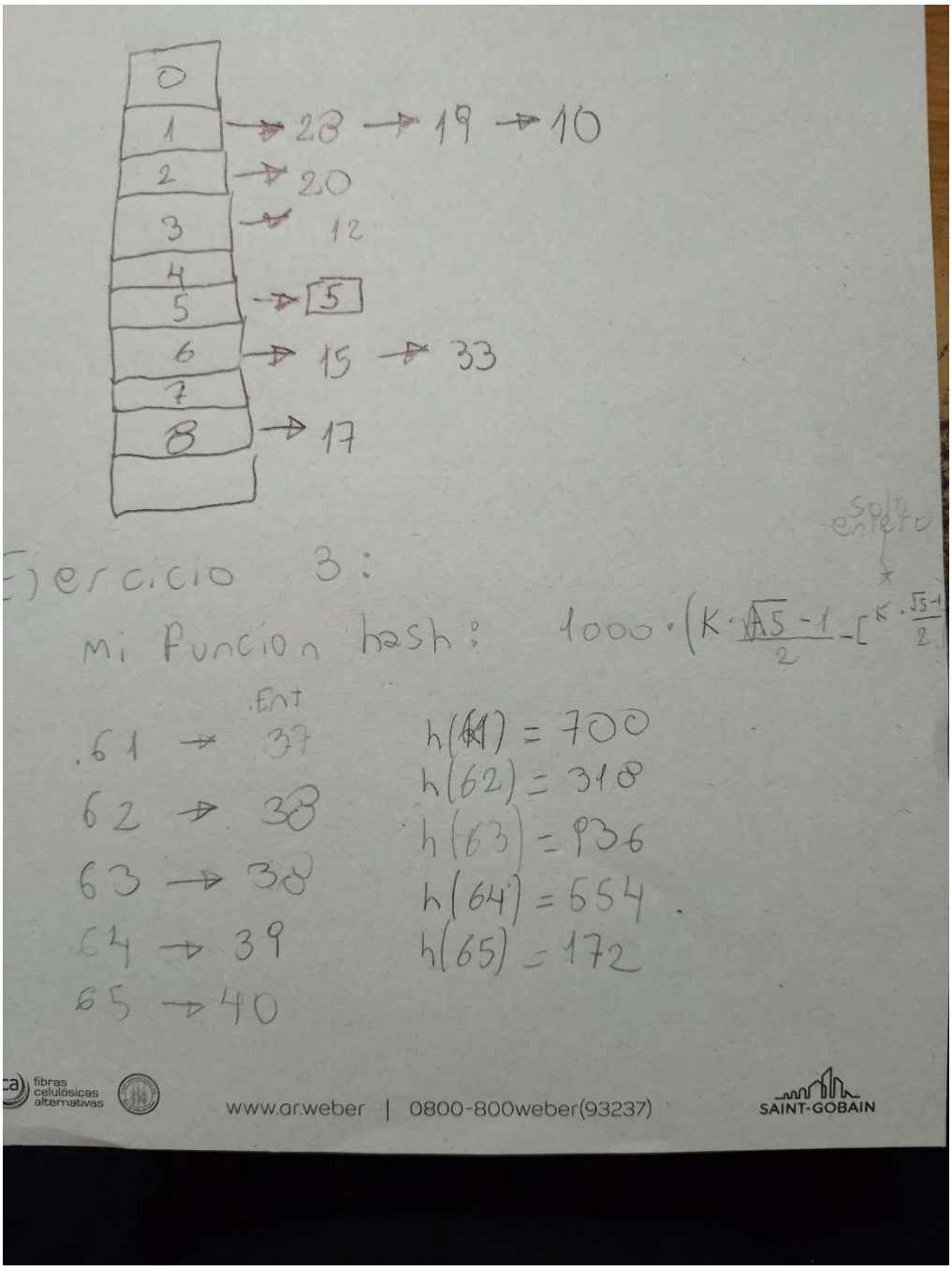
1 y 3-


0
1
2
3
4
5
6
7
8
28
19
10
20
12
5
15
33
17
Ejercicio 3:
Mi Funcion hash:
1000 · (K · √5 - 1 / 2 - [K · √5 - 1 / 2
solo entero
Ent
61 → 37
62 → 38
63 → 38
64 → 39
65 → 40
h(61) = 700
h(62) = 318
h(63) = 936
h(64) = 554
h(65) = 172
fibras celulósicas alternativas
www.ar.weber | 0800-800weber(93237)
SAINT-GOBAIN

