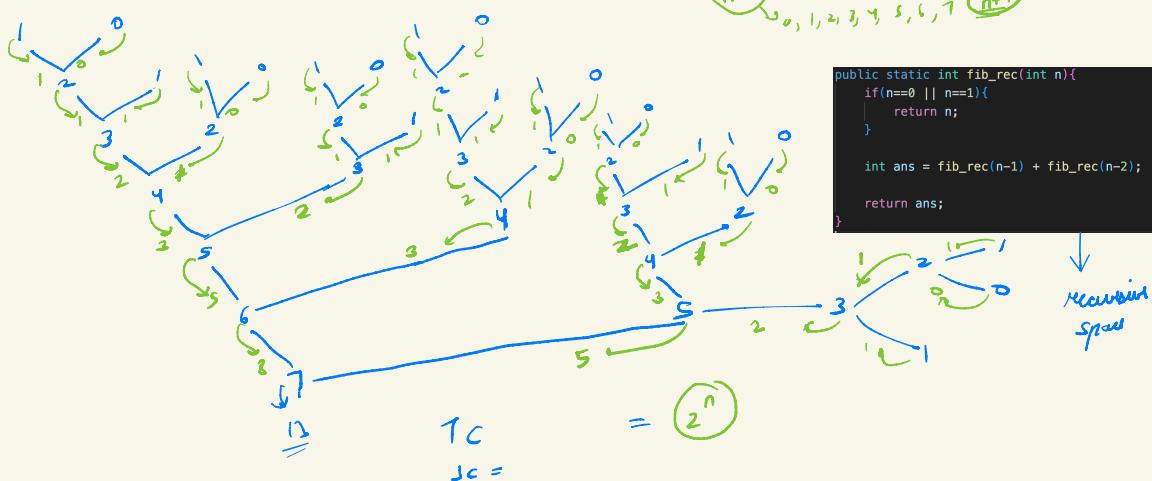
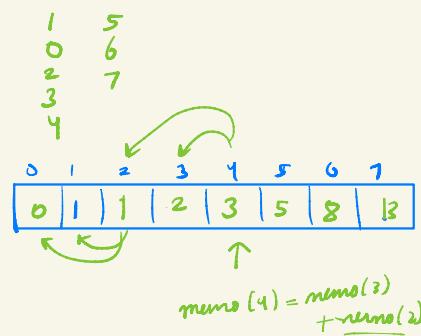
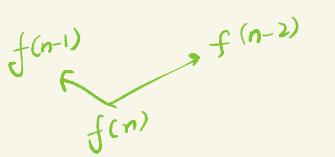



Get the n^{th} term
 $\Rightarrow 0, 1, 1, \textcircled{2}, 3, 5, 8, 13, 21, 34, 55, 89, \dots$
 $2 \rightarrow 1, 3 \rightarrow 2$ memorization



```
public static int fib_rec(int n){
    if(n==0 || n==1){
        return n;
    }
    int ans = fib_rec(n-1) + fib_rec(n-2);
    return ans;
}
```

overlapping subproblems



$$TC = O(n) \\ SC = O(n) + \text{rec. space}$$



```
public static int fib_memo(int n, int[] memo){
    if(n==0 || n==1){
        return memo[n]=n;
    }
    if(memo[n]!=0) return memo[n];
    int ans = fib_memo(n-1,memo) + fib_memo(n-2,memo);
    return memo[n]=ans;
}
```

- 1) Find the flow, write the for loop
- 2) Replace return with continue;
- 3) Remove memorization line {why?}
- 4) Replace calls with df.

```

public static int tab(int N, int[] memo){
    for(int n=0; n<=N; n++){
        if(n==0 || n==1){
            memo[n]=n;
            continue;
        }

        // if(memo[n]!=0) return memo[n];

        int ans = memo[n-1] + memo[n-2]; //fib_memo(n-1,memo) + fib_memo(n-2,memo)

        memo[n]=ans;
    }

    return memo[N];
}

```

$$n=0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12$$

$$Tc = O(n)$$

$$Sc = O(1)$$

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|----|----|----|----|----|-----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 | 144 |

```

public static int tabulation(int n){
    int[] dp = new int[n+1];

    dp[0] = 0;
    dp[1] = 1;

    for(int i=2; i<=n; i++){
        dp[i] = dp[i-1] + dp[i-2];
    }

    return dp[n];
}

