Todays Content: Pre-requisits : Recursion

- Back Tracking
- All numbers
- Subset sum
- Generate all permutations

Happy New Year Everyone

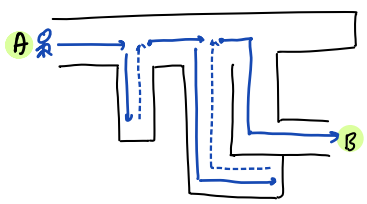
- Will share my
  - : Heaps : 2
  - : Greedy

Notes + pin links in message

Backtracking: Generating all solutions using recursion : Backtracking

1) All Subsets
2) All permutations
3) All Combinations

Maze:



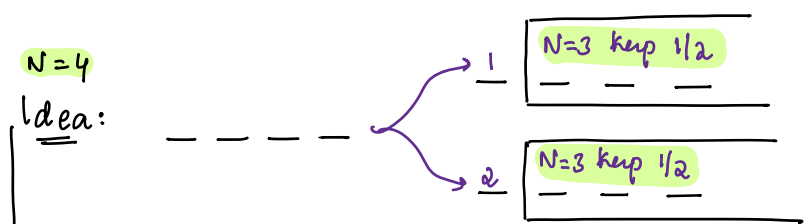
Idea: Generating solutions, choose path but we cannot go any further, come back & take another path.

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

N:2 :
1 1
1 2
2 1
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: 1. Using queue.
: Please Queue notes
2. Using bit manipulations

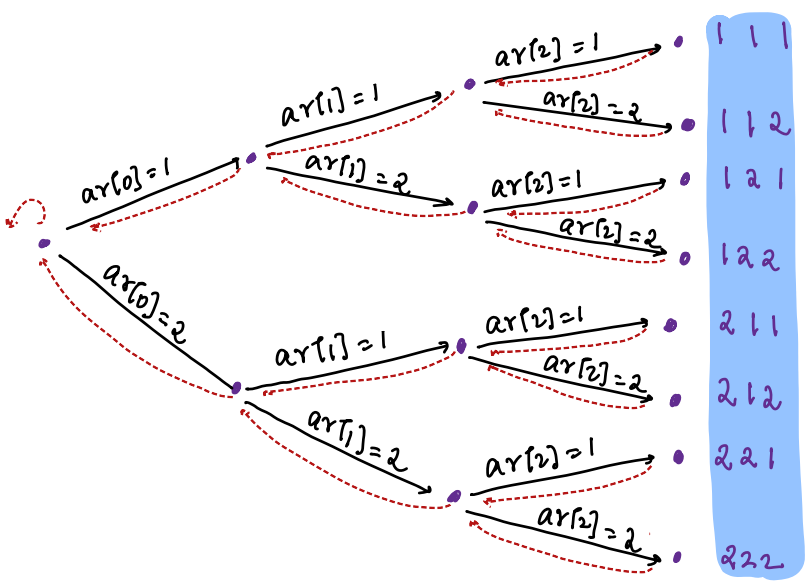


// Generate all n digit numbers using recursion : Backtracking

Tracing : N=3

End of path

	0	1	2
ar[3] =	2	2	2



func():

↳ current present we are at

→ parameters : ar[N] i N → size can indicate end

→ subproblems : 2

how many subProblem a single problem has $2 \cdot 2 \cdot 2 \cdots 2 = 2^N$ N digit numbers

→ return type: void

10:25 pm

```
void printAll( int ar[], int i, int N ){   TC: O(2^N * N)   SC: O(N + N)
                                                  array ↑   ↑ call stack size.
    if ( i == N ){
        print ar[]
        return;
    }
    // At i^th index choice
    ar[i] = 1
    printAll(ar, i+1, n)
    ar[i] = 2
    printAll(ar, i+1, n)
}
```

↳ Every function call will have its own copy

i & N : pass by value

↳ All functions will use same array.

: pass by reference

→ arr[] → map → pass an object it's pass by reference

→ list → ~

int ar[] = new int[]

HashMap<int, int> = new HashMap< >

ArrayList<int> = new ArrayList< >

: primitive datatypes

: int/float/double/long...

