5. Longest Palindromic Substring

Given a string `s`, return *the longest palindromic substring* in `s`.

Example 1:

```
Input: s = "babad"
Output: "bab"
Explanation: "aba" is also a valid answer.
```

Example 2:

```
Input: s = "cbbd"
Output: "bb"
```

Constraints:

- `1 <= s.length <= 1000`
- `s` consist of only digits and English letters.

```
# Find better approach
class Solution:
    def longestPalindrome(self, s: str) -> str:
        total = 0
        c= 0
        ans = ""
        for i in range(len(s)):
            for j in range(i+1,len(s)+1):
                if total < j-i and s[i:j] == s[i:j][::-1]:
                    ans = s[i:j]
                    total = j-i
        return ans
```

8. String to Integer (atoi)

Implement the `myAtoi(string s)` function, which converts a string to a 32-bit signed integer.

The algorithm for `myAtoi(string s)` is as follows:

1. **Whitespace**: Ignore any leading whitespace (`" "`).
2. **Signedness**: Determine the sign by checking if the next character is `'-'` or `'+'`, assuming positivity if neither present.
3. **Conversion**: Read the integer by skipping leading zeros until a non-digit character is encountered or the end of the string is reached. If no digits were read, then the result is 0.
4. **Rounding**: If the integer is out of the 32-bit signed integer range $[-2^{31}, 2^{31} - 1]$, then round the integer to remain in the range. Specifically, integers less than -2^{31} should be rounded to -2^{31}, and integers greater than $2^{31} - 1$ should be rounded to $2^{31} - 1$.

Return the integer as the final result.

Example 1:

Input: s = "42"

Output: 42

Explanation:

```
The underlined characters are what is read in and the caret is the current reader pos
Step 1: "42" (no characters read because there is no leading whitespace)
         ^
Step 2: "42" (no characters read because there is neither a '-' nor '+')
         ^
Step 3: "42" ("42" is read in)
           ^
```

Example 2:

Input: s = " -042"

Output: -42

Explanation:

```
Step 1: "___-042" (leading whitespace is read and ignored)
            ^
Step 2: "   -042" ('-' is read, so the result should be negative)
             ^
Step 3: "   -042" ("042" is read in, leading zeros ignored in the result)
               ^
```

Example 3:

Input: s = "1337c0d3"

Output: 1337

Explanation:

```
Step 1: "1337c0d3" (no characters read because there is no leading whitespace)
         ^
Step 2: "1337c0d3" (no characters read because there is neither a '-' nor '+')
         ^
Step 3: "1337c0d3" ("1337" is read in; reading stops because the next character is a
             ^
```

Example 4:

Input: s = "0-1"

Output: 0

Explanation:

```
Step 1: "0-1" (no characters read because there is no leading whitespace)
         ^
Step 2: "0-1" (no characters read because there is neither a '-' nor '+')
         ^
Step 3: "0-1" ("0" is read in; reading stops because the next character is a non-digi
          ^
```

Example 5:

Input: s = "words and 987"

Output: 0

Explanation:

Reading stops at the first non-digit character 'w'.

Constraints:

- `0 <= s.length <= 200`
- `s` consists of English letters (lower-case and upper-case), digits (`0-9`), `' '`, `'+'`, `'-'`, and `'.'`.

```
# find optimized version
class Solution:
    def myAtoi(self, s: str) -> int:
        res = 1
        i = 0
        if not s: return 0
        while i < len(s) and s[i] == " ":
            i+=1
        if i < len(s):
            if s[i] == "+":
                i+=1
            elif s[i] == "-":
                res = -1
                i+=1
        while i< len(s) and s[i] == "0":
            i+=1

        temp = ""
        while i < len(s):
            if not ("0" <= s[i] <= "9"):
                break
            temp += s[i]
            i+=1
        if temp:
            res *= int(temp)
        else:
            return 0
        if res in range(-2**31,2**31-1+1):
            return res
        elif res < -2**31:
            return -2**31
        else:
            return 2**31-1
        return 0
```

---

13. Roman to Integer

Roman numerals are represented by seven different symbols: `I`, `V`, `X`, `L`, `C`, `D` and `M`.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, `2` is written as `II` in Roman numeral, just two ones added together. `12` is written as `XII`, which is simply `X + II`. The number `27` is written as `XXVII`, which is `XX + V + II`.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not `IIII`. Instead, the number four is written as `IV`. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as `IX`. There are six instances where subtraction is used:

