# RECURSION.

Raj
DATE / /
PAGE

Q Count Good no. : An digit string is good if the
even index : Even no.
odd index : Odd no.
(zero may be at start)

Input: n = length of string
Output: good no. of length 'n'. $(0 < n <= 10^{15})$

* return in modulo $10^9+7$

```
long long helper(ll base, expo, mod)
{
    ll ans = 1;
    while (expo > 0)
    { if (expo%2 == 0) base = (base^2)%expo
      expo = expo/2;

      else { ans = (ans * base)%mod
             expo = expo - 1;
    }
    return ans

int countGoodNo. (long long n)
{
    ll even = (n+1)/2;  odd = n/2
    mood = 10^9 + 7
return (int) helper(5, even, mod) * helper(4, odd, mod);
}
```

10^9 = 10^9

Converted to int

0 1 2 3 4

5 x 4 x 5 x 4 x 5

$= 5^3 \times 4^2$

↓

Pow(5,3) * Pow(4,2)

Generalized

Pow(5, no of even index) * Pow(4, odd index)

Raj
DATE / /
PAGE

## Q Pow function (Fast calc.)

$\{x^y = (x^2)^{y/2}\}$

```
double fPow(double x, int n) {
    if (n==0) return 1
```

```
long N=n;
if (N%2==0)  return Pow(x*x, N/2);
else return x*(x, N-1);
```

Pow(x*x, N/2); → divide in half par

x*(x, N-1); → Spit 1 Baar

---

## Q Generate Paranthesis.

Given n pairs of paranthesis, generate all well formed combination of paranthesis

eg: n = 3

output { "((()))", "(()())", "(())()", "()(())", "()()()" }

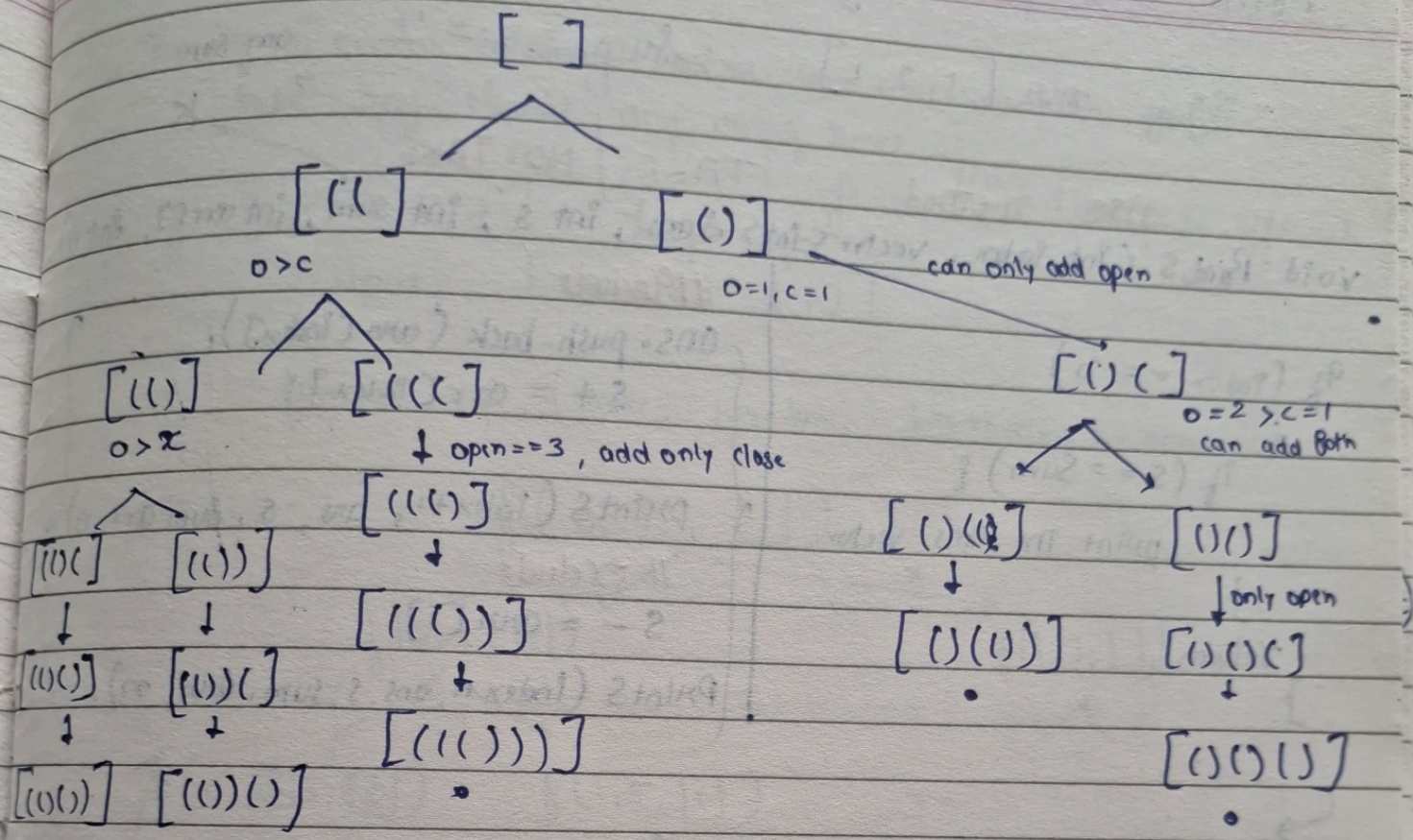
- Can never start with closing para. {close < open}

