DP with Bitmasking

- Gaurish Baliga

Representing Subsets - Normal Way

How to represent all the subsets of an array?

Arr = [10, 20, 30]

1 2 3 4 5 6 7 8 O(n)

Subsets: {}, {10}, {20}, {30}, {10, 20}, {20, 30}, {10, 30}, {10, 20, 30} 2^n

Space: $O(2^n * N)$

Bit inefficient

Operations on Subsets

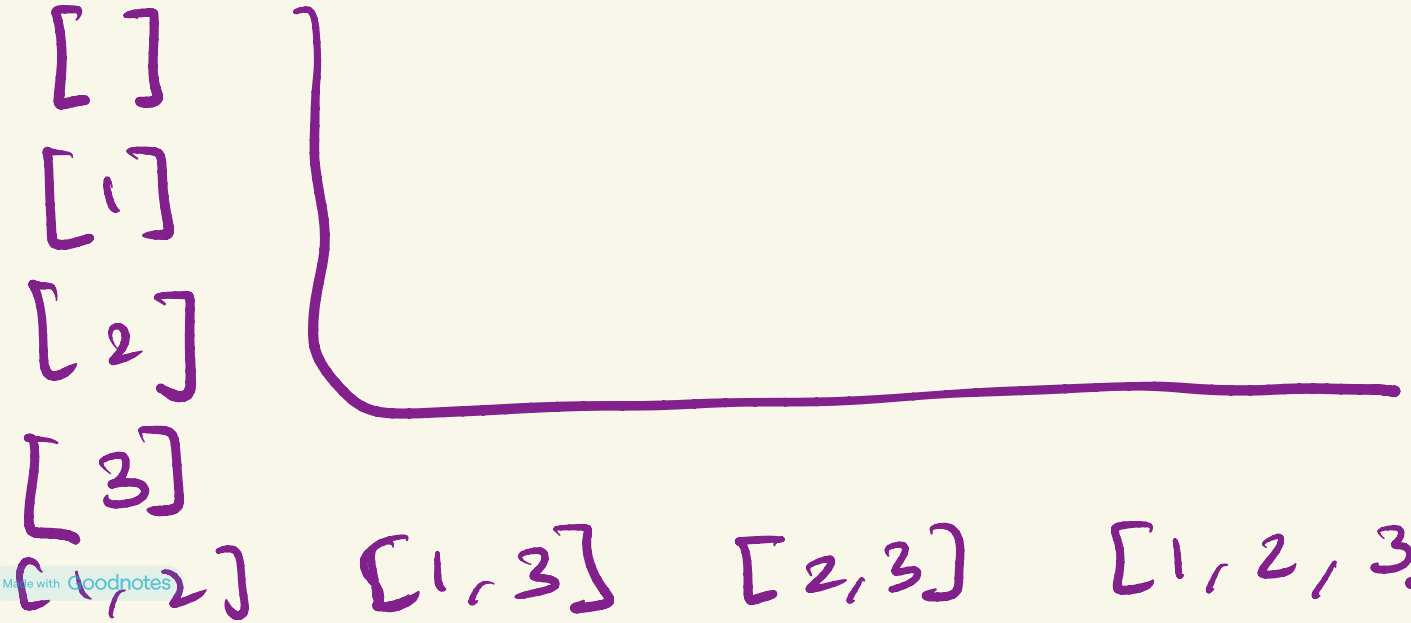
Given a Subset = {30, 40, 60} of an Arr = [10, 20, 30, 40, 50, 60], consider the following operations on it.

- Insert a arr[i] in the subset
- Search if arr[i] is in the subset
- Delete arr[i] from the subset if it exists

Now think about the different data structures we can use to perform all the above operations efficiently.

