leetcode

(0) 3Sum Closest

Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to target.

Return the sum of the three integers.

You may assume that each input would have exactly one solution.

```
3 <= nums.length <= 1000
-1000 <= nums[i] <= 1000
-10^4 <= target <= 10^4
```

arr :→ { 10, 40, 35, -5, 30, -10 }

target = 50

{ -10, -5, 10, 30, 35, 40 }

{-10, -5, 40} {-10, 10, 40} { -10, 30, 40} {-10, 30, 35}

25 40 60 45

{-5, 10, 40}

```java
public static int threeSumClosest(int[] nums, int target) {
  Arrays.sort(nums);

  int closestSum = nums[0] + nums[1] + nums[2];
  for(int i1 = 0; i1 < nums.length - 2; i1++){
      int i2 = i1 + 1, i3 = nums.length - 1;
      while(i2 < i3){
          int sum = nums[i1] + nums[i2] + nums[i3];

          if(sum == target){
              return target;
          } else if(sum > target){
              if(Math.abs(sum - target) < Math.abs(closestSum - target)){
                  closestSum = sum;
              }
              i3--;
          } else {
              if(Math.abs(sum - target) < Math.abs(closestSum - target)){
                  closestSum = sum;
              }
              i2++;
          }
      }
  }
  return closestSum;
}
```

(Q) Generate Valid Parantheses

N = 3

↳ Recursion & Backtracking

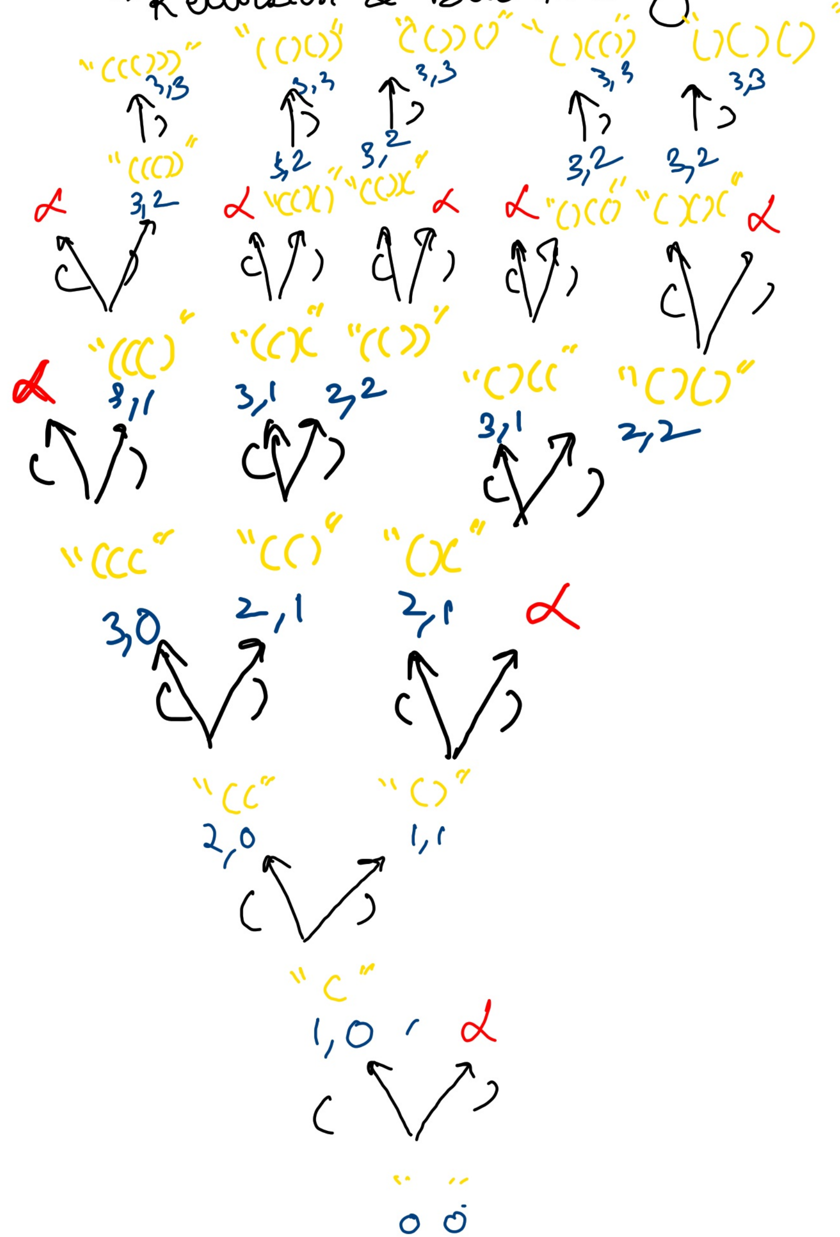
((()))

