

Todays Content: Pre-requisites: Recursion

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

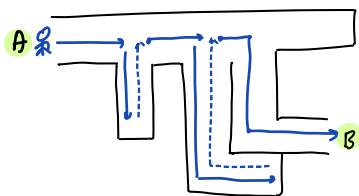
} Happy New Year Everyone

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

Notes + pen links in message

Backtracking: Generating all solutions using recursion: Backtracking

Maze:



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

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

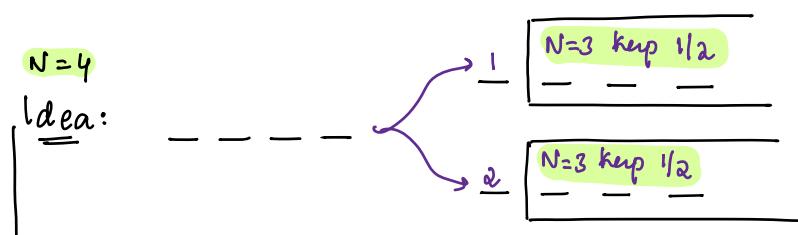
18) Given N digits print all N digit numbers formed only by 1 & 2  
in increasing order of numbers

N:2 :	<u>1</u> <u>1</u>	N:3 :	<u>1</u> <u>1</u> <u>1</u>
	<u>1</u> <u>2</u>		<u>1</u> <u>1</u> <u>2</u>
	<u>2</u> <u>1</u>		<u>1</u> <u>2</u> <u>1</u>
	<u>2</u> <u>2</u>		<u>1</u> <u>2</u> <u>2</u>
			<u>2</u> <u>1</u> <u>1</u>
			<u>2</u> <u>1</u> <u>2</u>
			<u>2</u> <u>2</u> <u>1</u>
			<u>2</u> <u>2</u> <u>2</u>

Idea: 1. Using queue.

: Please Queue notes

2. Using bit manipulations

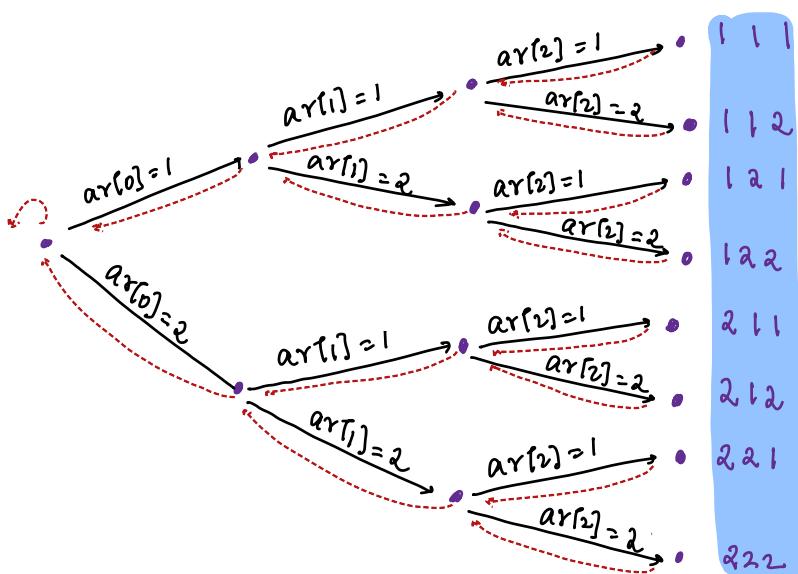


// Generate all n digit numbers using recursion: Backtracking

Tracing :  $N=3$

End of path

0	1	2	3	$ary[3] =$
0	1	2	—	0 1 2



func:

→ current present we are at

- parameters:  $ar[N]$  i  $\rightarrow$  size can indicate end
- Subproblems: 2  
how many subproblem a single problem has  $2 \times 2 \times 2 \times \dots = 2^N$  N digit numbers
- return type: void

10: 25 pm

void printAll( int ar[], int i, int N ) { TC:  $O(2^N * N)$  SC:  $O(N + N)$

```
if ( i == N ) {
    print ar[]
    return;
}
```

```
// At ith index choice
ar[i] = 1
printAll(ar, i+1, n)
ar[i] = 2
printAll(ar, i+1, n)
```

main() {

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

↳ Every function call will have its own copy  
( e.g. N : pass by value )

