```
Algorithm isTwoEquals(S)
Input S is a Sequence
Output is a Boolean

1  D := new Dictionary
1  P := S.first()
1  D.insertItem(P.element(), 1)
n  while !S.isLast(P) do
n      P := S.after(P)
n      value := P.element()
n      ed := D.findElement(value)
n      if ed = NO_SUCH_KEY then
n          D.insertItem(value, 1)
n      else
n          return true

   return false
```

$O(n)$

```
Algorithm countInversions(S)
Input: S is a sequence
    count := 0
    if S.size() > 1 then
        (S1, S2) := partition(S, n/2)
        count := countInversions(S1) + count
        count := countInversions(S2) + count
        count := count(S1, S2, S) + count
    return count

Algorithm count(S1, S2, S)
    Input: Sorted subsequence S1, S2 and S is a sequence
    count := 0
    while !S1.isEmpty() and !S2.isEmpty() do
        if S1.first().element() < S2.first().element() then
            S.insertLast(S1.remove(S1.first()))
        else
            count := count + S1.size()
            S.insertLast(S2.remove(S2.first()))
    while !S1.isEmpty() do
        S.insertLast(S1.remove(S1.first()))
    while !S2.isEmpty() do
        S.insertLast(S2.remove(S1.first()))
    return count.
```

A.

```
Algorithm findDeepestNode(T)
    D = newDictionary
    if T = null then
        return null
    findDeepestHelper(D, T.root, 0, T)
    greater := null
    for item to D.items() do
        if greater == null
            greater := item
            continue
        if greather.key() < item.key() then
            greater := item
    return greater

Algorithm findDeepestHelper(D, N, counter, T)
    Input D is dictionary, N is a Node, counter integer, T is a tree
    if N != null then
        D.insertItem(counter, N)
        findDeepestHelper(D, T.rightChild(N), counter+1, T)
        findDeepestHelper(D, T.leftChild(N), counter+1, T)
```

Algorithm bucketSort (S, N, Counting)

Input : S is a Sequence, N is a Size of Buckets,
Counting is a pointer ~~check~~ check word ~~letter~~.

A = new Array de size N that contains Sequence.

totalSize Word = ('Z' - 'A') / N

While ! S.isEmpty() do

(k,0) := S.remove (S.first())

~~temp := 'z' - k.charAt(k.lenght - counting)~~

~~if (temp < 0)~~

pointcheck := k.lenght - counting

if pointcheck < 0 then

A[0].insertFisst ((k,0))

else :

temp := 'z' - k.charAt (pointcheck)

if (temp %. totalSize Word < totalsizeWord /2 then

A[temp/totalSizeWord].insertFirst ((k,0))

else

A[temp/totalSizeWord].insertLast ((k,0))

For i:=0 to N-1 do

while ! A[i].isEmpty() do

F := A[i].first()

(k,0) ← A[i].remove (F)

S.insertLast ((k,0))