```

Recursion (↴)

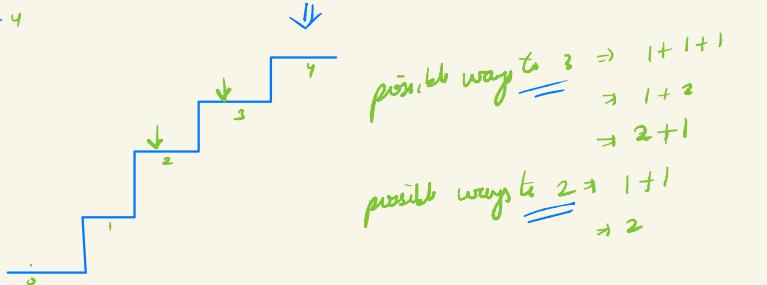


Memorization () add two lines

↓
Tabulation ()

$n=4$

0, 1, 1, 2, 3, 5, 8



$$\text{ways}(n) = \text{ways}(s) + \text{ways}(w)$$

↓ ↓

1 2

$$f(n) = f(n-1) + f(n-2)$$

↳ fibonacci

possible ways to 4 \Rightarrow

1+1+1+1
1+2+1
2+1+1

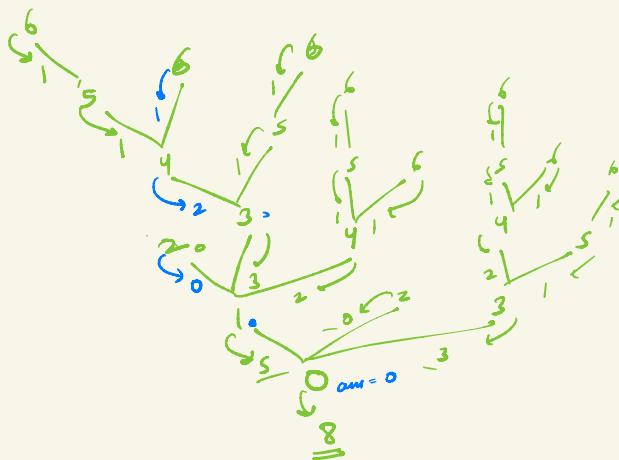
1+1+2
2+2

| | | | | | |
|---|---|---|---|---|---|
| 3 | 3 | 0 | 2 | 2 | 3 |
| 0 | 1 | 2 | 3 | 4 | 5 |

6

min(ans(i))ⁿ

memo²



idm = 5
jump = 12
 $\frac{5+2}{=}$

```
public int countWays(int idx, int[] arr){  

    if(idx == arr.length){  

        return 1;  

    }  
  

    int ans = 0;  

    for(int jump=1; jump<=arr[idx]; jump++){  

        if(idx + jump <= arr.length)  

            ans += countWays(idx+jump, arr);  

    }  
  

    return ans;  
}
```

| | | | | | |
|---|---|---|---|---|---|
| 3 | 3 | 0 | 2 | 2 | 3 |
| 0 | 1 | 2 | 3 | 4 | 5 |

| | |
|---|---|
| 2 | 3 |
| 6 | 1 |
| 5 | 0 |
| 4 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 8 | 5 | 0 | 3 | 2 | 1 | 1 |



```
public static int countWays_memo(int idx, int[] arr, int[] memo){
    if(idx == arr.length){
        return memo[idx]=1;
    }

    if(memo[idx]!=0) return memo[idx];

    int ans = 0;
    for(int jump=1; jump<=arr[idx]; jump++){
        if(idx + jump <= arr.length)
            ans += countWays_memo(idx+jump, arr,memo);
    }

    memo[idx] = ans;
}
```