2-

```
6  class DictionaryNode:
7    key = None
8    value = None
9
10
11 class Dictionary:
12   head = None
13
14
15 # def insert(D, key, value):
16 #   slot = key % 9
17 #   if D[slot] == None:
18 #     aList = linkedlist.LinkedList()
19 #     D[slot] = aList
20 #     dicNode = DictionaryNode()
21 #     dicNode.key = key
22 #     dicNode.value = value
23 #     linkedlist.add(aList, dicNode)
24 #   else:
25 #     slotList = D[slot]
26 #     dicNode = DictionaryNode()
27 #     dicNode.key = key
28 #     dicNode.value = value
29 #     linkedlist.add(slotList, dicNode)
30 #   return D
```

```
31
32  # def search(D, key):
33  #   slot = key % 9
34  #   if D[slot] == None:
35  #     return None
36  #   else:
37  #     currentNode = D[slot].head
38  #     while currentNode != None:
39  #       if currentNode.value.key == key:
40  #         return currentNode.value.value
41  #       currentNode = currentNode.nextNode
42  #     return None
43
```

```
# def delete(D, key):
#   slot = key % 9
#   if D[slot] == None:
#     return None
#   else:
#     currentNode = D[slot].head

#     while currentNode != None:
#       if currentNode.value.key == key:
#         linkedlist.delete(D[slot], currentNode.value)
#         if linkedlist.length(D[slot]) == 0:
#           D[slot] = None
#         return D
#       currentNode = currentNode.nextNode
```

4-

```
#FALTA MEJORAR, HAY ERRORES.
def checkPermutation(S, P):
  myHash = []

  if len(S) != len(P):
    return False

  for i in range(0, len(S)):
    myHash[i] = None

  for i in range(0, len(S)):
    insert(myHash, ord(S[i]), S[i])

  for i in range(0, len(P)):
    if search(myHash, P[i]) == None:
      return False

  return True
```

Es de o(n), debido a que debe de revisar letra por letra si es que esta dentro del hash.

5-

```
127 def areUniqueElements(L):
128     myHash = []
129     for i in range(0, linkedlist.length(L)):
130         myHash.append(None)
131     currentNode = L.head
132     for i in range(0, linkedlist.length(L)):
133         if search(myHash, currentNode.value) != None:
134             return False
135         insert(myHash, currentNode.value, currentNode.value)
136         currentNode = currentNode.nextNode
137     return True
```

En el peor caso es o(k) (siendo k la longitud de la lista)

8-

```
140 #REVISAR
141 def checkInside(S, P):
142     myHash = []
143     for i in range(0, 149):
144         myHash.append(None)
145 
146     lengthWords = len(P)
147     i = 0
148     while i + lengthWords <= len(P):
149         #Tomar letras de a 4 en la palabra mas larga
150         #uso k y j para lo que seria formar mi key, i es con quien recorro toda la palabra.
151         k = i
152         subString = S[i:i + lengthWords]
153         j = lengthWords - 1
154         aKey = 0
155         while j != -1:
156             aKey = (ord(subString[k]) * (10**j)) + aKey
157             k += 1
158             j -= 1
159         insert(myHash, aKey, i)
160         i += 1
161 
162     #Al terminar con todas las palabras que podia conformar con la mas larga, busco la palabra
    pequeña
163     aKeyAux = 0
164     j = lengthWords - 1
165     l = 0
166     while j != -1:
167         aKeyAux = (ord(P[l]) * (10**j)) + aKeyAux
168         l += 1
169         j -= 1
170     return search(myHash, aKeyAux)
```

El costo de este ejercicio será O(n – m) (siendo n la longitud de la palabra larga y m la longitud de la palabra pequeña).