(()())

(())()

()(())

()()()

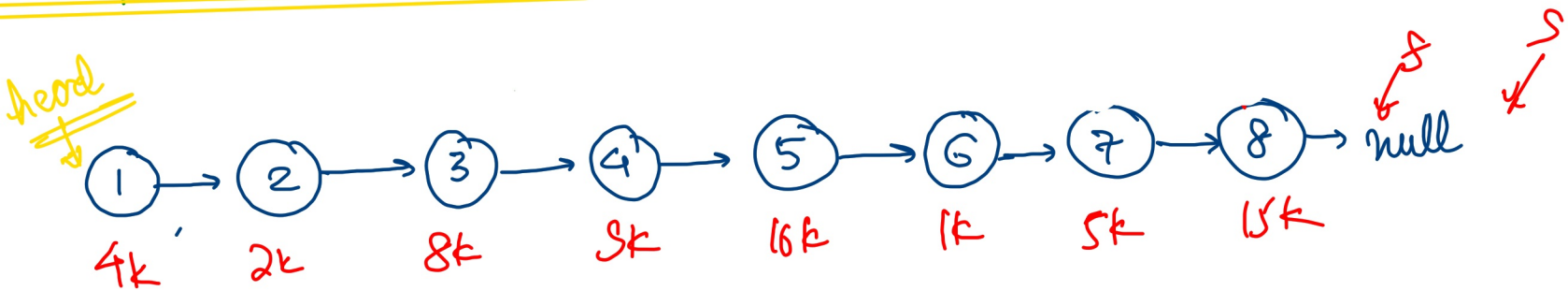


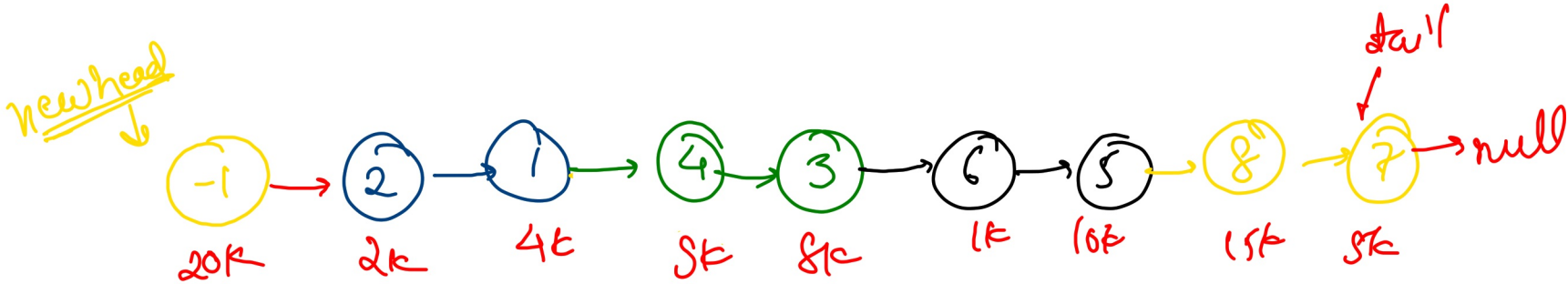
$(\text{calls})^{\text{height}}$

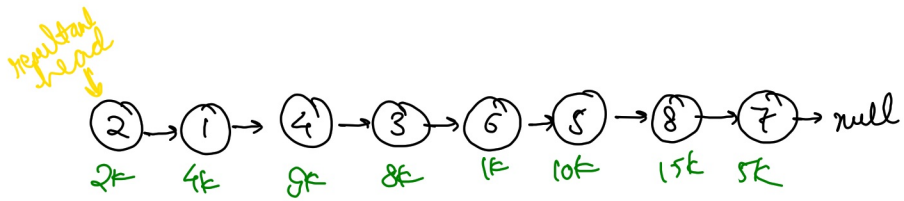
static List<String> res;

```java
public static void generateParanthesis(int n, int open, int close, String output){
    if(open == n && close == n){
        res.add(output);
        return;
    }

    // opening
    if(open < n){
        generateParanthesis(n, open + 1, close, output + "(");
    }

    // closing
    if(close < n && close < open){
        generateParanthesis(n, open, close + 1, output + ")");
    }
}
public static List<String> generateParenthesis(int n) {
   res = new ArrayList<>();
   generateParanthesis(n, 0, 0, "");
   return res;
}
```

