Ejercicio 1

```
def insert(T,element):
    if T.root == None:
        T.root=crearLista()
    element=element.lower()
    current=T.root
    total=len(element)-1
    for i,v in enumerate(element):
        if current[ord(v)-97] is None:
            newNode=TrieNode()
            newNode.key=v
            newNode.children=crearLista()
            if total==i:
                newNode.IsEndOfWord=True
                current[ord(v)-97]=newNode
            elif i==0:
                currentRet=newNode
                current[ord(v)-97]=newNode
                current=current[ord(v)-97].children
            else:
                newNode.parent=currentRet
                currentRet=newNode
                current[ord(v)-97]=newNode
                current=current[ord(v)-97].children
        else:
            currentRet=current[ord(v)-97]
            current=current[ord(v)-97].children


def search(T,element):
    current=T.root
    total=len(element)-1
    element=element.lower()
    for i,v in enumerate(element):
        if current[ord(v)-97] is not None:
            if i==total :
                if current[ord(v)-97].IsEndOfWord==True :
                    return True
                else:
                    return False
            else:
                current=current[ord(v)-97].children
        else:
            return False
```

Ejercicio 2

Con la implementación que hice de Trie, creo que el orden de complejidad es de O(m), ya que sabiendo el número de carácter a buscar en código ASCII es directo, sin la necesidad pasar por todos los nodos de la listas.

Ejercicio 3

```
def delete(T,element):
    current=T.root
    total=len(element)-1
    contador=0
    element=element.lower()
    inicio=TrieNode()
    bifurcacion=TrieNode()
    siguiente=""
    if search(T,element):
        for i,v in enumerate(element):
            if i==0:
                inicio=current[ord(v)-97]
                bifurcacion=inicio
            if current[ord(v)-97].children.count(None)<25:
                bifurcacion=current[ord(v)-97]
                siguiente=element[i+1]
            if i==total and current[ord(v)-97].children.count(None)<26:
                current[ord(v)-97].IsEndOfWord=False
                return True
            current=current[ord(v)-97].children
        if inicio==bifurcacion:
            T.root[ord(inicio.key)-97]=None
            return True
        else:
            bifurcacion.children[ord(siguiente)-97]=None
            return True
    else:
        return False
```