i+1)%Maxsize for
count number of times,use while



Count=4
c=0
Continue to
print Queue[i]
till c≠Count

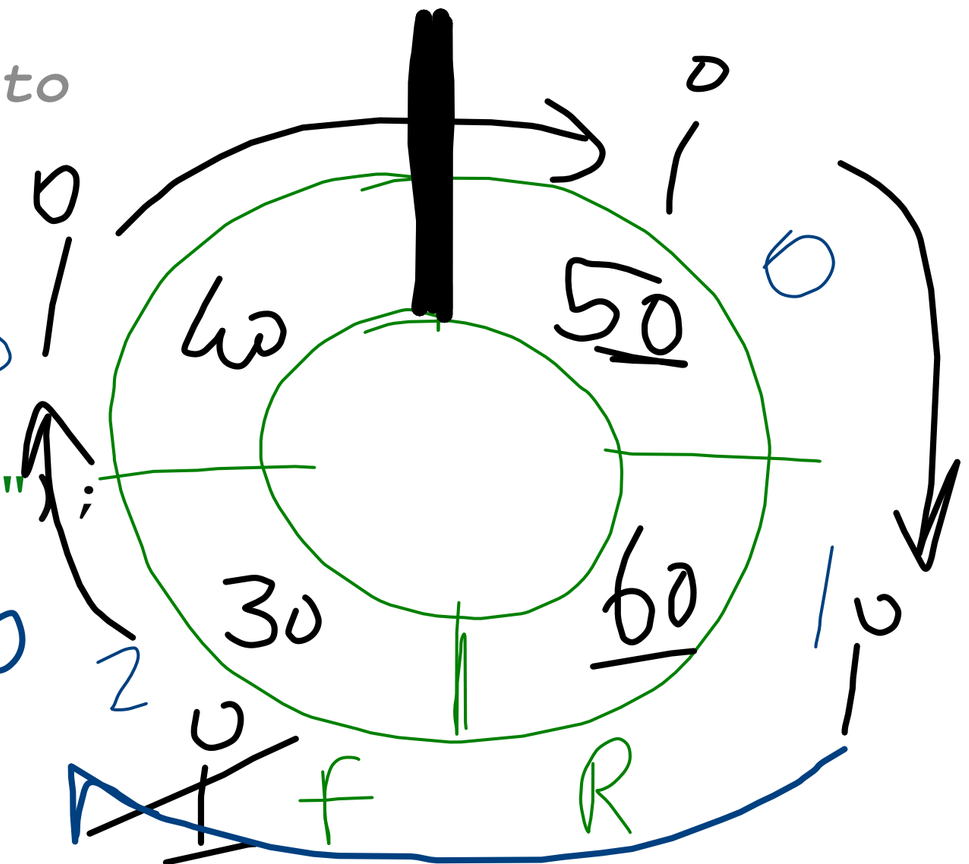
```
void print_queue()//in FIFO-->front to
rear
{
    int i=front,c=0;
    while(c<count)
    {
        System.out.print(queue[i]+" - ");
        i=(i+1)%MaxSize;
        c++;
    }

}
```

