Recursion in arraylists {Get}

1) Get subsequence
2) Get KPC
3) Get Stair paths
4) Get Maze paths
5) Get Maze path with Jumps

Previous classes

R&B 1: Intro
R&B 2: Arrays
R&B 3: Tower of Hanoi, DSA projects
Bin. Search ① & ②
Resume & LinkedIn profile building

Get:→ we need to return to main.

① Get Subsequences:

for n elements ⟹ 2^n subsets

Previously, we used bit representation to print.

Eg: {1 2 3} ⟹ {"", "1", "2", "3", ... "1,2,3"}

Level ⟹ element/item ; Level ⇒ parameters

Eg: {10, 20, 30, 40}

idx 0 1 2 3

Level ↓

getSubseq(idx, arr)

(>n) 4 level 5 → Base case
3 level 4 → 40
2 level 3 → 30
1 level 2 → 20
0 level 1 → 10
idx

if get type of question, return Arlst <>
else return nothing.

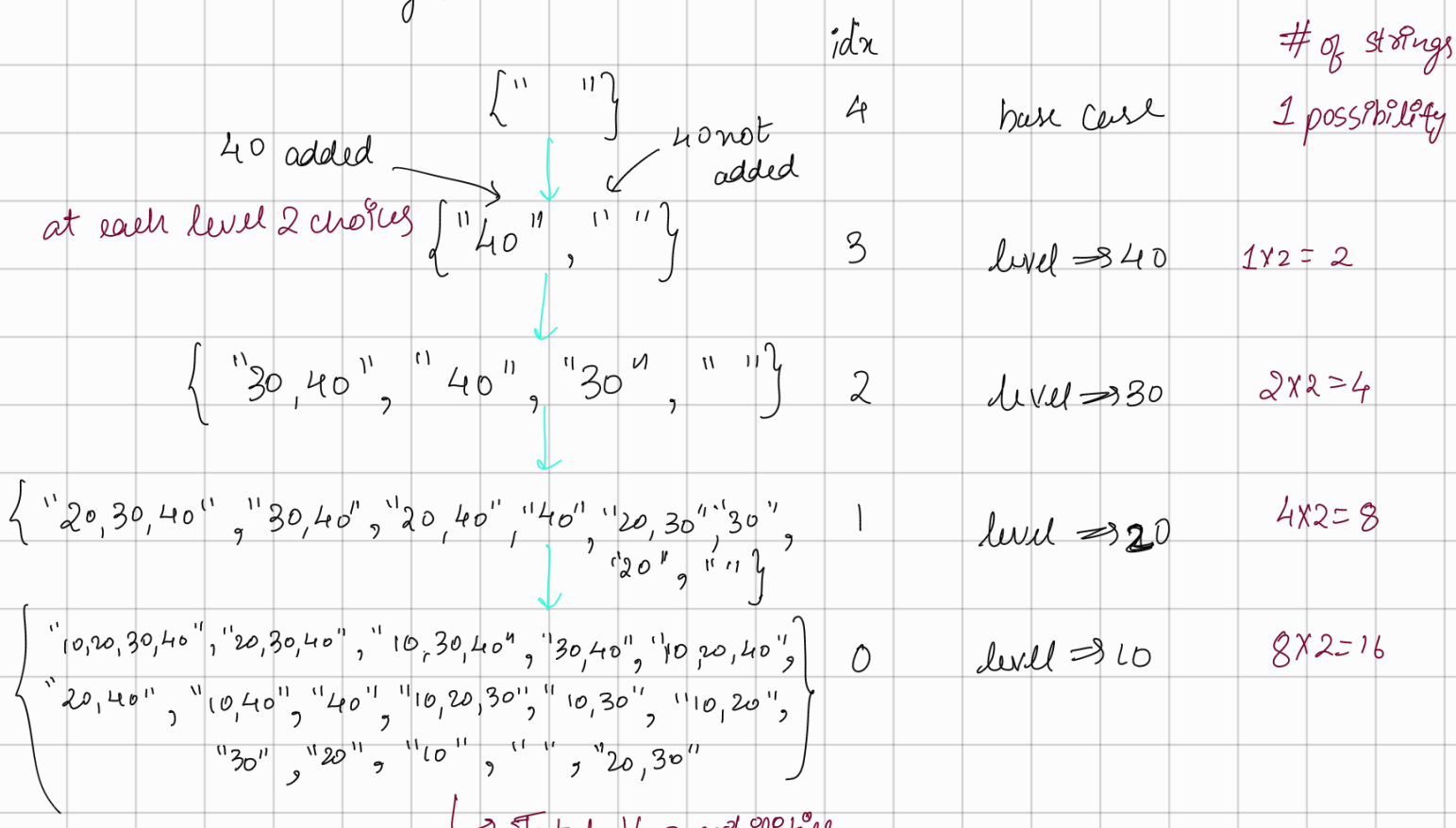
∴ a) Faith: Have faith on recursion to give Subseq. from idx+1 to arr. length.

