Ans 1 :

(A) Factorial :

Stack level 1 : fact(4) // n=4 and is not equal to 1.
Stack level 2 : fact(3)
Stack level 3 : fact(2)
Stack level 4 : fact(1)
Stack level 5 : fact(0) // now, n=1 so we return 1 from the function.

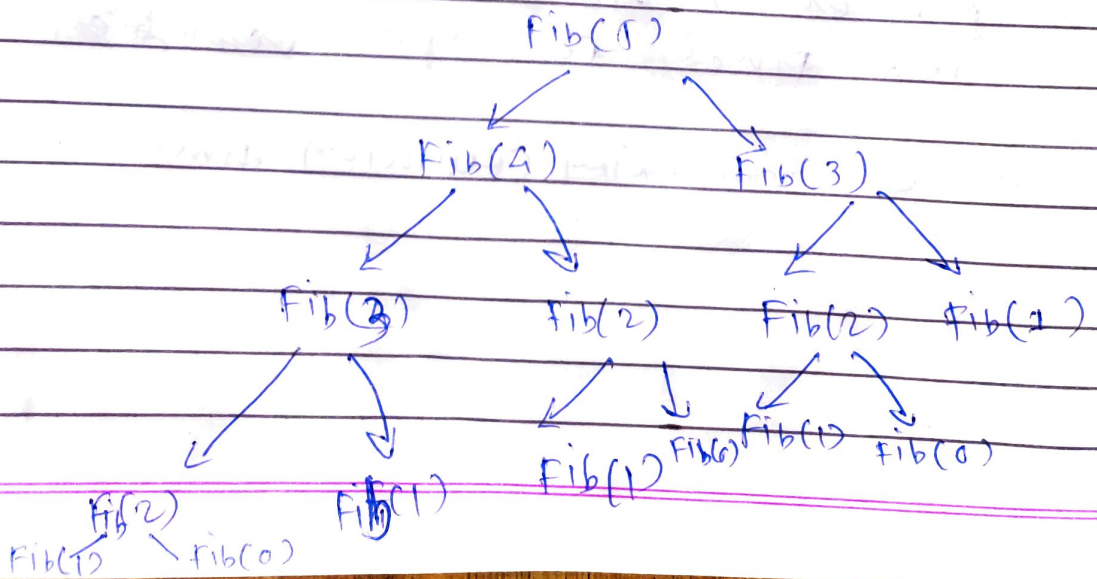
↓ returning values

Stack level 5 : 2 * fact(1) = 2
Stack level 4 : 3 * 2 = 6
Stack level 3 : 4 * 6 = 24
Stack level 2 : 5 * 24 = 120
Stack level 1 : 1 * 120 = 120

Ans

(B) Fibonacci :

Fibonacci stack tree



— Stack level 1 : Fib(5)

Stack level 2 : Fib(4) Fib(3)

Stack level 3 : fib(3)

Stack level 4 : Fib(2)

Stack level 5 : Fib(1) Fib(0)

Stack level 6 : Fib(2)

Stack level 7 : Fib(1) Fib(0)

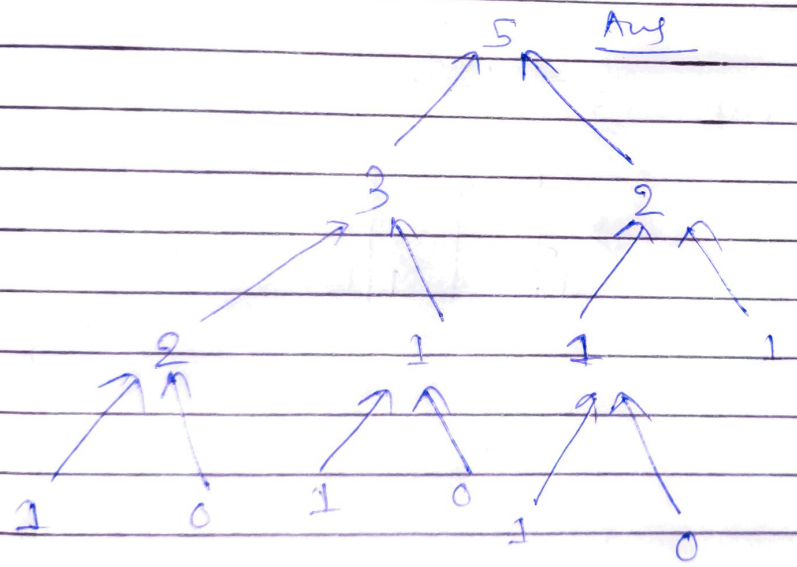
Stack level 8 : Fib(3)

Stack level 9 : Fib(2)

Stack level 10 : Fib(1)

Stack level 11 : Fib(0)

Result tree :



(c) Tower of hanoi.

