Recursion

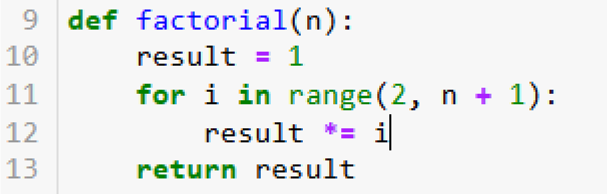
Suman Pandey
Email – suman17@gist.ac.kr
Office – 205, C3 building - C wing, building EECS
Phone – 010-3742-1791
Extension - 3165

What is recursion

- Repetition within a computer program
  - ***Iteration*** while-loop and for-loop
  - ***Recursion*** (elegant and powerful alternative to loops)
- An important technique in the study of data structures and algorithms.

Iterative definition of Factorial

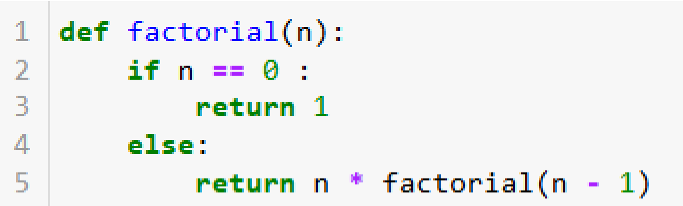
$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1 & \text{if } n \geq 1. \end{cases}$$



```python
def factorial(n):
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
```

