# 01. Grokking DSA

## Arrays

### 01. Running Sum of 1d Array

### Problem Statement

Given a one-dimensional array of integers, create a new array that represents the running sum of the original array.

The running sum at position `i` in the new array is calculated as the sum of all the numbers in the original array from the 0th index up to the `i`-th index (inclusive). Formally, the resulting array should be computed as follows: `result[i] = sum(nums[0] + nums[1] + ... + nums[i])` for each `i` from 0 to the length of the array minus one.

### Examples

#### Example 1

- **Input:** `[2, 3, 5, 1, 6]`
- **Expected Output:** `[2, 5, 10, 11, 17]`
- **Justification:**
  - For i=0: 2
  - For i=1: 2 + 3 = 5
  - For i=2: 2 + 3 + 5 = 10
  - For i=3: 2 + 3 + 5 + 1 = 11
  - For i=4: 2 + 3 + 5 + 1 + 6 = 17

#### Example 2

- **Input:** `[1, 1, 1, 1, 1]`
- **Expected Output:** `[1, 2, 3, 4, 5]`
- **Justification:** Each element is simply the sum of all preceding elements plus the current element.

#### Example 3

- **Input:** `[-1, 2, -3, 4, -5]`
- **Expected Output:** `[-1, 1, -2, 2, -3]`
- **Justification:** Negative numbers are also summed up in the same manner as positive ones.

## 02. Array Contains Duplicate

### Problem Statement

Given an integer array `nums`, return `true` if any value appears **at least twice** in the array, and return `false` if every element is distinct.

**Example 1**:

```
Input: nums= [1, 2, 3, 4]
Output: false
Explanation: There are no duplicates in the given array.
```

**Example 2**:

```
Input: nums= [1, 2, 3, 1]
Output: true
Explanation: '1' is repeating.
```

---

## 03. Left and Right Sum Differences

### Problem Statement

Given an input array of integers `nums`, find an integer array, let's call it `differenceArray,` of the same length as an input integer array.

Each element of `differenceArray`, i.e., `differenceArray[i],` should be calculated as follows: take the sum of all elements to the left of index `i` in array `nums` (denoted as `leftSum[i]`), and subtract it from the sum of all elements to the right of index `i` in array `nums` (denoted as `rightSum[i]`), taking the absolute value of the result. Formally:

**differenceArray[i] = | leftSum[i] - rightSum[i] |**

If there are no elements to the left/right of `i`, the corresponding sum should be taken as 0.

### Examples

**Example 1:**

- **Input:** `[2, 5, 1, 6]`
- **Expected Output:** `[12, 5, 1, 8]`
- **Explanation:**
    - For i=0: |(0) - (5+1+6)| = |0 - 12| = 12
    - For i=1: |(2) - (1+6)| = |2 - 7| = 5

    - For i=2: |(2+5) - (6)| = |7 - 6| = 1
    - For i=3: |(2+5+1) - (0)| = |8 - 0| = 8

**Example 2:**

- **Input:** `[3, 3, 3]`
- **Expected Output:** `[6, 0, 6]`
- **Explanation:**
    - For i=0: |(0) - (3+3)| = 6
    - For i=1: |(3) - (3)| = 0
    - For i=2: |(3+3) - (0)| = 6

**Example 3:**

- **Input:** `[1, 2, 3, 4, 5]`
- **Expected Output:** `[14, 11, 6, 1, 10]`
- **Explanation:**
    - Calculations for each index i will follow the above-mentioned logic.

---

## 04. Find the Highest Altitude

## Problem Statement

Given an array of integers representing a journey on a bike, wherein each number indicates a climb or descent in altitude, measured as a gain or loss of height.

The rider starts at altitude 0 and determines the highest altitude the biker reaches at any point during the journey.

### Examples

**Example 1**

- **Input:** `[-5, 1, 5, 0, -7]`
- **Expected Output:** `1`
- **Justification:** The altitude changes are `[-5, -4, 1, 1, -6]`, where `1` is the highest altitude reached.

**Example 2**

- **Input:** `[4, -3, 2, -1, -2]`
- **Expected Output:** `4`
- **Justification:** The altitude changes are `[4, 1, 3, 2, 0]`, where `4` is the highest altitude reached.