- `I` can be placed before `V` (5) and `X` (10) to make 4 and 9.
- `X` can be placed before `L` (50) and `C` (100) to make 40 and 90.
- `C` can be placed before `D` (500) and `M` (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

Example 1:

```
Input: s = "III"
Output: 3
Explanation: III = 3.
```

Example 2:

```
Input: s = "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.
```

Example 3:

```
Input: s = "MCMXCIV"
Output: 1994
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.
```

Constraints:

- `1 <= s.length <= 15`
- `s` contains only the characters `('I', 'V', 'X', 'L', 'C', 'D', 'M')`.
- It is **guaranteed** that `s` is a valid roman numeral in the range `[1, 3999]`.

```
class Solution:
    def romanToInt(self, s: str) -> int:
        hashmap = {
            "I"     :        1,
 "V"             :5,
"X"             :10,
"L"           :50,
"C"             :100,
"D"             :500,
"M"             :1000,
        }
        total = 0
        for i in range(len(s)):
            cur = hashmap.get(s[i])
            if i != len(s)-1 and  cur < hashmap.get(s[i+1]) :
                total -= cur
            else:
                total += cur
        return total
```

14. Longest Common Prefix

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string `""`.

Example 1:

```
Input: strs = ["flower","flow","flight"]
Output: "fl"
```

Example 2:

```
Input: strs = ["dog","racecar","car"]
Output: ""
Explanation: There is no common prefix among the input strings.
```

Constraints:

- `1 <= strs.length <= 200`
- `0 <= strs[i].length <= 200`
- `strs[i]` consists of only lowercase English letters if it is non-empty.

```
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        n = len(strs)
        m = float("inf")
        for arr in strs:
            m = min(m,len(arr))
        ans = ""
        for i in range(m):
            ele = strs[0][i]
            for j in range(n):
                if ele != strs[j][i]:
                    return ans
            else:
                ans += ele
        return ans
```

151. Reverse Words in a String

Given an input string `s`, reverse the order of the **words**.

A **word** is defined as a sequence of non-space characters. The **words** in `s` will be separated by at least one space.

Return *a string of the words in reverse order concatenated by a single space.*

Note that `s` may contain leading or trailing spaces or multiple spaces between two words. The returned string should only have a single space separating the words. Do not include any extra spaces.

Example 1:

```
Input: s = "the sky is blue"
Output: "blue is sky the"
```

Example 2:

```
Input: s = "  hello world  "
Output: "world hello"
Explanation: Your reversed string should not contain leading or trailing spaces.
```

Example 3:

```
Input: s = "a good   example"
Output: "example good a"
Explanation: You need to reduce multiple spaces between two words to a single space i
```

Constraints:

- `1 <= s.length <= 10^4`
- `s` contains English letters (upper-case and lower-case), digits, and spaces `' '`.
- There is **at least one** word in `s`.

Follow-up: If the string data type is mutable in your language, can you solve it **in-place** with `O(1)` extra space?

```
class Solution:
    def reverseWords(self, s: str) -> str:
        result = []
        temp = ""
        n =len(s)-1
        for i in range(len(s)):
            if s[i]!=" ":
                temp += s[i]
            elif s[i] == " " and temp !="":
                result.append(temp)
                temp=""
            if i == n and temp != "":
                result.append(temp)
        return " ".join(result[::-1])
```

---

205. Isomorphic Strings

Given two strings `s` and `t`, *determine if they are isomorphic*.

Two strings `s` and `t` are isomorphic if the characters in `s` can be replaced to get `t`.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.

Example 1:

Input: s = "egg", t = "add"

Output: true

Explanation:

The strings `s` and `t` can be made identical by:

- Mapping `'e'` to `'a'`.
- Mapping `'g'` to `'d'`.

Example 2:

Input: s = "foo", t = "bar"

Output: false

Explanation:

The strings `s` and `t` can not be made identical as `'o'` needs to be mapped to both `'a'` and `'r'`.

Example 3:

Input: s = "paper", t = "title"

Output: true

Constraints:

- `1 <= s.length <= 5 * 10^4`
- `t.length == s.length`
- `s` and `t` consist of any valid ascii character.

```
class Solution:
    def isIsomorphic(self, s: str, t: str) -> bool:
        char_index_s = {}
        char_index_t = {}

        for i in range(len(s)):
            if s[i] not in char_index_s:
                char_index_s[s[i]] = i

            if t[i] not in char_index_t:
                char_index_t[t[i]] = i

            if char_index_s[s[i]] != char_index_t[t[i]]:
                return False

        return True
```

242. Valid Anagram

Given two strings `s` and `t`, return `true` if `t` is an anagram of `s`, and `false` otherwise.