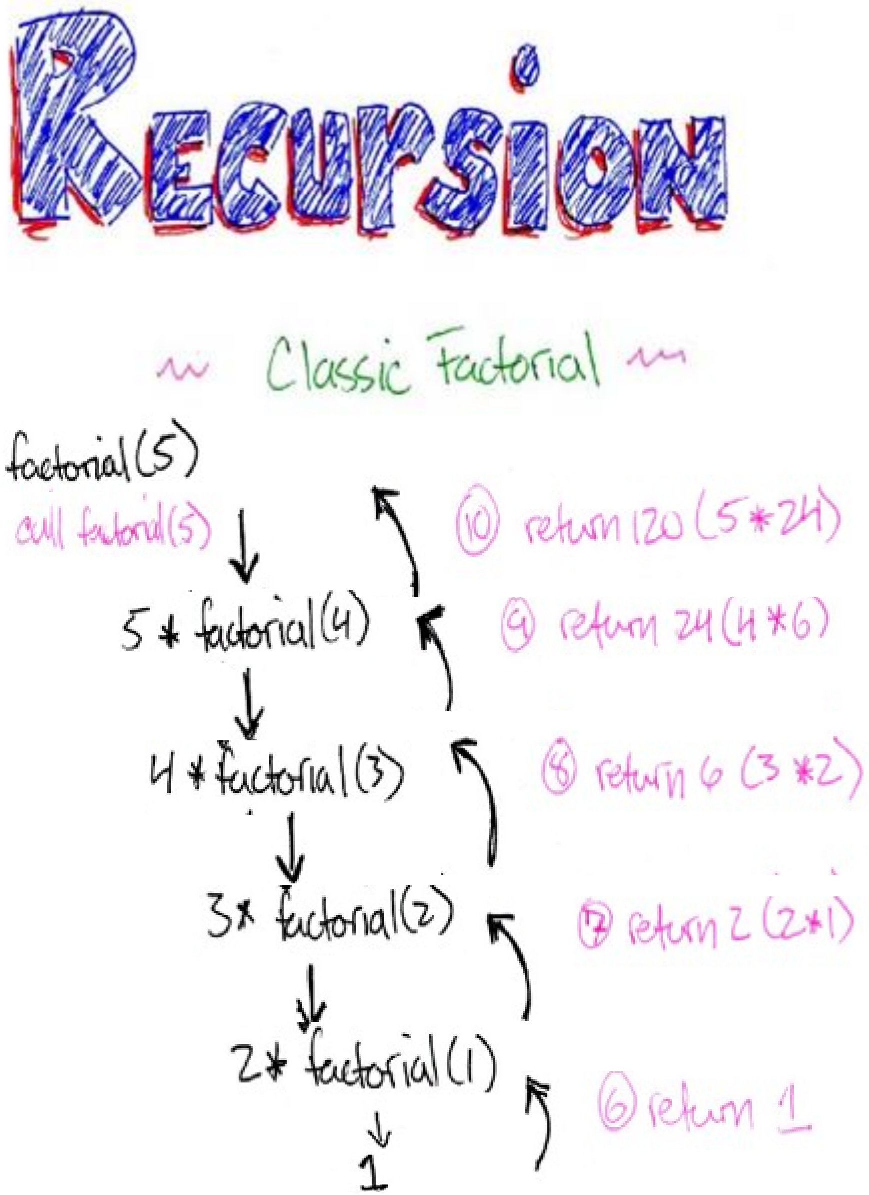
Recursive definition of factorial

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n \geq 1. \end{cases}$$



```python
def factorial(n):
    if n == 0 :
        return 1
    else:
        return n * factorial(n - 1)
```

How does Recursive function work in memory



Stack

~ Stack Overflows ~

results from too much data being pushed onto the stack. The memory /capacity of the stack is exceeded.

Why Recursion

- Used in many Data Structure
  - Array/String problems
  - Tree -> entirely depend on recursion
  - Graph -> DFS/BFS
  - Stack/Heap -> partially based on recursion
- Used in several high level Algorithm
  - DP
  - Backtracking
  - Divide and Conquer

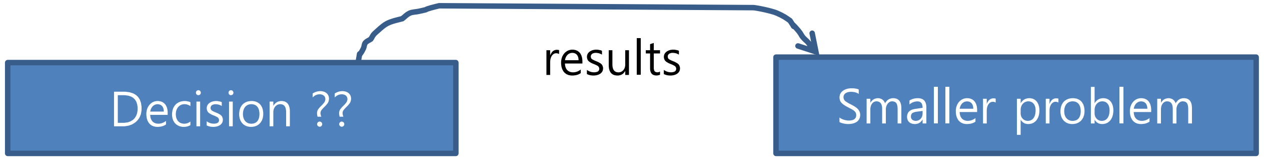
Note: If you are going in an coding interview without preparing recursion, you are doomed. Loops will not be enough.

Approach for recursion

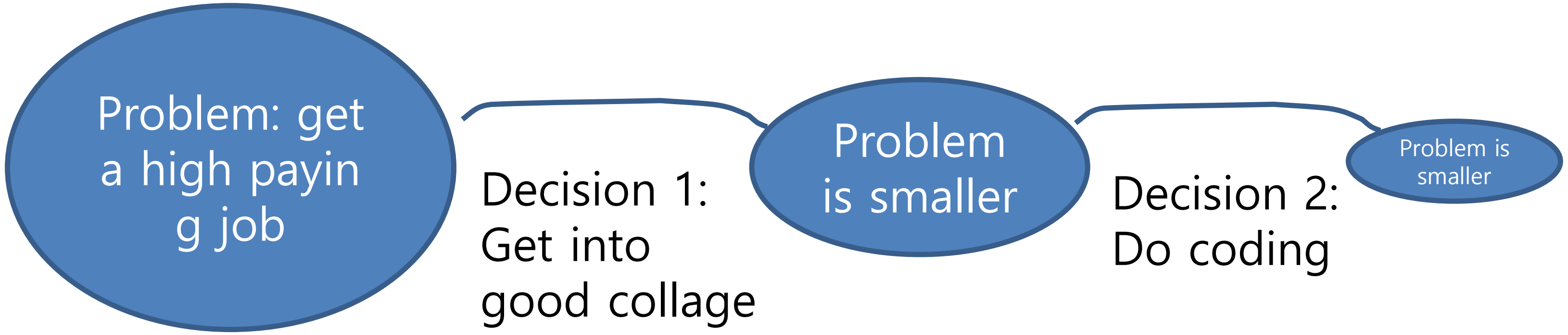
- General approach for recursion
  - Reduces the size of input
- **Better approach for recursion**
  - We take a decision, as a result the size of input is reduced

Input should be smaller in each function call

- How does the program become smaller?
  - We take certain decisions based on which the program becomes smaller.



