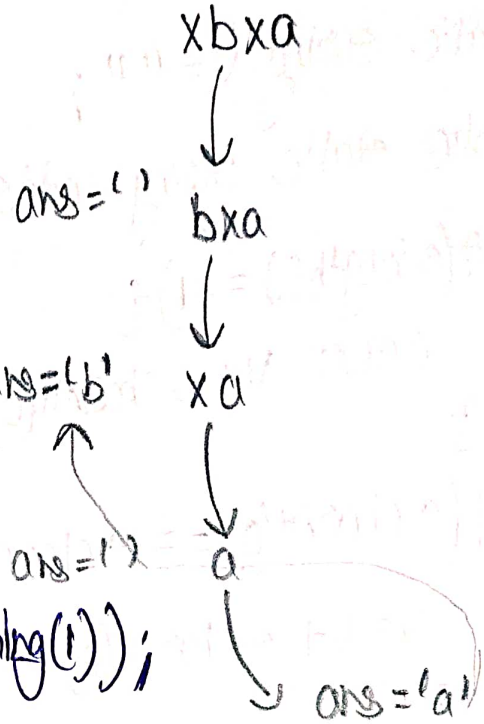
Recursion and String

Remove X

```
{
  Public static string removeX(String str){
    if(str.length()==0){
      return str;
    }
    String ans="";
    if(str.charAt(0)!='x'){
      ans = ans + str.charAt(0);
    }
    String smallAns = removeX(str.substring(1));
    return ans + smallans;
  }
```



Replace Character Recursively → YOU NEED TO REPLACE occurence of character c_1 with character c_2.

```
{
  static String str = "";
Public static String replace(String input, char c1, char c2)
{ if(input.length()==0){
    return str;
  }
  if(input.charAt(0)==c1){
    str = str + c2;
  }
  else{str = str + input.charAt(0);}
return replace(input.substring(1),c1,c2); }}
```

abacd
a x
↓
xbxcd

```
{
    if(input.length==0){
        return "";
    }
    char c;
    if(input.charAt(0)==c1){
        c=c2;
    }
    else{
        c= input.charAt(0);
    }
    String ans = replace(input.substring(1),c1,c2);
    return (c+ans);
}
}
```

Remove Duplicate Recursively

```
# {
Public static String removeConsecutiveDuplicates(String s){
    if(s.length()==1)
    { return s;
    }
    String ans = "";
    if(s.charAt(0) != s.charAt(1)){
        ans = ans + s.charAt(0);
    }
String smallAns = removeConsecutiveDuplicates(s.substring(1));
return ans+smallAns;
}
```

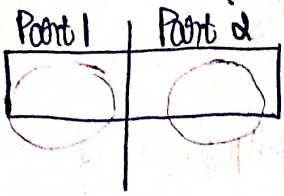
aabccba → abccba

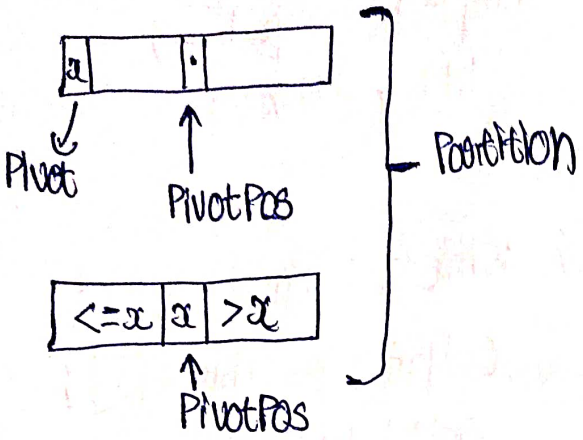
```
{
  static string v="a";
Public static string removeConsecutiveDuplicates(string s){
  if(s.length()==1){
    return s;
  }
if(s.charAt(0)==s.charAt(1)){
    return removeConsecutiveDuplicates(s.substring(1));
  }
  else{
    v= removeConsecutiveDuplicates(s.substring(1));
    return s.charAt(0)+v; ⟶ WRNG RCV STRINGS KRA GAA.
    }
}
}
```

Merge Sort — X — (DIVIDE and conquer technique)