? - ith subset → jth operation

[1, 2, 3]



[1, 2, 3]

[]

(1)

(2)

(3)

[1, 2]

[1, 3]

[2, 3]

[7, 2, 3]

1 put 2

1 check 2

1 delete 2

4 put 2

4 check 7

set <int>s [2^n];

↑ subset

→ $O(\log_2 n)$ inset

→ $O(\log_2 n)$ delete

→ $O(\log_2 n)$ check

s[i]

↳ ith subset

Tradeoffs in Data Structures

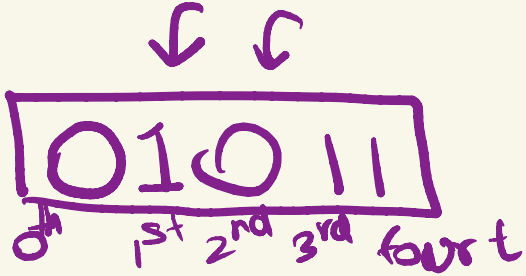
Operation/DS	Vector	Set
Search	Iterate - O(N)	s.find() - O(logN)
Insert	v.push_back() - O(1)	s.insert() - O(logN)
Delete	Search then delete - O(N)	s.erase() - O(logN)
Space	$O(N \cdot 2^N)$	$O(N \cdot 2^N)$

Represent a set as a binary number

0 → not there

1 → it is there

0, 10, 20, 30, 40



n

O(1) space

Let's use Bitmasks to represent subsets!

Let's visualise the array of size N as a binary number of N bits. For every subset, we will mark the existing elements as 1 in the binary string and the non-existing elements as 0. Example: Arr = [10, 20, 30, 40]

{}	0000	{30}	0100	{40}	1000	{30, 40}	1100
{10}	0001	{10, 30}	0101	{10, 40}	1001	{10, 30, 40}	1101
{20}	0010	{20, 30}	0110	{20, 40}	1010	{20, 30, 40}	1110
{10, 20}	0011	{10, 20, 30}	0111	{10, 20, 40}	1011	{10, 20, 30, 40}	1111

* check if ith element is there

→ [mask & (1 << i) ≠ 0] ??

int mask = 2^i

* How do you set a bit??

(mask |= (1 << i);).

↙

Binary repr

Let’s use Bitmasks to represent subsets!

Now convert every the binary strings into decimal numbers

{}	0000	0
{10}	0001	1
{20}	0010	2
{10, 20}	0011	3
{30}	0100	4
{10, 30}	0101	5
{20, 30}	0110	6
{10, 20, 30}	0111	7

{40}	1000	8
{10, 40}	1001	9
{20, 40}	1010	10
{10, 20, 40}	1011	11
{30, 40}	1100	12
{10, 30, 40}	1101	13
{20, 30, 40}	1110	14
{10, 20, 30, 40}	1111	15

Let's use Bitmasks to represent subsets!

Space used by every subset = O(1), Total space = $O(2^N)$

$n \leq 60$

Operations

Search for arr[i] => Check if the i^{th} bit is 1 in subset represented by bitmask

Insert arr[i] => Make the i^{th} bit as 1 in subset

Delete arr[i] => Make the i^{th} bit as 0 in subset

O(1)

True O(1) time

Some additional operations

Union of 2 subsets = $S_1 \mid S_2$

Intersection of 2 subsets = $S_1 \& S_2$

Elements existing in only one of the 2 subsets = $S_1 \wedge S_2$

* imp

number of operations $\lesssim 10^8$

$n \rightarrow 2^n < 10^8$

DP w/ bitmasking $n \leq 20$

Limitation on N

Can we generate bitmasks for an array of size 10^6

How high can the size of the array (N) be for us to use bitmasks?

Use of Bitmasks in DP

A lot of times we need information on which elements are picked and which are skipped in our state, that's where bitmasks become very useful.

Example: dp[i][bitmask] = some answer such that we are standing on the ith element of the array and bitmask represents the values we have chosen or skipped from 0 to i.

→ n find the number of perm. of an array of size n

↓ * all unique elements

$n!_0$ ✗

⇒ How would you simulate this through code??