Example 1:

Input: s = "anagram", t = "nagaram"

Output: true

Example 2:

Input: s = "rat", t = "car"

Output: false

Constraints:

- `1 <= s.length, t.length <= 5 * 10^4`
- `s` and `t` consist of lowercase English letters.

Follow up: What if the inputs contain Unicode characters? How would you adapt your solution to such a case?

```
from collections import Counter

class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        cnt_s = Counter(s)
        cnt_t = Counter(t)
        if len(s) != len(t):
            return False
        for key,val in cnt_s.items():
            if cnt_s.get(key) != cnt_t.get(key):
                return False
        return True
```

451. Sort Characters By Frequency

Given a string `s`, sort it in **decreasing order** based on the **frequency** of the characters. The **frequency** of a character is the number of times it appears in the string.

Return *the sorted string*. If there are multiple answers, return *any of them*.

Example 1:

```
Input: s = "tree"
Output: "eert"
Explanation: 'e' appears twice while 'r' and 't' both appear once.
So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.
```

Example 2:

```
Input: s = "cccaaa"
Output: "aaaccc"
Explanation: Both 'c' and 'a' appear three times, so both "cccaaa" and "aaaccc" are v
Note that "cacaca" is incorrect, as the same characters must be together.
```

Example 3:

```
Input: s = "Aabb"
Output: "bbAa"
Explanation: "bbaA" is also a valid answer, but "Aabb" is incorrect.
Note that 'A' and 'a' are treated as two different characters.
```

Constraints:

- `1 <= s.length <= 5 * 10^5`
- `s` consists of uppercase and lowercase English letters and digits.

```
# Use heap for better time complexity
from collections import Counter
class Solution:
    def frequencySort(self, s: str) -> str:
        cnt_s = Counter(s)
        res = ""
        for key,val in sorted(cnt_s.items(),key = lambda x:x[1],reverse = True):
            res+= key*val
        return res
```

796. Rotate String

Given two strings `s` and `goal`, return `true` *if and only if* `s` *can become* `goal` *after some number of* ***shifts*** *on* `s`.

A **shift** on `s` consists of moving the leftmost character of `s` to the rightmost position.

- For example, if `s = "abcde"`, then it will be `"bcdea"` after one shift.

Example 1:

```
Input: s = "abcde", goal = "cdeab"
Output: true
```

Example 2:

```
Input: s = "abcde", goal = "abced"
Output: false
```

Constraints:

- `1 <= s.length, goal.length <= 100`

- `s` and `goal` consist of lowercase English letters.

```
class Solution:
    def rotateString(self, s: str, goal: str) -> bool:
        if len(s) != len(goal):return False
        for i in range(len(goal)):
            if goal == s[i:]+s[:i]:
                return True
        return False
```

1021. Remove Outermost Parentheses

A valid parentheses string is either empty `""`, `"(" + A + ")"`, or `A + B`, where `A` and `B` are valid parentheses strings, and `+` represents string concatenation.

- For example, `""`, `"()"`, `"(())()"`, and `"(()(()))"` are all valid parentheses strings.

A valid parentheses string `s` is primitive if it is nonempty, and there does not exist a way to split it into `s = A + B`, with `A` and `B` nonempty valid parentheses strings.

Given a valid parentheses string `s`, consider its primitive decomposition: `s = P_1 + P_2 + ... + P_k`, where `P_i` are primitive valid parentheses strings.

Return `s` *after removing the outermost parentheses of every primitive string in the primitive decomposition of* `s`.

Example 1:

```
Input: s = "(()())(())"
Output: "()()()"
Explanation:
The input string is "(()())(())", with primitive decomposition "(()())" + "(())".
After removing outer parentheses of each part, this is "()()" + "()" = "()()()".
```

Example 2:

```
Input: s = "((()))(())(()(()))"
Output: "()()()()(())"
Explanation:
The input string is "((()))(())(()(()))", with primitive decomposition "((()))" + "(()
After removing outer parentheses of each part, this is "()()" + "()" + "()(())" = "()
```

Example 3:

```
Input: s = "()()"
Output: ""
Explanation:
The input string is "()()", with primitive decomposition "()" + "()".
After removing outer parentheses of each part, this is "" + "" = "".
```

Constraints:

- `1 <= s.length <= 10^5`
- `s[i]` is either `'('` or `')'`.
- `s` is a valid parentheses string.

```
class Solution:
    def removeOuterParentheses(self, s: str) -> str:
        ans = ""
        c = 0
        ind = 0
        for i in range(len(s)):
            if s[i] == "(":
                c+=1
            else:
                c-=1
                if c==0:
                    ans += s[ind+1:i]
                    ind = i+1
        return ans
```

1539. Kth Missing Positive Number

Given an array `arr` of positive integers sorted in a **strictly increasing order**, and an integer `k`.