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |
| 3 | 3 | 0 | 2 | 2 | 3 |

dp(s)

memo(4)
5
6

memo(s)

number of ways
to reach n from s

break till
10:45

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 8 | 5 | 0 | 3 | 2 | 1 | 1 |

idx=6 5+3=8
jump=3

ans=5+0+3=8

```
public int countWays_tabulation(int idx, int[] arr, int[] memo){
    for(int id=arr.length; id>=0; id--){
        if(id == arr.length){
            memo[id]=1;
            continue;
        }

        int ans = 0;
        for(int jump=1; jump<=arr[id]; jump++){
            if(id + jump <= arr.length)
                ans += memo[id+jump]; //countWays_memo(idx+jump, arr,memo);
        }

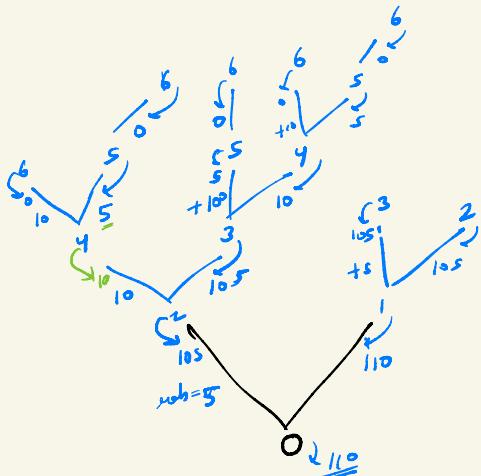
        memo[id] = ans;
    }

    return memo[0];
}
```

| | | | | | |
|---|---|----|-----|----|---|
| 0 | 1 | 2 | 3 | 4 | 5 |
| 5 | 5 | 10 | 100 | 10 | 5 |

↑

105



```
public static int findMax(int idx, int[] arr){
    if(idx == arr.length){
        return 0;
    }

    int rob=arr[idx];
    int notRob=0;

    if(idx+2<=arr.length)
        rob += findMax(idx+2, arr);

    notRob = findMax(idx+1, arr);

    int ans = Math.max(rob,notRob);
    return ans;
}
```

ans 0 1 2 3 4 5
↓ ↓ ↓
5 5 10 100 10 5
↑ ↑ ↑

day sum → 9
why
rec mem ()
tab ()

| | | | | | | |
|-----|-----|-----|-----|----|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 110 | 110 | 105 | 105 | 10 | 5 | 0 |

memo → ↓
dp(i)
maximum choice
if at start from i th
idm
dp(i)

$$m = 5 + 105$$

$$nr \geq 110$$

```
public int findMax_tabulation(int idx, int[] arr, int[] memo){
    for(idx=arr.length; idx>=0; idx--){
        if(idx == arr.length){
            memo[idx]=0;
            continue;
        }

        int rob=arr[idx];
        int notRob=0;

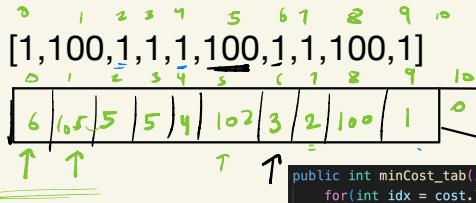
        if(idx+2<=arr.length)
            rob += memo[idx+2]; //findMax_memo(idx+2, arr,memo);

        notRob = memo[idx+1]; //findMax_memo(idx+1, arr,memo);

        int ans = Math.max(rob,notRob);
        memo[idx]=ans;
    }

    return memo[0];
}
```

$dp(idm)$
 \downarrow
 min cost from
 idm to end



(10) idm
 $dp = 1$
 $ts = 0$

```
public int minCost_tab_mostOptimized(int[] cost){  

    int n = cost.length;  

    int idxp1 = cost[n-1];  

    int idxp2 = 0;  

    int ansIdx = cost[0];  

    for(int idx = n-2; idx>=0; idx--){  

        int ansIdx = cost[idx] + Math.min(idxp1, idxp2);  

        idxp2 = idxp1;  

        idxp1 = ansIdx;  

    }  

    return Math.min(idxp1, idxp2);  

}
```

```
public int minCost_tab(int[] cost, int[] dp){  

    for(int idx = cost.length; idx>=0; idx--){  

        if(idx == cost.length){  

            dp[idx]=0;  

            continue;  

        }  

        int oneStep = dp[idx+1]; //minCost_memo(cost,idm+1,dp);  

        int twoStep = Integer.MAX_VALUE;  

        if(idx + 2 <= cost.length){  

            twoStep = dp[idx+2]; //minCost_memo(cost,idm+2,dp);  

        }  

        int ans = cost[idx] + Math.min(oneStep,twoStep);  

        dp[idx]=ans;  

    }  

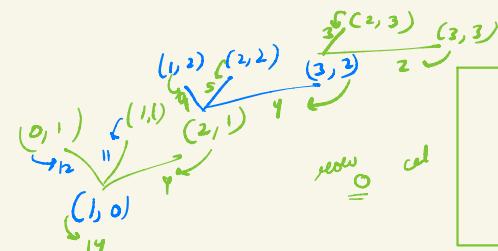
    return Math.min(dp[0],dp[1]);  

}
```