Naive solution

vector → {1, 2, 3 ... n}

cont = 1

while (next_permutation()) → bool O(n)

count ++

}

TC → O(n! · n)

Recursion → memoising

index

```
f ( i , mask) {
  if (i==n) return 1;
  sum = 0
  for (int j = 0 ; j < n ; j++) {
    if (jth bit not set) sum +=
        f(i+1, mask | (1<<j))
```

int sum = f(0,0);

dp[i][mask]

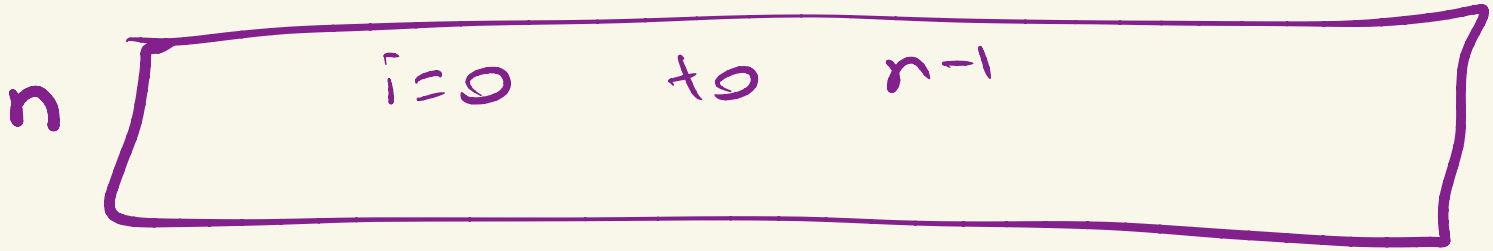
↳ number of ways to place elmst such that we are place elmt on the ith index and state of used elmst is mask

