Sage and binary numbers

```
public int solve(int A) {
        int curr = -1;
        int prev = -1;
        int max = 0;
        for (int i = 31; i >= 0; i--) {
            int bitValue = A & (1 << i);
            if (bitValue != 0) {
                prev = curr;
                curr = i;
            }
            if (prev != -1) {
                max = Math.max(max, prev - curr);
            }
        }
        return max;
    }
```

Lucky Numbers

```
// Earlier brute force : O(n2 * root(n))
    // TC : O(nlogn)
    // SC : O(n)
    public int solve(int A) {
        int[] count = new int[A + 1];

        for (int i = 2; i <= A; i++) { // O(n)
            if (count[i] == 0) {
                for (int j = 2 * i; j <= A; j = j + i) { // O(logn)
                    count[j]++;
                }
            }
        }

        int cnt = 0;
        for (int i = 2; i <= A; i++) {
            if (count[i] == 2) {
                cnt++;
            }
        }

        return cnt;
    }
```

Chef and Cooking

```
// TC : O(n)
    // SC : O(1)
    public long solve(int[] A) {
        long cs = A[0];
        long ans = A[0];
        for (int i = 1; i < A.length; i++) {
            if (A[i] > A[i - 1]) {
                cs += A[i];
            } else {
                cs = A[i];
            }
            ans = Math.max(ans, cs);
        }
        return ans;
    }
```