Stack would be used to implement tower of hanoi algorithm.

Approach:

let three stacks used with same capa as number of dist and will have uni name as 's', 'a', and 'd'.

Step 1:

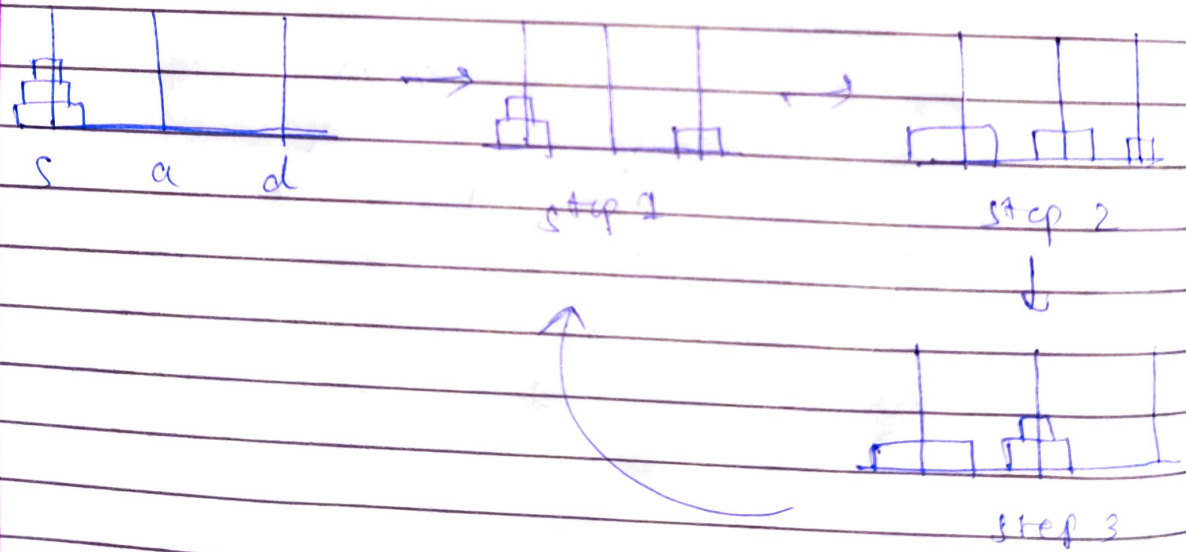
Shift First disk s → d

Step 2:

Shift 2nd disk s → a

Step 3:

Shift 3rd disk d → a



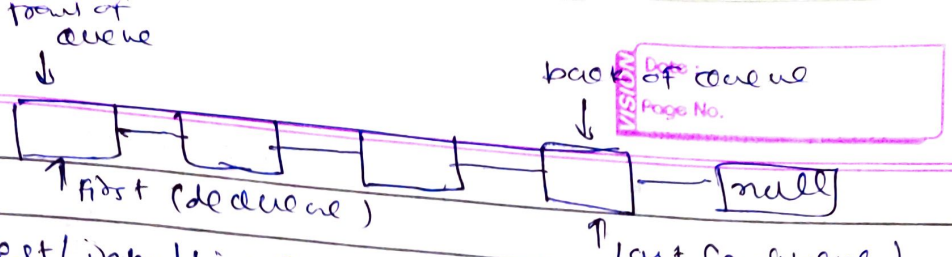
If these three step keeps repeating for every disk at the top of source rod the it will transfer all disk to destination rod.

- From this, the pattern ~~ate~~ it will be.