( )  open = 1, close = 1

- We have open limit of n, can't go more
- open + close <= 2n

((()(
↓
((()(
↓
((()(

for n = 3




```
void para(int n, close, open, vector<string>&v, strings)
{
    if (open<n) para(n, close, open+1, v, s+'('); // add open bracket
    if(close<open) para(n, close+1, open, v, s+')'); // add close bracket
    if (open + close = 2*n){
        v.push_back(s);
        return;
    }
}
vector<string> getparanthesis (int n)
{ Vector<string>ans = para(n, 0, 0, ans, "");
}
```

1) Put open
↓
(open < n)

2) Put Close
↓
(close < open)

got a completed answer

(unique elements)

Raj

DATE / /
PAGE

# Print Subsequence of Sum 'K'

eg: arr [1, 2, 1] Sum = K = 2

arr sum → s
K

Use method of {TAKE / NOT TAKE}

```
void PrintS (int index, vector<int> &ans, int s, int sum, int arr[], int n)
{
    if (index == n)
    {
        if (s == sum) {
            print the ans vector
        }
        return;
    }
    #include
    ans.push_back (arr[index]);
    s += arr[index];
    printS (index+1, ans, s, sum, arr, n);
    // exclude.
    s -= arr[i]
    PrintS (index+1, ans, s, sum, arr, n);
```

Q Modify Q. : Print any One Subseq. of Sum K.

- Make PrintS as a bool funct^n

- At every recursion call, check if condition true then return true
- If False: then only do recursive call.

```
ans.push_back(arr[index]); s+=arr[index];
if (PrintS(index+1, ans, s, sum, arr, n) == true) return true;
s -= arr[i]
ans.pop_back();
if (PrintS (.....) == true) return true
return false
```