  - Ex:


Problem: get a high payin g job
Decision 1: Get into good collage
Problem is smaller
Decision 2: Do coding
Problem is smaller

  - Because of these decisions the problem is becoming smaller
    So in recursion, we are not reducing the size of the input, its reducing itself based on the decision you take

Two ways for thinking about Recursion

First Approach

- Base condition -> Hypothesis -> Induction

Second Approach

- Choices -> Decision
  - This makes problems smaller
- Make Recursion Tree
  - Each branch of the tree will be your choices

There is no fixed rule, my explanation might just help you approach a problem easily

Example First Approach

Problem – Print 1 to n (use recursion) – Using **Base condition** **->** **Hypothesis** **->** **induction**

Hypothesis

Print (n) - > print 1 to n

Print(n) -> 1,2,3…… n

Print(n-1) -> 1,2,3…..n-1

Induction **-** What you do with output

```
Print(n)
{
        Print(n-1)
        print n
}
```

Base condition **-** its based on smallest valid input

```
Print(n)
{
   if ( n == 1 )
         print n
         return
   Print(n-1)
   print << n
}
```

Beauty of Hypothesis and Induction

- Lets fall in love with recursion
- Example : Print n to 1

Reducing size of input (assume function will work properly with reduced input)

1 to n

Print(n) -> 1 to n
Print(n-1) -> 1 to n-1

```
Print(n)
{
   if ( n == 1 )
            print n
            return
   Print(n-1)
   print << n
}
```

Reducing size of input (assume function will work properly with reduced input)

n to 1

Print(n) -> n to 1
Print(n-1) -> n-1 to 1

```
Print(n)
{
   if ( n == 1 )
            print n
            return
  print << n
  Print(n-1)
 }
```

Try doing Factorial Problem with this approach now !!

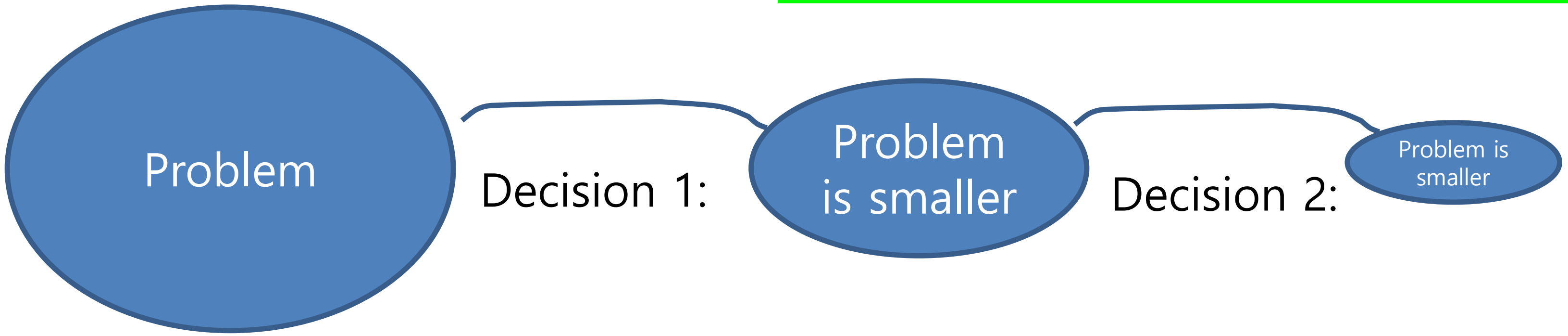
Two ways for thinking about Recursion

First Approach

- Base condition -> Hypothesis -> Induction

Second Approach

- Choices -> Decision
  - This makes problems smaller
- Make Recursion Tree
  - Each branch of the tree will be your choices



There is no fixed rule, my explanation might just help you approach a problem easily