#
```
Public static void mergeSort(int[] input) {
  if(input.length<=1) {
    return;
  }
  int mid = input.length/2;
  int part1[] = new int[mid];
  int part2[] = new int[input.length-mid];
  for(int i=0; i<mid; i++) {
    Part1[i] = input[i];
  }
  int k=0;
  for(int i=mid; i<input.length; i++) {
    Part2[k] = input[i];
    k++;
  }

  mergeSort(part1);
  mergeSort(part2);
  merge(part1, part2, input)
}

Public static void merge(int[] input1, int input2[], int[] output) {
int i=0, j=0, k=0;
while(i<input1.length && j<input2.length) {
  if(input1[i] < input2[j]) {
    output[k] = input1[i];
    k++; i++;
  }
else {
    output[k] = input2[j];
    j++;
    k++;
  }
while(i<input1.length) {
  output[k] = input1[i];
    k++; i++;
}

while(j<input2.length) {
  output[k] = input2[j];
    j++; k++;
}
}
```

Quick SORT (ALGORITHM)






```
void quicksort(input[], si, ei)
{ //Base case  if(si >= ei)
                return;
  int pivotPos = partition(input, si, ei)
  quicksort(input, si, pivotpos-1);
  quicksort(input, pivotpos+1, ei);
}

Partition(input, si, ei) {
  // Find Pivot
  // Place Pivot at its posn.
  // Ensure towards left  <=x
     towards right >x
  // return pivot pos
}
```

10	18	19	9	2	6	11

Pivot = Input[si]

count = 3 (How many no are smaller than Pivot.)

9	18	19	10	2	6	11

Pivotpos = si + count

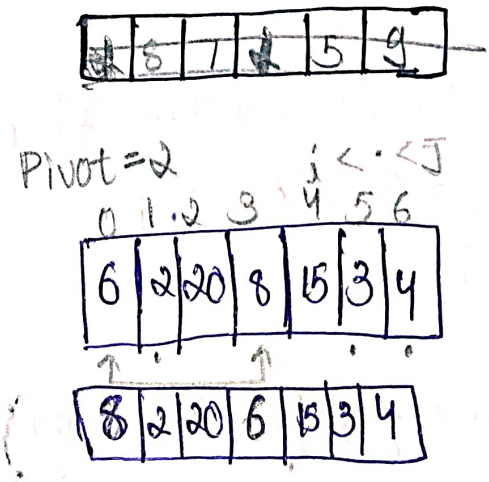
//swap [si] <--> [si + count]

—— Two Pointer concept

while(i < pivotpos && j > pivotpos)

#

```
Public static void quicksort(int[] input) {
    quicksort(input, 0, input.length-1);
}

Public static void quicksort (int[] input, int si, int end) {
if(si>=end) {
    return;
    }
    int pivotpos = partition(input, si, end);
    quicksort(input, si, pivotpos-1);
    quicksort (input, pivotpos+1, end);
}
Public static int partition(int[] input, int si, int end) {
    int pivot = input[si];
    int count = 0;
    for(int j = si+1; j<=end; j++) {
        if(input[j] <= pivot) {
            count++;
        } }
    int pivotpos = si+count;
    int temp = input[si];
    input[si] = input[si+count];
    input[si+count] = temp;
int i = si, j = end;
while(i<pivotpos && j>pivotpos) {
if(input[i]<=input[pivotpos]) { i++; }
if(input[j] > input[pivotpos]) { j--; }
if(input[i]>input[pivotpos] && input[j]<=input[pivotpos]) {
    int temp = input[i]; input[i] = input[j]; input[j] = temp;
} }
return pivotpos;
```