Pattern:

Shift 'n-1' disk from s → a
shift last disk from s → d
shift 'n-1' disks from d → a

Ans: 2




```
class TestLinkedListQueue
{
    private Node first, last;
    private int N;
    private class Node
    {
        String item;
        Node next;
    }

    public void enqueue (String item)
    {
        Node oldlast = last;
        last = new Node();
        last.item = item;
        last.next = null;
        if (isEmpty())
            first = last;
        else
            oldlast.next = last;
        N++;
    }

    public String dequeue()
    {
        String item = first.item;
        first = first.next;
        if (isEmpty())
            last = null;
        N--;
        return item;
    }
```

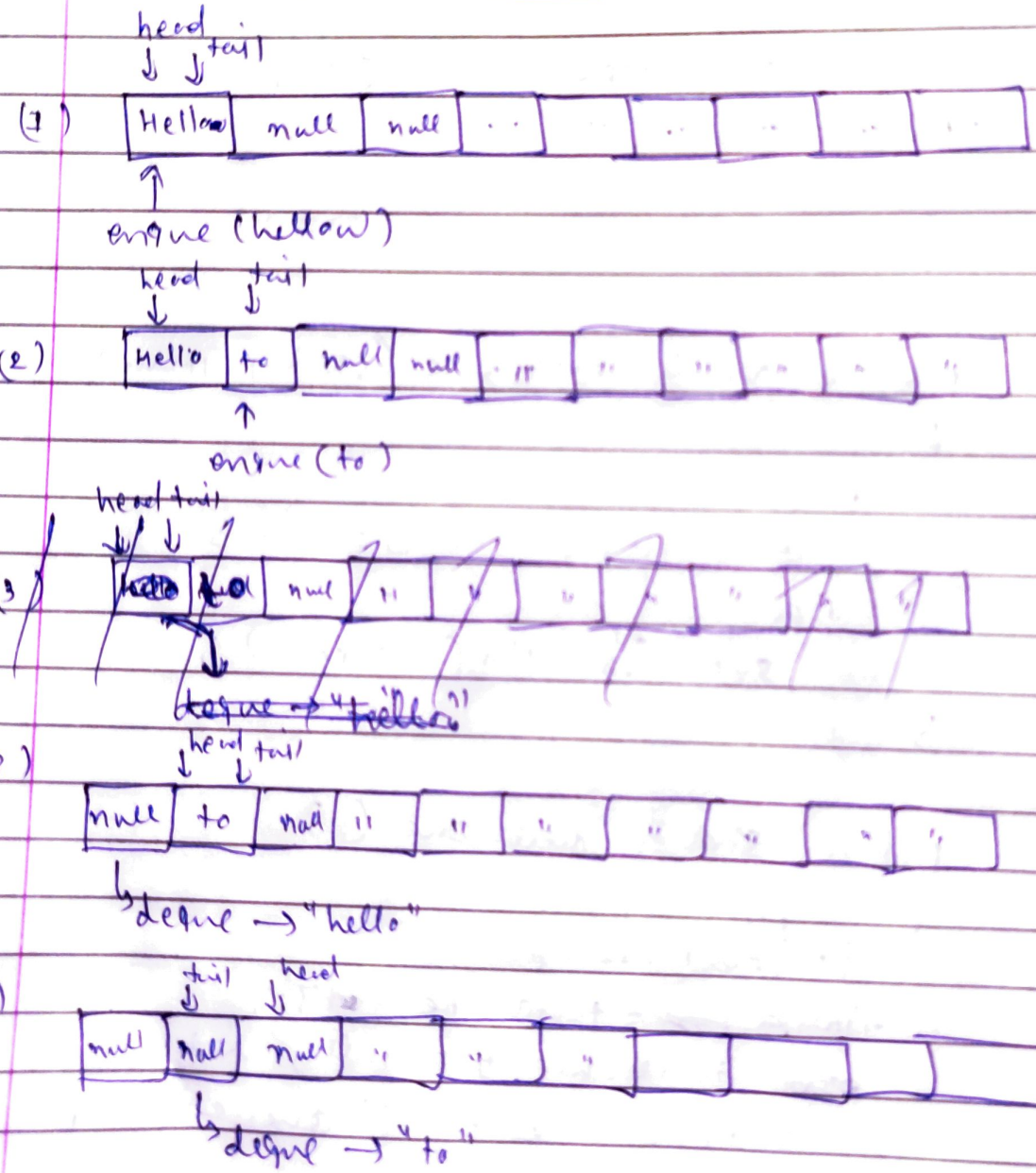
```
public boolean isEmpty()
{
    return first == null;
}

public int size()
{
    return N;
}

}
```