Q Swap Nodes in Pair

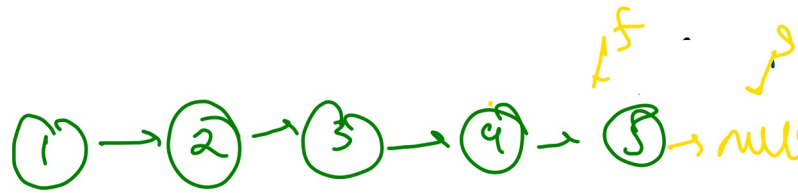






Pseudo code

① resHead = new ListNode(-1)
resTail = resHead;



② first = head;
second = head.next;

③ while (first != null) {

ListNode temp = second.next;

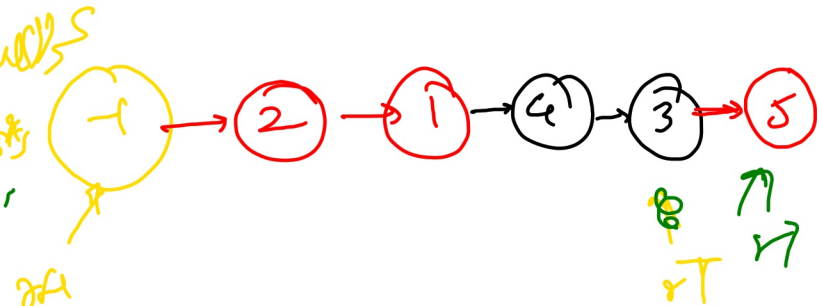
① resTail.next = second;
second.next = first;
tail = first;

② first = temp;
second = temp.next;

}

④ tail.next = null;

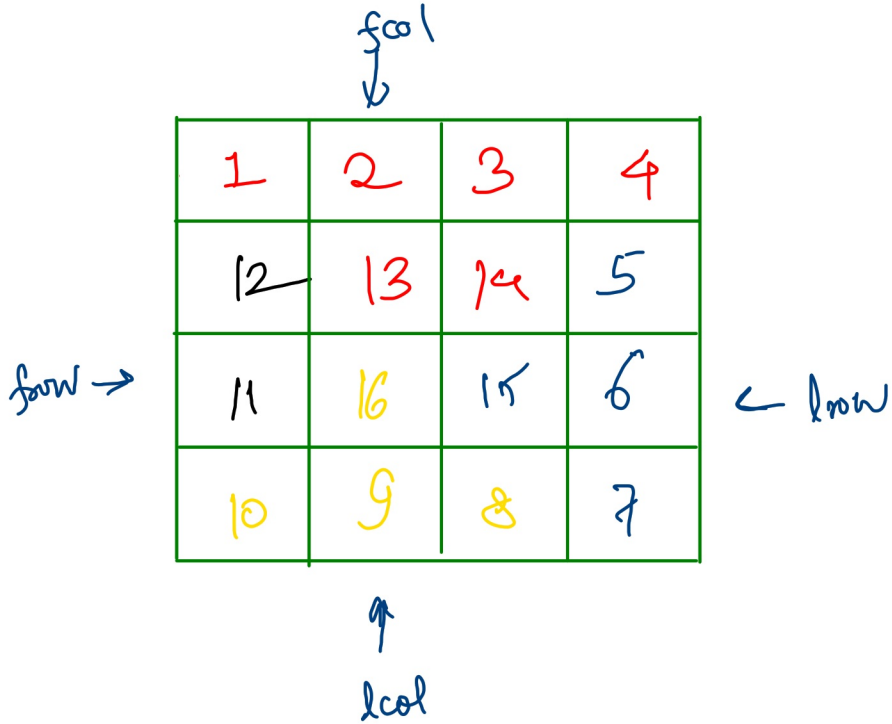
⑤ return res.head.next;



```
if(head == null || head.next == null){
  return head;
}

Node first = head;
Node second = head.next;
Node resHead = new Node(-1, null);
Node resTail = resHead;

while(first != null){
    if(first.next == null){
        resTail.next = first;
        resTail = first;
        break;
    }

    Node temp = second.next;

    resTail.next = second;
    second.next = first;
    resTail = first;

    first = temp;
    if(temp == null) second = null;
    else second = temp.next;
}

resTail.next = null;
return resHead.next;
```