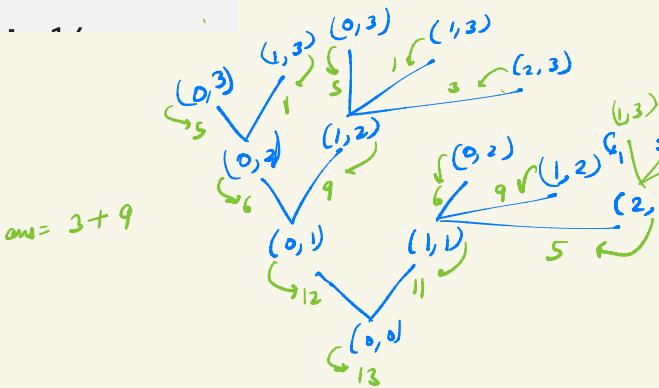
0 {{1, 3, 1, 5},
 1 {2, 2, 4, 1},
 2 {5, 0, 2, 3},
 3 {0, 6, 1, 2}};

.. ..

$0 \leq i \leq n$
 $0 \leq j \leq m$
 $n \times m$



row 0 col



ans = 3 + 9

```
public static int getMax_rec(int[][] mine, int i, int j){  

    if(j==mine[0].length-1){  

        return mine[i][j];  

    }  

    int up = 0;  

    int same = 0;  

    int down = 0;  

    if(i-1 >= 0){  

        up = getMax_rec(mine, i-1, j+1);  

    }  

    same = getMax_rec(mine, i, j+1);  

    if(i+1<mine.length){  

        down = getMax_rec(mine, i+1, j+1);  

    }  

    int ans = mine[i][j] + Math.max(up,Math.max(same,down));  

    return ans;  

}
```

```

{{1, 3, 1, 5},
{2, 2, 4, 1},
{5, 0, 2, 3},
{0, 6, 1, 2}};

```

| | 0 | 1 | 2 | 3 |
|---|----|----|---|---|
| 0 | 13 | 12 | 6 | 5 |
| 1 | 14 | 11 | 4 | 1 |
| 2 | 16 | 9 | 5 | 2 |
| 3 | 17 | 11 | 4 | 2 |

```

public static int goldmine_tab(int[][] mine, int[][] dp){
    for(int j = mine[0].length-1; j>=0; j--){
        for(int i=0; i<mine.length; i++){
            if(j==mine[0].length-1){
                dp[i][j] = mine[i][j];
                continue;
            }

            int up = 0;
            int same = 0;
            int down = 0;

            if(i-1 >= 0){
                up = dp[i-1][j+1]; //getMax_memo(mine, i-1, j+1,dp);
            }

            same = dp[i][j+1];//getMax_memo(mine, i, j+1,dp);

            if(i+1<mine.length){
                down = dp[i+1][j+1]; //getMax_memo(mine, i+1, j+1,dp);
            }

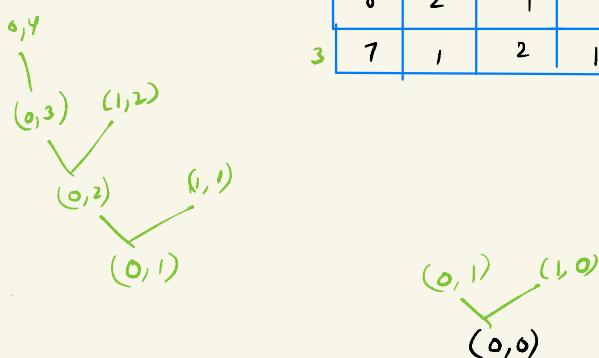
            int ans = mine[i][j] + Math.max(up,Math.max(same,down));
            dp[i][j] = ans;
        }
    }

    int ans = 0;
    for(int i=0; i<mine.length; i++){
        ans = Math.max(ans,dp[i][0]);
    }

    return ans;
}

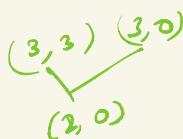
```

