```
def next_number(s):
    result = []
    i = 0
    while i < len(s):
        count = 1
        while i + 1 < len(s) and s[i] == s[i + 1]:
            i += 1
            count += 1
        result.append(str(count) + s[i])
        i += 1
    return ''.join(result)


# print(next_number("1211"))
# s = "1"
# n = 4
# for i in range(n):
#     s = next_number(s)
#     print(s)

def spreadsheet_encode_column(col_str):
    num = 0
    count = len(col_str) - 1
    for s in col_str:
        temp = num
        num += 26 ** count * (ord(s) - ord('A') + 1)
        count -= 1
    return num


# print(spreadsheet_encode_column("ZZ"))

# Palindrome String

s = "Was it a cat I saw?"


# s1 = ''.join([i for i in s if i.isalpha()]).replace(" ", "").lower()
# print(s == s[::-1])
# s2 = ([i for i in s if i.isalpha()])

def is_palindrome(s):
    i = 0
    j = len(s) - 1

    while i < j:
        while not s[i].isalnum() and i < j:
            i += 1
        while not s[j].isalnum() and i < j:
            j -= 1
        if s[i].lower() != s[j].lower():
            return False
        i += 1
        j -= 1
    return True


# print(is_palindrome(s))

# Anagram
s1 = 'fairy tales'
s2 = 'rail safety'

s1 = s1.replace(" ", "")
s2 = s2.replace(" ", "")


# Requires n logn time (Since any comparison
```

```
# based sorting algorthims requires atleast
# n log n times to sort)
# print(sorted(s1) == sorted(s2))

def is_anagram(s1, s2):
    ht = dict()

    if len(s1) != len(s2):
        return False

    for i in s1:
        if i in ht:
            ht[i] += 1
        else:
            ht[i] = 1

    for i in s2:
        if i in ht:
            ht[i] -= 1
        else:
            ht[i] = 1

    for i in ht:
        if ht[i] != 0:
            return False
    return True


# print(is_anagram(s1, s2))

# Palindrome Permutation Problem
palin_prem = "Tact Coa"
not_palin_prem = "This is not a palindrome permutation"


def is_palin_perm(input_str):
    input_str = input_str.replace(" ", "")
    input_str = input_str.lower()

    d = dict()

    for i in input_str:
        if i in d:
            d[i] += 1
        else:
            d[i] = 1

    odd_count = 0
    for k, v in d.items():
        if v % 2 != 0 and odd_count == 0:
            odd_count += 1
        elif v % 2 != 0 and odd_count != 0:
            return False
    return True


# print(is_palin_perm(palin_prem))

# Checking Permutation problem
is_permutation_1 = "God"
is_permutation_2 = "dog"

not_permutation_1 = "Not"
not_permutation_2 = "top"


# TC : O(n logn) SC: O(1)
def is_prem_1(str_1, str_2):
```

```
    str_1 = str_1.lower()
    str_2 = str_2.lower()

    if len(str_1) != len(str_2):
        return False

    str_1 = ''.join(sorted(str_1))
    str_2 = ''.join(sorted(str_2))

    n = len(str_1)
    for i in range(n):
        if str_1[i] != str_2[i]:
            return False
    return True


def is_prem_2(str_1, str_2):
    str_1 = str_1.lower()
    str_2 = str_2.lower()

    if len(str_1) != len(str_2):
        return False

    d = dict()

    for i in str_1:
        if i in d:
            d[i] -= i
        else:
            d[i] = 1
    for i in str_2:
        if i in d:
            d[i] -= 1
        else:
            d[i] = 1
    return all(value == 0 for value in d.values())


print(is_prem_1(is_permutation_1, is_permutation_2))
print(is_prem_1(not_permutation_1, not_permutation_2))

# Unique and Non unique string
unique_str = "AbCDefg"
non_unique_str = "non unique str"


def normalize(input_str):
    input_str = input_str.replace(" ", "")
    return input_str.lower()


def is_unique_1(input_str):
    input_str = normalize(input_str)
    d = dict()
    for i in input_str:
        if i in d:
            return False
        else:
            d[i] = 1
    return True


print(is_unique_1(unique_str))


def int_to_str(input_int):
    if input_int < 0:
        is_negative = True
```

```
        input_int *= -1
    else:
        is_negative = False

    output_str = []
    while input_int > 0:
        output_str.append(chr(ord('0') + input_int % 10))
        input_int //= 10
    output_str = output_str[::-1]

    output_str = ''.join(output_str)

    if is_negative:
        return '-' + output_str
    else:
        return output_str


print(int_to_str(-123))


def str_to_int(input_str):
    out_int = 0

    if input_str[0] == '-':
        start_idx = 1
        is_negative = True
    else:
        start_idx = 0
        is_negative = False

    for i in range(start_idx, len(input_str)):
        place = 10 ** (len(input_str) - (i + 1))
        digit = ord(input_str[i]) - ord('0')
        out_int += place * digit

    if is_negative:
        return -1 * out_int
    else:
        return out_int


print(str_to_int("-123"))

AS = [2, 3, 3, 5, 7, 11]
BS = [3, 3, 7, 15, 31]


def intersect_sorted_array(A, B):
    i = 0
    j = 0
    interesection = []

    while i < len(A) and j < len(B):
        if A[i] == B[j]:
            if i == 0 or A[i] != A[i - 1]:
                interesection.append(A[i])
            i += 1
            j += 1
        elif A[i] < B[j]:
            i += 1
        else:  # A[i] > B[j]
            j += 1
    return interesection


print(intersect_sorted_array(AS, BS))
```