RMO:-

add of A[15][20] =

$1500 + [(15+15) \times 26 + 5] \times 1$

$= 1500 + [30 \times 26 + 5] \times 1$

$= 1500 + [780 + 5] \times 1$

$= 1500 + 785$

$= 2285$

CMO:-

add of A[15][20] =

$1500 + [5 \times 26 + 30] \times 1$

$= 1500 + [130 + 30] \times 1$

$= 1660$

Q. A[30][20], w=8 bytes, Find location of A[5][10] if A[4][5] is stored at 3000.

→ R = 30, C = 20, w = 8

Given,

$$3000 = B + [4 \times 20 + 5] \times 8$$

$$\Rightarrow 3000 = B + 85 \times 8$$

$$\Rightarrow 3000 = B + 680$$

$$\Rightarrow B = 3000 - 680$$

$$\Rightarrow B = 2320$$

add of $A[5][10] = 2320 + [5 \times 20 + 10] \times 8$

$$= 2320 + [110] \times 8$$

$$= 2320 + 880$$

$$= 3200$$

Q. A matrix B[10][20] is stored in the memory with ~~each~~ w=2. If base address at B[2][1] is 2140, find address of B[5][4] if matrix is in CMO.

→ Here, Base address is not of [0][0] but [2][1], we will consider it as LR and LC in case of fixed size array.

So,

add of A[5][4] = 2140 + [illegible] 10 × (4-1) + (5-2)] x2

= 2140 + [30 + 3] x 2

= 2140 + 66

= 2206

→ Recursion :-

A function is called recursive if a statement within the body of a function calls the same function. This process is called recursion.

```
void f()
{
   ...
   f();
   ...
}
```

direct recursion.

```
void f()
{
   ...
   g();
   ...
}
void g()
{
   ...
   f();
   ...
}
```

indirect recursion.

In case of indirect recursion, a function call second function but that second function calls first caller function.

A recursive function can go infinite loop, to avoid infinite running of recursive function, function should have -

• base condition :- There should must be one base condition, such that, when this condition is met function stops calling itself recursively.

• Iterative condition :-

The recursive calls should progress in such a way that each time a recursive call is made it comes closer to the base condition.

→ Difference b/w recursive and iterative

• In recursive function call itself, until base condition met.

In iterative approach, loop runs until condition fails.

• Iterative approach involves four steps: initialization, condition, execution and updation.

In recursive function, only base condition is specified.

- Recursion is slower than iteration due to overhead of maintaining stack whereas iteration don't have such process.

- Recursion take more memory than iteration.

→ Tail recursion :-

A recursive function is said to be tail recursive if there is no pending operations performed on return from a recursive call.

ex.- factorial as non-tail recursive and tail recursive :-

factorial as non-tail recursive -

```
int fact(int n){
    if(n==0) return 1;
    else return(n * fact(n-1));
}
```

Factorial as tail recursion

```
int fact(int n, int result){
    if(n==0) return result;
    else return( fact(n-1, n*result));
}
```

Recursive function to print fabonicci series up to n^{th} element

```
int fab(int n){
    static int n1=0, n2=1, n3;
    if(n>1){
        n3 = n1 + n2;
        n1 = n2;
        n2 = n3;
        cout<< n3 <<" ";
        fab(n-1);
    }
}

int main(){
    int n;
    cout<<"input integer "<<endl;
    cin>> n;
```

```
if(n==0) cout<<" 0 ";
else if(n==1) cout<<" 0 1 ";
  else
  {
     cout<<" 0 1 ";
     fab(n);
  }
  return 0;
}
```

→ Applications of recursion :-

→ Tower of Hanoi :-

In this puzzle we have 3 rods and n-different size disk. Our work is to move entire stack of disk from initial given rod to destination rod using auxalary rod, and while doing this we have to follow all these rules.

- Only one disk can be moved at a time.
- You can only remove top disk disk from one steck and put it on top of another stack.

- Do not put bigger disk on smaller disk.

In this problem, we have assumed
initial rod as 'A'
aux rod as 'B'
destination rod as 'C'

NOTE:- Minimum no of steps needed to perform to move 'n' rods from 'A' to 'C' is

$2^n - 1$

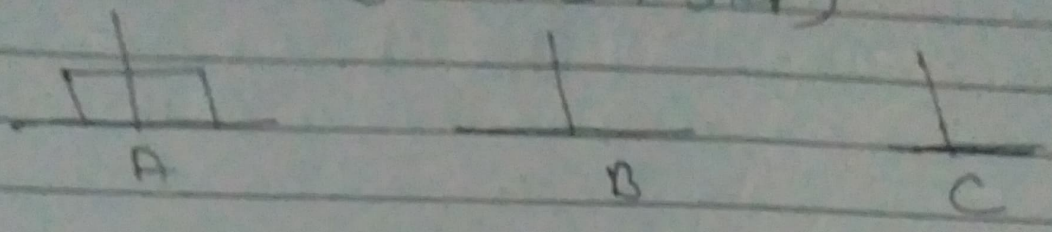
→ Steps to following while solving it in one by one step:- (for subjective question).