Ans: 3

```
class TestArrayQueue
{
    public String[] testQ;
    public int head, tail;

    TestArrayQueue()
    {
    this.testQ = new String[10];
    head = 0;
    tail = 0;
    }

    public void enqueue(String item)
    {
        if (tail < 10)
            testQ[tail++] = item;
        else
            System.out.println("error");
    }

    public String dequeue()
    {
        if (head < tail) {
    String item = testQ[head++];
            testQ[head] = null; return item; }
        System.out.println("Queue is empty");
    }

    public int size()
    {  if (!isEmpty())
            return tail - head;
    }
```

```
public boolean isEmpty()
{
    if (head > tail)
        return true;
    else
        return false;
}
```

→ Queue operation:



(5) Here head pointer is greater than tail which means queue is empty

Ans: 4

(B) Input String = "Hello to the World";
total character = 18

→ hashcode eqn = $s[0] * 31^{(15-1)}$

$$= \sum_{i=0}^{n-1} s[i]\, 31^{(17-i)}$$

$$= \underset{H}{s[0]}\, 31^{16} + \underset{e}{s[1]}\, 31^{15} + \underset{l}{s[2]}\, 31^{14} + \underset{l}{s[3]}\, 31^{13} + \underset{o}{s[4]}\, 31^{12} + \underset{\text{" "}}{s[5]}\, 31^{11} + \underset{t}{s[6]}\, 31^{10} + \underset{o}{s[7]}\, 31^{9} + \underset{\text{" "}}{s[8]}\, 31^{8} + \underset{t}{s[9]}\, 31^{7} + \underset{h}{s[10]}\, 31^{6} + \underset{e}{s[11]}\, 31^{5} + \underset{\text{" "}}{s[12]}\, 31^{4} + \underset{w}{s[13]}\, 31^{3} + \underset{o}{s[14]}\, 31^{2} + \underset{r}{s[15]}\, 31^{1} + \underset{l}{s[16]}\, 31^{1} + \underset{d}{s[17]}\, 31^{0}$$

$$= 72\,31^{17} + 101\,31^{16} + 101\,31^{15} + 108\,31^{14} + 111\,31^{13} + 32\,31^{12} + 116\,31^{11} + 111\,31^{10} + 32\,31^{9} + 116\,31^{8} + 104\,31^{7} + 101\,31^{6} + 32\,31^{5} + 87\,31^{4} + 111\,31^{3} + 114\,31^{2} + 108\,31^{1} + 100\,31^{0}$$

= It is very big value which Java program is not able to show and that why it display negative value due to overflow.

= By applying function in Java code the Answer is,

= 1699696879868464200000000000

(A) hashcode:

hashcode function returns integer value after calculating some arithmatic operation on its reference value. hashcode will be same any object with same class type.

equals:

equals function do deep comparision between object and check whether both objects have same data in it. If two objects are same after comparision with equal function then hashcode function will also return the same integer value but This will not be correct for vice-versa case.

compareTo:

compareTo function can be used after extending comparable interface. It compare given objects and return negative value if left object is smaller than right object, return positive value if left object is greater than right object. If both objects are same then it return zero. CompareTo method is used to do sorting for objects.

Ans (7)

A*B/C+(D+E-(F*(G/H)))

current	Stack	Expression
A	empty	A
*	*	A
B	*	AB
/	/	AB*
C	/	AB*C
+	+	AB*C/
(	(+	AB*C/
D	(+	AB*C/D
+	+(+	AB*C/D
E	+(+	AB*C/DE
-	-(+	AB*C/DE+
(	(-(+	AB*C/DE+
F	(-(+	AB*C/DE+F
*	*(-(+	AB*C/DE+F
(	(*(-(+	AB*C/DE+F(
G	(*(-(+	AB*C/DE+FG
/	/(*(-(+	AB*C/DE+FG
H	/(*(-(+	AB*C/DE+FGH
)	*(-(+	AB*C/DE+FGH/
)	-(+	AB*C/DE+FGH/*
)	+	AB*C/DE+FGH/*-
	empty	AB*C/DE+FGH/*-+

Postfix