Given an integer `n`, return all the numbers in the range `[1, n]` sorted in lexicographical order.

You must write an algorithm that runs in `O(n)` time and uses `O(1)` extra space.

Example 1:

```
Input: n = 13
Output: [1,10,11,12,13,2,3,4,5,6,7,8,9]
```

Example 2:

```
Input: n = 2
Output: [1,2]
```

lexicographical order (in numbers) mean → 1 will come first, then 2, then 3 & so on ....

Ex : 1, 10, 11, 12, 13, 2, ...

Thought Process

n = 13. limit = [1, 13]

1

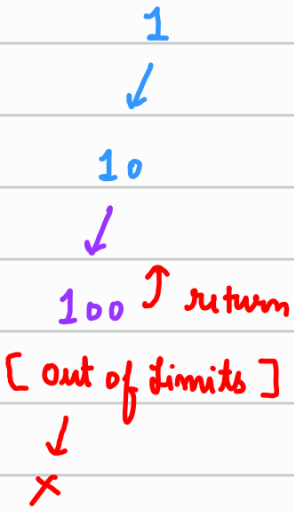
print { 1,

1

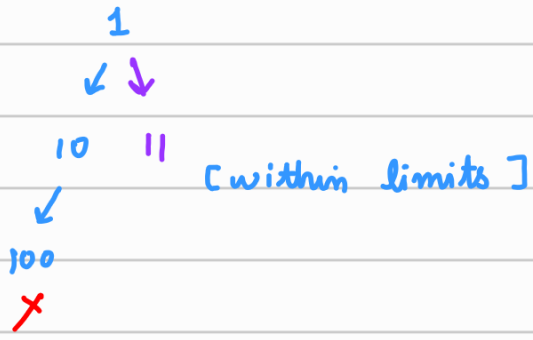
10

within limits

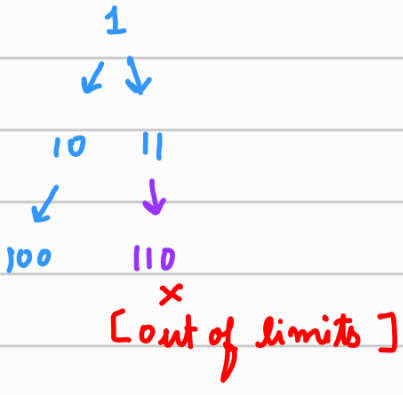
print { 1, 10,

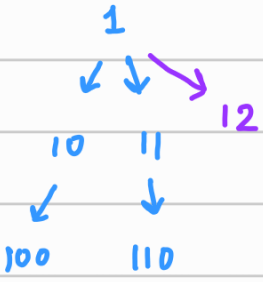


out of limits, no need to go further, return.
∴ will go till 2 digits, making it 3 digits makes it 00 b.