- If no of disks are odd start by putting first disk from initial rod to destination rod and then progress.
- If no of disks are even start by putting first disk from initial rod

to auxiliary rod and then progress.

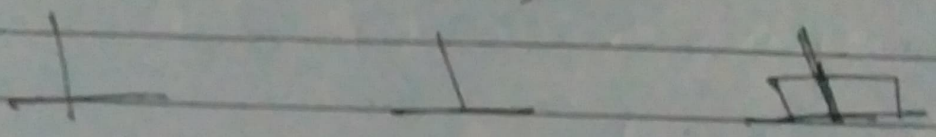
Ex. -

- For $n=1$ $(2^1-1 = 1 \text{ step})$

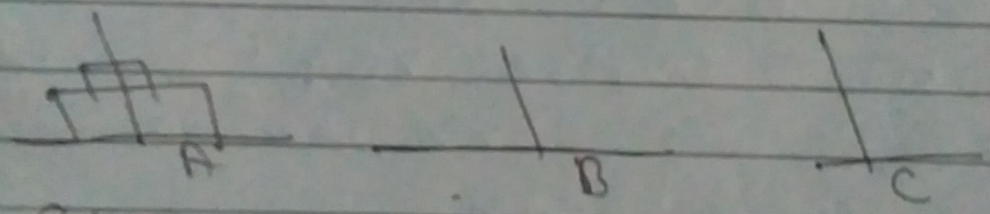


Since 'n' is odd, put from A to C

$\langle A, C \rangle$