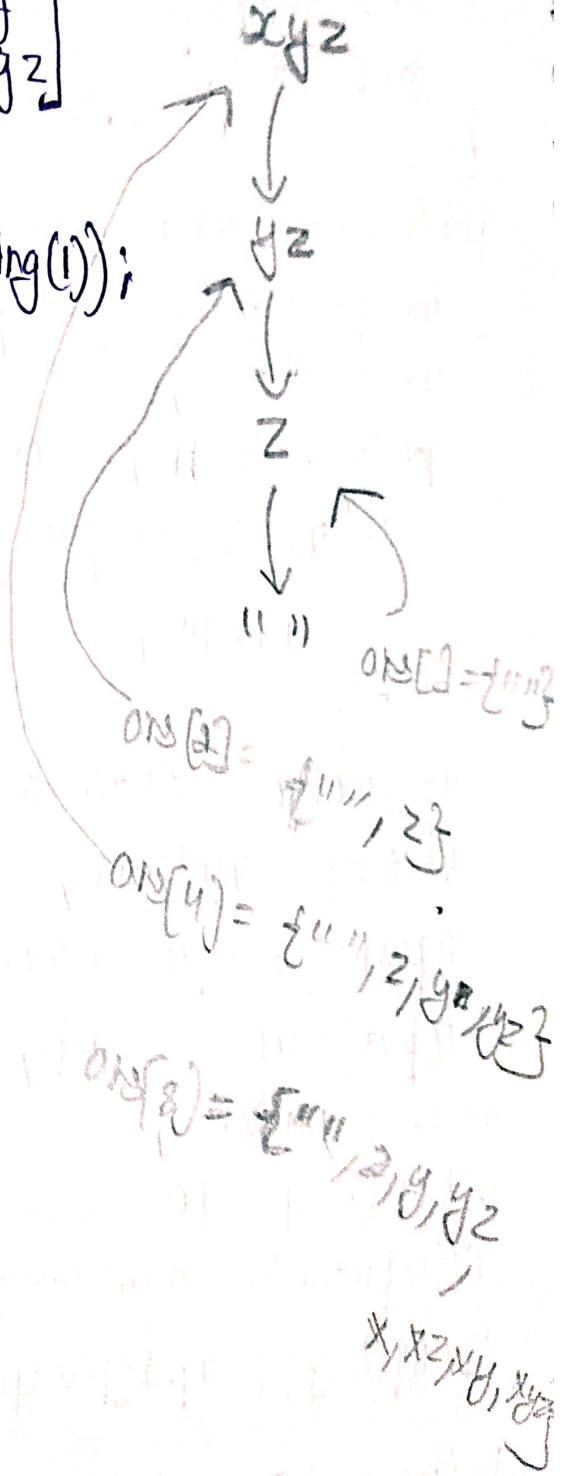
Return Subsequences

xyz ⟶ 'yz'

```
#
Public class subsequences {
Public static string[] findSubsequences(string str)
{
  if(str.length()==0){
    string ans[] = {""};
    return ans;
  }

string smallAns[] = findSubsequences(str.substring(1));
string ans[] = new string[2* smallAns.length];

int k=0;
for(int i=0; i<smallans.length ; i++) {
   ans[k] = smallAns[i];
   k++;
}

for(int i=0; i<(smallAns.length);i++) {
   // ans[i+smallAns.length]
  ans[k]= str.charAt(0) + smallAns[i];
   k++;
}
return ans;
}
```

2
""
z
y
yz
x
xz
xy
xyz

""
z
y
yz



Return keypad

2- a,b,c
3- d,e,f
4- g,h,i

234

3X3X3
=27
ways

Base case

n==0 {
return
}

2 → abc
3 → def
4 → ghi
5 → jkl
6 → mnop
7 → qrst
8 → uv
9 → wxyz
0 → .;
1 → ?!