0 1 2 3

4 4 4

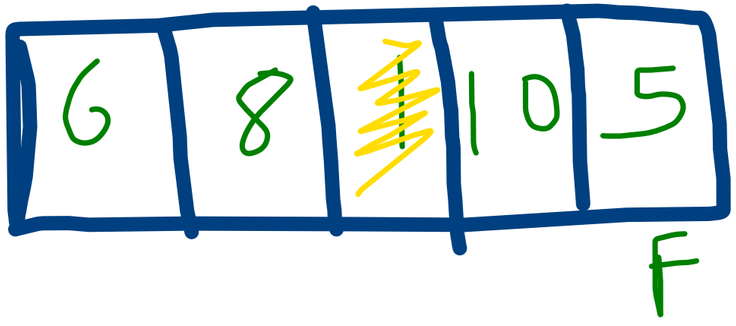
Count=4 c<=0 i=2
1 i=3
2
3 4

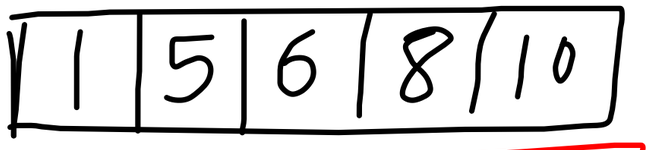


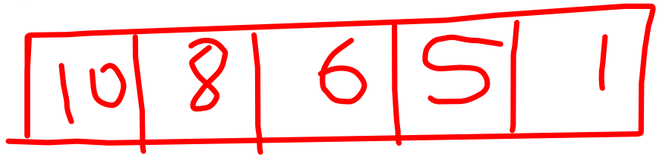
30 40 50 60

Parents ①  ② Friends ③

R



APQ : 

DPQ 

In order to implement a priority queue, we would modify enqueue of linear queue which will keep data in a sorted order as per need.

4.D-queue

It is a double-ended cube where insertion and removal is possible from either side.

Enqueue(10) : R

Enqueue(200) : L

(20) : R

|| (300) : L

(30) : Full

APPLICATIONS

1.Playlist

2.In communication as message sequencing buffer.

Communication or data will only make sense if it is processed in a sequence.

3.In streaming media.

4.In data transfer

5.In any type of scheduling task, job, or process

Generate binary number series up to n, where n is entered by the user.

