Output

1)
```
int magic(int N) {
    if (N == 0)  return 0
    else return magic(N/2) * 10 + (N % 2)
}
```

TC = O(log(N))

N → N/2 ...

```
magic(7) {
    return magic(3) * 10 + 1        // 11 * 10 + 1 = 111
    magic(3) {
        return magic(1) * 10 + 1    // 1 * 10 + 1 = 11
        magic(1) {
            return magic(0) * 10 + 1    // 0 * 10 + 1 = 1
            magic(0) { return 0 }
        }
    }
}
```

SCROLL 0

2)
```
fun(char s[], int x) {
    print(s)
    char temp
    if (x < s.length/2) {
        Temp = s[x]
        s[x] = s[s.length - x - 1]      } swap(x, s.length - x - 1)
```

```
        s[s.length - x - 1] = temp
        fun(s, x+1)
    }
}
```

x

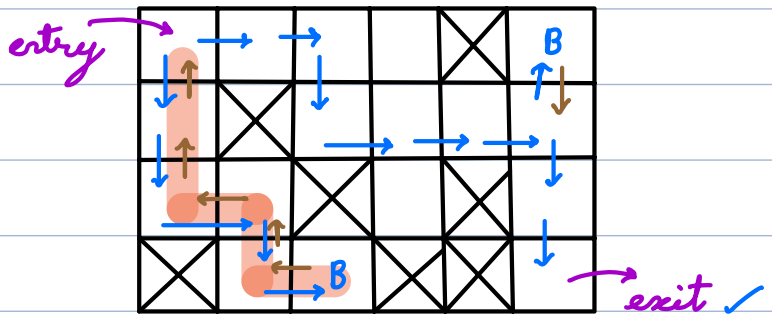
0	1	2	3	4	5
~~S~~	~~E~~	~~E~~	~~T~~	~~L~~	~~L~~
L	L	O	R	C	S

N/2 = 3

TC = O(N)

Backtracking

↳ try all possibilities using recursion.



blocked

↳ restart ✗

↳ take a step back & try alternate choice.

Q→ Given an integer A, generate all valid parenthesis '(' ')' pairs of length 2A.

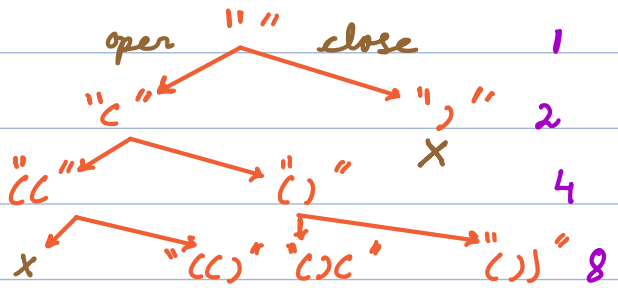
Traveling left to right,

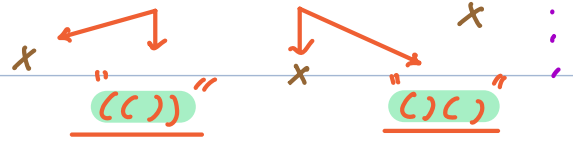
at all points # close ')' <= # open '('

Overall # close ')' = # open '('

A = 1 ()

A = 2





A = 3 "((()))" "(()())" "(())()" "()(())" "()()()"

```
void solve (str, N, oper, close) {
    if (str.length == 2*N) { // Base case
        print(str)
        return
    }
    if (oper < N)   solve(str + '(', N, oper+1, close)
    if (close < oper) solve(str + ')', N, oper, close+1)
}
```

("" , 0 , 0)

TC <= $O(2^N)$ SC = $O(N)$

Subarray → continuous part of array.

Subsequence → Sequence is an array after removing some elements, keeping the order of remaining elements same.

A = [1 2 3 4 5 6 7] (0 1 2 3 4 5 6)

[1 3 4] ✓

[1 4 3] ✗

Subset → Set of some array elements.

