Create a function that will test if a string is a valid PIN or not via a regular expression.

A valid PIN has:

- Exactly 4 or 6 characters.
- Only numeric characters (0-9).
- No whitespace.

Examples

```
validate("121317") → true

validate("1234") → true

validate("45135") → false

validate("89abc1") → false

validate("900876") → true

validate(" 4983") → false
```

Notes

- Empty strings should return `false` when tested.

You have a pack of 5 randomly numbered cards, which can range from 0-9. You can win if you can produce a higher **two-digit** number from your cards than your opponent. Return `true` if your cards win that round.

Examples

```
winRound([2, 5, 2, 6, 9], [3, 7, 3, 1, 2]) → true
// Your cards can make the number 96
// Your opponent can make the number 73
// You win the round since 96 > 73

winRound([2, 5, 2, 6, 9], [3, 7, 3, 1, 2]) → true

winRound([1, 2, 3, 4, 5], [9, 8, 7, 6, 5]) → false

winRound([4, 3, 4, 4, 5], [3, 2, 5, 4, 1]) → false
```

Given an array of integers, return the **largest gap** between the **sorted elements** of the array.

For example, consider the array:

```
[9, 4, 26, 26, 0, 0, 5, 20, 6, 25, 5]
```

... in which, after sorting, the array becomes:

```
[0, 0, 4, 5, 5, 6, 9, 20, 25, 26, 26]
```

... so that we now see that the largest gap in the array is between 9 and 20 which is `11`.

Examples

```
largestGap([9, 4, 26, 26, 0, 0, 5, 20, 6, 25, 5]) → 11
// After sorting: [0, 0, 4, 5, 5, 6, 9, 20, 25, 26, 26]
// Largest gap between 9 and 20 is 11

largestGap([14, 13, 7, 1, 4, 12, 3, 7, 7, 12, 11, 5, 7]) → 4
// After sorting: [1, 3, 4, 5, 7, 7, 7, 7, 11, 12, 12, 13, 14]
// Largest gap between 7 and 11 is 4

largestGap([13, 3, 8, 5, 5, 2, 13, 6, 14, 2, 11, 4, 10, 8, 1, 9]) → 2
// After sorting: [1, 2, 2, 3, 4, 5, 5, 6, 8, 8, 9, 10, 11, 13, 13, 14]
// Largest gap between 6 and 8 is 2
```

Create a function that finds how many prime numbers there are, up to the given integer.

Examples

```
primeNumbers(10) → 4
// 2, 3, 5 and 7

primeNumbers(20) → 8
// 2, 3, 5, 7, 11, 13, 17 and 19

primeNumbers(30) → 10
// 2, 3, 5, 7, 11, 13, 17, 19, 23 and 29
```

A number is said to be **Harshad** if it's *exactly divisible* by the **sum** of its digits. Create a function that determines whether a number is a Harshad or not.

Examples

```
isHarshad(75) → false
// 7 + 5 = 12
// 75 is not exactly divisible by 12

isHarshad(171) → true
// 1 + 7 + 1 = 9
// 9 exactly divides 171

isHarshad(481) → true

isHarshad(89) → false

isHarshad(516) → true

isHarshad(200) → true
```

Create a function that will remove any repeated character(s) in a word passed to the function. Not just consecutive characters, but characters repeating anywhere in the string.

Examples

```
unrepeated("teshahset") → "tesha"

unrepeated("hello") → "helo"

unrepeated("aaaaa") → "a"

unrepeated("WWE!!!") → "WE!"

unrepeated("call 911") → "cal 91"
```

Given a string, reverse all the words which have odd length. The even length words are not changed.

Examples

```
reverseOdd("Bananas") → "sananaB"

reverseOdd("One two three four") → "enO owt eerht four"

reverseOdd("Make sure uoy only esrever sdrow of ddo length")
→ "Make sure you only reverse words of odd length"
```

We can assign a value to each character in a word, based on their position in the alphabet (a = 1, b = 2, ... , z = 26). A **balanced word** is one where the sum of values on the left-hand side of the word equals the sum of values on the right-hand side. For odd length words, the middle character (balance point) is ignored.

Write a function that returns `true` if the word is balanced, and `false` if it's not.

Examples

```
balanced("zips") → true
// "zips" = "zi|ps" = 26+9|16+19 = 35|35 = true

balanced("brake") → false
// "brake" = "br|ke" = 2+18|11+5 = 20|16 = false
```

Notes

- All words will be lowercase, and have a minimum of 2 characters.
- Palindromic words will always be balanced.

Create a program that converts a phone number with letters to one with only numbers.

Number	Letter
0	none
1	none
2	ABC
3	DEF
4	GHI
5	JKL
6	MNO
7	PQRS
8	TUV
9	WXYZ

Examples

```
textToNum("123-647-EYES") → "123-647-3937"

textToNum("(325)444-TEST") → "(325)444-8378"

textToNum("653-TRY-THIS") → "653-879-8447"

textToNum("435-224-7613") → "435-224-7613"
```

Notes

- All inputs will be formatted as a string representing a proper phone number in the format `XXX-XXX-XXXX` or `(XXX)XXX-XXXX`, using numbers and capital letters.