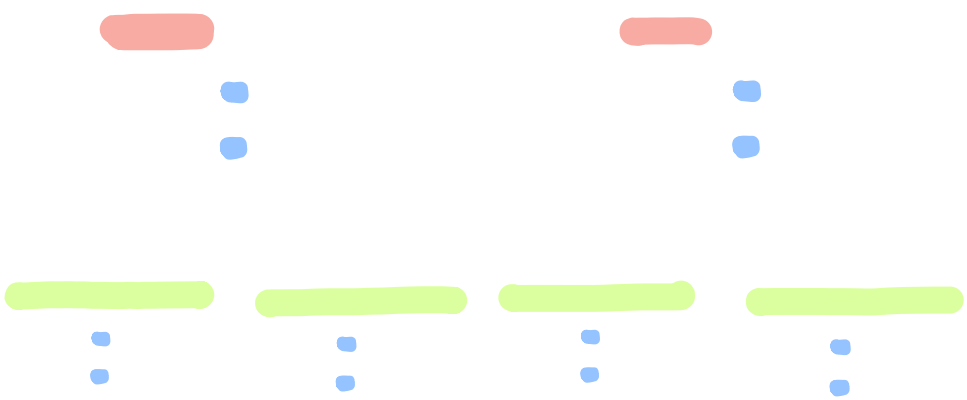
```
main(){
    int ar[N] = {0}
    printall(ar, 0, N)
}
```

```
main(){
    int ar[3] = {0}
    printAll(ar, 0, 3)
}
```

ar[3] = {1 }

```
void printAll(ar[], i=0, N=3)
    if (i==N) { print ar[] return; }
    ar[i] = 1
    printAll(ar, i+1, n)
    ar[i] = 2
    printAll(ar, i+1, n)
}
```

