```
#!/usr/local/bin/python3
import re
from data_structure.graph import TreeNode
from data_structure.trie import *
import sys
from threading import *
from time import sleep
from random import randint
from data_structure.linked_list import *

def is_power_of_2(n, d = None):
    last = 0
    count = -1
    while last < 1 and n > 0:
        last += (n & 1)
        n >>= 1
        count += 1
    return n == 0 and last == 1 and (count == d if d else True)


def outthink(n, p, q):
    for x in range(1, n+1):
        if x % p == 0 and (str(q) in str(x)):
            print('%d: outthink' % x)
        elif x % p == 0:
            print('%d: out(divided by %d)' % (x, p))
        elif str(q) in str(x):
            print('%d: think' % x)


def fib(n):
    def fibiter(n):
        if n <= 0:
            return n
        result = x = y = 1
        for _ in range(3, n + 1):
            result = x + y
            x, y = y, result
        return result
    def fibrecur(n, mem):
        if n <=1:
            return n
        if mem[n] == None:
```

```
            if mem[n-2] == None:
                mem[n-2] = fibrecur(n-2, mem)
            if mem[n-1] == None:
                mem[n-1] = fibrecur(n-1, mem)
            mem[n] = mem[n-2] + mem[n-1]
        return mem[n]
    def fibrecur_naive(n):
        if n <= 1:
            return n
        return fibrecur_naive(n-1) + fibrecur_naive(n-2)

    if 1 <= n <= 26:
        print (chr(96 + n) * fibrecur(n, [None] * (n+1)))
    return fibrecur(n, [None] *(n+1))



def passage_parser(passages):
    def clean(sentence):
        s = str.lower(sentence.strip()) # remove leading and trailing spaces,
convert to lower
        s = re.sub(r' +', ' ', s) # remove duplicated spaces
        s = re.sub(r'[^a-z0-9 ]', '', s) # remove non-alphanumetic chars
        return s
    passages = passages.split('|')
    ps = [clean(x) for x in passages]
#     print(ps)
    skipped = set()
    length = len(ps)
    for i in range(length):
        if i in skipped:
            continue
        for j in range(i + 1, length):
            if j in skipped:
                continue
            if re.search(ps[j], ps[i]):
                skipped.add(j)
            elif re.search(ps[i], ps[j]):
                skipped.add(i)
#     print(skipped)
    result = [passages[i] for i in range(length) if i not in skipped]
#     print(result)
    return '|'.join(result)
```

```
def BST_checker(root):
    '''
    check if a tree is BST.
    '''
    class NotBST(Exception):
        pass

    def get_range(r):
        assert r is not None
        low = high = r.val
        lh = rl = None
        if r.left is not None:
            low, lh = get_range(r.left) # once returned, left subtree is BST
            if lh > r.val:
                raise NotBST()

        if r.right is not None:
            rl, high = get_range(r.right)
            if rl < r.val:
                raise NotBST()

        if lh and rl and lh == rl:
            raise NotBST()
        return low, high

    if root is None:
        return True
    else:
        try:
            get_range(root)
            return True
        except NotBST:
            return False




def summary(lines):
    d = {}
    for l in lines:
        date, num, item = l.split(',')
        d[date] = d.get(date, {})
```

```
        d[date][item] = d[date].get(item, 0)
        d[date][item] += int(num)
#     result = []
    for date in sorted(d.keys()):
        total = sum(d[date].values())
        unique_items = len(d[date])
        print( '%s,%d,%.2f,%d' % (date, total, float(total)/unique_items,
unique_items))
#         result.append(s)
#     return result




def thermo_test(times, nThreads):
    def thermo(ID, times, mutex, bar, r):
        for i in range(times):
            with mutex:
                print('-%d-: thermo %d is reading temp' % (i, ID))
                if (randint(0, 3) % 3 == 0):
                    sleep(100)
            if ID == 0:
                print('-%d-: thread %d records the temp' % (i, ID))
                r.append(randint(0, 10))