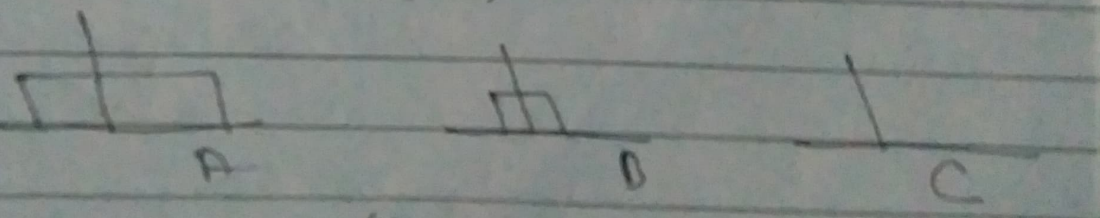


- For $n=2$ (even) $(2^2-1 = 3 \text{ steps})$

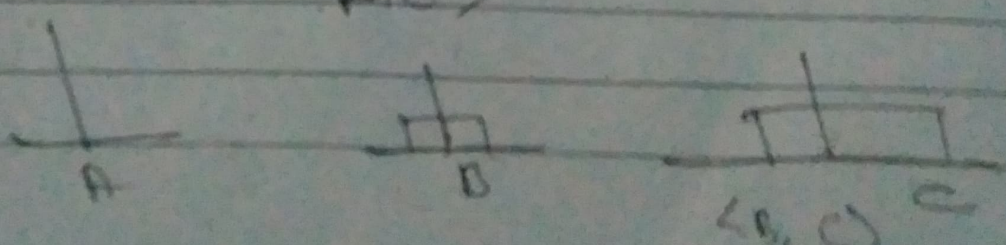


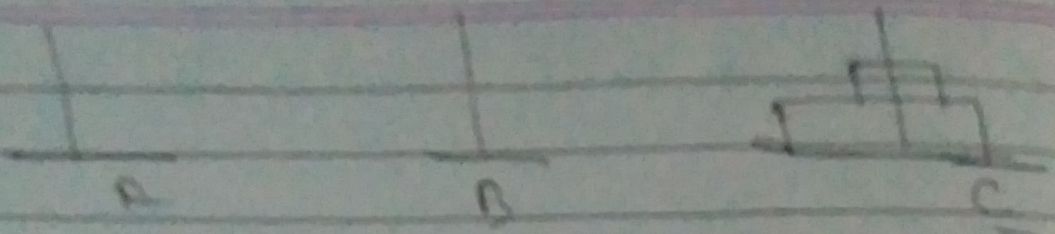
Since, n is even move from A to B

$\langle A, B \rangle$

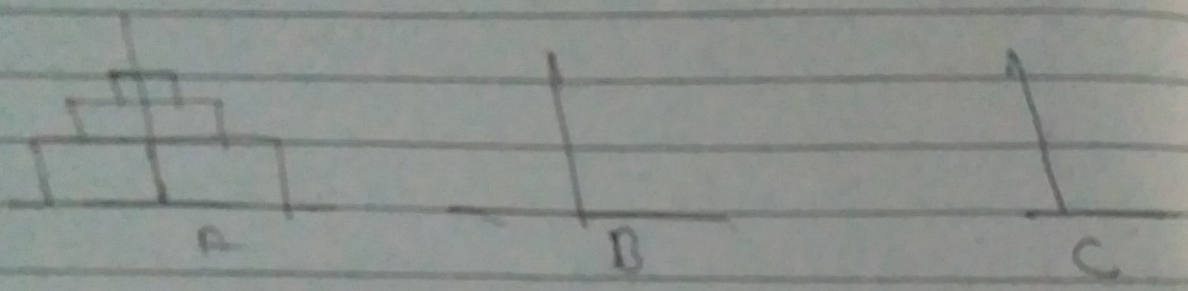


$\langle A, C \rangle$

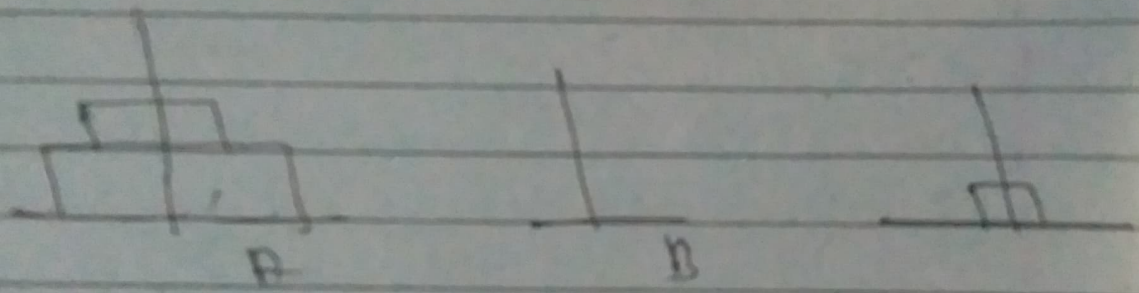




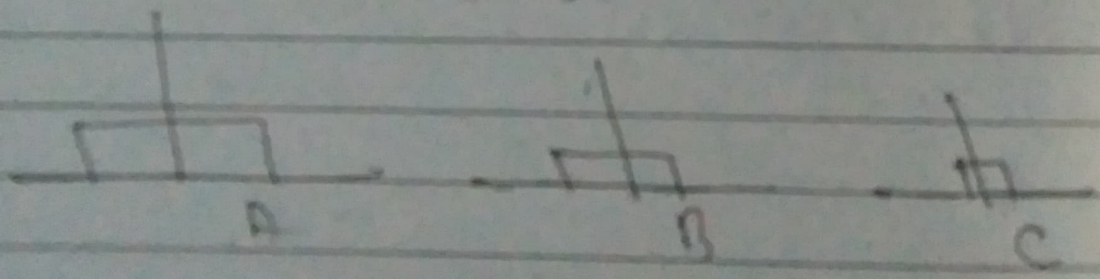
For n=3 again (odd) ($2^3-1 = 7$ steps)