n=5

1-----1

10----2

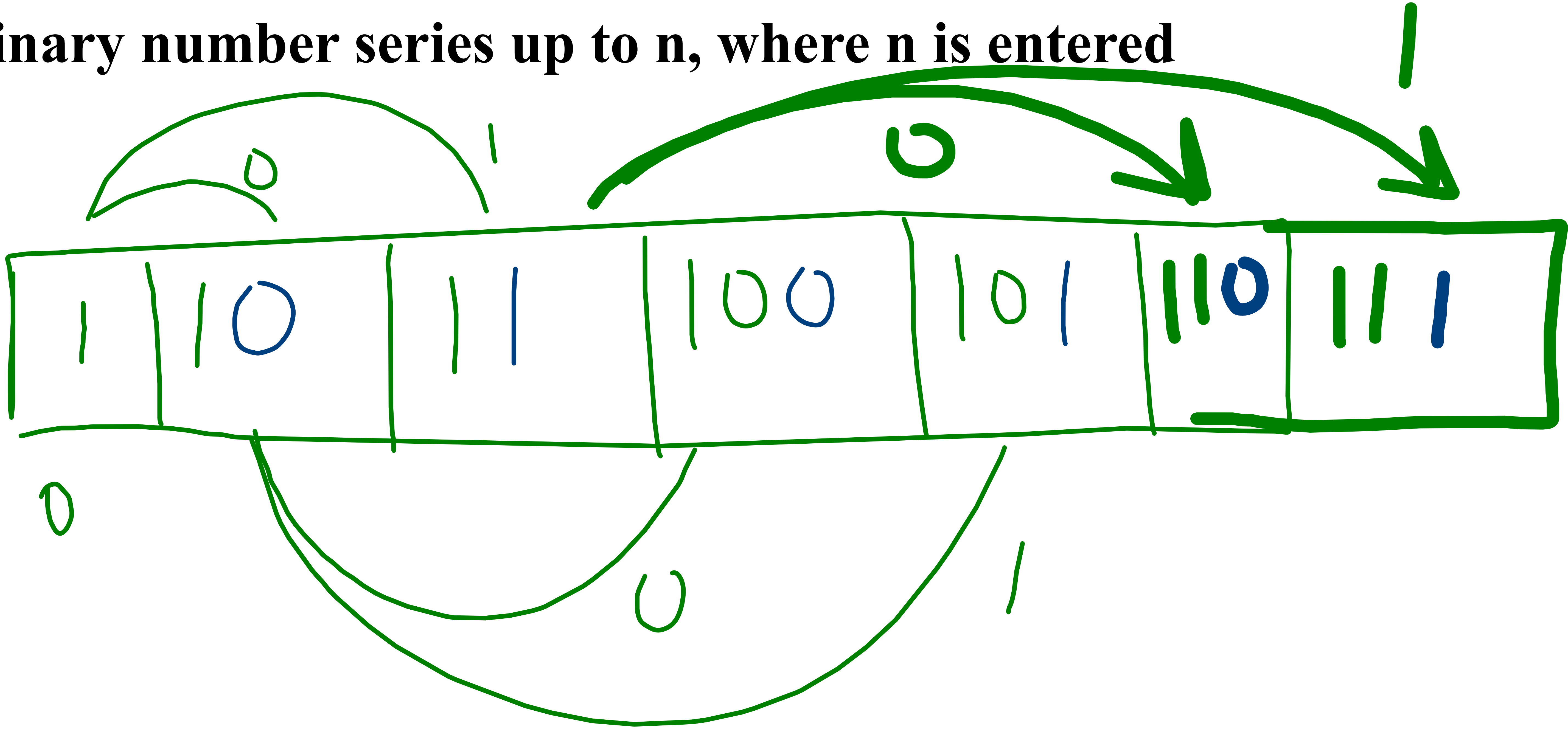
11----3

100--4

101--5

110--6

111--7



Loop

till

n

0,1,2 ... n

n=?

i=0<1

if1<q[i]+"0"

if2<-q[i]+"1"

String array

q[n]

n Reached

Print Queue

function

n

main

String g[n]

g[0] = "1"

for n-1

g[i+1] = g[i] + "0"

g[i+2] = g[i] + "1"

for each ∈ g[]

[
]