X (Not

# Q Combinational Sum

Raj
DATE / /
PAGE

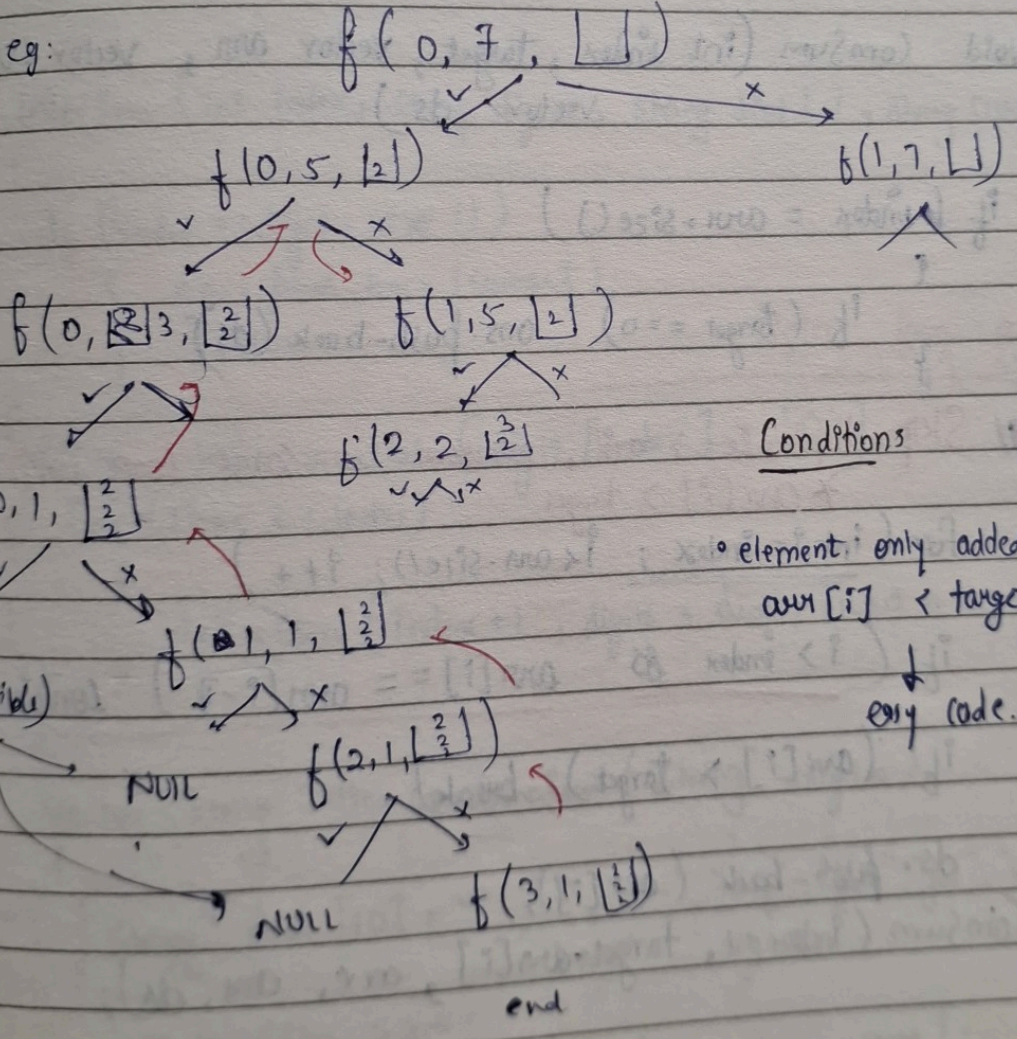
Sum of all element in array whose sum = k, where you can take any element any no. of times → Pick / Not pick approach

eg: [2, 3, 6, 7] target = 7

output: [[2, 2, 3], [7]]

function f (index, req. sum, data structure for answer)

eg:




end

Conditions

- element only added when arr[i] < target

easy code.

