Today's Content:

↳ Back Tracking → 1, 2, 3

1) All numbers
2) Subset sum
3) General way

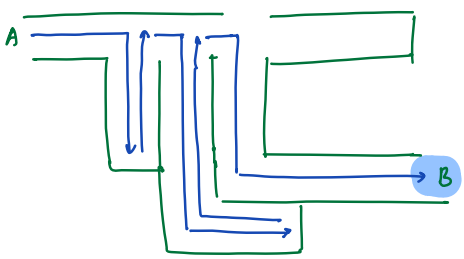
Back Tracking:

↳ Generating all possibilities to get correct ans

Try out all combinations, if a particular is not working re-trace & pick a new combination.

Maze:

↳ Harry potter Goblet of fire:



Note: All Back tracking codes are written by Recursion

Q) Given N digits print all N digit numbers formed only only 1 & 2 in increasing order of numbers

N=1: 1
2

N=2: 1 1] 11 < 12
1 2] 12 < 21
2 1] 21 < 22
2 2

N=3: 1 1 1
1 1 2
1 2 1
1 2 2
2 1 1
2 1 2
2 2 1
2 2 2

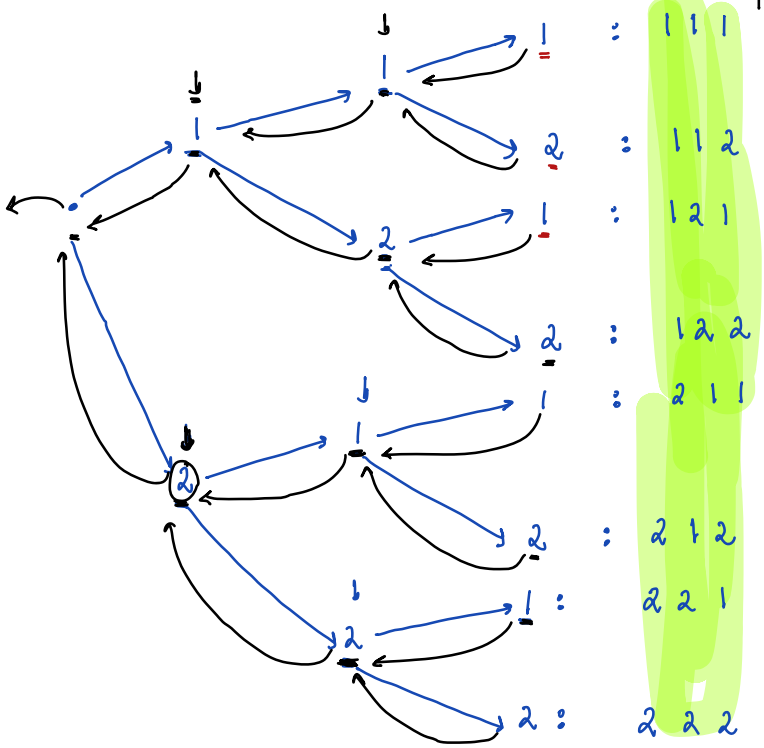
Idea: If we replace 1 → 0 & 2 → 1, Can be solved using bit-masking

0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1

TC: $O(2^n * n)$

N=4: 4 position to fill — — — —
1 — — — — 3 more position to fill
2 — — — — 3 more position to fill

N=3: 0 — 1 — 2 — 3 → we filled the number print it & backtrack



→ func C): →

parameters: N, ar[N], i → at what index we are currently filling data

function calls: 2

return type: void

pass by reference

```
void printall ( int ar[], int N, int i) {
    if ( i == N) { // We have a number hence print & return
        print (arr)
        return;
    }
    ar[i] = 1;  // made
    printall (ar, N, i+1);
    ar[i] = 2; }
    printall (ar, N, i+1);
}
```

(int ar[]: where we are keeping data; int i: current index, where we are keeping data)

pass by refera:

All function calls are going to same array

TC: $[2^N * N]$

SC: $[N + N]$

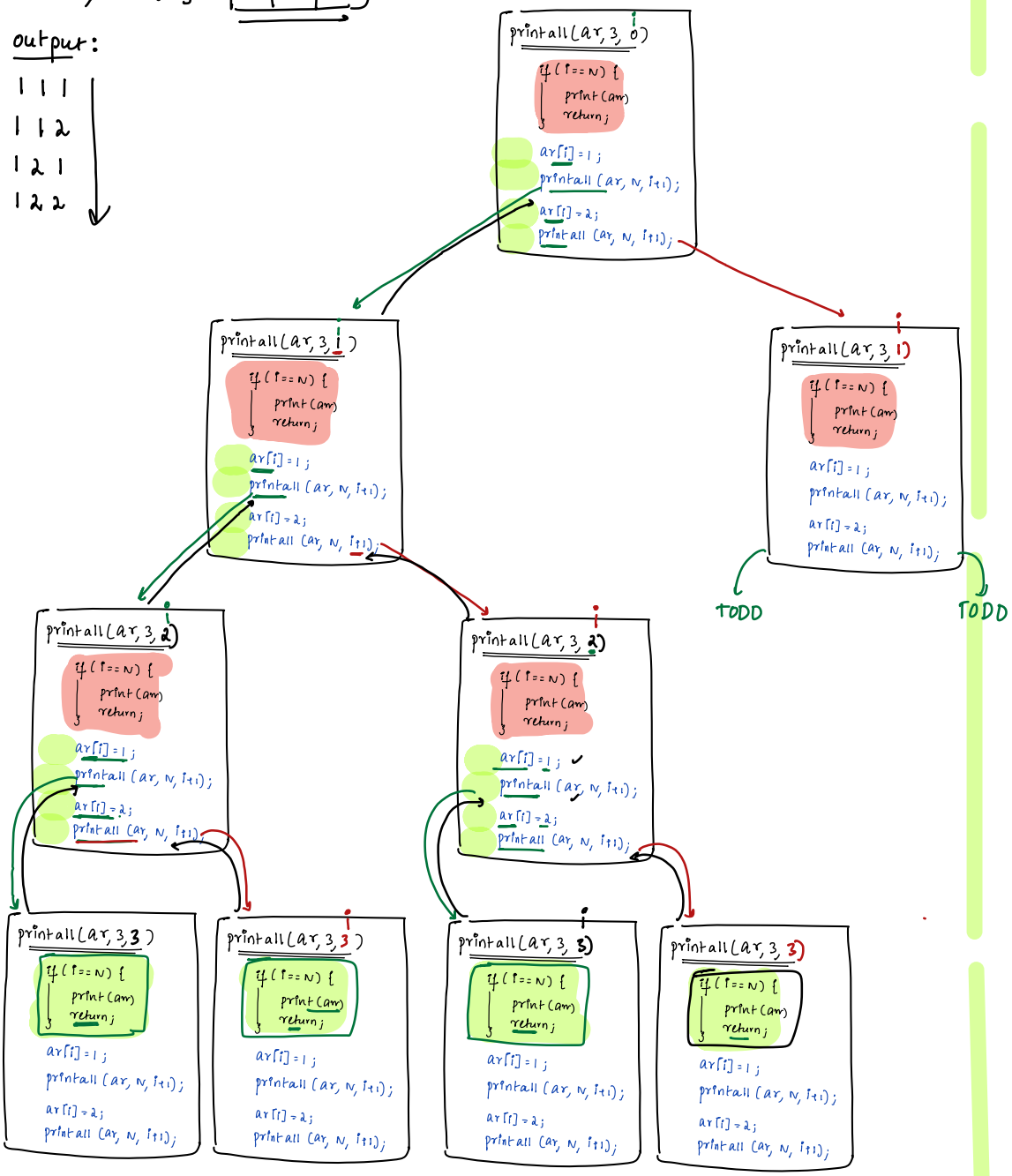
- N → ar[]
- N → Call stack size

```
main C ) {
    int N;
    Read N
    int ar[N]
    printall ( ar, N, 0)
}
```

// N=3, ar[3] = | 0: 2 | 1: 2 | 2: 2 |

output:

1 1 1
1 1 2
1 2 1
1 2 2

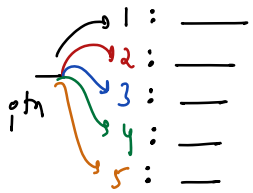


// Given N print all N digit numbers formed by digits = {1 2 3 4 5}
in increasing order

func() :

parameters : ar[], N, i

functions calls :




```
void printall (int ar[], int N, int i) {
    if (i == N) { // We have a number hence print & return
        print(ar)
        return;
    }

    ar[i] = 1;
    printall (ar, N, i+1);

    ar[i] = 2;
    printall (ar, N, i+1);

    ar[i] = 3;
    printall (ar, N, i+1);

    ar[i] = 4;
    printall (ar, N, i+1);

    ar[i] = 5;
    printall (ar, N, i+1);
}
```

```
for(int v=1; v<=5; v++) {
    ar[i] = v;
    printall(ar, N, i+1)
}
```

