22 February 2025 10:02

402. Remove K Digits

Solved

Medium | Topics | Companies

Given string num representing a non-negative integer `num`, and an integer `k`, return *the smallest possible integer after removing* `k` *digits from* `num`.

Example 1:

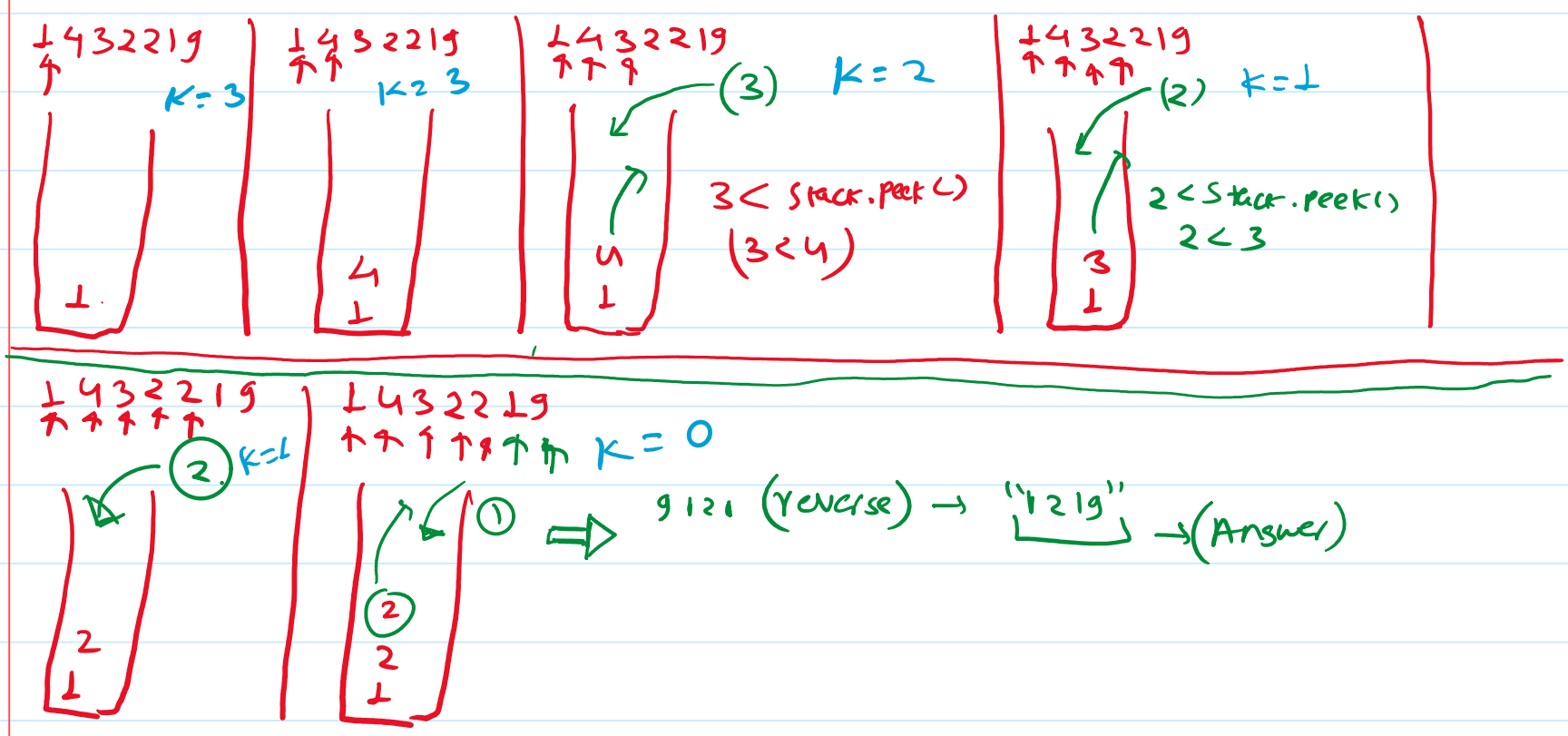
```
Input: num = "1432219", k = 3
Output: "1219"
Explanation: Remove the three digits 4, 3, and 2 to form the new number 1219 which is the smallest.
```

Example 2:

```
Input: num = "10200", k = 1
Output: "200"
```

Example 3:

```
Input: num = "10", k = 2
Output: "0"
```



---

logic - • if Number is arranged in increases then it is the smallest element

i,e 1234567 this is the Smallest Number...

Edge Case for [K]

K <= N

K == N

Direct return "0"

→ Because All digit remove

"00100"

→ "100"

→ eliminate the excess "0"

(K=3) And All element are in se order

Suppose "123456"

6
5
4
3
2
1

(Still K=3) Then eliminate the last K digit for Smallest

i,e 123456

→ "123" Ans

Code -

Way 1. जो भी stack में जाएगा तो Then Pop out from there

Code -

```
fun(string s)
{
// Create Stack
stack<char> st
for(i=0 ⟶ n-1)
{ // insert into stack with condition check
while(!st.isEmpty() && k>0 && st.Peek() > s[i])
{ st.pop();
k = k-1;
}
// And insert it
st.push(s[i]);
}
```

```
// if 9se Order Case
   while( K>0)  St.POP(), K--

// if empty return "0"
   if(st.isEmpty()) return "0";

// pop out each thing from stack
   res=" ";
   while(!St.Empty())
      {
        res = rest st.pop();
      }

// 32th Back से 0 को हटा देते
while (res.size != 0 && res.back()=='0')
        res.pop_back();
// reverse it
   reverse(res);

// if empty return —
   if(rev.empty()) return "0";

   return res;
}
```

2nd way – only pur only k Element on stack

```
public String removeKdigits(String num, int k) {
    if(k>=num.length()) return "0";
    Stack<Character> stack=new Stack<>();

    int i=0 ,n=num.length();
    while(i<n && k>0){
      while(!stack.isEmpty() && num.charAt(i)<stack.peek() && k>0)
      {stack.pop(); k--;}
        stack.push(num.charAt(i)); i++;
    }
    while(!stack.isEmpty() && k>0){
      stack.pop(); k--;
    }

    StringBuilder sb=new StringBuilder("");

    while(!stack.isEmpty()){
      sb.append(stack.pop());
    }

    sb.reverse();
    sb.append(num.substring(i,num.length()));
    while(sb.length()>0 && sb.charAt(0)=='0')
    sb=new StringBuilder(sb.substring(1));
    if(sb.length()==0) return "0";

      return sb.toString();
  }
```

TC -- O(3N) +O(k)

space complexity : 0(N) due to using the stack

Note in Dry run