Raj
DATE / /
PAGE

# Comb. Sum II

- There will be duplicate elements in array
- each element in given arr be okay & used once
- final ans should not contain duplicates

- Sort given array
- Use a loop to skip duplicates

eg: [10, 1, 2, 7, 6, 1, 5] target = 8

output: [[1,1,6], [1,2,5], [1,7], [2,6]]

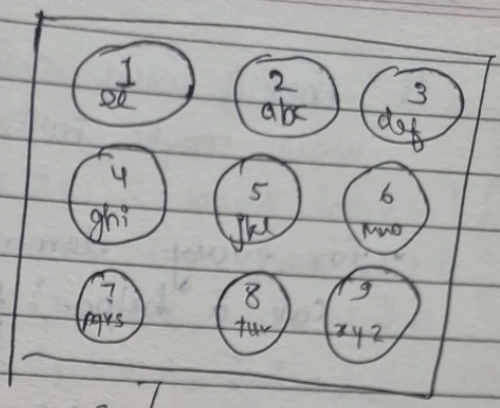
```
void ComSum (int index, target, vector arr, vector<vector ans
                                    vector ds)
{ if (index = arr.size())
    {
      if (target == 0) ans.push_back(ds);
    }
// Skip duplicate, add, remove in some loop until
      * arr[i] > target
  for (int i= index ; i< arr.size(); i++)
  {
    if ( i > index && arr[i] == arr[i-1] ) continue
    if (arr[i] > target) break;
    ds.push_back (arr[i]);
  ComSum (index+1, target-arr[i], arr, ans, ds);
}
```

# Letter Combination

Raj
DATE / /
PAGE

String of digit (2-9) contain letters

return all possible letter combination that the number could represent




eg: 2 3 = Digit (String)

[ad, ae, af, bd, be, bf, cd, ce, cf]

```
void letterCom (int index, string digits, string dial[], string curr, vector of string ans)
{
    if (index == digit.size())
    {
        ans.push_back(current);
        return;
    }
    String letter = dial[digit[index] - '0'];
    for (char c : letter)
    {
        letterCom(index+1, digits, dial, curr+c, ans);
    }
}
```

Select from each char of String of that no

```
vector<String letterCombination(string digits)
{
    String dial[10] = "", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "xyz";
    vector<string> ans
    letterCom(0, digits, dial, "", ans);
    return ans
}
```

map string to number

# Word Search in matrix

Raj
DATE / /
PAGE

Q. Check if word exist in matrix or not eg word: SEE
Word maybe horizontal or vertical

| | | | |
|---|---|---|---|
| A | B | C | E |
| S | F | C | B |
| X | Y | Z | S |
| A | D | E | E |
| H | E | L | L |

- for every element in matrix call a "helper" function
- Recursive call for up|down|right|left

(check all boundings at each call.

```
bool dfs (vector<vector<char>>& board, int r, c, index, string word)
{
  if(r<0 || c<0 || r> board.size() || c. board[0].size() ||
        board[r][c] != word[index])
  {
    return false;
  }
  if (word.size()-1 == index)
  {
    return true;
  }
```

false → out of bounds →
→ char not matching

full match

```
// Mark current cell at '#' then go on
char temp = board[r][c]
board[r][c] = '#'
```