All functions will use same array.  
: pass by reference

→ arr[] → map → pass an object it's pass  
→ list → → by reference  
int ar[] = **new** int[]  
HashMap<int, int> = **new** HashMap<>  
ArrayList<int> = **new** ArrayList<>

: primitive datatypes

: int / float / double / long ...

```

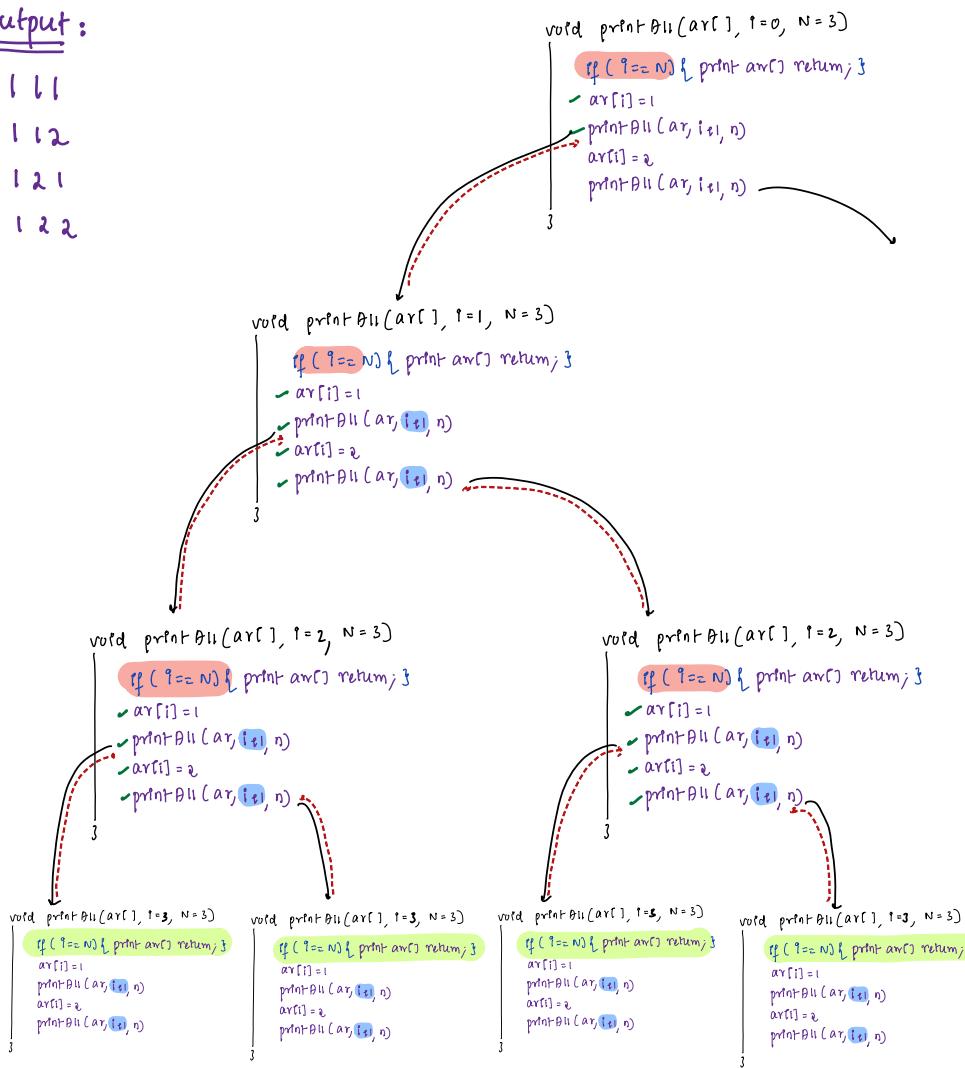
main() {
    int ar[3] = {0, 1, 2};
    printAll(ar, 0, 3);
}

```

$$ar[3] = \{1 \ 2 \ 2\}$$

output :