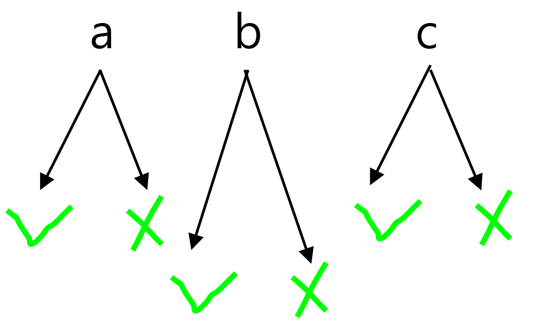
Identify – Choice + Decision

- Identify if it is a recursive problem ? How ?
  - You will be given choices, and based on those choices you will have to take decision
    - **Choice + Decision**
  - Recursive Tree -> This is most important part
    - If you have designed the recursive tree, then writing a code for it, will be a cakewalk
- Lets take an example
    - Subset Problem

      input -> “abc”

      output -> a ab
      b bc
      c ac
      abc
    - How will you know this is recursion problem.

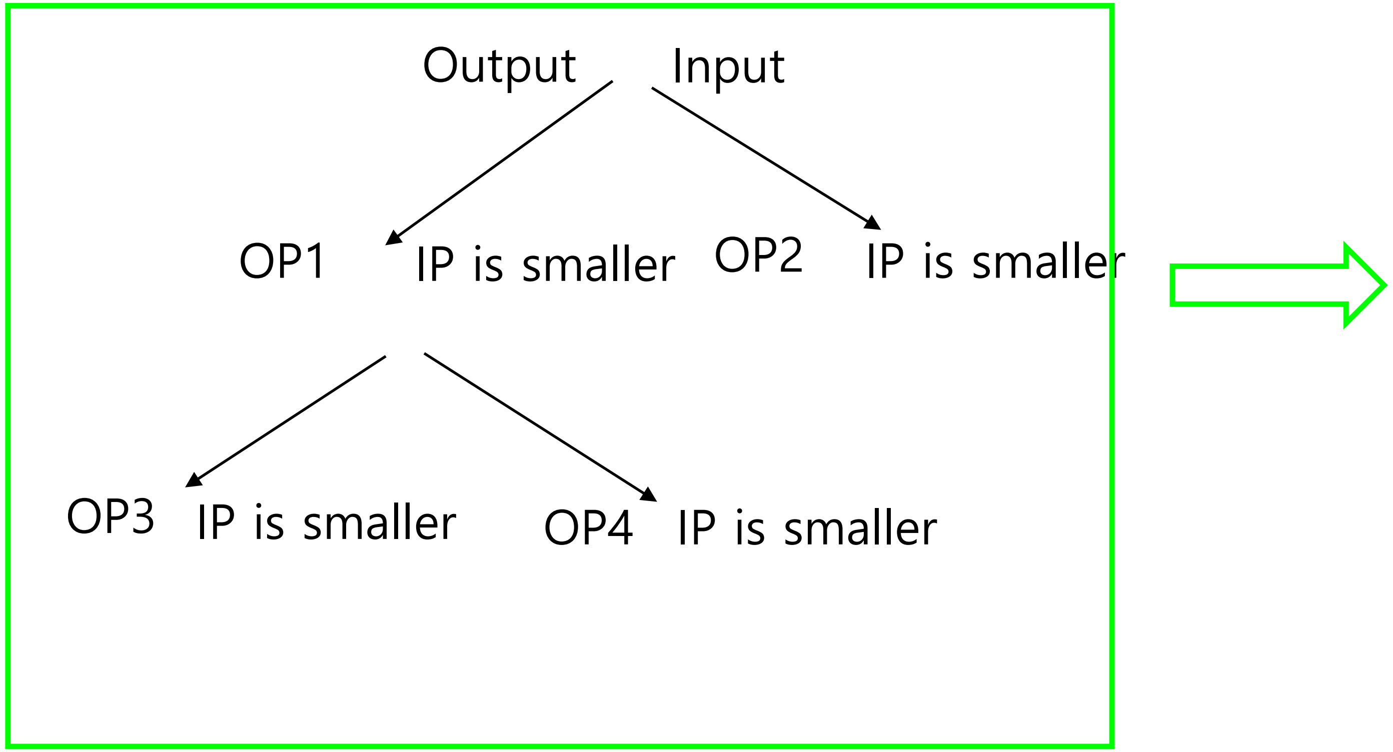
Choices + Decisions



We have **choice**, we can include or not include
Decision – include, or not include
As a result of your decision, your input will become smaller