| | 0 | 1 | 2 | 3 |
|---|---|---|----|---|
| 0 | 4 | 9 | 11 | 1 |
| 1 | 2 | 1 | 9 | 5 |
| 2 | 8 | 2 | 4 | 3 |
| 3 | 7 | 1 | 2 | 1 |

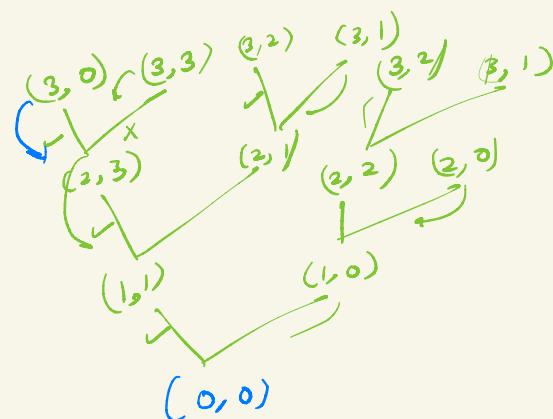


content
decreasing

{1, 2, 3}



$$\text{ans} = 0 + 3 + 2 + 1 + 2 + 1 + 7 + 0$$

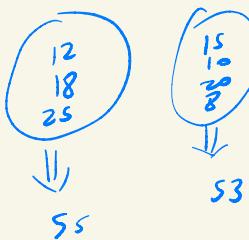


```
int ans = 0;
public void getAllXor(int[] nums, int idx, int cxor){
    if(idx==nums.length){
        ans += cxor;
        return;
    }

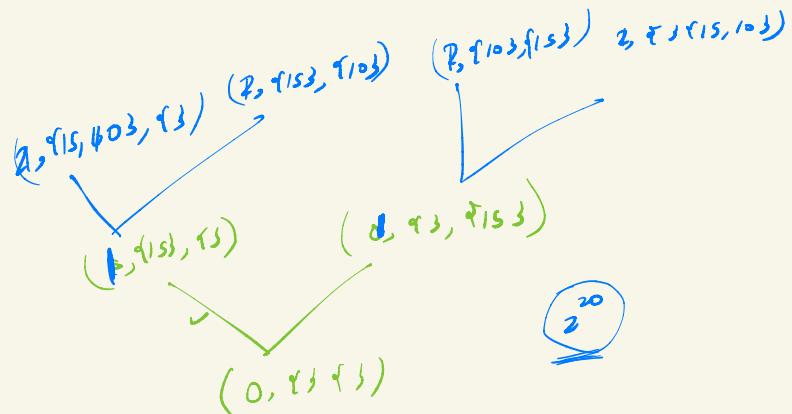
    getAllXor(nums, idx+1, cxor^nums[idx]);
    getAllXor(nums, idx+1, cxor);
}
```

$$\begin{array}{r} 01 \\ 3 \\ \hline 10 \\ \hline 11 \end{array} \quad \begin{array}{r} 3 \\ 11 \\ \hline 00 \\ \hline 11 \\ \hline 10 \end{array} \quad \begin{array}{r} 10 \\ 11 \\ \hline 01 \\ \hline 10 \end{array}$$