Seq = { - - - - - - - - - - }

Max_Profit = 0 Buy_day = 0 Sell_day = 0

for Buy ← 1 to 15

    for Sell ← 15 to Buy_day

        Profit = Sell - Buy

        If Profit > Max Profit

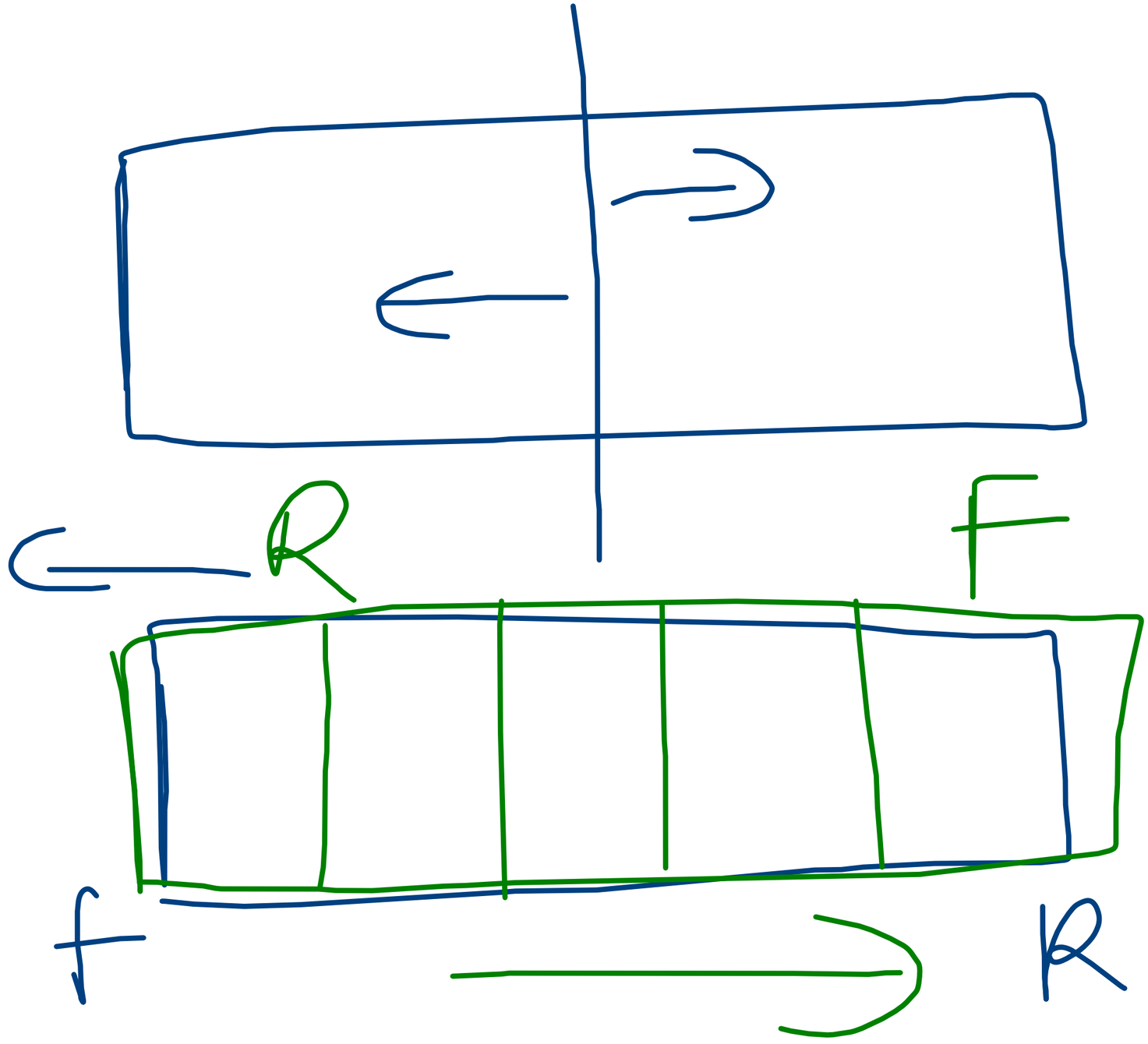
        & update

        Max_Profit, B_day, S_d

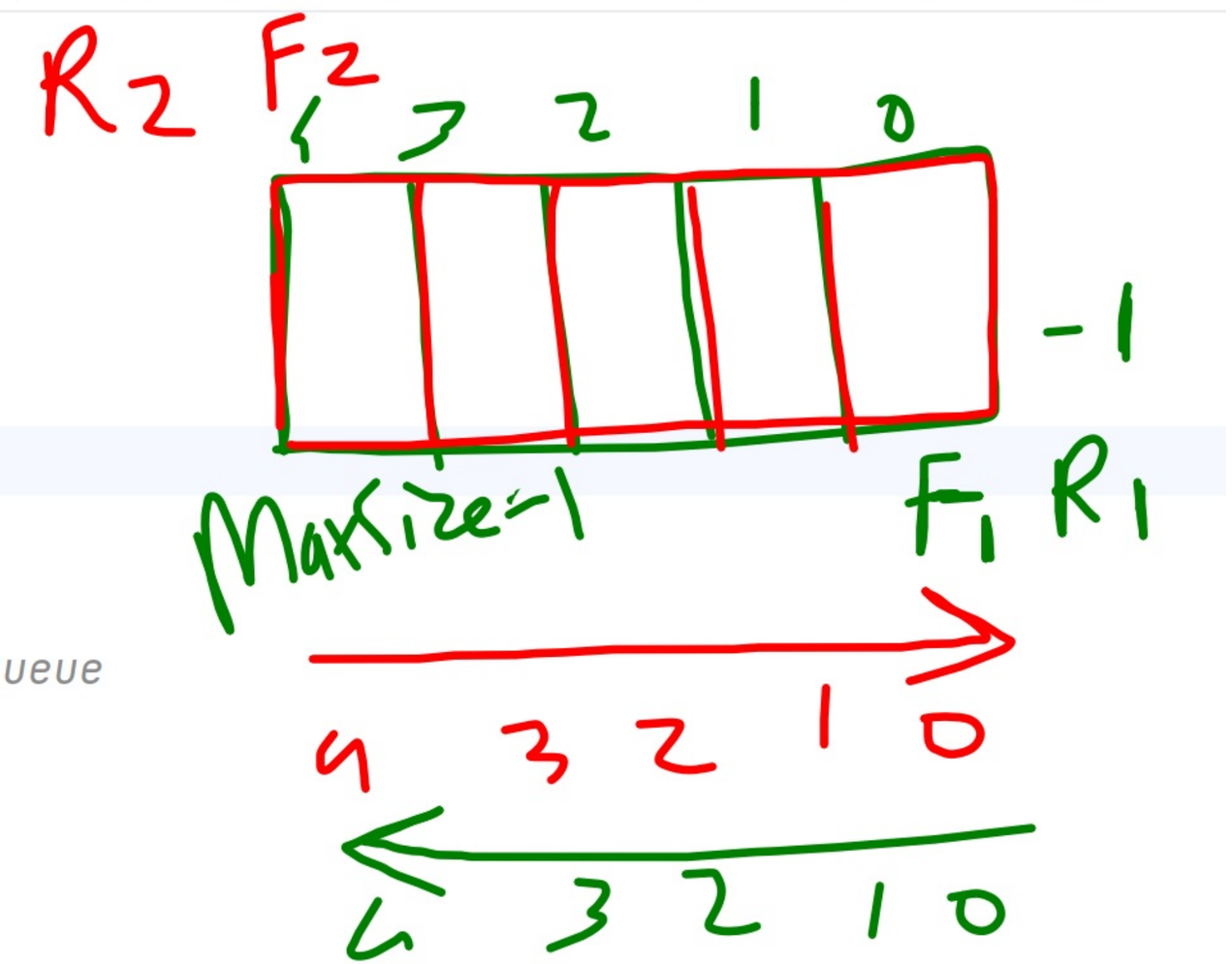
methods:String,collections
write your own notes