Q) Spiral Matrix -II

N = 4



ele = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

1. Top wall
   left to right
   [frow][i]
2. Right wall
   Top to bottom
   [i][lcol]
3. Bottom wall
   Right to left
   [lrow][i]
4. left wall
   bottom to top
   [i][fcol]

(Q) length of last Word

1.Given a string s consists of upper/lower-case alphabets and empty space characters ' ', return the length of last word in the string.

2.If the last word does not exist, return 0.

3 Note: A word is defined as a character sequence consists of non-space characters only.

4.You have to complete the function length of the last word

```java
public static int lengthOfLastWord(String s) {
    int count = 0;
    int lastWordLength = 0;
    for(int i=0; i<s.length(); i++){
        if(s.charAt(i) == ' '){
            if(count > 0){
                lastWordLength = count;
                count = 0;
            }
        }
        else {
            count++;
        }
    }

    if(count > 0) return count;
    return lastWordLength;
}
```

code

(1) " ___ hello ___ pepcoder " : 8

(2) " hello ___ pepcoder ___ " : 8

(3) " hellopepcoder " : 13

(4) " _____ " : 0

(Q) Search Insert Position

:
- if ele found : return idx : mid
- if ele found : return idx of ceil : left

} ⇒ lower bound

	0	1	2	3	4	5	6	7	8	9	
arr : {	10,	20,	40,	60,	70,	80,	100,	120,	140,	150 }	10

1.You are given a sorted array and a target value.
2.You have to complete the function searchInsert() that should return the index if the target is found.
If not, return the index where it would be if it were inserted in order.

constraint :⇒ no duplicates

```java
public static int searchInsert(int[] nums, int target) {
    int left = 0, right = nums.length - 1;

    while(left <= right){
        int mid = left + (right - left) / 2;

        if(nums[mid] == target){
            return mid;
        } else if(nums[mid] < target){
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return left;
}
```

(0) # Count And Say

Constraint $N \leq 30$

The **count-and-say** sequence is a sequence of digit strings defined by the recursive formula:

- `countAndSay(1) = "1"`
- `countAndSay(n)` is the way you would "say" the digit string from `countAndSay(n-1)`, which is then converted into a different digit string.

To determine how you "say" a digit string, split it into the **minimal** number of groups so that each group is a contiguous section all of the **same character.** Then for each group, say the number of characters, then say the character. To convert the saying into a digit string, replace the counts with a number and concatenate every saying.

"3322251"

two 3's, three 2's, one 5, and one 1

2 3 + 3 2 + 1 5 + 1 1

"23321511"

Given a positive integer `n`, return *the* n^{th} *term of the* ***count-and-say*** *sequence.*

1. Given a sequence follows this pattern:

```
1
11
21
1211
111221
```

Where,
1 is read as "one 1" or 11.
11 is read as "two 1s" or 21.
21 is read as "one 2, then one 1" or 1211

2. You will be given a number n, you need to print the nth number of this series
3. Input and output is handled for you
4. It is a functional problem ,please do not modify main()

$CRS(2) = say(CRS(1))$

$= say("1")$

$= "11"$

$CRS(3) = say(CRS(2))$

$= say("11")$

$= "21"$

$CRS(4) = say(CRS(3))$

$= say("21") = "1211"$

$CRS(5) = say(CRS(4)) = say("1211") = "111221"$

$CRS(n) \rightarrow CRS(n-1)$, $say(str)$

```
# if (n == 1)
    return "1";
```

"33222511"

int count = 0 1 2 0 1 2 3 0 1 0 1 2

char digit = '3' '2' '5' '1'

String re = "" + 2 + '3'

= "23" + 3 + '2'

= "2332" + 1 + '5'

