## ⑤ Sum of digits using Recursion

Q. Given 567
return → 5+6+7 → 18.

→ Program →

```
int Sum(int n){
if(n==0)
return 0;
    int temp = n%10;
    return temp + Sum(n/10)
```

Sum →

56④

time = O(digits in no:)
space = O( )

---

## ⑥ Rope cutting problem.

→ n = 5
a = 2, b = 5, c = 1 } Ans 5 pieces of length 1.

Sol: if a/b/c → 1 return n.
if n = even a/b/c = 2 return (n/2);

} Couldn't figure out why recursion was needed.

---

Answer: → int maxPieces(int n, int a, int b, int c)

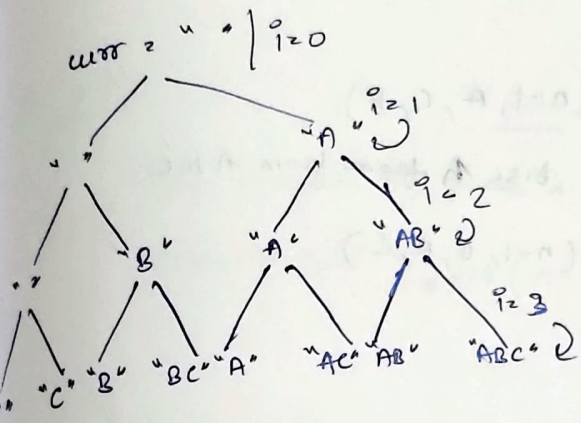
```
{ if(n==0) return 0;
  if(n<0) return -1;
  int res = max(maxPieces(n-a,a,b,c)
                maxPieces(n-b,a,b,c)
                maxPieces(n-c,a,b,c));
  if(res==-1)
      return -1;
  return res;
}
```

Time → O(3^n)

## ⑦ Generate Subsets

I/P → ABC

O/P → "", "A", "B", "C", "AB", "BC", "AC", "ABC".




$O(2^n)$

Not allowed in Java.

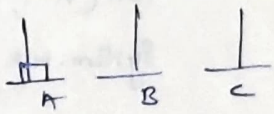
```
void subsets (String s,
              String curr = "",
              int i = 0)
{ if (i == s.length())
  { print(curr);
    return;
  }
  subsets (s, curr, i+1);
  subsets (s, curr + s[i],
           i+1);
}
```

---

## ⑧ Tower of Hanoi

**Rules**

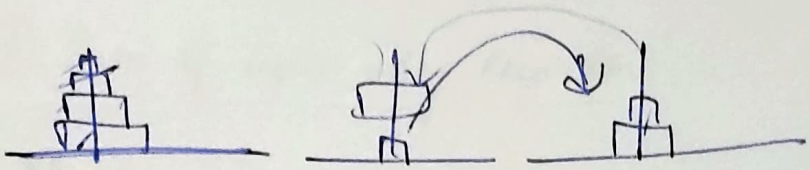
1) Only 1 disc can move at a time.
2) Always larger below smaller.
3) Only top disc can be moved.




**Q** → Move all disc given from A to C.

**Sol** → $2^n - 1$ moves for n disks. → $T(n) = 2T(n-1) + 1$ Recurrion tree method.

idea → Move (all) disks from A to B → Step 1.
Move last disk from A to C → Step 2.
Move (all) disks from B to C → Step 3.



TOH (n, A, B, C).

- TOH (n-1, A, C, B).
- Move disc ~~A from~~ from A to C.
- TOH (n-1, B, A, C).

Code

```
Void TOH (int n, char A, char B, char C)
{
    if (n == 1)
    { System.out.println("Move 1 from "+A+" to "+C);
      return;
    }
    TOH(n-1, A, C, B);
    System.out.println("Move " + n + " from " + A + " to " + C);
    TOH(n-1, B, A, C);
}
```

# Josephus Problem

n → total people.

k → kill every kth person.

Find out who survives.

* Code

Func → jos(int n, int k).

Base case ⇒ one person is left.

Recursive call ⇒ * Jos(n-1, k) → n-1 as a person will be killed every time.
↳ (1)

⇒ * But we need to make changes as when func is freshly called again it will always label 0 as 1st person.

Not the next person to one getting killed → which we want.

⇒ General Case

⇒ In first → k-1 person is killed.

& k will become the first person.

∴ Jos(n-1, k) + k — (2)

⇒ In the equation too, (+k) can become > n. ∴ we use %.

∴ Final program is :—

```
⇒ int jos(int n, int k)
{
  if(n==1)
    {return 0;}
  else
    {return (jos(n-1,k)+k)%n;}
}
```

O(n)