f(i, mask){

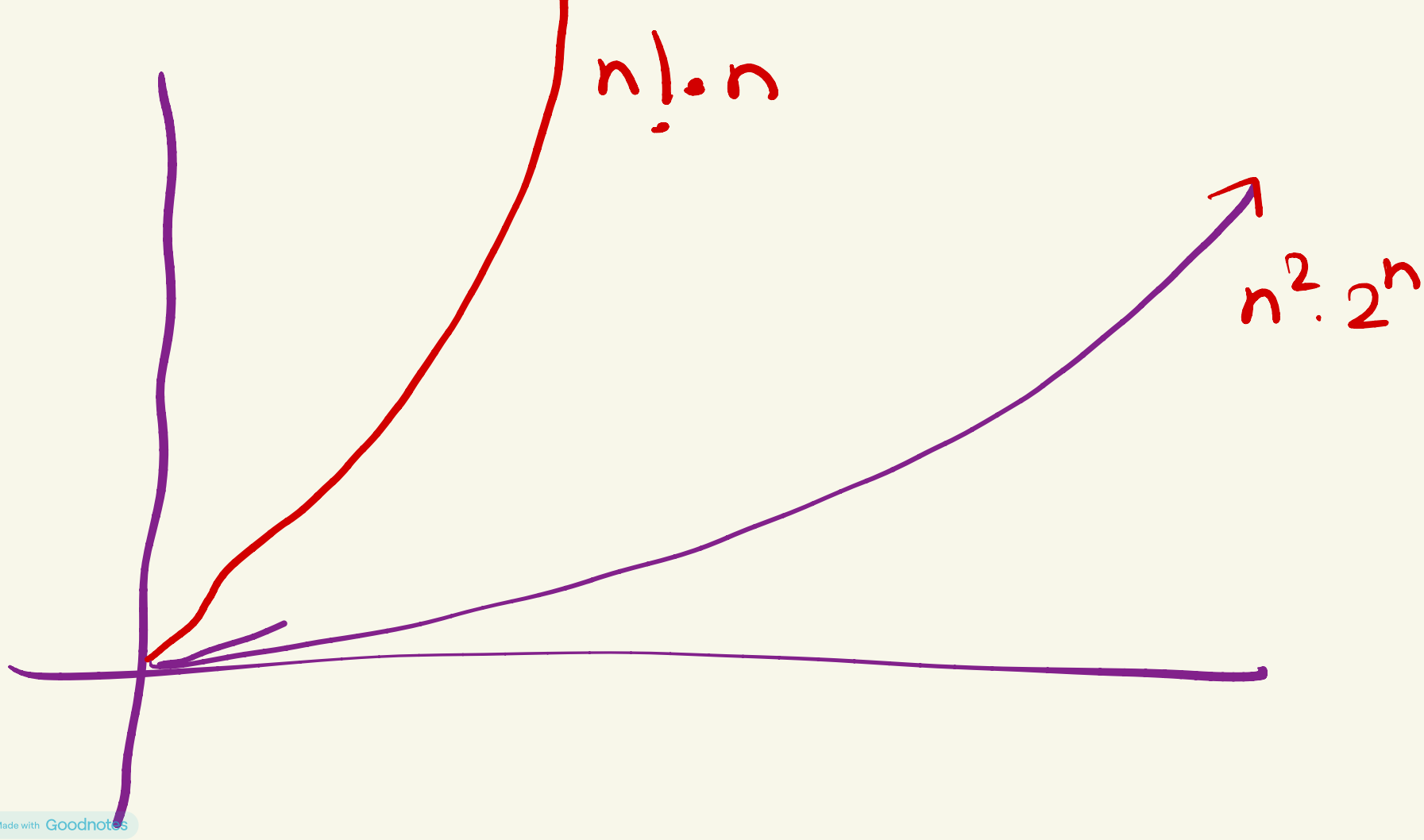
i → n

mask → 2^n

$n! \cdot n$



y TC → $O(n^2 \cdot 2^n)$



Problem 1: Link

Atcoder DP

Problem D

$n \leq 21$

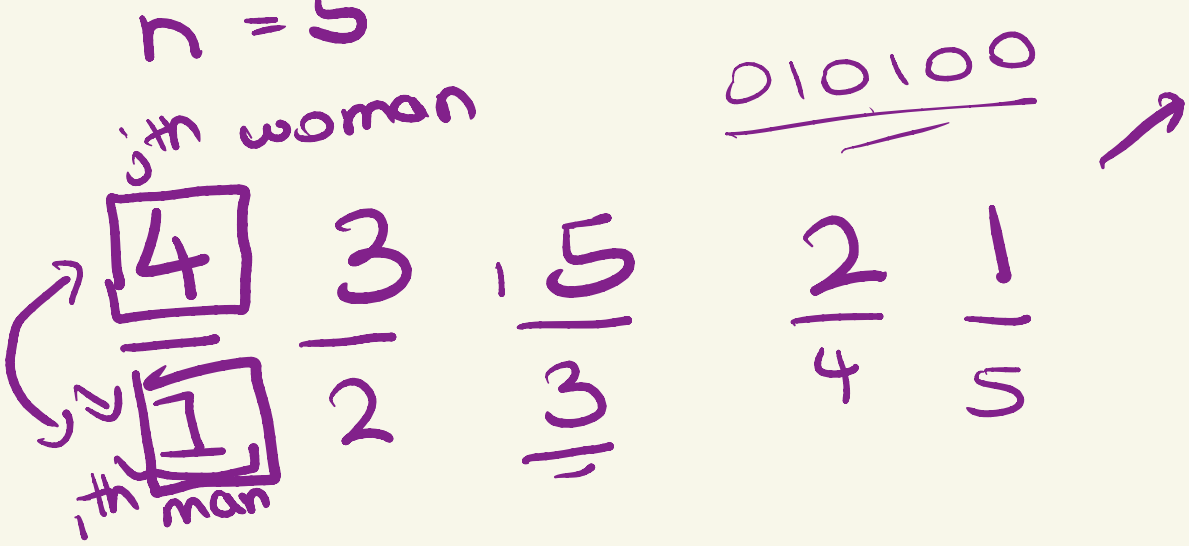
$a_{ij} = 1$ i^{th} man is compatible with the j^{th} woman

number of pairing

n disjoint pairs (i, j) $a_{ij} = 1$

→ number of permutations

$n = 5$



(1, 4)
(2, 3)
(3, 5)
(4, 2)
(5, 1)

jth → jth index

* jth emt should be free * $a_{ij} = 1$

int dp[n][1 ≤ cn]

v < map < set < int >, int >> (n)