            try:
                bar.wait()
            except RuntimeError:
                print('thermo %d detects aborted barrier. Exits.' % ID)
                return

    mutex = BoundedSemaphore(1)
    bar = Barrier(3)
    result = []
    for i in range(nThreads):
        t = Thread(target = thermo, args = (i, times, mutex, bar, result))
        t.start()
    for _ in range(times):
        try:
            bar.wait(3)
        except RuntimeError:
            print('one of the thermo is broken...result is %s' % result)
            break
```

```
def find_missing_sorted(A, start):
    ''' find a missing number from sorted array A which starts from $start
    '''
    if len(A) == 0:
        return None
    low, high = -1, len(A)
    while high - low > 1:
        mid = (low + high)//2
        if A[mid] > start + mid:
            high = mid
        else:
            low = mid + 1
    if 0 <= low < len(A) and start + low > A[low]:
        return start + low
    else:
        return start + high
```

```
def integer_to_words(num):
    less_than_20 = ['', 'One', 'Two', 'Three', 'Four', 'Five', 'Six', 'Seven',
'Eight', 'Nine', 'Ten', 'Eleven', 'Twelve', 'Thirteen', 'Fourteen', 'Fifteen',
'Sixteen', 'Seventeen', 'Eighteen', 'Nineteen']
    less_than_100 = ['', '', 'Twenty', 'Thirty', 'Forty', 'Fifty', 'Sixty',
'Seventy', 'Eighty', 'Ninety']
    thousand = ['', 'Thousand', 'Million', 'Billion']

    def i2w(num):
        assert 0<= num < 1000
        if num == 0: # to take care of the recursive solution
            r = ''
        elif num < 20:
            r = less_than_20[num]
        elif 20 <= num < 100:
            r = '%s %s' % (less_than_100[num//10], less_than_20[num % 10])
        else:
            r = '%s Hundred %s' % (less_than_20[num//100], i2w(num % 100))
        return r.strip()
```

```
    # below lines could be an easy solution to the problem when 0 < num <= 1000
    if num == 0:
        return 'Zero'
    if num == 1000:
        return 'One Thousand'
    if 0 <= num < 1000:
        return i2w(num)

    result = ''
    i = 0
    while num > 0:
        num, remainder = divmod(num, 1000)
        if remainder:
            result = ('%s %s %s' % (i2w(remainder), thousand[i], result)).strip()
        i += 1
    return result




def nth_reverse_node(head, n):
    length = 0
    curr = head
    while curr:
        length += 1
        curr = curr.next
    if 0 < n <= length:
        curr = head
        for _ in range(length - n):
            curr = curr.next
        return curr.val if curr else None


def insert_slot(A, target):
    low, high = 0, len(A) - 1
    while low < high:
        mid = (low + high)//2
        if A[mid] > A[high]:
            if A[low] < target < A[mid]:
                high = mid
            else:
                low = mid + 1
        else:
            if A[mid] < target:
```

```
                low = mid + 1
            else:
                high = mid
    return low if target < A[low] else low + 1


def main():
    # outthink(20, 3, 1)
    #
#     print(fib(5))
#
#     t = Trie(['abc', 'ba', 'bbc', 'bc', 'bcd', 'bcz','cd'])
#     print(t.starts_with('z', 8, True))
#
#
#     print(passage_parser('''IBM cognitive computing|IBM "cognitive" computing
is a revolution| ibm cognitive computing|'IBM Cognitive Computing' is a
revolution? '''))
#
#     print(passage_parser(''' IBM cognitive computing|IBM "cognitive" computing
is a revolution|the cognitive computing is a revolution'''))
#
#     print(BST_checker(TreeNode.from_list([5,3,9,-1,3,5,12])))
#
#
#     instr = ['2018-01-01,5,cookies', '2016-04-02,7,pumpkins','2017-04-
03,2,peaches', '2017-04-02,6,apples']
#     instr1 = ['2017-06-02,5,apples', '2017-06-02,2,pears', '2017-06-
01,3,pineapples']
#     print(summary(instr))
#
#     thermo_test(5, 2)
#
#     print(find_missing([1,2,3,4,5], 0))

#     print(integer_to_words(10000012))
#
#     print(nth_reverse_node(create_LL([1,2,3]), 4))
#
    print(insert_slot([3, 5, -5], 1))

if __name__ == '__main__':
    main()
```