#

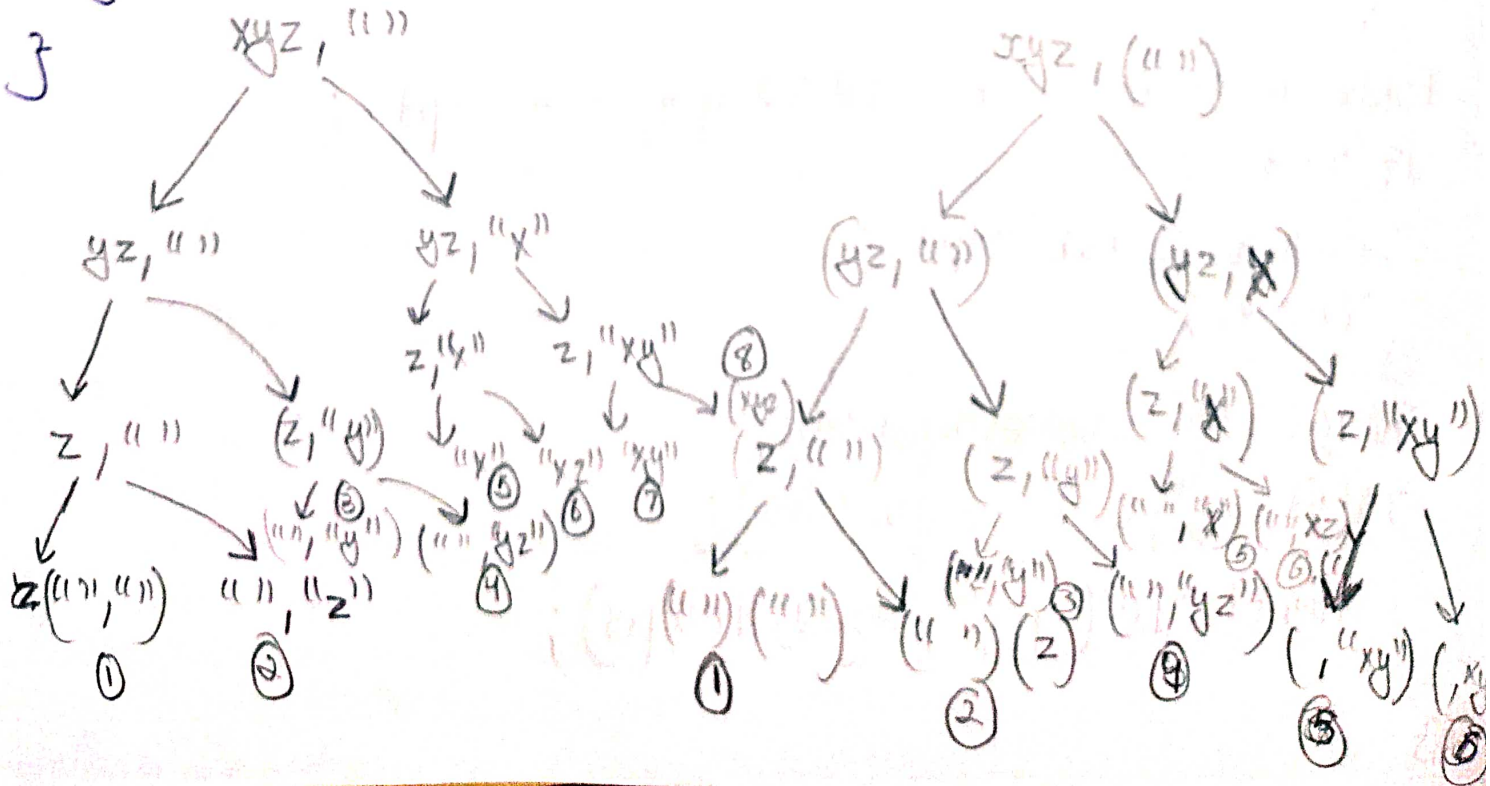
```
Public class keypad combinations {
Public static char[] numberOptions(int n){
switch(n){
 case 2:
  char options2[] = {'a','b','c'};
  return options2;
 case 3:
   char options3[] = {'d','e','f'};
   return options3;
 case 4
   char options4[] = {'g','h','i'};
   return options4;
 default:
   char options[] = {'\0'};
   return options;
  }
}
```

```
Public static keypad. String[] returnkeypad(int n) {
    if(n==0) {
        String ans[] = {""};
        return ans;
    }
    int lastDigit = n % 10;
    int smallerNumber = n/10;
    String smallAns[] = returnkeypad(smallerNumber);
    char options[] = numberOptions[lastDigit];
    String ans[] = new String[smallAns.length * options.length];
    int k=0;
    for(int i=0; i<smallAns.length; i++) {
        for(int j=0; j<options.length; j++) {
            ans[k] = smallAns[i] + options[j];
            k++;
        }
    }
    return ans;
}

Public static void main(String[] args) {
    int num = 234;
    String[] ans = returnkeypad(num);
    for(int i=0; i<ans.length; i++) {
        System.out.println(ans[i]);
    }
```

