这题的思路是：设 a 是 n+1 位二进制数所包含 1 的个数，b 是 n 位二进制数所包含 1 的个数，则 a=b+1 。因为 n+1 位二进制数是在对应 n 位二进制数的前面加一个 1。

因此，当我们得到 n 位二进制数所包含 1 数目的数组后，我们把数组中的每一个数加 1，便得到了 n+1 位二进制数所包含 1 数目的数组。

第一个版本的实现是这样子的：

```
class Solution(object):
    def countBits(self, num):
        """
        :type num: int
        :rtype: List[int]
        """
        if num == 0:
            return [0]

        last_list, current_list, next_list = [1], [0, 1], []
        for i in range(1, num+1):
            if len(next_list) == 0:
                next_list = last_list[:] + [x+1 for x in last_list]
                last_list = next_list[:]
            current_list.append(next_list.pop(0))
        return current_list


temp = Solution()
print temp.countBits(20)
```

第一版实现不好的地方在于**预计算了所有 n+1 位二进制数所包含 1 的个数**。因此，第二个版本的实现中不再做这种预计算。

```
class Solution(object):
    def countBits(self, num):
        """
        :type num: int
        :rtype: List[int]
        """
        result, temp_list = [0], [0]

        for i in range(1, num+1):
            if len(temp_list) == 0:
                temp_list = result[:]
            result.append(temp_list.pop(0)+1)

        return result

temp = Solution()
print temp.countBits(20)
```

第二版实现不好的地方在于**从 n 位跳到 n+1 位的时候，我们需要复制数组 result** 。实际上，我们只需要一个类似于指针的东西在 result 数组中滑动就可以了，并不需要临时数组。

```python
#!/usr/bin/env python
# coding=utf-8

class Solution(object):
    def countBits(self, num):
        """
        :type num: int
        :rtype: List[int]
        """
        result, temp_list = [0], [0]
        num1, num2 = 1, 0

        for i in range(1, num+1):
            result.append(result[num2]+1)
            num2 += 1
            if num2 == num1:
                num1 *= 2
                num2 = 0

        return result

temp = Solution()
print temp.countBits(20)

```