[1 2] ✓

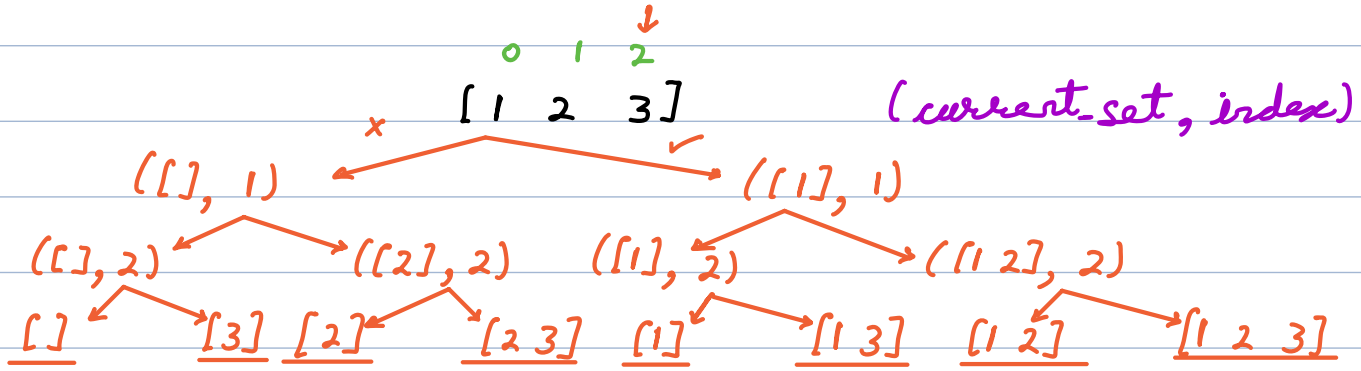
[4 2] ✓

Q → Given an array with distinct elements.
Print of possible subset / subsequence.

A = [1 2 3]

[] [1 2]
[1] [1 3]
[2] [2 3]
[3] [1 2 3]

subsets = 2^N

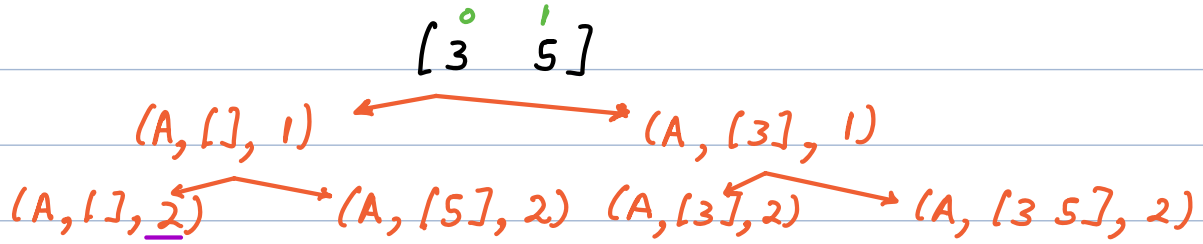


input [] 0

```
void subset ( A[], cur[], idx) {
    N = A.length
    if (idx == N) {  // Base case
        print (cur)  // ans.add(cur)   copy of cur to add
        return
    }
    subset (A, cur, idx+1)  // exclude

    cur.add (A[idx])  // Do
    subset (A, cur, idx+1)  // include
    cur.remove(cur.length()-1)  // Undo
}
```

(A, [], 0)



ans → [ ]

[5]

[3]

[3 5]

array → call by reference
int → call by value

TC = O(2^N) SC = O(N)

#permutations with unique characters = N!

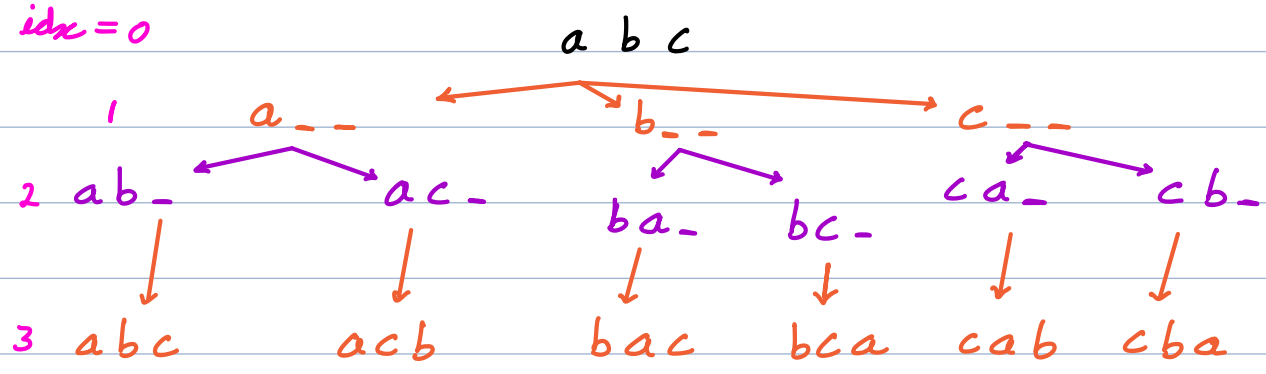
5 * 4 * 3 * 2 * 1 = 5!

5 balls

Q→ Given an array with distinct elements, print all permutations (without updating input).

Eg → a b c

abc bac cab
acb bca cba



```
void permute(A[], idx, ans[], vst[]) {
    N = A.length
    if (idx == N) {                    // Base case
        print(ans)
        return
    }
    for i → 0 to (N-1) {               // All possibilities
        if (!vst[i]) {                 // Valid possibility
            vst[i] = true
            ans[idx] = A[i]            } // Do
            permute(A, idx+1, ans, vst) // Recursion
            vst[i] = false             // Undo
        }
    }
}
```

TC = O(N!) SC = O(N)