Create a function that takes a string and returns the number of alphanumeric characters that occur more than once.

Examples

```
duplicateCount("abcde") → 0

duplicateCount("aabbcde") → 2

duplicateCount("Indivisibilities") → 2

duplicateCount("Aa") → 0
// Case sensitive
```

Notes

- Duplicate characters are case sensitive.
- The input string will contain only alphanumeric characters.

Write a function that adds two numbers. The catch, however, is that the numbers will be strings.

Examples

```
addStrNums("4", "5") → "9"

addStrNums("abcdefg", "3") → "-1"

addStrNums("1", "") → "1"

addStrNums("1874682736267235927359283579235789257", "3265298357298572 9") → "187
```

Notes

- If there are any non-numerical characters, return `"-1"`.
- An empty parameter should be treated as `"0"`.
- Your function should be able to add any size number.
- Your function doesn't have to add negative numbers.
- Zeros at the beginning of the string should be trimmed.
- **Bonus**: Don't use `BigInteger` or `BigDecimal` classes.

Create a function that takes a number as an argument and returns `true` if the number is a valid credit card number, `false` otherwise.

Credit card numbers must be between 14-19 digits in length, and pass the Luhn test, described below:

1. Remove the last digit (this is the "check digit").
2. Reverse the number.
3. Double the value of each digit in odd-numbered positions. If the doubled value has more than 1 digit, add the digits together (e.g. 8 x 2 = 16 → 1 + 6 = 7).
4. Add all digits.
5. Subtract the last digit of the sum (from step 4) from 10. The result should be equal to the check digit from step 1.

Examples

```
validateCard(1234567890123456) → false

// Step 1: check digit = 6, num = 123456789012345
// Step 2: num reversed = 543210987654321
// Step 3: digit array after selective doubling: [1, 4, 6, 2, 2, 0, 9, 8, 5, 6,
// Step 4: sum = 58
// Step 5: 10 - 8 = 2 (not equal to 6) → false

validateCard(1234567890123452) → true
```

You are given two strings `s` and `t`.

String `t` is generated by randomly shuffling string `s` and then adding one more letter at a random position.

Return the letter that was added to `t`.

Examples

```
findTheDifference("bcefg", "bcdefg") → 'd'

findTheDifference("abcd", "abcde") → 'e'

findTheDifference("aiou", "aeiou") → 'e'

findTheDifference("mnoqrst", "mnopqrst") → 'p'
```

Given two strings, `s1` and `s2`, select only the characters in each string where the character in the same position in the **other** string is in uppercase. Return these as a single string.

To illustrate, given the strings `s1 = "heLLo"` and `s2 = "GUlp"`, we select the letters "he" from `s1`, because "G" and "U" are uppercase. We then select "lp" from `s2`, because "LL" is in uppercase. Finally, we join these together and return `"help"`.

Examples

```
selectLetters("heLLO", "GUlp") → "help"

selectLetters("1234567", "XxXxX")  → "135"

selectLetters("EVERYTHING", "SomeThings") →  "EYSomeThings"
```

Notes

Create a function which validates whether a given array **alternates** between *positive* and *negative* numbers.

Examples

```
alternateSign([3, -2, 5, -5, 2, -8]) → true

alternateSign([-6, 1, -1, 4, -3]) → true

alternateSign([4, 4, -2, 3, -6, 10]) → false
```

Notes

- Lists can be of any length.
- It doesn't matter if an array begins/ends with a positive or negative, as long as it alternates.
- If a list contains 0, return `false` (as it is neither positive nor negative).

Create a method that takes an array of integers and returns a new array, sorted in ascending order (smallest to biggest).

- Sort integer array in ascending order.
- If the function's argument is `null`, an empty array, or `undefined`; return an empty array.
- Return a new array of sorted numbers.

Examples

```
sortNumsAscending([1, 2, 10, 50, 5]) → [1, 2, 5, 10, 50]

sortNumsAscending([80, 29, 4, -95, -24, 85]) → [-95, -24, 4, 29, 80, 85]

sortNumsAscending(null) → []

sortNumsAscending([]) → []
```

Create a method that takes a string as its argument and returns the string in reversed order.

Examples

```
reverse("Hello World") → "dlroW olleH"

reverse("The quick brown fox.") → ".xof nworb kciuq ehT"

reverse("Edabit is really helpful!") → "!lufpleh yllaer si tibadE"
```

Write a function that takes an array of elements and returns only the ints.

Examples

```
returnInts([9, 2, "space", "car", "lion", 16]) → [9, 2, 16]

returnInts(["hello", 81, "basketball", 123, "fox"]) → [81, 123]

returnInts([10, "121", 56, 20, "car", 3, "lion"]) → [10, 56, 20, 3]

returnInts(["String",  true,  3.3,  1]) → [1]
```

Create a function that takes a string and returns a string ordered by the length of the words. From shortest to longest word. If there are words with the same amount of letters, order them alphabetically.

Examples

```
sortByLength("Hello my friend") → "my Hello friend"

sortByLength("Have a wonderful day") → "a day Have wonderful"

sortByLenght("My son loves pineapples") → "My son loves pineapples"
```