introduton to recursion

Рекурсия - это концепция, которая подразумевает вызов функции самой себя.

Рекурсия позволяет разбить проблему на более мелкие подпроблемы и решить каждую подпроблему независимо.

parts of recursion

1. Base case
2. Recursive case

stack and recursion

В случае рекурсивного алгоритма каждый раз, когда функция вызывает саму себя, ее состояние заталкивается в стек, а когда функция возвращается, ее состояние выгружается из стека.

pros and cons

+	-
Чистые и элегантные решения	Накладные расходы
	Бесконечные циклы
	Сложность
	Отладка

complexity

master theorem
table
tree

log n

binary search
power

$a^n = (a^{n/2})^2 = a^{n/2} * a^{n/2}$
$a^n = a^{n/2} * a^{n/2} * a$

```
int Pow(int val, int n)
    if (n == 0)
        return 1
    int result = Pow(val, n / 2)
    result *= result
    if (n mod 2 != 0)
        return result * val
   return result
```

n

print array
effective factorial

```
int factorial(int n)
 if n == 0
   return 1
 else
   return n * factorial(n - 1)
```

```
int Fib(int n, int lhs = 1, int rhs = 1)
   if (n <= 1)
      return rhs
   return Fib(n - 1, rhs, lhs + rhs)
```

```
int SumAscending(n):
    if n == 0
        return 0
    else
        return n + SumAscending(n - 1)
```

```
int SumDescending(n, result=0):
    if n == 0
        return result
    else
        return SumDescending(n - 1, result + n)
```

n log n

merge sort

```
void MergeSort(arr, left = 0, right = arr.size)
   if (right - left <= 1)
      return
   mid = (left + right) / 2
   MergeSort(arr, left, mid)
   MergeSort(arr, mid, right)
   res = Merge(arr, left, mid, right)
   /* заменяем элементы исходного массива arr */
   /* с позиции left до right на отсортированные элементы из res */
   arr[left:right] = res
```

n ^ 2

print table
buble sort
lcs

```
void BubbleSort(array int arr, int n)
   if (arr.IsEmpty)
      return

   for (int i = 1; i < n; ++i)
      if arr[i - 1] > arr[i]
         Swap(arr[i - 1], arr[i])

   return BubbleSort(arr, n - 1)
```

```
void PrintTable(int n)
   if n == 1
      return

   for (int i = 1; i <= n; ++i)
      Print(i * n)
   Print()

   return PrintTable(n - 1)
```

2 ^ n

naive fibonacci

```
int Fib(int n)
  if (n >= 1)
    return 1
  return Fib(n - 1) + Fib(n - 2)
```

tail recursion

Хвостовая рекурсия - это особый тип рекурсии, при котором последней операцией, выполняемой функцией перед возвратом, является вызов самой себя

backtracing

Бэктрекинг - это общая техника, которая рассматривает поиск всех возможных комбинаций, постепенное построение комбинации в решения и отказ от комбинации, как только будет установлено, что она не является решением.

A. Генератор скобок

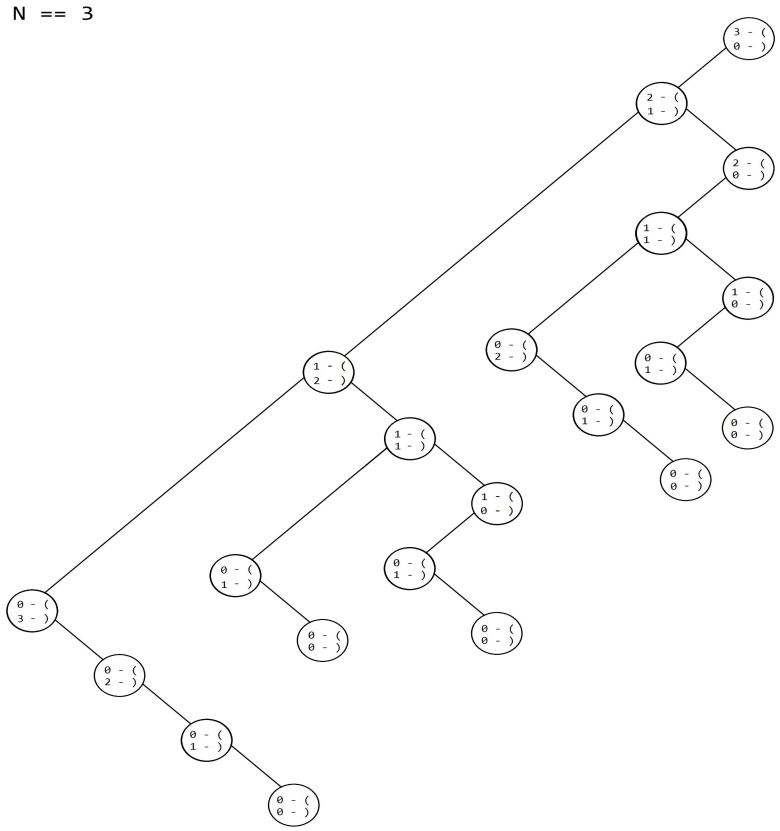
Сгенерировать все правильные скобочные последовательности заданной длины

```
void Helper(array <string> result, string current, int open, int close)
    if (open == 0 and close == 0)
        result.add(current)
        return

    if (open > 0)
        Helper(result, current + (, open - 1, close + 1)

    if (close > 0)
        Helper(result, current+ ) , open, close - 1)
```

N == 3



B. Комбинации

Сгенерировать все возможные комбинации по
введенным числам

```
array <string> alphabet = [ "", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz" ]

void Backtrack(string digits, array <string> alphabet, array <string> result, string buffer)
    if (buffer.size == digits.size)
        result.add(buffer)
        return

    for (char letter in alphabet[digits[buffer.size] - '0'])
        buffer.add(letter)
        Backtrack(digits, alphabet, result, buffer)
        buffer.pop_back
```

practical tasks

dynamic programming
trees and graphs
filesystem
backtracking

traverse tree

```
void TraverseTree(node)
    if node is Null
        return

    Print(node.value)
    TraverseTree(node.left)
    TraverseTree(node.right)
```