Return *the* `k^th` ***positive*** *integer that is* ***missing*** *from this array.*

Example 1:

```
Input: arr = [2,3,4,7,11], k = 5
Output: 9
Explanation: The missing positive integers are [1,5,6,8,9,10,12,13,...]. The 5th miss
```

Example 2:

```
Input: arr = [1,2,3,4], k = 2
Output: 6
Explanation: The missing positive integers are [5,6,7,...]. The 2nd missing positive
```

Constraints:

- `1 <= arr.length <= 1000`
- `1 <= arr[i] <= 1000`
- `1 <= k <= 1000`
- `arr[i] < arr[j]` for `1 <= i < j <= arr.length`

Follow up:

Could you solve this problem in less than O(n) complexity?

1614. Maximum Nesting Depth of the Parentheses

Given a **valid parentheses string** `s`, return the **nesting depth** of `s`. The nesting depth is the **maximum** number of nested parentheses.

Example 1:

Input: s = "(1+(2*3)+((8)/4))+1"

Output: 3

Explanation:

Digit 8 is inside of 3 nested parentheses in the string.

Example 2:

Input: s = "(1)+((2))+(((3)))"

Output: 3

Explanation:

Digit 3 is inside of 3 nested parentheses in the string.

Example 3:

Input: s = "()(())((()()))"

Output: 3

Constraints:

- `1 <= s.length <= 100`
- `s` consists of digits `0-9` and characters `'+'`, `'-'`, `'*'`, `'/'`, `'('`, and `')'`.
- It is guaranteed that parentheses expression `s` is a VPS.

```
class Solution:
    def maxDepth(self, s: str) -> int:
        total = 0
        c = 0
        for i in s:
            if i == "(":
                c+=1
            elif i == ")":
                c-=1
            total = max(total,c)
        return total
```

1781. Sum of Beauty of All Substrings

The **beauty** of a string is the difference in frequencies between the most frequent and least frequent characters.

- For example, the beauty of `"abaacc"` is `3 - 1 = 2`.

Given a string `s`, return *the sum of **beauty** of all of its substrings.*

Example 1:

```
Input: s = "aabcb"
Output: 5
Explanation: The substrings with non-zero beauty are ["aab","aabc","aabcb","abcb","bc
```

Example 2:

```
Input: s = "aabcbaa"
Output: 17
```

Constraints:

- `1 <= s.length <= 500`
- `s` consists of only lowercase English letters.

```
class Solution:
    def beautySum(self, s: str) -> int:
        beutysum = 0
        n = len(s)

        for i in range(n):
            freq = defaultdict(int)
            for j in range(i,n):
                freq[s[j]]+=1
                max_freq = max(freq.values())
                min_freq = min(freq.values())
                beutysum+=(max_freq-min_freq)
        return beutysum
```

1903. Largest Odd Number in String

You are given a string `num`, representing a large integer. Return *the **largest-valued odd** integer (as a string) that is a **non-empty substring** of* `num`*, or an empty string* `""` *if no odd integer exists.*

A **substring** is a contiguous sequence of characters within a string.

Example 1:

```
Input: num = "52"
Output: "5"
Explanation: The only non-empty substrings are "5", "2", and "52". "5" is the only od
```

Example 2:

```
Input: num = "4206"
Output: ""
Explanation: There are no odd numbers in "4206".
```

Example 3:

```
Input: num = "35427"
Output: "35427"
Explanation: "35427" is already an odd number.
```

Constraints:

- `1 <= num.length <= 10^5`
- `num` only consists of digits and does not contain any leading zeros.

```
import sys
sys.set_int_max_str_digits(55555555)
class Solution:
    def largestOddNumber(self, nums: str) -> str:
        ans = ""
        lenn = 0
        for i in range(len(nums)-1,-1,-1):
            if int(nums[i]) % 2 == 1:
                return nums[:i+1]
        return ans
```