**Example 3**

- **Input:** `[2, 2, -3, -1, 2, 1, -5]`
- **Expected Output:** `4`
- **Justification:** The altitude changes are `[2, 4, 1, 0, 2, 3, -2]`, where `4` is the highest altitude reached.

---

## Matrix

## 05. Richest Customer Wealth

## Problem Statement

Given a 2D array `accounts` of `m x n` size, `accounts[j]` represents money in the `jth` bank account of the `ith` customer.

Return the wealth of the richest individual from a set of bank customers.

Imagine every customer has multiple bank accounts, with each account holding a certain amount of money. The total wealth of a customer is calculated by summing all the money across their multiple accounts.

### Examples

1. **Example 1:**
    - **Input:** `[[5,2,3],[0,6,7]]`
    - **Expected Output:** `13`

---

## 06. Matrix Diagonal Sum

## Problem Statement

Given a square matrix (2D array), calculate the sum of its two diagonals.

The two diagonals in consideration are the primary diagonal that spans from the top-left to the bottom-right and the secondary diagonal that spans from top-right to bottom-left. If a number is part of both diagonals (which occurs only for odd-sized matrices), it should be counted only once in the sum.

### Examples

1. **Example 1:**
    - **Input:**

```
[[1,2,3],
 [4,5,6],
 [7,8,9]]
```

- **Expected Output:** 25

---

## 07. Row with Maximum 1_s

## Problem Statement

Given a binary matrix that has dimensions , consisting of ones and zeros, determine the row that contains the highest number of ones and return two values: the zero-based index of this row and the actual count of ones it possesses.

If there is a tie, i.e., multiple rows contain the same maximum number of ones, we must select the row with the lowest index.

### Examples

#### Example 1:

- **Input:** `[[1, 0], [1, 1], [0, 1]]`
- **Expected Output:** `[1, 2]`
- **Justification:** The second row `[1, 1]` contains the most ones, so the output is `[1, 2]`.

---

## Stack

## 08. Balanced Parentheses

## Problem Statement

Given a string `s` containing `(`, `)`, `[`, `]`, `{`, and `}` characters. Determine if a given string of parentheses is balanced.

A string of parentheses is considered balanced if every opening parenthesis has a corresponding closing parenthesis in the correct order.

#### Example 1:

**Input:** String s = "{[()]}";
**Expected Output:** true
**Explanation:** The parentheses in this string are perfectly balanced. Every opening parenthesis '{', '[', '(' has a corresponding closing parenthesis '}', ']', ')' in the correct order.

---

## 09. Reverse a String

## Problem Statement

Given a string, write a function that uses a stack to reverse the string. The function should return the reversed string.

## Examples

**Example 1:**

```
Input: "Hello, World!"
Output: "!dlroW ,olleH"
```

**Example 2:**

```
Input: "OpenAI"
Output: "IAnepO"
```

**Example 3:**

```
Input: "Stacks are fun!"
Output: "!nuf era skcatS"
```

---

## 10. Decimal to Binary

## Problem Statement

Given a positive integer `n`, write a function that returns its binary equivalent as a string. The function should not use any in-built binary conversion function.

### Examples

**Example 1:**

```
Input: 2
Output: "10"
Explanation: The binary equivalent of 2 is 10.
```

**Example 2:**

```
Input: 7
Output: "111"
Explanation: The binary equivalent of 7 is 111.
```

**Example 3:**

```
Input: 18
Output: "10010"
```

```
Explanation: The binary equivalent of 18 is 10010.
```

## 11. Next Greater Element

### Problem Statement

Given an array, print the Next Greater Element (NGE) for every element.

The Next Greater Element for an element x is the first greater element on the right side of x in the array.

Elements for which no greater element exist, consider the next greater element as -1.

### Examples

**Example 1:**

```
Input: [4, 5, 2, 25]
Output: [5, 25, 25, -1]
```

**Example 1:**

```
Input: [13, 7, 6, 12]
Output: [-1, 12, 12, -1]
```

**Example 1:**

```
Input: [1, 2, 3, 4, 5]
Output: [2, 3, 4, 5, -1]
```

## 12. Sorting a Stack

### Problem Statement

Given a stack, sort it using only stack operations (push and pop).