Problem 1 Solution

$O(n^2 \cdot 2^n)$

pair ith man
state of women represented by mask

```
int f(int index, int mask)
{
    if(index == n) return 1;
    if(dp[index][mask] != -1) return dp[index][mask];

    int ways = 0;

    for(int i = 0; i < n; i++)
    {
        if(((1 << i) & mask) == 0 && grid[index][i])
            ways = (ways + f(index + 1,mask | (1 << i))) % MOD;
    }

    return dp[index][mask] = ways;
}
```

ith woman is not taken
compatibility

5-6 lines

f(i, mask) → match from the ith man ahead

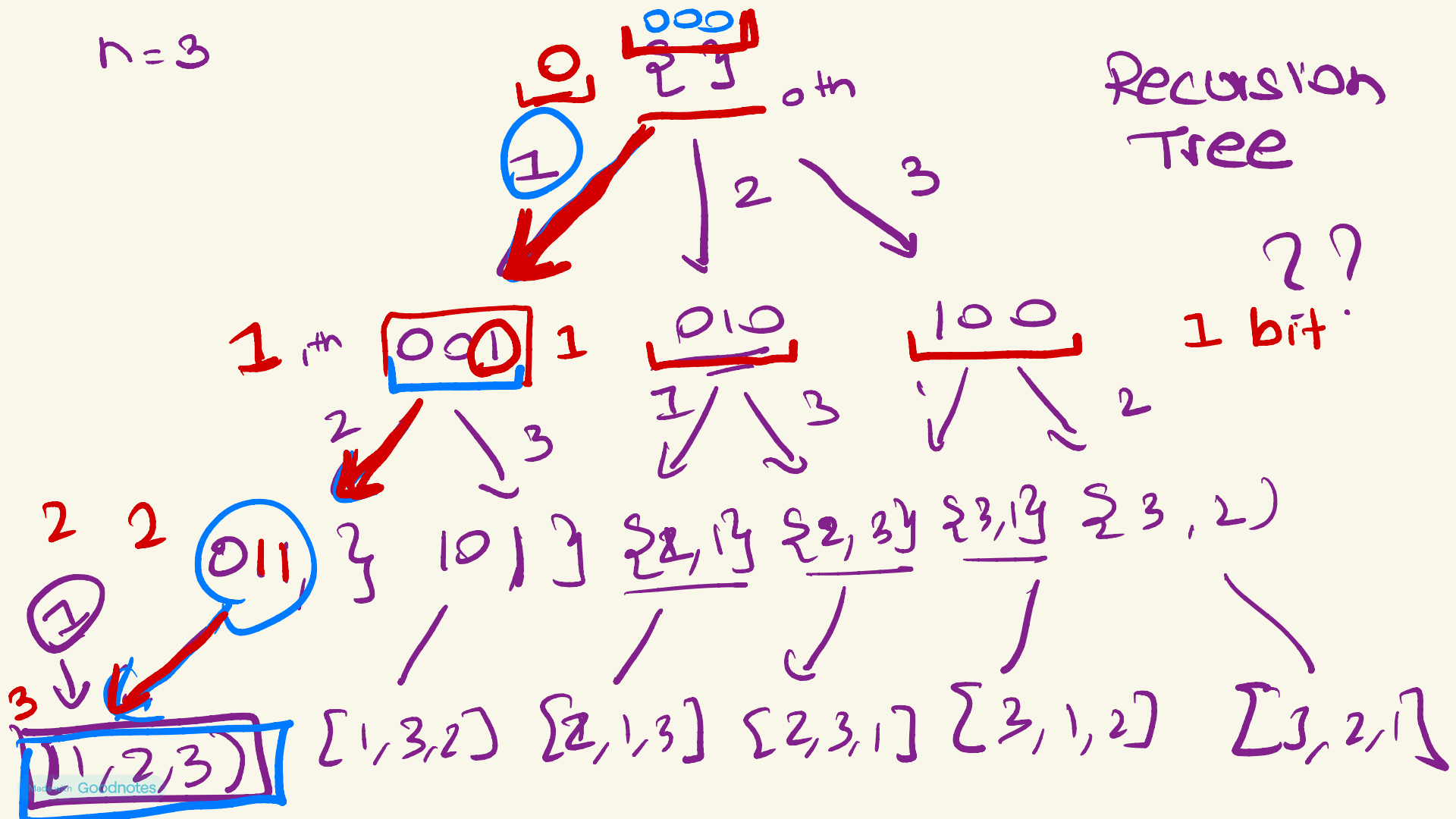
Problem 1 Solution (Optimized)

```
int f(int index, int mask)
{int index = __builtin_popcount(mask);
    if(index == n) return 1;
    if(dp[index][mask] != -1) return dp[index][mask];

    int ways = 0;

    for(int i = 0; i < n; i++)
    {
        if(((1 << i) & mask) == 0 && grid[index][i])
            ways = (ways + f(index+1,mask | (1 << i))) % MOD;
    }

    return dp[index][mask] = ways;
}
```