TC: $5^N * \{N\}$

SC: N + N

ar — Call stack size

10: 23 → 10:30pm

2Q) Given N array elements count no: of subsets with sum == k

↳ arr ⇒ +ve, 0, -ve

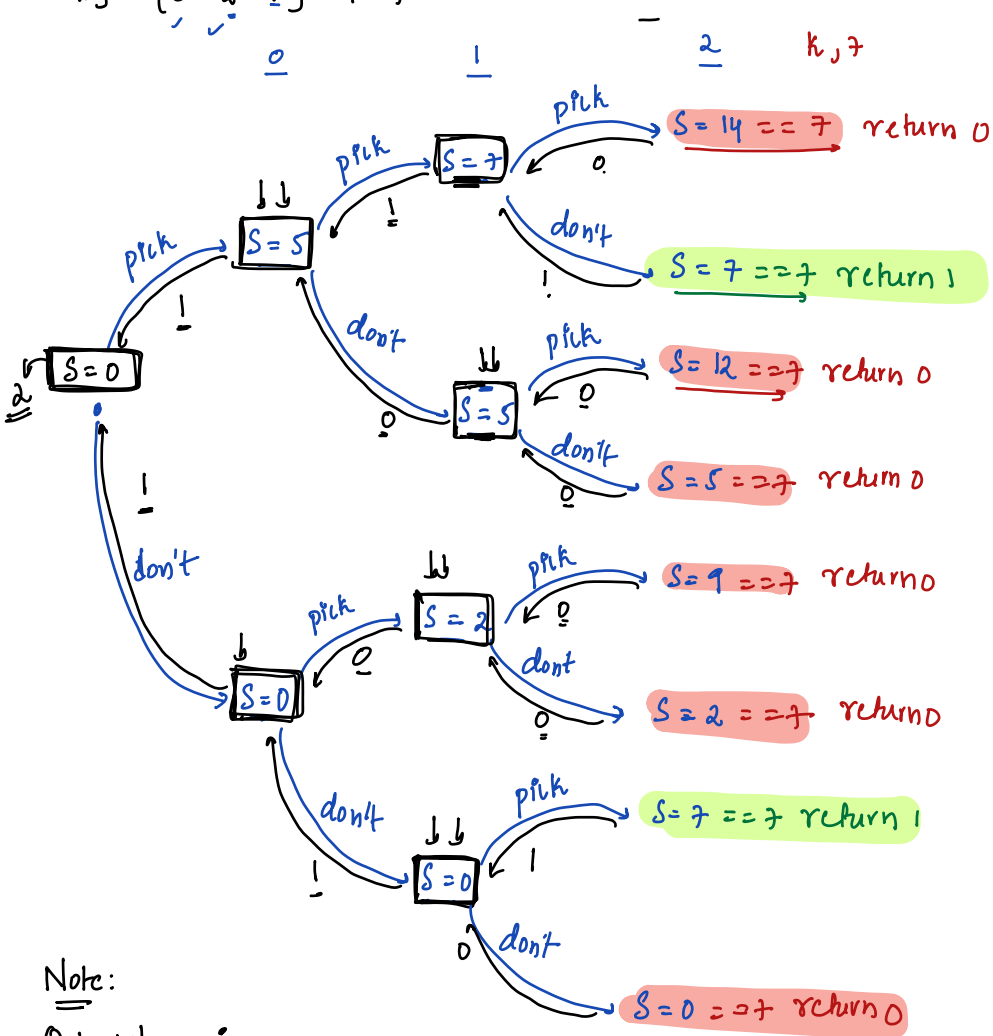
Eg1: [10 2 7 6 1 5]

k=8 : {2 6} {1 7} {5 2 1} ⇒ ans = 3

Ideal: Generate all subsets sum & compare == k

a) Using Bit manipulations TC: O(2^N * N)

ar[3] = [5 2 7] k = 7



Note:

Only when i == N,

we are checking if sum == k

func: given arr[] size target sum current position current sum

parameters : ar[], N, k, i, sum

functioncalls : 2 function calls

returntype : int

```
int Subset(int ar[], int N, int k, int i, int sum){
    if(i == N){ if(sum == k) return 1 else return 0}
    int c = 0
    sum = sum + ar[i]// updated
    c = c + Subset(ar, N, k, i+1, sum)        } pick ith element
    sum = sum - ar[i]//Subtrat
    c = c + Subset(ar, N, k, i+1, sum)        } leave ith element
    return c;
}
```

