1 Problem 1

1.1 Easy but inefficient

Algorithm 1 Locker(A[0..n – 1])

for i ← 0 to n – 1 do
    A[i] ← 0
end for

for i ← 1 to n do
    j ← i – 1
    while j < n do
        j ← j + i
    end while
end for

openCount ← 0
for i ← 0 to n – 1 do
    if A[i] = 1 then
        print (i + 1)-th locker door is open
        openCount ← openCount + 1
    else
        print (i + 1)-th locker door is closed
    end if
end for
print openCount locker doors are open
1.2 Better

If a locker door gets toggled odd number of times, it remains open at the end. If $i$ has odd number of divisors, $i$-th locker door gets toggled odd number of times and thus remains open at the end. As divisors exist in pair, there could be odd number of divisors only if for one of the divisors, the quotient is the divisor itself. In short, when $i$ is a perfect square. Thus if we can find out which numbers are perfect square, we know that those doors remain open at the end, but all other doors remain closed.

Algorithm 2 LockerBetter(A[0..n − 1])

```
for i ← 0 to n − 1 do
    A[i] ← 0
end for

c ← ⌊√n⌋

for i ← 0 to c − 1 do
    A[i · i] ← 1
end for

for i ← 0 to n − 1 do
    if A[i] = 1 then
        print (i + 1)-th locker door is open
    else
        print (i + 1)-th locker door is closed
    end if
end for

print c locker doors are open
```

The inefficient one is $Θ(n \lg n)$ whereas the efficient one is $Θ(n)$.

2 Problem 2

2.1

The Count array records how many elements are smaller than an element of the input array. The lower the number, the earlier the element should appear in the sorted list. And you should show something beside the sorted array!

2.2

It is not stable, because if you take an input array whose two consecutive elements are equal, then the else of the algorithm gets executed making the left one’s count bigger and then at the end, the left one moves to the right.

2.3

Not in-place. It uses two extra arrays whose size depends on the input size, $n$. 