printAll

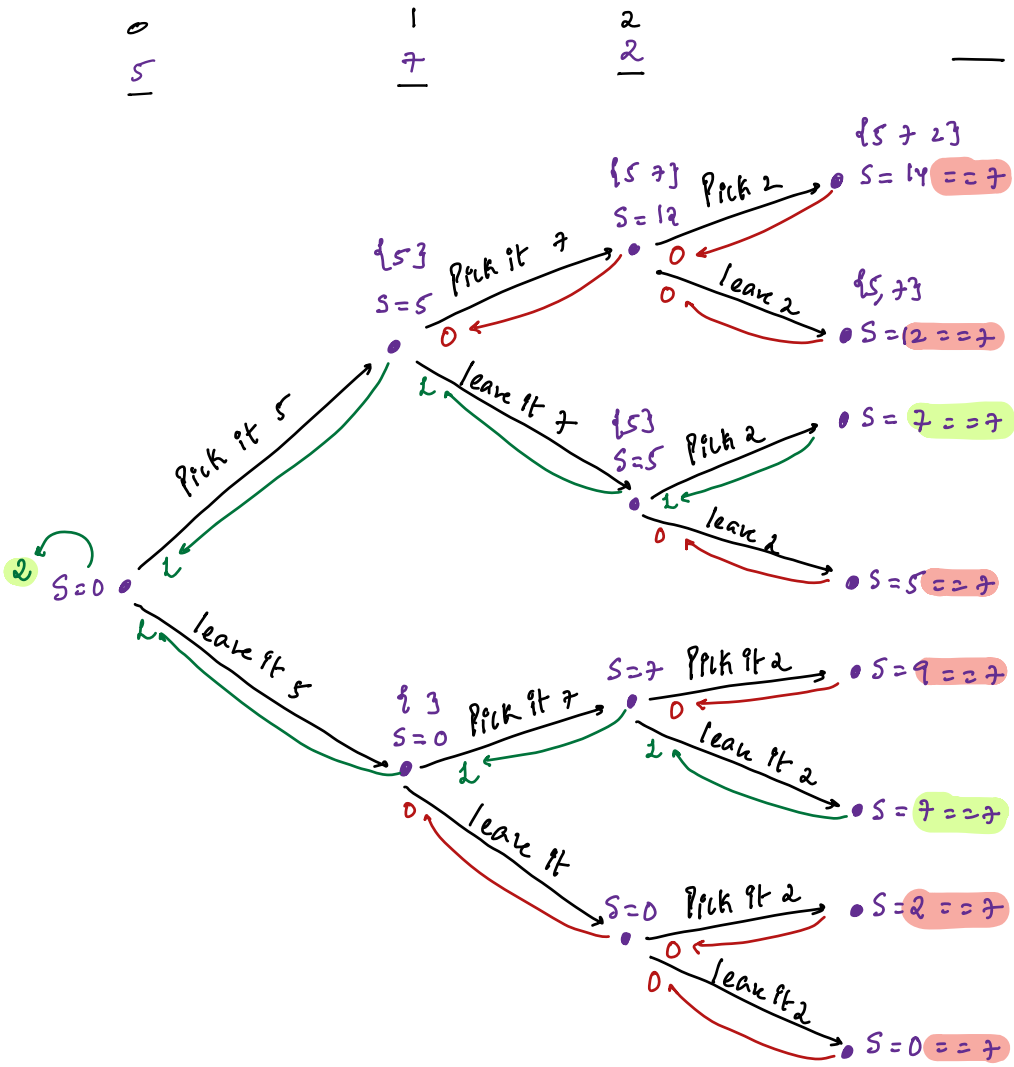


Q) Given ar[N] elements, count of subsets with sum == k

Eg1: ar[3] = { 5 (0) 7 (1) 2 (2) } k = 7, ans = 2

Subsets: {5 2} {7}

Idea: Generate all subset sums == k : backtracking



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

SubProblems: 2

returntype: int : count of subsets

```
int countSubSumC(int ar[], int N, int k, int i, int sum){
    if(i==N){
        if(sum==k){return 1}
        else{return 0}
    }
    // At ith index choices
    sum = sum + ar[i] // Picking ith ele sum gets updated
    int c1 = countSubSumC(ar, N, k, i+1, sum) // subsets with sum=k
    sum = sum - ar[i] // leave ith ele, sum we are reducing
    int c2 = countSubSumC(ar, N, k, i+1, sum) // subsets with sum=k
    return c1+c2
}
```

undoing updates on sum

TC: $2^n * 1$

SC: O(N): Stack size

Note: In back tracking

: variable updates

fun(_ , _ , _)

: undo updates in above variable

// Given ar[N] distinct elements print all permutations / all arrangements } Backtracking