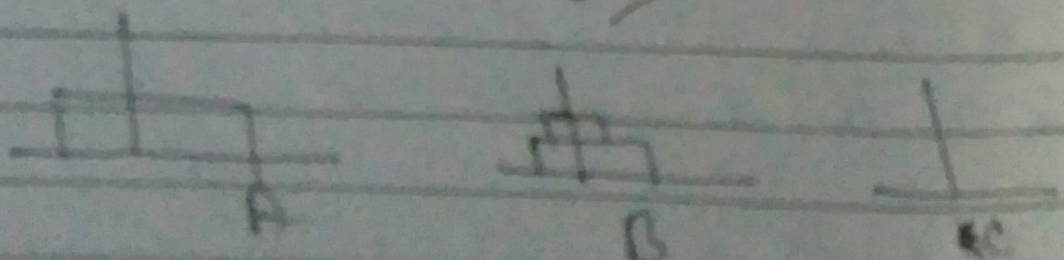
<A, C>



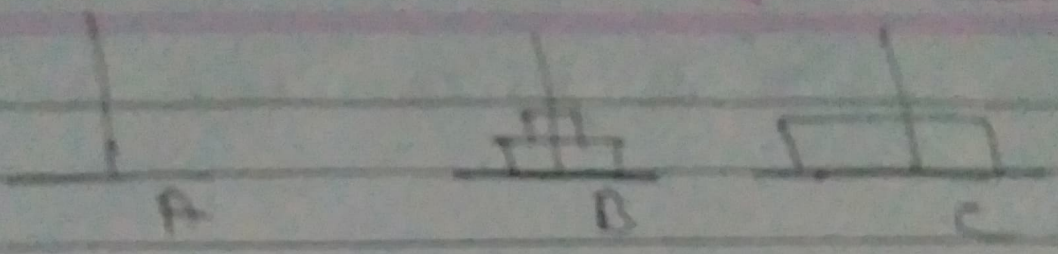
<A B>

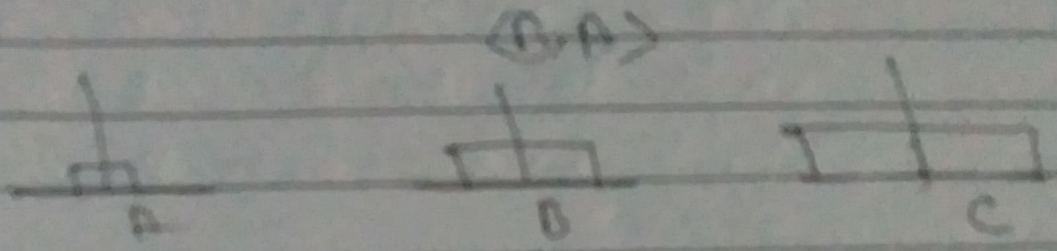


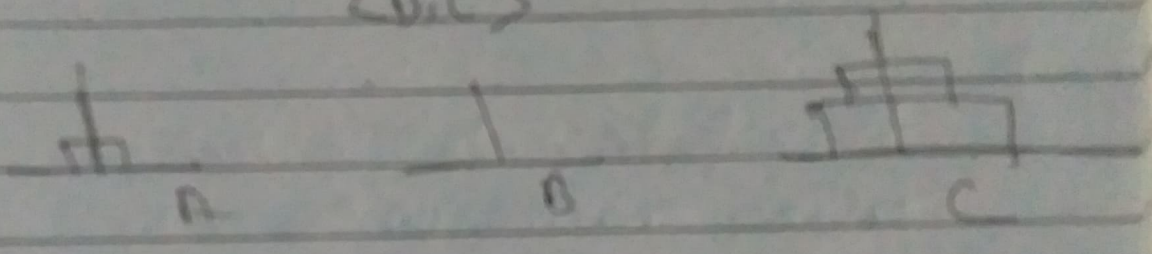
<C, B>

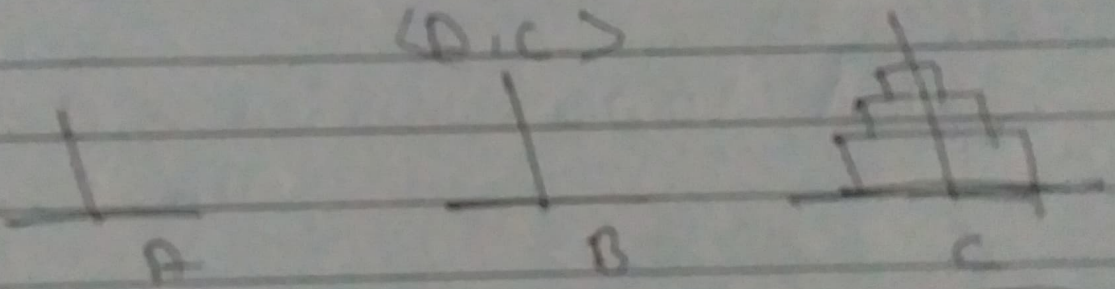


<A, C>









As you can see pattern in these proces of moving disks, if we want to put 'n' disk from 'A' to 'C' we first put 'n-1' disk from 'A' to 'B' then 'n' disk from A to C then again 'n-1' disks from 'B' to 'C'.

So, for programming purpose steps will be.

Shift 'n-1' disks from A (initial) to B (aux.)
shift last disk from A (initial) to C (destina
Shift 'n-1' disks from B (aux.) to C (des.)

Code :-

```
#include <iostream>
using namespace std;

void towerofhanoi(int n, char from, char
                  aux, char to){

    if(n==1){
        cout<<" shift disk 1 from"<<from
            <<" to "<<to<<endl;
        return;
    }

    towerofhanoi(n-1, from, to, aux);
    cout<<" shift disk "<<n<<" from "<<
        from<<" to "<<to<<endl;

    towerofhanoi(n-1, aux, from, to);
}
```

```
int main(){
    int n = 4;
    towerOfHanoi(n, 'A', 'B', 'C');
    return 0;
}
```