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

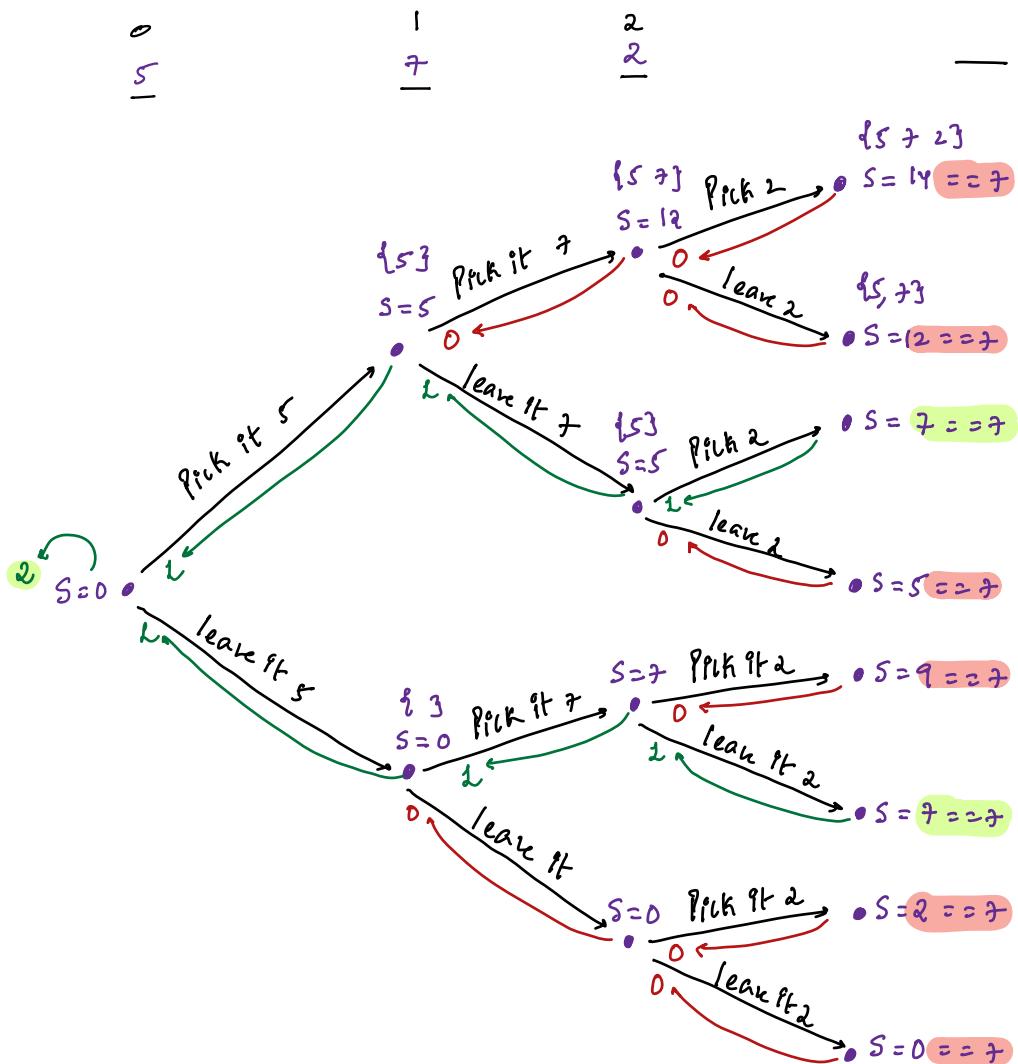


Q) Given  $\text{arr}[N]$  elements, count of subsets with sum == k

Ex1:  $\text{arr}[3] = \{ 5, 7, 2 \}$   $k=7$ , ans=2

Subsets:  $\{5, 2\} \{7\}$

Idea: Generate all subset sums == k  
 : backtracking



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

SubProblems: 2

return type: int : count of subsets

int countSubSumC(int arr[], int N, int k, int i, int sum) {

```
if (i == N) {  
    if (sum == k) { return 1 }  
    else { return 0 }  
}
```