You can use an additional temporary stack, but you may not copy the elements into any other data structure (such as an array). The values in the stack are to be sorted in descending order, with the largest elements on top.

### Examples

```
1. Input: [34, 3, 31, 98, 92, 23]
   Output: [3, 23, 31, 34, 92, 98]

2. Input: [4, 3, 2, 10, 12, 1, 5, 6]
   Output: [1, 2, 3, 4, 5, 6, 10, 12]
```

```
3. Input: [20, 10, -5, -1]
   Output: [-5, -1, 10, 20]
```

---

## 13. Simplify Path

### Problem Statement

Given an absolute file path in a Unix-style file system, simplify it by converting ".." to the previous directory and removing any "." or multiple slashes. The resulting string should represent the shortest absolute path.

#### Examples:

```
1.
   Input: "/a//b////c/d//././/.."
   Output: "/a/b/c"

2.
   Input: "/../"
   Output: "/"

3.
   Input: "/home//foo/"
   Output: "/home/foo"
```

---

## 14. Remove All Adjacent Duplicates In String

### Problem Statement

Give a string `s` and convert it into a valid string.

A string is considered valid if it does not have any two adjacent duplicate characters. The task is to remove the minimum number of characters from `s` such that it becomes valid. The removal of characters should be done in a way that no further removals are required.

### Examples

#### Example 1

- Input: `"abbaca"`
- Expected Output: `"ca"`
- Description: We remove 'b' from "abbaca" to get "aaca", then remove 'a' from "aaca" to get "ca"

## 15. Removing Stars From a String

### Problem Statement

Given a string `s`, where `*` represents a star. We can remove a star along with its closest non-star character to its left in a single operation.

The task is to perform as many such operations as possible until all stars have been removed and return the resultant string.

### Examples

#### Example 1

- Input: `"abc*de*f"`
- Expected Output: `"abdf"`
- Description: We remove `c` along with `*` to get `"abde*f"`, then remove `e` along with `*` to get `"abdf"`

## 16. Make The String Great

### Problem Statement

Given a string of English lowercase and uppercase letters, make the string "good" by removing two adjacent characters that are the same but in different cases.

Continue to do this until there are no more adjacent characters of the same letter but in different cases. An empty string is also considered "good".

### Examples

#### Example 1

- Input: `"AaBbCcDdEeff"`
- Output: `"ff"`

## **Queue**

## 17. Reverse a Queue

### Problem Statement

Given a `queue`, return it after reversing its elements.

**Examples:**

1. **Input:** Queue = [1, 2, 3, 4, 5]
   **Output:** [5, 4, 3, 2, 1]
   **Explanation:** The input queue elements are reversed.
2. **Input:** Queue = [10, 20, 30, 40, 50]
   **Output:** [50, 40, 30, 20, 10]
   **Explanation:** The input queue elements are reversed.
3. **Input:** Queue = [5, 7, 9]
   **Output:** [9, 7, 5]
   **Explanation:** The input queue elements are reversed.

---

## 18. Implement Stack using Queues

## Problem Statement

Implement a stack using two queues. The stack should support standard operations like push (add an element to the top of the stack) and pop (remove an element from the top of the stack).

**Examples:**

1. **Input:** Push operations: [1, 2, 3], Pop operations: 2
   **Output:** [1]
   **Explanation:** After pushing 1, 2, 3 the stack looks like [1, 2, 3]. Then we perform 2 pop operations, removing 3 and 2, so the output is [1].

---

## 19. Generate Binary Numbers from

## Problem Statement

Given an integer `N`, generate all binary numbers from 1 to N and return them as a list of strings.

**Examples:**

1. **Input:** N = 2
   **Output:** ["1", "10"]
   **Explanation:** The binary representation of 1 is "1", and the binary representation of 2 is "10".
2. **Input:** N = 3
   **Output:** ["1", "10", "11"]
   **Explanation:** The binary representation of 1 is "1", the binary representation of 2 is "10", and the binary representation of 3 is "11".