Identify – Recursive Tree

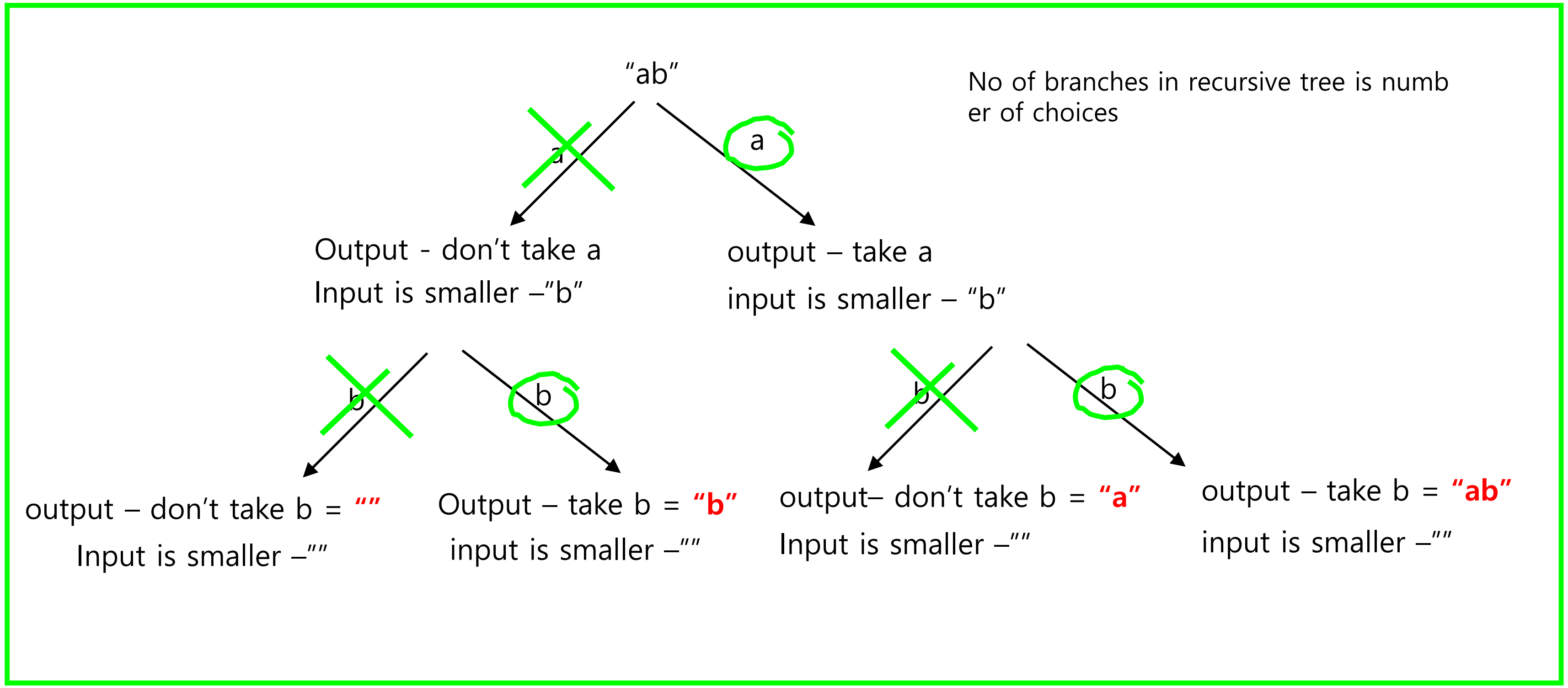
- **Recursive Tree** -> This is most important part
  - If you have designed the recursive tree, then writing a code for it, will be a cakewalk
  - How can your represent your decisions in the best way-> use recursive tree



Recursive Tree

Identify – Recursive Tree

- **Recursive Tree** -> This is most important part
  - If you have designed the recursive tree, then writing a code for it, will be a cakewalk
- How can your represent your decisions -> use recursive tree