$\left\{ \begin{array}{l} TC: 2^N * 1 \\ SC: O(N): \text{Stack size} \end{array} \right.$

// At  $i^{th}$  index choices

undoping updates on sum [   
      sum = sum + arr[i] // Picking  $i^{th}$  ele sum gets updated  
      int  $c_1 = \text{countSubSum}(arr, N, k, i+1, \text{sum})$  // Subsets with sum = k  
      sum = sum - arr[i] // Leave  $i^{th}$  ele sum we are reducing  
      int  $c_2 = \text{countSubSum}(arr, N, k, i+1, \text{sum})$  // Subsets with sum = k  
      return  $c_1 + c_2$  ]

Note: In back tracking

: variable updates

fun( — , — , — )

: undo updates in above variable

// Given arr[N] distinct elements print all permutations:  
 $\downarrow$   
 $\text{arr[3]} = \{6, 9, 14\}$

} Backtracking  
 all arrangements

Permutations / arrangements } for n elts =  $\frac{N \times N-1 \times N-2 \times N-3 \times \dots \times 1}{N!}$   
 $\downarrow$   
 $\begin{array}{ccc} 6 & 9 & 14 \\ \underline{-} & \underline{-} & \underline{-} \\ 6 & 14 & 9 \end{array}$

$\begin{array}{ccc} 9 & 6 & 14 \\ \underline{-} & \underline{-} & \underline{-} \\ 9 & 14 & 6 \end{array}$  parameters: arr[], i, N

$\begin{array}{ccc} 9 & 14 & 6 \\ \underline{-} & \underline{-} & \underline{-} \\ 14 & 6 & 9 \end{array}$  subproblems: At  $i^{\text{th}}$  index we swap with all indices from  $[i, N-1]$

$\begin{array}{ccc} 14 & 9 & 6 \\ \underline{-} & \underline{-} & \underline{-} \end{array}$  return type: void

void allpermute(int arr[], int N, int i){

if ( $i == n$ ) {  
 print(arr[]);  
 return;  
}}

TC:  $(N! * N)$

SC:  $O(N)$

// At  $i^{\text{th}}$  index choices = Swap  $i^{\text{th}}$  index =  $[i, N-1]$

$j = i; j < n; j++$  } // choices

// at arr[i] we want arr[j]

Swap arr[i]  $\leftrightarrow$  arr[j] // fix  $i^{\text{th}}$  index with arr[j]

allpermute(arr, N, i+1)