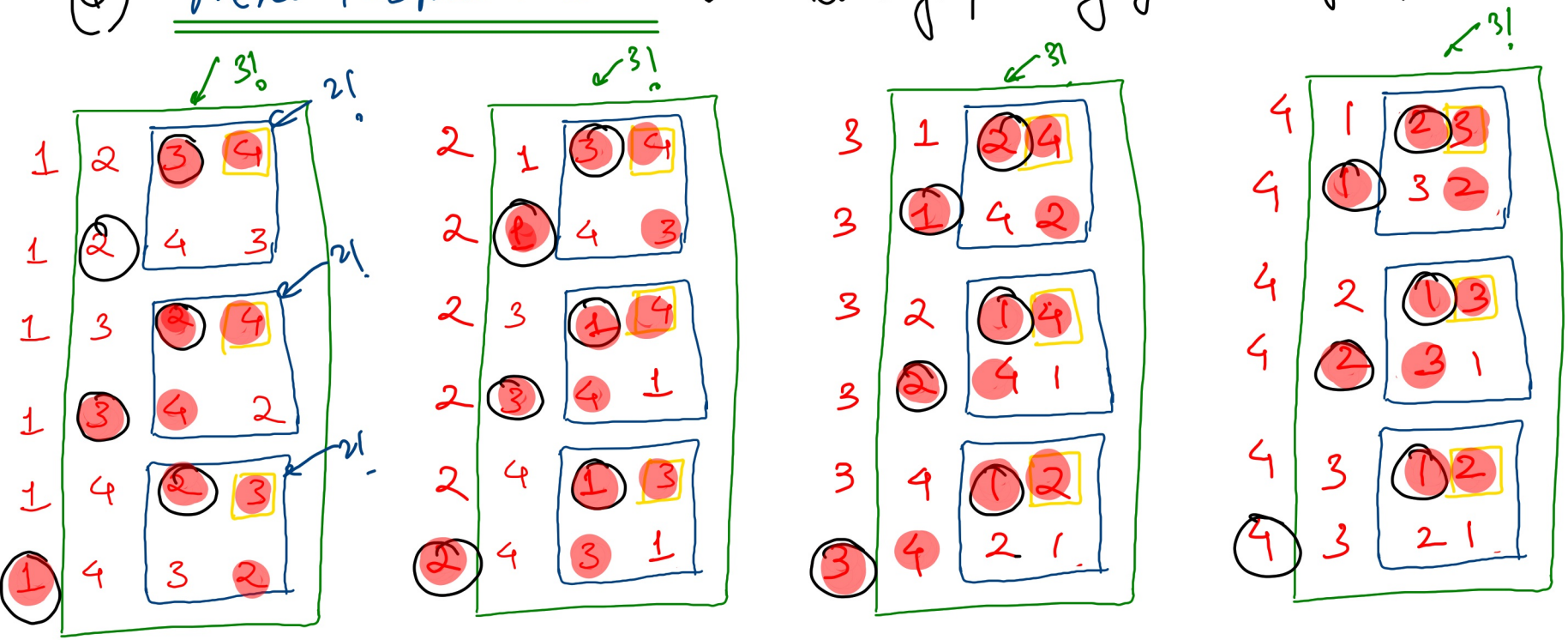
= "233215" + 2 + '1'

= "23321521"

```java
public static String say(String str){
    int count = 0;
    char digit = str.charAt(0);
    String res = "";

    for(char ch: str.toCharArray()){
        if(ch != digit){
            res = res + count + digit;
            digit = ch;
            count = 0;
        }

        count++;
    }

    res = res + count + digit;
    return res;
}
```

```java
public static String countAndSay(int n) {
    if(n == 1) return "1";
    String cnsSmall = countAndSay(n - 1);
    return say(cnsSmall);
}
```

(1) Next Permutation :→ lexicographically just larger permutation.



Algorithm

O(N) ① Find the first **DIP** element from right
- DIP: arr[i] > arr[i+1]

+

O(N) ② Replace the dip element by **just greater value**
- just greater value → in right side.

+

O(N) ③ Now, **sort the right hand** side array of dip element.
- sort the right hand → reverse the array.

O(3N) = O(N)

O(N) solution

Constraints

N < 10^5

3. The replacement must be in-place and use only constant extra memory.

→ O(1) extra space

Corner Case

If no dip element found ⇒ largest permutation

→ convert to smallest permutation

→ Sort the entire array/string

2. If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

eg: 5 4 3 2 1 ⟶ 1 2 3 4 5