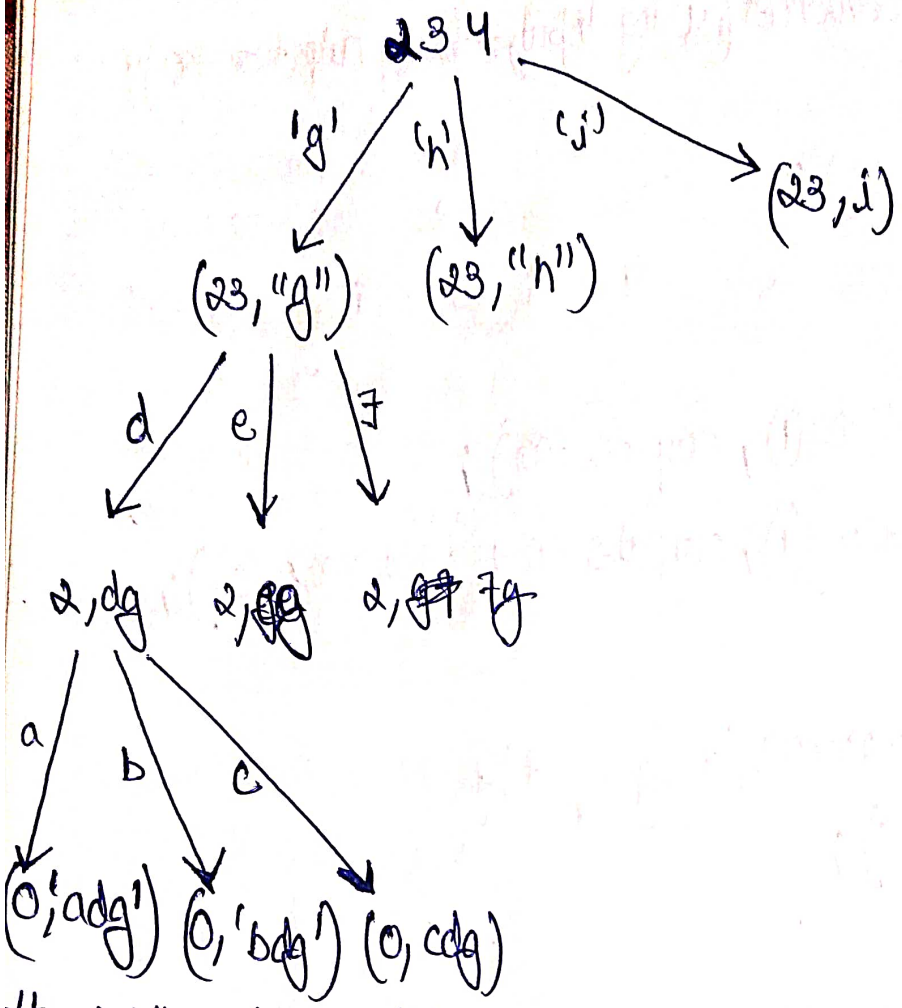
{ — RETURN TYPE DENA JARRURI H
return {""}; — ERROR DAGA
}

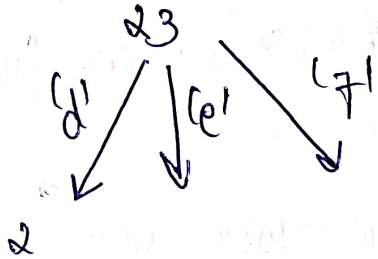
Print Subsequence of a String

```
Public class PrintSubsequences {
  Public static void PrintSubsequences(String input, String outputSoFar){
  If(input.length()==0){
   System.out.println(outputSoFar);
   return;
  }

  PrintSubsequences(input.subString(1), outputSoFar);
  PrintSubsequences(input.subString(1), outputSoFar + input.charAt(0));
}

  Public static void printSubsequences(String input){
    PrintSubsequences(input, "");
  }

  Public static void main(String[] args){
    PrintSubsequences("xyz");
  }
}
```



Print keypad combinations






```
# Public class solution {
    Public static String[] values(int n) {

    }
Public static void PrintKeyPad(int input, String output) {
 if(input==0) {
  System.out.println(output);
   return;
 }
String[] key = values(input%10);
for(int i=0 ; i<key.length ; i++) {
   PrintKeyPad(input/10, key[i] +output);
 }
}
```