O(n)

SC O(2^n)

TC O(n·2^n)


n=3
Recursion Tree
000
0
0th
1
2
3
1 th
001
1
010
100
1 bit
??
2
3
1
3
2
2 2
011
10
{2,1}
{2,3}
{3,1}
{3,2}
1
3
(1,2,3)
[1,3,2]
[2,1,3]
[2,3,1]
[3,1,2]
[3,2,1]

$\times O(n^2 \cdot 2^n) \longrightarrow 10^9$?TLE?!

$O(n \cdot 2^n)$?

↳ index & mask

dependent on each other?!

index = number of set bits in mask

$n * 2^n$ dp[i][mask]

→ 2^n useful states

n = 3

0,000,

1,001

1,010

2,011

1,100

mask

2,101

2,110

3,111

$O(2^n)$

X

2,111

X never possible

Problem 2

TC → $O(n^2 \cdot 2^n)$

Given a list of points on a 2D plane, rearrange these points in any way such that in the final permutation of points, the sum of distances of the adjacent elements is minimized.

Constraints: [N <= 15], [-1e9 <= Xi, Yi <=1e9]

manhattan dis

abs $(x_1 - x_2)$ + abs $(y_2 - y_1)$

Points : [{0, 0}, {5, 6}, {1, 2}]

Best permutation -> [{0, 0}, {1, 2}, {5, 6}]]

Ans = Dist(P1, P3) + Dist(P3, P2) = 11

3 8

$\rightarrow \sum \text{cost}(arr_i, arr_{i+1})$

$i \rightarrow$ we want to place something

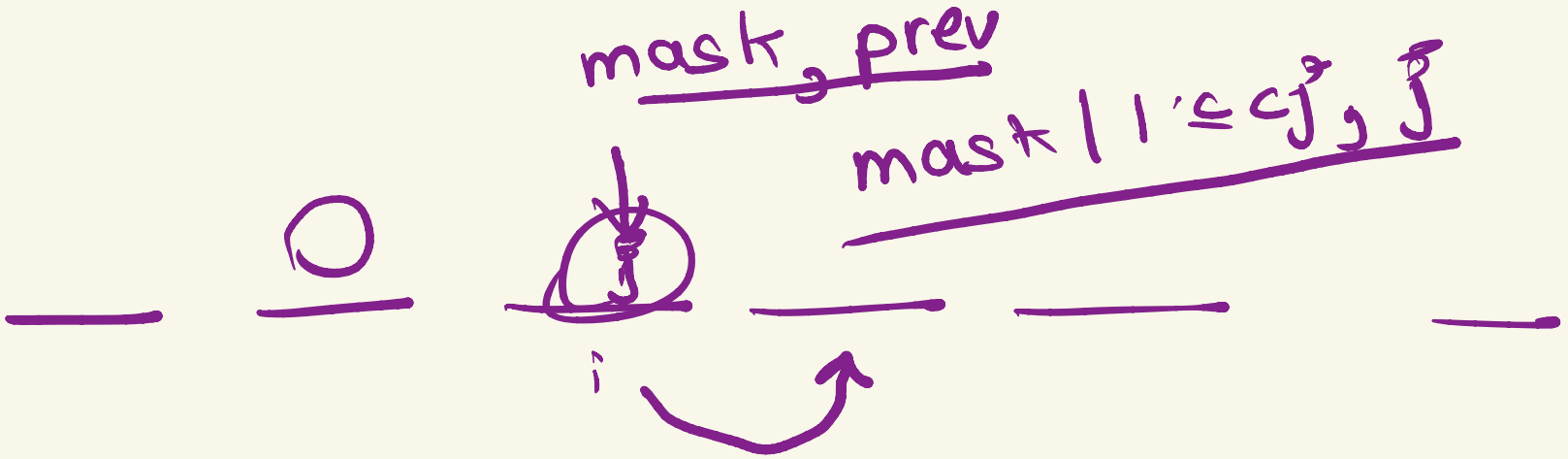
$dp[mask][prev] \Rightarrow n \times 2^n$