A<S> getSS(idx, arr); // Expectation
A<S> getSS(idx+1, arr); // Faith

When idx == arr.length, return base case Arraylist of empty string

for empty list, only one possible subseq. - ie - {""}
So in basecase, return {""} // Empty ArrayList.
↑ size is not zero, but 1.

Now moving down in levels as



↳ Total 16 possibilities.

Complexity of making new list & adding in String will be same

☆ In get Subsequence category the smallest problem always lies in the base case

∴ CODE:

```
if(i == str.length()) {          // Base Case
    ArrayList<String> base = new ArrayList<>();
    base.add("");                // Add Empty String in base case
    return base;
}
// Faith
ArrayList<String> smallAns = gss(i+1, str);   // Faith
ArrayList<String> ans = new ArrayList<>();
```

```
for(String s: smallAns)
        ans.add(s);
for(String s: smallAns)
        ans.add(str.charAt[i] + s);
return ans;
```

} Faith meets expectation
work for calculating
answer from index
i+1 to i.

② Get KeyPad Combination.

0 → .; , 1 → abc , 2 → def . . . 9 → yz

Eg: 12 → {"ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"}

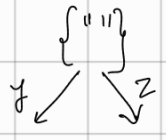
649

649 ← 49 ← 9 ← empty (base case)

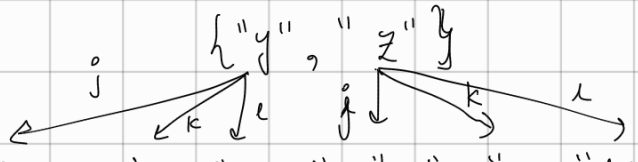
idx 0 idx 1 idx 2 idx 3

for base case, return ArrayList with a single empty string

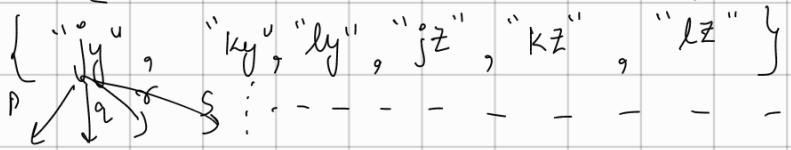
idx = 3



idx = 2 ⇒ "9"



idx = 1 → "4"



idx = 0 → "6"

→ for all 4 possibilities → p, q, r & s

total size = (size of "9" string) × (size of "4" string) (size of "6" string).

= 2 × 3 × 4

= 24

∴ CODE

```
if(i == str.length()){
    ArrayList<String> base = new ArrayList<>();
    base.add("");
    return base;
}

//Faith
ArrayList<String> smallAns = getKPC(i+1, str);

// Expectation
ArrayList<String> ans = new ArrayList<>();
for(Character letter : dtoc[str.charAt(i)-'0'].toCharArray())
    for(String s: smallAns)
        ans.add(letter+s);

return ans;
```

☆ Static member functions can only access static data members
☆ Non static member function can access both static as well as non static data members