how to use:code,generic

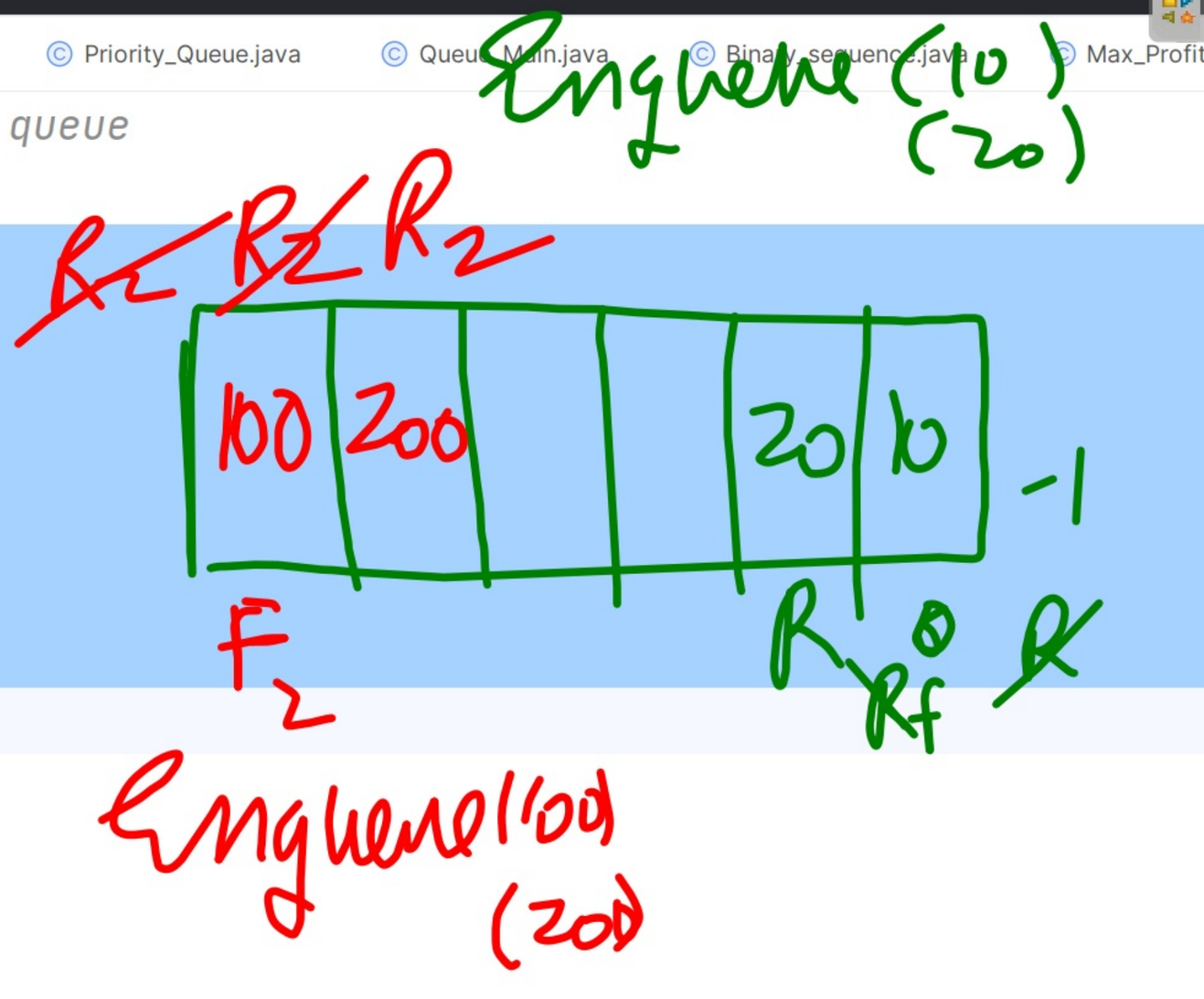
application or nature


R
F
f
R

```
7       void create_Queue(int size)
8       {
9           rear1=-1;
10          front1=0;
11          rear2=MaxSize;
12          front2=MaxSize-1;
13          MaxSize=size;
14          queue=new int[MaxSize];
15      }
16      //enqueue:accepts an element in queue
17  //rear+1
18      void enqueue(int e)
19      {
20          queue[++rear]=e;
21      }
```

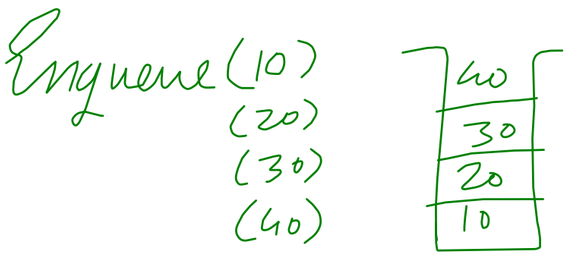


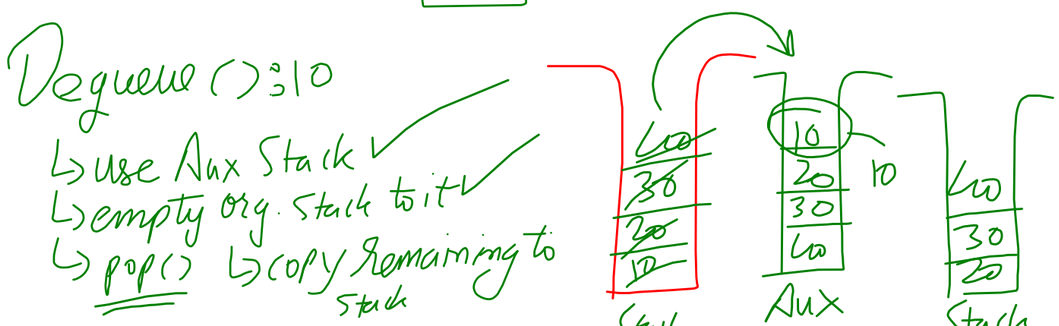
```
    //enqueue:accepts an element in queue
//rear+1
    void enqueue1(int e)
    {
        queue[++rear1]=e;
    }
    void enqueue2(int e)
    {
        queue[--rear2]=e;
    }
    boolean is_full()
    {
        return(rear==MaxSize-1);
    }
```

