Counting Bits

Given a non negative integer number **num**. For every numbers **i** in the range **0 ≤ i ≤ num** calculate the number of 1's in their binary representation and return them as an array.

Example: For `num = 5` you should return `[0,1,1,2,1,2]`.

Follow up:

- It is very easy to come up with a solution with run time **O(n*sizeof(integer))**. But can you do it in linear time**O(n)** /possibly in a single pass?
- Space complexity should be **O(n)**.
- Can you do it like a boss? Do it without using any builtin function like **__builtin_popcount** in c++ or in any other language.

1. You should make use of what you have produced already.
2. Divide the numbers in ranges like [2-3], [4-7], [8-15] and so on. And try to generate new range from previous.
3. Or does the odd/even status of the number help you in calculating the number of 1s?

Credits:
Special thanks to @ syedee for adding this problem and creating all test cases.

Solution 1

An easy recurrence for this problem is f[i] = f[i / 2] + i % 2.

```java
public int[] countBits(int num) {
    int[] f = new int[num + 1];
    for (int i=1; i<=num; i++) f[i] = f[i >> 1] + (i & 1);
    return f;
}
```

written by lixx2100 original link here

Solution 2

```
class Solution {
public:
    vector<int> countBits(int num) {
        vector<int> ret(num+1, 0);
        for (int i = 1; i <= num; ++i)
            ret[i] = ret[i&(i-1)] + 1;
        return ret;
    }
};
```

written by fengcc original link here

Solution 3

```
class Solution {
public:
    vector<int> countBits(int num) {

        vector<int> res(num+1,0);

        for(int i = 1; i < res.size();i++)
            res[i] = i%2 + res[i/2];

        return res;
    }
};
```

written by m.shyamkrishnan original link here

From Leetcoder.