---

## 20. Palindrome Check using Queue

### Problem Statement

Given a string, determine if that string is a palindrome using a queue data structure.

A palindrome is a word, number, phrase, or other sequence of characters that reads the same forward and backward, ignoring spaces, punctuation, and capitalization.

**Examples:**

1. **Input:** "madam"
   **Output:** True
   **Explanation:** The word "madam" reads the same forwards and backwards.
2. **Input:** "openai"
   **Output:** False
   **Explanation:** The word "openai" does not read the same forwards and backwards.

---

## 21. Zigzag Iterator

### Problem Statement

Given two `1d` vectors, implement an iterator to return their elements alternately.

For example, given two `1d` vectors:

```
v1 = [1, 2]
v2 = [3, 4, 5, 6]
```

By calling `next()` repeatedly until `hasNext()` returns `false`, the order of elements returned by next should be: [1, 3, 2, 4, 5, 6].

Example 1:

```
i = ZigzagIterator([1, 2], [3, 4, 5, 6])
print(i.next())   # returns 1
print(i.next())   # returns 3
print(i.next())   # returns 2
print(i.next())   # returns 4
print(i.next())   # returns 5
print(i.next())   # returns 6
print(i.hasNext())  # returns False
```

---

## 22. Max of All Subarrays of Size *k*

## Problem Statement

Given an integer array and an integer `k`, design an algorithm to find the maximum for each and every contiguous subarray of size `k`.

**Examples:**

1. **Input**: array = `[1, 2, 3, 1, 4, 5, 2, 3, 6]`, k = 3
   **Output**: `[3, 3, 4, 5, 5, 5, 6]`
   **Description**: Here, subarray `1,2,3` has maximum 3, `2,3,1` has maximum 3, `3,1,4` has maximum 4, `1,4,5` has maximum 5, `4,5,2` has maximum 5, `5,2,3` has maximum 5, and `2,3,6` has maximum 6.

---

## LinkedList

## 23. Reverse Linked List

## Problem Statement:

Given the head of a singly linked list, your task is to reverse the list and return its head. The singly linked list has nodes, and each node contains an integer and a pointer to the next node. The last node in the list points to `null`, indicating the end of the list.

## Examples

### Example 1:

- **Input:** `[3, 5, 2]`
- **Expected Output:** `[2, 5, 3]`
- **Justification:** Reversing the list `[3, 5, 2]` gives us `[2, 5, 3]`.

---

## 24. Remove Duplicates from Sorted List

## Problem Statement:

Given a sorted linked list, remove all the duplicate elements to leave only distinct numbers. The linked list should remain sorted, and the modified list should be returned.

## Examples

### Example 1:

- **Input**: 1 -> 1 -> 2
- **Output**: 1 -> 2

- **Justification**: Since 1 is repeated, we remove the duplicate to leave a sorted list of unique numbers.

**Example 2:**

- **Input**: 1 -> 2 -> 2 -> 3
- **Output**: 1 -> 2 -> 3

---

## 25. Merge Two Sorted Lists

## Problem Statement

Given the head of two sorted linked lists, `l1` and `l2`.

Return a new sorted list created by merging together the nodes of the first two lists.

## Examples

1. **Example 1:**
   - **Input:**
     ```
     [1, 3, 5]
     [2, 4, 6]
     ```
   - **Expected Output:**
     ```
     [1, 2, 3, 4, 5, 6]
     ```
   - **Justification:** Merging the two sorted linked lists, node by node, results in a single sorted linked list containing all elements from both input lists.

---

## 26. Find if Doubly Linked List is a Palindrome

## Problem Statement

Given a doubly linked list, determine whether it is a palindrome.

A doubly linked list is a palindrome if it reads the same backward as forward, utilizing the previous and next pointers of the nodes.

## Examples

1. **Example 1:**
   - **Input:** 1 <-> 2 <-> 3 <-> 2 <-> 1
   - **Output:** true
   - **Justification:** The list reads the same backward as forward.

2. **Example 2:**
    - **Input:** 1 <-> 2 <-> 2 <-> 3
    - **Output:** false
    - **Justification:** Reading backward, the list is 3 <-> 2 <-> 2 <-> 1, which is not the same as reading forward.

