Dhruthzuci Tech Solution - Assignment

NAME: Rai Srihari

Reg.no: 11807979

Mail: sriharirai77@gmail.com

Q1)Write API:
Using Python Flask or ExpressJS, Write a REST API that reads the body and returns JSON.

```
# API Method POST
# URL : /find_symbols_in_names

# Input JSON Body of the API:
{
   "chemicals": ['Amazon', 'Microsoft', 'Google'],
   "symbols": ['I', 'Am', 'cro', 'Na', 'le', 'abc']
}

# Output: display the chemical names with their symbol surrounded by square brackets:
{
   "result": "[Am]azon, Mi[cro]soft, Goog[le]"
}
```

```
from functools import reduce

class TrieNode:

   def __init__(self):
      self.c = dict()
      self.sym = None

def bracket(words, symbols):

   root = TrieNode()
   for s in symbols:
      t = root

      for char in s:
         if char not in t.c:
            t.c[char] = TrieNode()

         t = t.c[char]

      t.sym = s
```

```
    result = dict()

    for word in words:
        i = 0
        symlist = list()

        while i < len(word):
            j, t = i, root

            while j < len(word) and word[j] in t.c:
                t = t.c[word[j]]

                if t.sym is not None:
                    symlist.append((j+1-len(t.sym), j+1, t.sym))
                j += 1

            i += 1

        if len(symlist) > 0:
            sym = reduce(lambda x, y: x if x[1]-x[0] >= y[1]-y[0] else y, symlist)
            result[word] = "{}[{}]{}".format(word[:sym[0]], sym[2], word[sym[1]:])

    return tuple(word if word not in result else result[word] for word in words)

bracket(['amazon', 'microsoft', 'google'], ['i', 'am', 'cro', 'na', 'le', 'abc'])
>>> ('[am]azon', 'mi[cro]soft', 'goog[le]')
```

Q2) Given two lays, write a function to compute their intersection.
Example 1:
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2]

Example 2:
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [9,4]
Note:
Each element in the result must be unique.
The result can be in any order.

CODE ->

```
def intersect(l1, l2, m, n):

    if (m > n):
        t = l1
        l1 = l2
        l2 = t

        temp = m
        m = n
```

```
        n = temp

    l1.sort()

    for i in range(0, n):
        if (binarySearch(l1, 0, m - 1, l2[i]) != -1):
            print(l2[i], end=" ")

def binarySearch(l, l, r, x):

    if (r >= l):
        m = int(l + (r - l)/2)
        if (l[m] == x):
            return m


        if (l[m] > x):
            return binarySearch(l, l, m - 1, x)

        return binarySearch(l, m + 1, r, x)

    return -1


l1 = [8, 2, 6, 3, 4, 7]
l2 = [4, 9, 7, 21, 8]
m = len(l1)
n = len(l2)

intersect(l1, l2, m, n)
```

Q3)Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.
An input string is valid if:
Open brackets must be closed by the same type of brackets.
Open brackets must be closed in the correct order.
Note that an empty string is also considered valid.
Example 1:
Input: "()"
Output: true

Example 2:
Input: "()[]{}"
Output: true

Example 3:
Input: "(]"
Output: false

CODE ->

```
def bb(expression):
	st = []

	for ch in expression:
		if ch in ["(", "{", "["]:
			st.append(ch)

		else:
			if not st:
				return False

			cc = st.pop()

			if cc == '(':
				if ch != ")":
					return False

			if cc == '{':
				if ch != "}":
					return False

			if cc == '[':
				if ch != "]":
					return False


	if st:
		return False

	return True


if __name__ == "__main__":
	expression = "[[{()(})[{}{}]]]"

	if bb(expression):
		print("true")

	else:
		print("false")
```

Q4)Given a non-empty array of integers, every element appears twice except for one. Find that single one.
Note:
Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?
Example 1:
Input: [2,2,1]
Output: 1

Example 2:
Input: [4,1,2,1,2]
Output: 4

CODE ->

```
from collections import Counter

t = dict(Counter([2,2,1,3,3,4,6,7,7,6,0,0,4]))

for element in t:
    if (t[element]==1):
        print({t[element]})
```