ar[3] = { 6 9 14 }

Permutations / arrangements for n elements = N * N-1 * N-2 * N-3 * ..... — 1

→ N! permutations

6 9 14
6 14 9
9 6 14
9 14 6
14 6 9
14 9 6

parameters: ar[], i, N

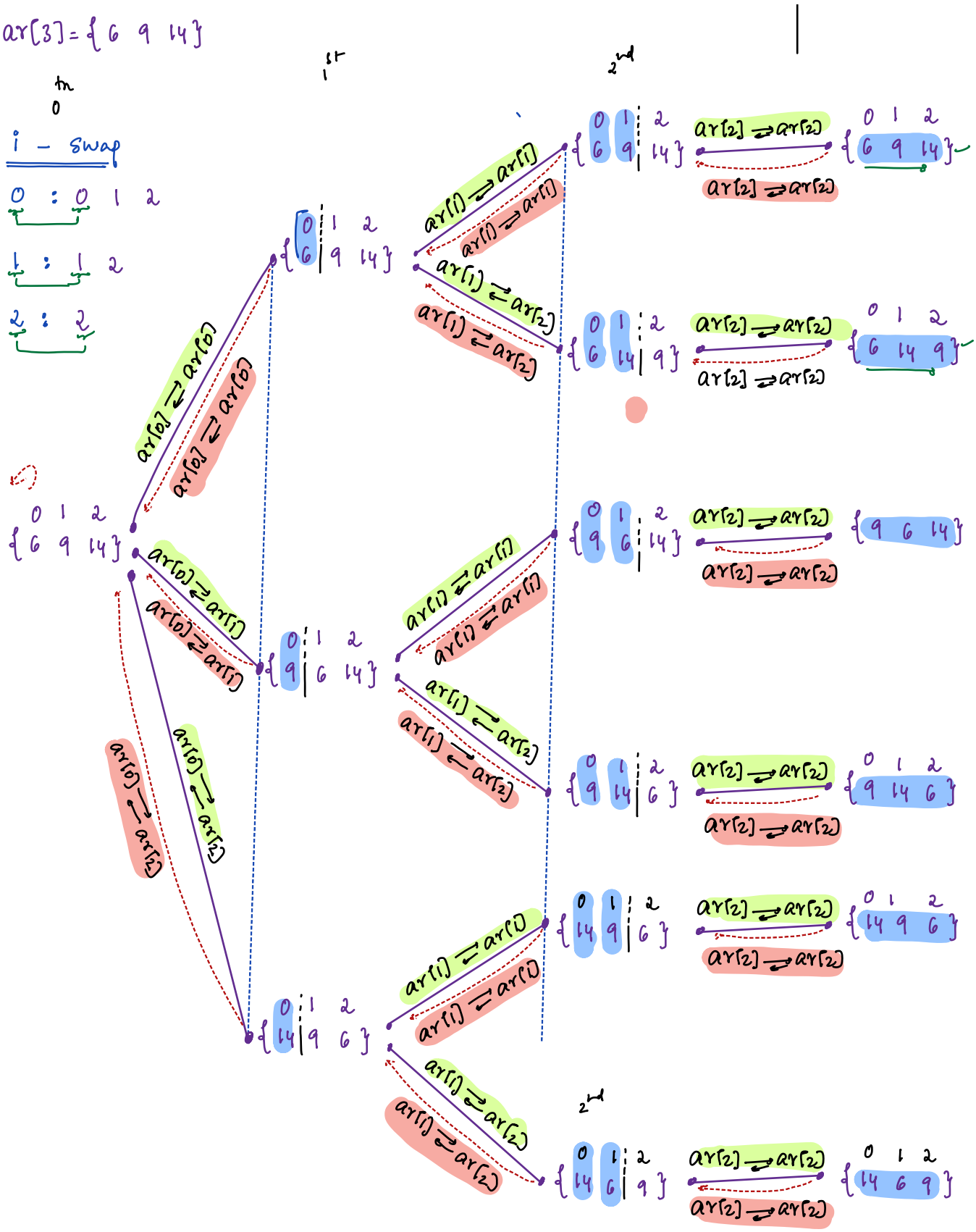
Subproblems: At i^{th} index we swap with all indices from [i, N-1]

return type: void

```
void allpermute(int ar[], int N, int i){
    if(i==n){
        print(ar[])
        return;
    }
    // At i^th index choices = swap i^th index = [i, n-1]
    for(j=i; j<n; j++){ // choices
        // at ar[i] we want ar[j]
        swap ar[i] ⇄ ar[j] // for i^th index with ar[j]
        allpermute(ar, N, i+1)
        swap ar[i] ⇄ ar[j] // undo update
    }
}
```

TC: (N! * N)

SC: O(N

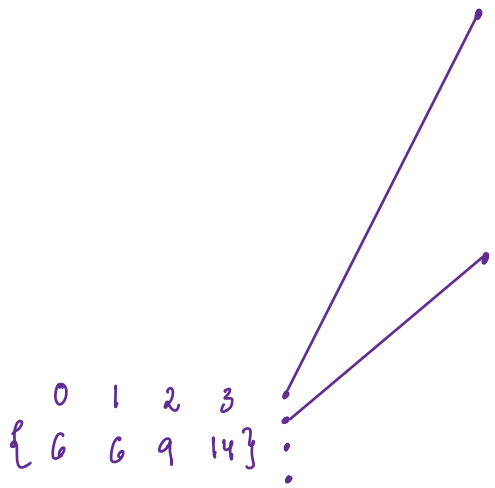

arr[3] = { 6 9 14 }
1st
2nd
i – swap
0 : 0 1 2
1 : 1 2
2 : 2
arr[0] ⇄ arr[0]
arr[1] ⇄ arr[1]
arr[2] ⇄ arr[2]
{ 6 9 14 }
{ 6 14 9 }
{ 9 6 14 }
{ 9 14 6 }
{ 14 9 6 }
{ 14 6 9 }

Permutations with Repeatim : TODO array duplicates, get all distinct permutations

ar[4] = { 6 6 9 14 }

0th

ir
1



arr[4] = { 6 6 9 14 }