7
15 = 10 20 8 12 18 25 ↑
0

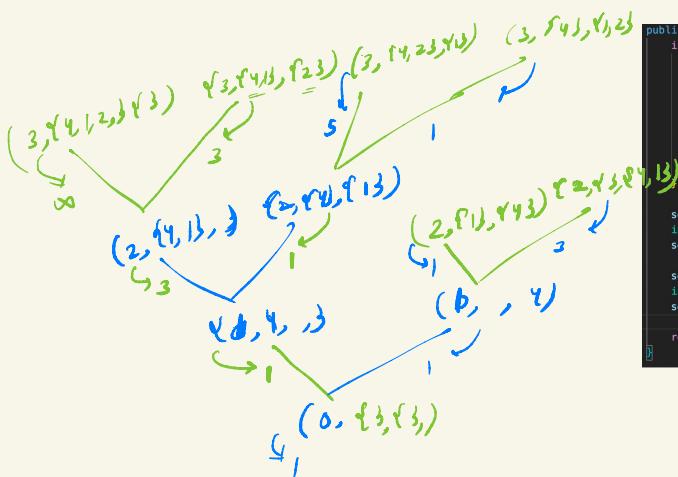


②



$\{4, 1, 2\}$

$\{4\} \{1, 2\}$



```

public static int minAbsoluteDiff(int[] arr, int idx, ArrayList<Integer> set1, ArrayList<Integer> set2) {
    if(idx==arr.length){
        int n = arr.length;
        if(n==0 && set1.size()==set2.size()){
            return Math.abs(set1.size() - set2.size());
        } else if(n==1 && Math.abs(set1.size()-set2.size())==1){
            return Math.abs(set1.size() - set2.size());
        }
        return Integer.MAX_VALUE;
    }

    set1.add(arr[idx]);
    int ans1 = minAbsoluteDiff(arr, idx + 1, set1, set2, set1+arr[idx], set2);
    set1.remove(set1.size()-1);

    set2.add(arr[idx]);
    int ans2 = minAbsoluteDiff(arr, idx + 1, set1, set2, set1, set2+arr[idx]);
    set2.remove(set2.size()-1);

    return Math.min(ans1,ans2);
}

```

1 → Valid
0 → Invalid



| | | | | |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 |

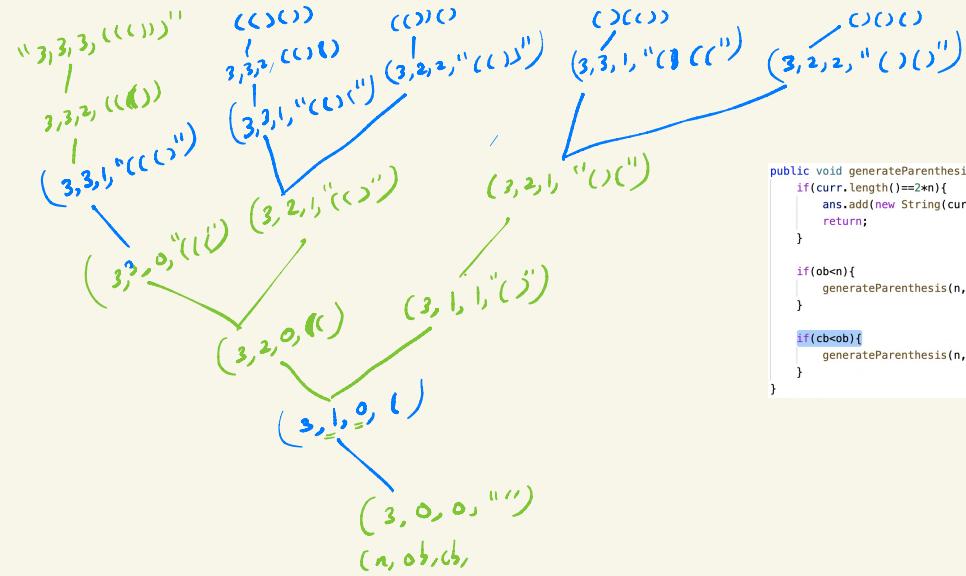
$ab = \infty \cdot 0$
 $y(ab < 0)$
return false

$N=3$
X
 2^m

"(())C"
") (()"
" (())"
" (() "
" () (() "
 $ob++$
 $ob-$
 $ob = \beta + \alpha + z - 1$

"()) C"
OCJ

C ()
() ()
ob = 19 / 772 = 0
ob = 19 / 772 = 0
 $\frac{1}{P}$



```

public void generateParenthesis(int n, ArrayList<String> ans,int ob,
{
    if(curr.length()==2*n){
        ans.add(new String(curr));
        return;
    }

    if(ob<n){
        generateParenthesis(n,ans,ob+1,cb,curr+"(");
    }

    if(cb<ob){
        generateParenthesis(n,ans,ob,cb+1,curr+ ")");
    }
}

```