```
Algorithm sum(L)
    if L.isEmpty() then return 0
    return sumHelper(L, L.first())

Algorithm sumHelper(L, p)
    if L.isLast(p) then
        return p.element()
    s := sumHelper(L, L.after(p))
    return s + p.element()

Algorithm sum(T)
    if T.isEmpty() then return 0
    return sumHelper(T, T.root())
Algorithm sumHelper(T, p)
    if T.isExternal(p) then
        return 0
    lsum := sumHelper(T, T.leftChild(p))
    rsum := sumHelper(T, T.rightChild(p))
    return lsum + rsum + p.element()
```

$b^{z} = b^{a \log_{b} x} = ((b^{\log_{b} x})^{a})$

$b^z = x^a$
$\log_b b^z = \log_b x^a$
$z = \log_b x^a$

5 → false

3 → false

```
Algorithm isEven(n)
    if n = 0 then
        return true
    return ! isEven(n-1)
Algorithm isEven(n)
    if n = 0 then
        return true
    if n = 1 then
        return false
    return isEven(n-2)
```

```
Algorithm findSmallestHelper(T, p)
        if T.isExternal(p) then
                return +infinity
        Lmin := findSmallestHelper(T, T.leftChild(p))
        Rmin := findSmallestHelper(T, T.rightChild(p))
        return MIN(MIN(Lmin, Rmin), p.element())

Algorithm shuffle(S)
        last := S.size()
        while last > 1 do
                r := randomInt(last)
                last := last – 1
                S.swapElements(S.atRank(r), S.atRank(last))

Algorithm shuffleRec(S)
        last := S.size()
        shuffleHelper(S, last)
Algorithm shuffleHelper(S, last)
        if last > 1 then
                r := randomInt(last)
                last := last – 1
                S.swapElements(S.atRank(r), S.atRank(last))
                shuffleHelper(S, last)


Algorithm findSmallerKeys(T, x)
        S := new Sequence
        if T.isEmpty() then return S
        findSmallerHelper(T, T.root(), x, S)
        return S
Algorithm findSmallerHelper(T, p, x, S)
        if T.isExternal(p) \/  p.element() > x then
                return
        findSmallerHelper(T, T.leftChild(p), x, S)
        S.insertLast(p.element())
        findSmallerHelper(T, T.rightChild(p), x, S)
```

() => (())
(1, 2) => (), (2), (1), (1,2))
(1,2,3) => (), (3), (2), (2,3), (1), (1,3), (1,2), (1,2,3))

```
Algorithm powerSet(S)
        R := new Sequence
        if S.isEmpty() then
                R.insertLast(S)
                return R
        e := S.remove(S.last())
        R1 := powerSet(S)
        for each sub in R1.elements() do
                R.insertLast(sub)
                sub2 := copy(sub)
                sub2.insertLast(e)
                R.insertLast(sub2)
        return R

Algorithm powerSet(S)
        if S.isEmpty() then
                R := new List
                S.insertLast(R)
                return S
        e := S.remove(S.last())
        R := powerSet(S)
        return powerSetHelper(R, R.first(), e)

Algorithm powerSetHelper(R, p, e)
        sub := copy(p.element())
        sub.insertLast(e)
        R.insertAfter(p, sub)
        if R.isLast(p) then
                return R
        return powerSetHelper(R, R.after(p), e)
```