---

## 27. Swap Nodes in Pairs

## Problem Statement

Given a singly linked list, swap every two adjacent nodes and return the head of the modified list.

If the total number of nodes in the list is odd, the last node remains in place. Every node in the linked list contains a single integer value.

## Examples

1. **Input**: `[1, 2, 3, 4]`
   **Output**: `[2, 1, 4, 3]`
   **Justification**: Pairs (1,2) and (3,4) are swapped.
2. **Input**: `[7, 8, 9, 10, 11]`
   **Output**: `[8, 7, 10, 9, 11]`
   **Justification**: Pairs (7,8) and (9,10) are swapped. 11 remains in its place as it has no adjacent node to swap with.

---

## Tree & Binary Search Tree

## 28. Maximum Depth (or Height) of Binary Tree

## Problem Statement

Determine the depth (or height) of a binary tree, which refers to the number of nodes along the longest path from the root node to the farthest leaf node. If the tree is empty, the depth is 0.

**Example:**

1. 
    - Input

```
     1
    / \
   2   3
```

```
 / \
4   5
```

**Output:** 3
**Explanation:** The longest path is `1->2->4` or `1->2->5` with 3 nodes.

2. 
   - **Input:**

     ```
     1
      \
       2
        \
         3
     ```

   - **Expected Output:** 3
   - **Justification:** There's only one path `1->2->3` with 3 nodes

---

## 29. Balanced Binary Tree

## Problem Statement

Determine if a binary tree is height-balanced.

A binary tree is considered height-balanced if, for each node, the difference in height between its left and right subtrees is no more than one.

**Examples:**

1. **Input:**

```
    3
   / \
  9  20
    /  \
   15   7
```

- **Expected Output:** true
- **Justification:** Every node in the tree has a left and right subtree depth difference of either 0 or 1.

2. **Input:**

```
        1
      /   \
     2     2
    / \   / \
   3   3 3   3
```

```
  / \         / \
 4   4       4   4
```

- **Expected Output:** true

## 30. Minimum Difference Between BST Nodes

## Problem Statement

Given a Binary Search Tree (BST), you are required to find the smallest difference between the values of any two different nodes.

In a BST, the nodes are arranged in a manner where the value of nodes on the left is less than or equal to the root, and the value of nodes on the right is greater than the root.

#### Example

*Example 1:*

- Input:

```
    4
   / \
  2   6
 / \
1   3
```

- Expected Output: 1
- Justification: The pairs (1,2), (2,3), or (3,4) have the smallest difference of 1.

## 31. Range Sum of BST

## Problem Statement

Given a Binary Search Tree (BST) and a range defined by two integers, L and R, calculate the sum of all the values of nodes that fall within this range. The node's value is inclusive within the range if and only if L <= node's value <= R.

**Examples:**

**Example 1:**

Input:

```
Tree:
   10
  /  \
 5    15
/ \    \
3   7   18
Range: [7, 15]
```

Expected Output: 32

Justification: The values that fall within the range [7, 15] are 7, 10, and 15. Their sum is 7 + 10 + 15 = 32.

---

## 32. Kth Smallest Element in a BST

## Problem Statement

Given a `root` node of the Binary Search Tree (BST) and integer 'k'. Return the Kth smallest element among all node values of the binary tree.

**Examples:**

1. **Example 1:**
   **Input:**

   ```
       8
      / \
     3    10
    / \     \
   1    6     14
      /   \   /
     4    7   13
   ```

   k = 4
   **Expected Output:** 6
   **Justification:** The in-order traversal of the tree is [1, 3, 4, 6, 7, 8, 10, 13, 14]. The 4th element in this sequence is 6.

---

## 33. Closest Binary Search Tree Value

## Problem Statement

Given a binary search tree (BST) and a target number, you need to find a node value in the BST that is closest to the given target. A BST is a tree where for every node, the values in the left subtree are smaller than the node, and the values in the right subtree are greater.

**Examples:**

*Example 1:*

Input:

```
    Tree:
       5
     /   \
    3      8
  / \    / \
 1    4 6    9
```

Target: 6.4

Expected Output: 6

Justification: The values 6 and 8 are the closest numbers to 6.4 in the tree. Among them, 6 is closer.