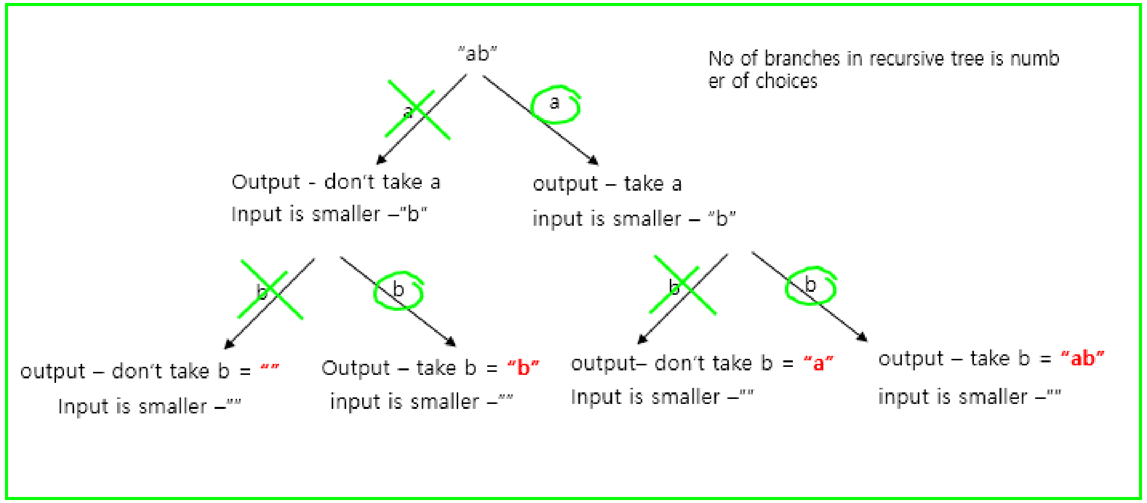
No of branches in recursive tree is number of choices

Code

```
def subset(string, out):
    if len(string)==0: # base case, smallest size of input
        print(out)
        return

    out1 = out+string[0]
    string = string[1:len(string)] #decreasing input size
    subset(string, out1) # recursive call when you take char in output
    subset(string, out) # recursive call when you dont take char in ouptut

# Driver Code
if __name__ == "__main__":
    str = "abcd"
    out = ""
    subset(str,out)
```



"ab"

No of branches in recursive tree is numb
er of choices

Output - don't take a
Input is smaller –"b"

output – take a
input is smaller – "b"

output – don't take b = ""
Input is smaller –""

Output – take b = "b"
input is smaller –""

output– don't take b = "a"
Input is smaller –""

output – take b = "ab"
input is smaller –""

Example

- Merge Sort

```
Alg.: MERGE-SORT(A, p, r)
        if p < r        smallest valid base case
            $q \leftarrow \lfloor (p + r)/2 \rfloor$
            MERGE-SORT(A, p, q)          } input made
            MERGE-SORT(A, q + 1, r)      } smaller
            MERGE(A, p, q, r)    - Output
```

- Quick Sort

```
Alg.:QUICKSORT(A, l, h)
        if ( l < h )     smallest valid base case
            j = PARTITION(A, l , h)   Output
            QUICKSORT(A, l, j)        } input made
            QUICKSORT(A, j+1, h)      } smaller
```

Some Problems based on Recursion

- Print 1 to n / n to 1
- Sort an Array / Sort element of Stack
- Delete the middle element in the stack
- Remove duplicate from string
- Count the number of occurrences in string

This problem could be solved with a l oop. But you must understand the fo undation of recursion, and hence you should practice these

- Subset
- Permutation with C spaces
  - Permutation with case change
  - Letter case permutation
- Generate a balanced parenthesis

Approach for recursion

- General approach for recursion
  - Reduces the size of input
- **Better approach for recursion**
  - We take a decision, as a result the size of input is reduced

Summary

- First approach
  - Base condition -> Hypothesis -> Induction
- Second approach
  - Take a Decision for choices -> as a result Input is smaller
  - Represent using Recursive Tree using, Think about the input/output of recursion
    - Take input
    - Make choices (no of choices are the branches of the recursive tree)
    - After making the choice you have taken the decision, as a result your input is smaller
    - Keep making these choices until your input is smallest valid input