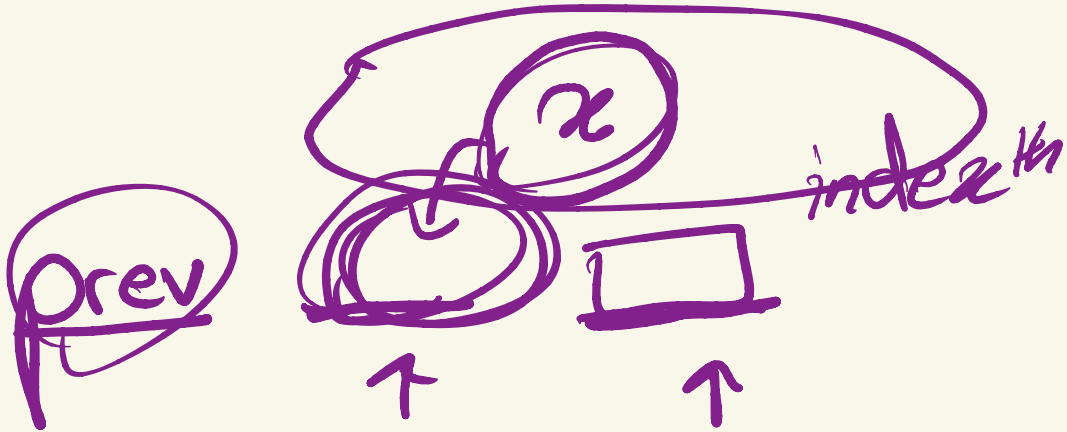
index = __builtin_popcount(mask)

answer = INF

i = 0 to i = n - 1

ans = min(ans, cost(i, prev) + f(mask | 2^i, i))





Solution 1 - TC: $O(n^2 2^n)$, SC: $O(n^2 2^n)$

```
state:
    dp[i][bitmask][last element] = minimum sum of distances in the suffix [i... n - 1]
    such tha bitmask represents the elements in the first i - 1 elements and last elements
    represents the last point

transition:
    check for jth point from (0 to n - 1)
    can you pick the jth point as the ith element in the final array or not

    if(bitmask & (1 << j)){ whether jth bit is set or not
        continue;
    }else{
        dp[i][bitmask][last element] = min(dp[i][bitmask][last element],
        (bitmask != 0 ? dist(j, last element) : 0) + dp[i + 1][bitmask | (1 << j)][j]
    }

base case:
    dp[n][(1 << n) - 1][anything] = 0

final subproblem
    dp[0][0][anything]
```

Solution 2: TC: $O(n^2 2^n)$, SC: $O(n*2^n)$

```
state:
    dp[bitmask][last element]
    i = set_bits(bitmask)
    = minimum sum of distances in the suffix [i... n - 1] such tha bitmask represents the
    elements in the first i - 1 elements and last elements represents the last point

transition:
    check for jth point from (0 to n - 1)
    can you pick the jth point as the ith element in the final array or not

    if(bitmask & (1 << j)){ whether jth bit is set or not
        continue;
    }else{
        dp[bitmask][last element] = min(dp[bitmask][last element],
        (bitmask != 0 ? dist(j, last element) : 0) + dp[bitmask | (1 << j)][j]
    }

base case:
    dp[(1 << n) - 1][anything] = 0

final subproblem
    dp[0][anything]
```