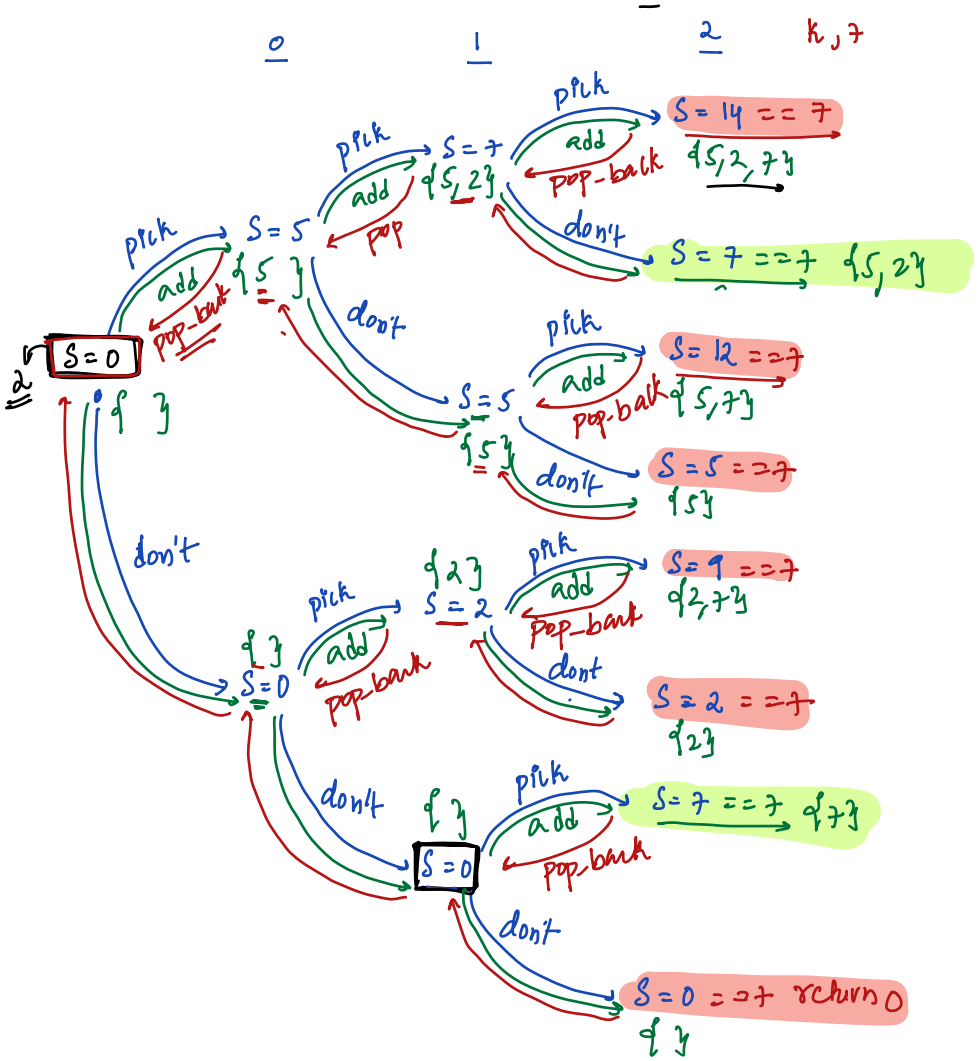
TC: $2^N * \{1\}$ SC: O(N) to stack size

// 208) Print all Subset with Sum == k

add : will add element at back

pop back : will remove last element

array_list is pass by reference



func : given arr[] size target sum current position current sum contains subset elements

parameters : ar[], N, k, i, sum, list ↳ pass by reference

function calls : 2 function calls

return type : void

```
void printSubSum(int ar[], int N, int k, int i, int sum, list<int> l){
    if(i==N){
        if(sum==k){ print(l) }
        return
    }

    sum = sum + ar[i]      } made changes
    l.add(ar[i])
    printSubsum(ar, N, k, i+1, sum, l)

    sum = sum - ar[i]      } revert changes
    l.pop_back
    printSubsum(ar, N, k, i+1, sum, l)
}
```

// In general: Backtracking → parameter, function calls: (How many choices are present), return type.

```
fun(_____)
    BaseCase:
        return;

    At i^th index:
        Make changes
        fun(_____)
        revert changes

    Make other calls:
}
```