Swap arr[i]  $\leftrightarrow$  arr[j] // undo update

$$\text{ar}[3] = \{6 \ 9 \ 14\}$$

tn  
0

$$\underline{0} : \underline{0} \mid 2$$

1 2

2 : 2



0 1 2

۲۰

1

10

11

92

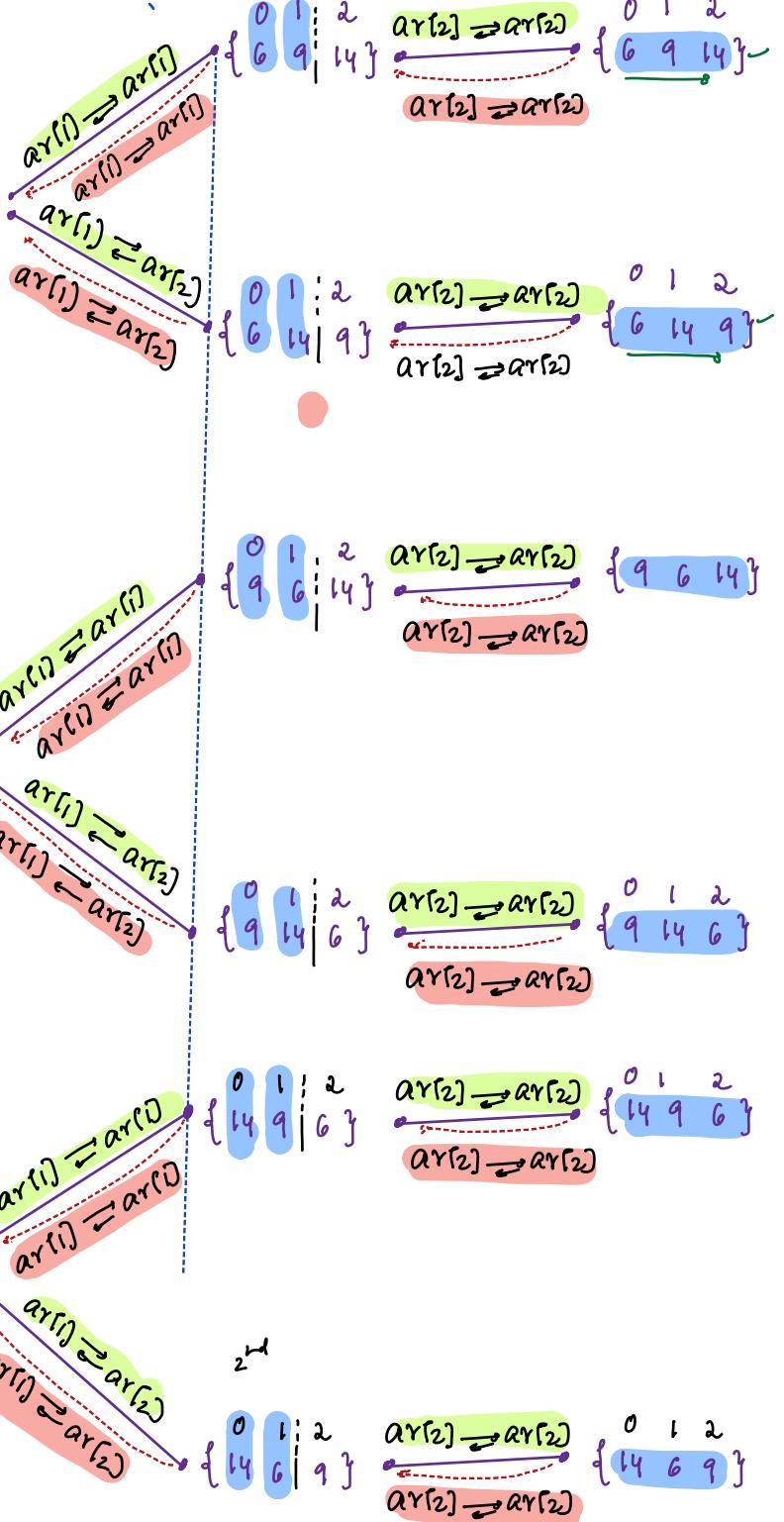
٩٥

1

1

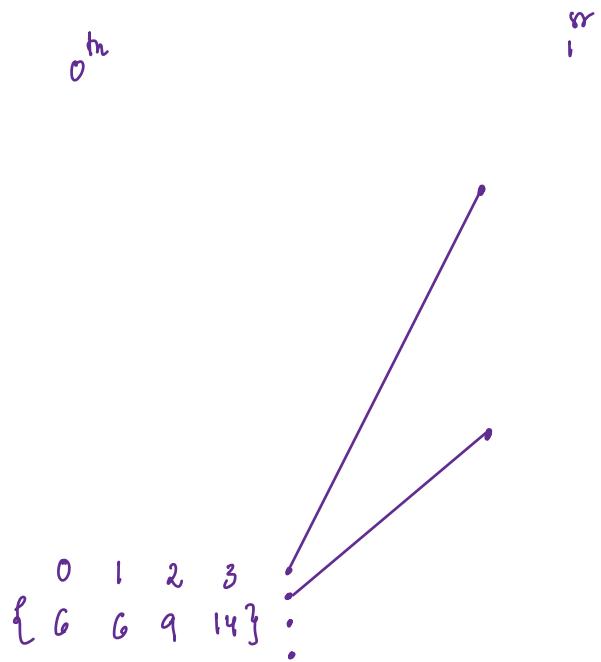
1

2



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

$$\text{arr}[4] = \{ 6 \ 6 \ 9 \ 14 \}$$



$$ar[4] = \{6, 6, 9, 14\}$$

$$\{6:2, 9:1, 14:1\}$$

0 1 2 3