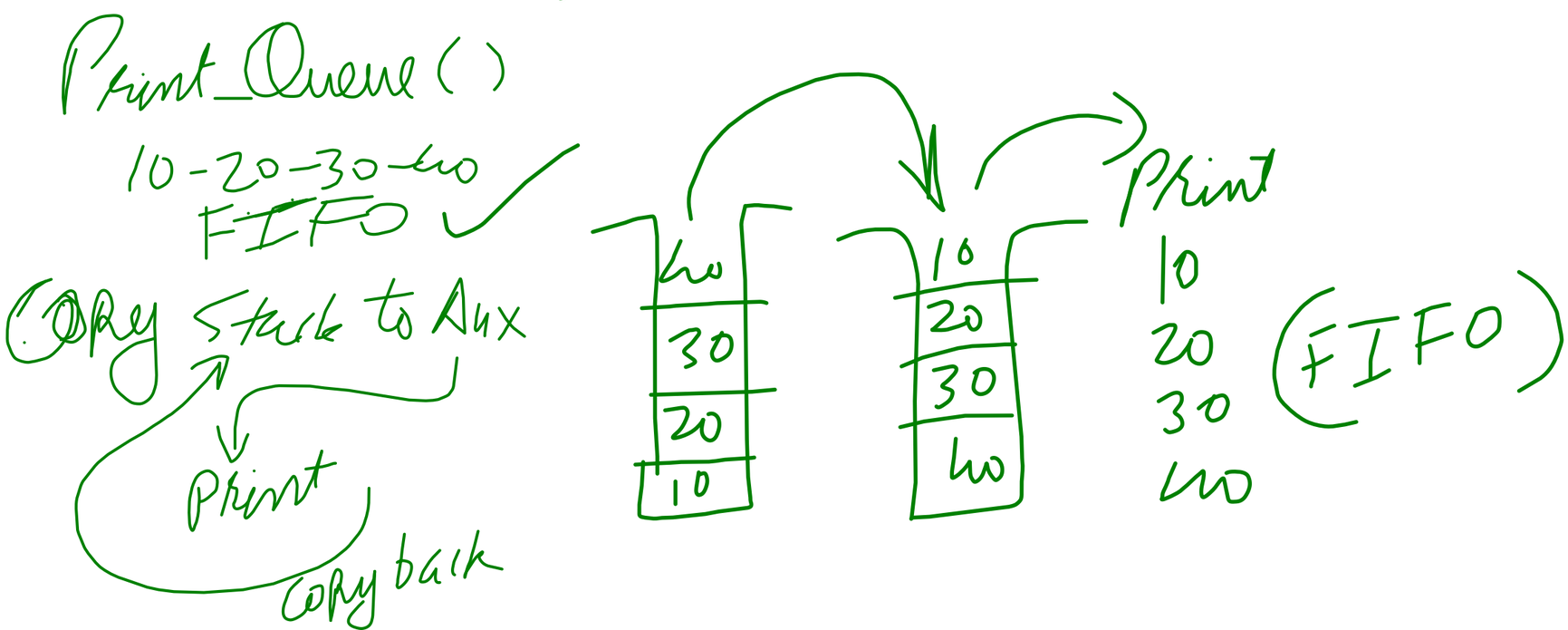


Implement two stacks in a single array where both stacks operate from opposite ends.
Push 1, Push 2, Pop 1, Pop 2, Peek 1, Peek 2, Print 1, Print 2

Implement a queue which is FIFO structure using stack structure.
Should support Enqueue /Dequeue operation along with printing of contents in Queue FIFO manner.







Implement stack using two queues.