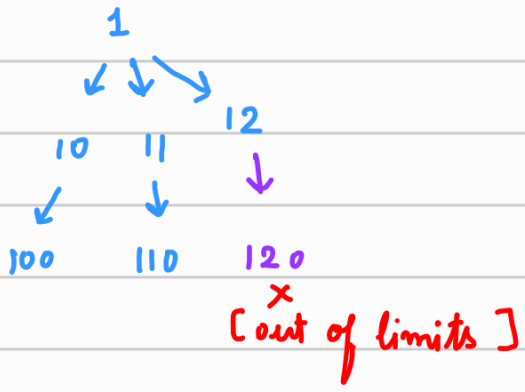


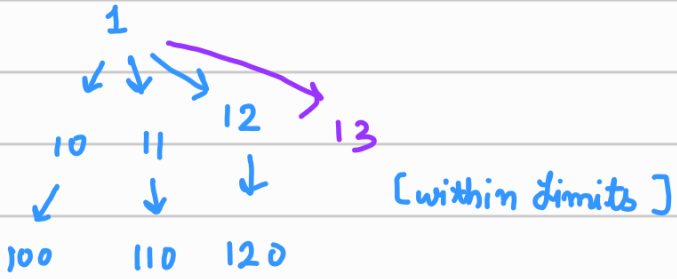
print { 1, 10, 11,



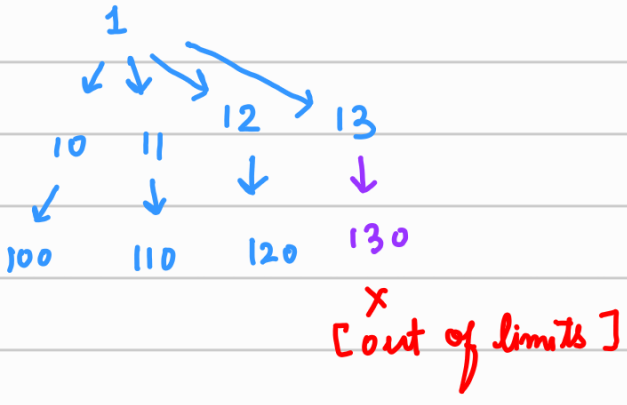


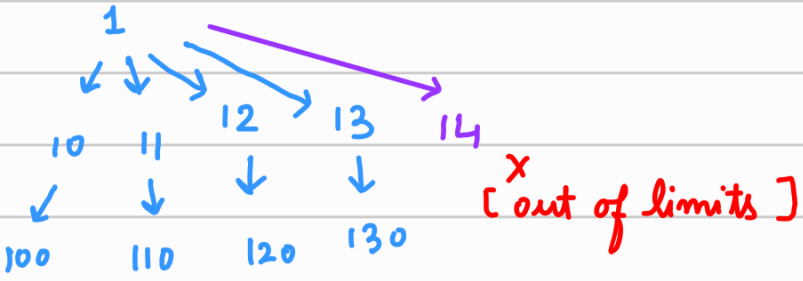
print { 1, 10, 11, 12,





print : {1, 10, 11, 12, 13,



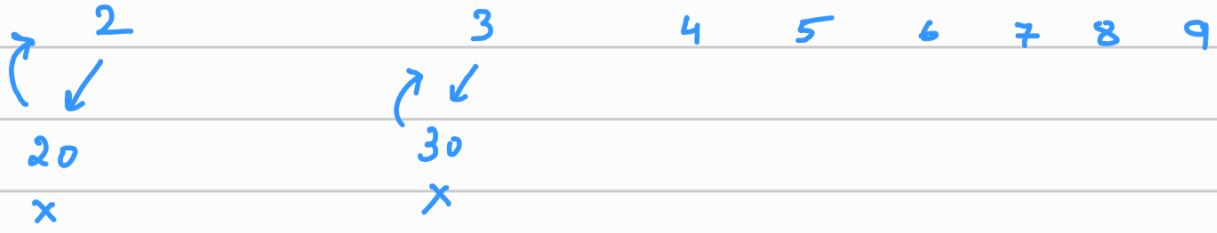


- we are going to as dipth as possible & then returning & doing the same again.

we found out nos starting with 1 in lexicographical

order.

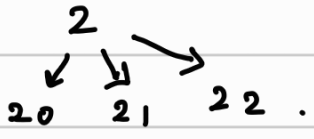
Similarly, we will do for all other nos (2, 3, 4 ....)
(within limits)



will go till 9, 10 already covered in 1.

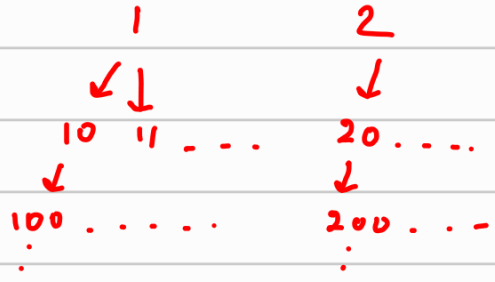
Ex : Limit → [1, 22]

22 will already have been covered in 2.



∴ starting fts 1, 2, 3 9
all the other numbers will be covered in this only
[∵ starting ft will go till 9]

Ex :



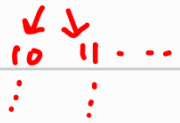
This way all the other nos will be covered in 1 ..... 9 only.

Story to code

```
for ( startNum = 1 ; startNum <= 9 ; startNum++ ) {

    Solve ( startNum , n , result ) ; // for every starting no we will call dfs & solve to its depth.
```


```
Solve ( int currNum , int n , vector<int> result ) {

    if ( currNum > n ) // out of limit.
        return ;

    // else
    result.push_back ( currNum ); // put the no in result same to as we were printing valid no in dry run.
```

// Ex : we got 1, now we will see what we can get with 1 → , similarly → 2, 3, 4.....9




```
for ( int append = 0 ; append <= 9 ; append++ ) {

    int newNum = ( currnum * 10 ) + append ;
```

// Ex : 1 * 10 + 0 = 10

1 * 10 + 1 = 11

```
if ( newNum > n ) return;   // if new num is out of limit
// else
    solve ( newNum, n, result); // call recursion to go further deep down.
```

Ex: 1
↓
10
↓
100
↓
⋮

if we added 10 as new no to result go further down. (depth) and see how much more ans we can find

TC: $O(n)$

ultimately we are visiting all the numbers.
& total nos are n (as range is till n)
∴ $O(n)$

SC: $O(\text{no of digits in } n) = O(\log_{10} n)$

no extra space taken, recursion → recursive stack space used.

Ex: n = 13 [1, 13]

1
↓
10
↓
100 ✗ invalid

we went till depth of 2 (before it was invalid)

no of digit in limit = 2, dept = 2.
(limit = 13)

depth depends on no of digit (in n)

CODE

```cpp
//Approach (Simple Recursion - DFS)
//T.C : O(n) - We visit each number (1 to n) only once.
//S.C : O(d) - where d is the number of digits in n i.e. log10(n)
class Solution {
public:

    void solve(int curr, int n, vector<int>& result) {
        if(curr > n)
            return;

        result.push_back(curr);

        for(int nextDigit = 0; nextDigit <= 9; nextDigit++) {
            int nextnum = curr*10 + nextDigit;

            if(nextnum > n)
                return;

            solve(nextnum, n, result);
        }
    }

    vector<int> lexicalOrder(int n) {
        vector<int> result;

        for(int num = 1; num <= 9; num++)
            solve(num, n, result);

        return result;
    }
};
```