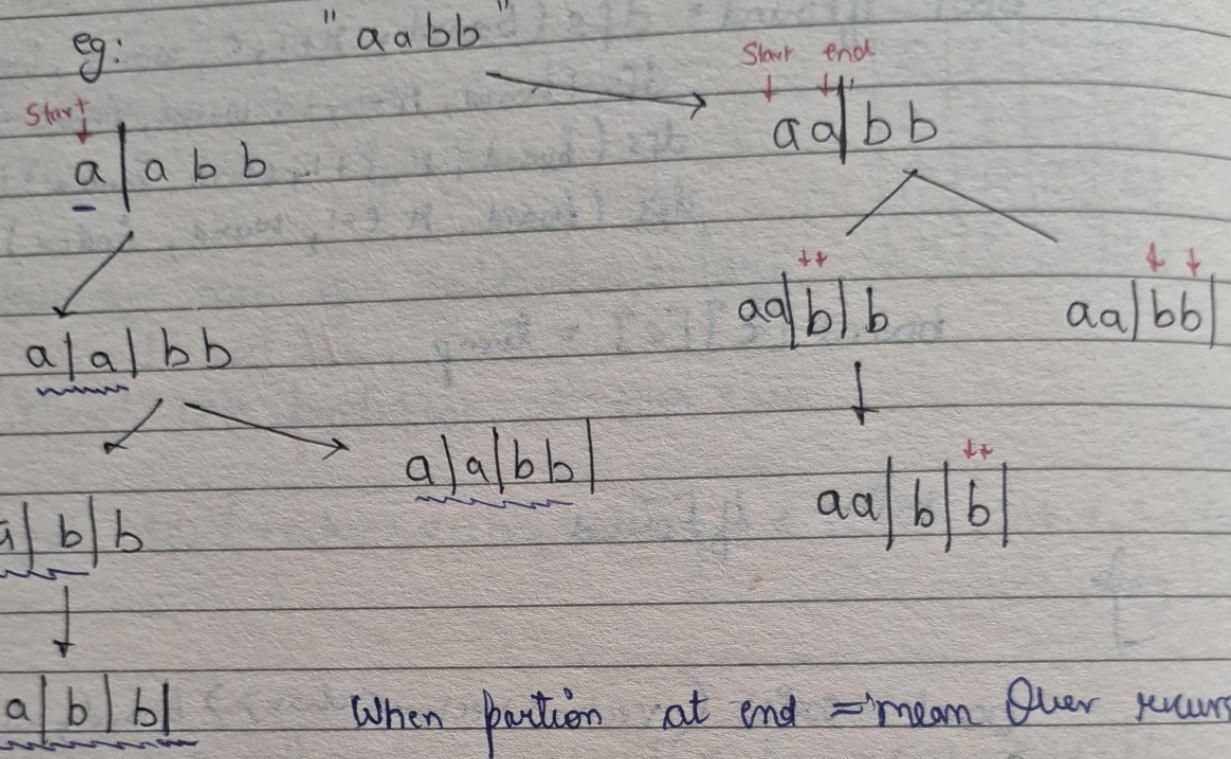
Raj
DATE / /
PAGE

```
// Explore all four direction

bool ifFound = dfs(board, r+1, c, word, index+1) ||
               dfs(board, r-1, c, word, index+1) ||
               dfs(board, r, c+1, word, index+1) ||
               dfs(board, r, c-1, word, index);

board[r][c] = temp; // back track and
                    change that char to original

return ifFound;
}

bool exist(vector<vector<char>> board, string word)
{
    int rows = board.size();
    int cols = board[0].size();

    for(int i=0; i<rows; i++)
    {
        for(int j=0; j<cols; j++)
        {
            if(dfs(board, r, c, word, 0))
            {
                return true
            }
        }
    }
}
```

# Palindrome Partition

Raj
DATE / /
PAGE

Q Partition a string such that every substr is palindrome




When partition at end ⇒ mean Over recursion

1) Select a part for Partition, check if it is palindrome

2) If Yes, then only add it and explore further by recursion

```
vector<vector<string>> Partition (string s)
{
    define vector and
    vec of vector
    backtrack(0, s, path, result);
    return result;
```

Raj
DATE / /
PAGE

```
// function to check Palindrome.
bool isPalindrome(const string& s, int left, int right)
{
    while (left < right) {
        if (s[left] != s[right]) return false;
        left++; right--;
    }
    return true;
}

// Backtracking function
void backtrack(int start; const string& s, vector<string>& path, vec<vec<str>> result)
{
    if (start == s.size())
    { result.push_back(path);
    } return;
```

Iterate until a palindrome is found, then only explore by recursion

```
for (int end = start; end < s.size(); ++end)
{
    if (isPalindrome(s, start, end);
    {
        path.push_back(s.substr(start, end-start+1))   // start → index, end-start+1 → length
        backtrack(end+1, s, path, result);
        path.pop();
    }
}
}
```

Choose
explore
Un-choose (bcl)