③ Get Stair Path. <Fundamental Question>

CLIMB STAIRS

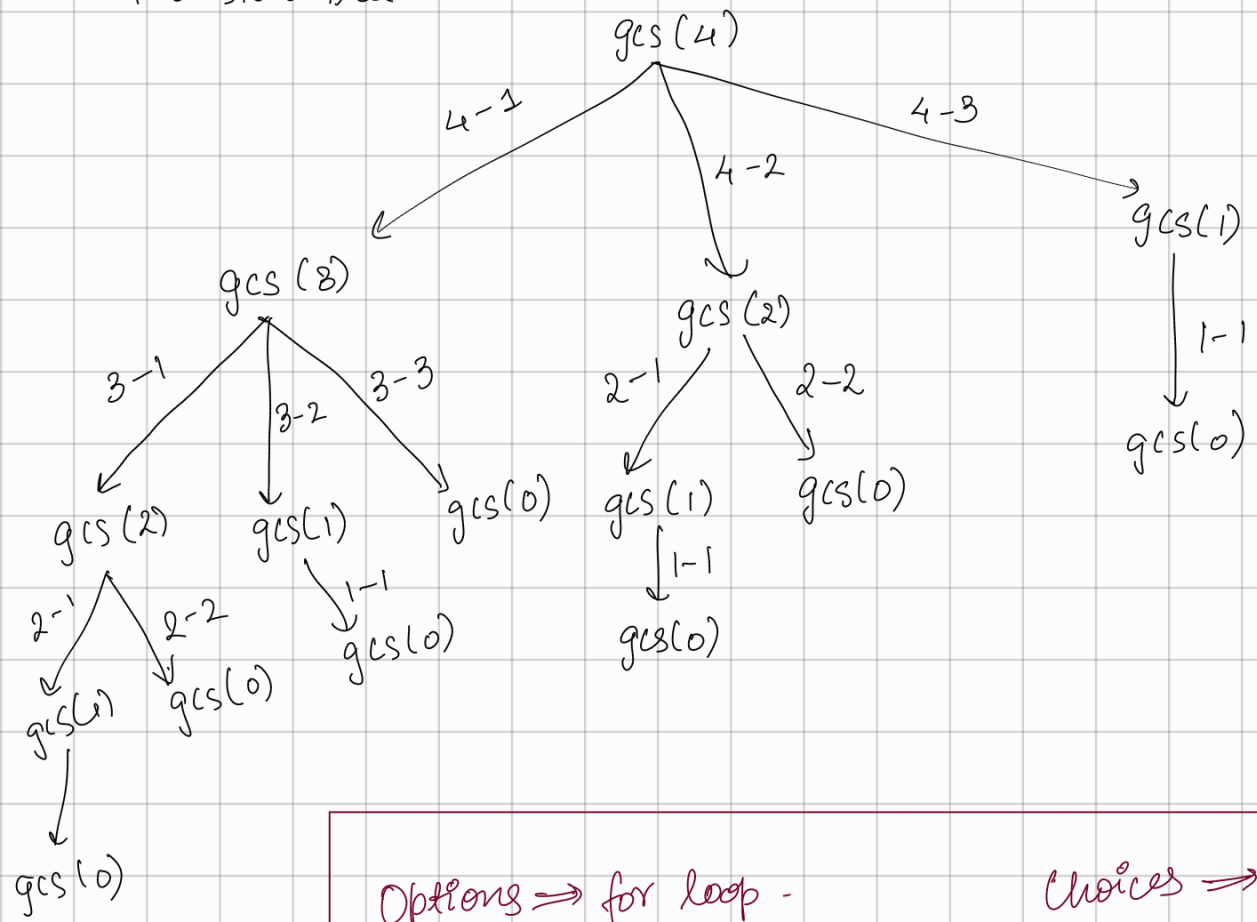
Print no. of ways to climb n stairs in 1/2/3 climbs per step

A<S> gcs(n)

Source = n = 4

destination ⇒ 0

Recursion Tree



Options ⇒ for loop - Choices ⇒ levels

∴ CODE:

```
if(n==0){
    ArrayList<String> base = new ArrayList<>();
    base.add("");
    return base;
} else if(n<0){
    ArrayList<String> base = new ArrayList<>();
    return base;
}

// faith.
ArrayList<String> smallAns1 = getStairPaths(n-1);
```

```
ArrayList<String> smallAns2 = getStairPaths(n-2);
ArrayList<String> smallAns3 = getStairPaths(n-3);

// Expectation
ArrayList<String> ans = new ArrayList<>();
for(String s: smallAns1) ans.add('1' + s);
for(String s: smallAns2) ans.add('2' + s);
for(String s: smallAns3) ans.add('3' + s);

return ans;
```

To avoid negative calls, check for condition before calling subproblem