---

## 34. Merge Two Binary Trees

### Problem Statement

Given two binary trees, `root1` and `root2`, merge them into a single, new binary tree.

If two nodes from the given trees share the same position, their values should be summed up in the resulting tree. If a node exists in one tree but not in the other, the resulting tree should have a node at the same position with the value from the existing node.

### Example

Input:

```
Tree 1:         Tree 2:
     1                2
    / \              / \
   3    2           1    3
  /                  \    \
 5                    4    7
```

Output:

```
Merged Tree:
      3
     / \
    4   5
   / \   \
  5   4   7
```

---

## Hash Table (aka Hashmap or Dictionary)

### 35. First Non-repeating Character

### Problem Statement

Given a string, identify the position of the first character that appears only once in the string. If no such character exists, return -1.

#### Examples

1. **Example 1:**
   - Input: "apple"
   - Expected Output: 0
   - Justification: The first character 'a' appears only once in the string and is the first character.
2. **Example 2:**
   - Input: "abcab"
   - Expected Output: 2
   - Justification: The first character that appears only once is 'c' and its position is 2.

---

### 36. Largest Unique Number

### Problem Statement

Given an array of integers, identify the highest value that appears only once in the array. If no such number exists, return -1.

#### Examples:

1. **Example 1:**
   - Input: [5, 7, 3, 7, 5, 8]
   - Expected Output: 8

    - Justification: The number 8 is the highest value that appears only once in the array.

2. **Example 2:**
    - Input: [1, 2, 3, 2, 1, 4, 4]
    - Expected Output: 3
    - Justification: The number 3 is the highest value that appears only once in the array.
3. **Example 3:**
    - Input: [9, 9, 8, 8, 7, 7]
    - Expected Output: -1

---

## 37. Maximum Number of Balloons

## Problem Statement

Given a string, determine the maximum number of times the word "balloon" can be formed using the characters from the string. Each character in the string can be used only once.

**Examples:**

1. **Example 1:**
    - Input: "balloonballoon"
    - Expected Output: 2
    - Justification: The word "balloon" can be formed twice from the given string.
2. **Example 2:**
    - Input: "bbaall"
    - Expected Output: 0
    - Justification: The word "balloon" cannot be formed from the given string as we are missing the character 'o' twice.

---

## 38. Longest Palindrome

## Problem Statement:

Given a string, determine the length of the longest palindrome that can be constructed using the characters from the string. You don't need to return the palindrome itself, just its maximum possible length.

**Examples:**

1.

- **Input**: "applepie"
- **Expected Output**: 5
- **Justification**: The longest palindrome that can be constructed from the string is "pepep", which has a length of 5. There are are other palindromes too but they all will be of length 5.

2. 
   - **Input**: "aabbcc"
   - **Expected Output**: 6

---

## 39. Ransom Note

## Problem Statement

Given two strings, one representing a ransom note and the other representing the available letters from a magazine, determine if it's possible to construct the ransom note using only the letters from the magazine. Each letter from the magazine can be used only once.

**Examples:**

1. **Example 1:**
   - Input: Ransom Note = "hello", Magazine = "hellworld"
   - Expected Output: true
   - Justification: The word "hello" can be constructed from the letters in "hellworld".

---

## HashSet

## 40. Counting Elements

## Problem Statement

Given a list of integers, determine the count of numbers for which there exists another number in the list that is greater by exactly one unit.

In other words, for each number `x` in the list, if `x + 1` also exists in the list, then `x` is considered for the count.

**Examples:**

1. **Example 1:**
   - Input: [4, 3, 1, 5, 6]
   - Expected Output: 3

    - Justification: The numbers 4, 3, and 5 have 5, 4, and 6 respectively in the list, which are greater by exactly one unit.

---

## 41. Jewels and Stones

## Problem Statement

Given two strings. The first string represents types of jewels, where each character is a unique type of jewel. The second string represents stones you have, where each character represents a type of stone. Determine how many of the stones you have are also jewels.

**Examples:**

1. **Example 1:**
    - Input: Jewels = "abc", Stones = "aabbcc"
    - Expected Output: 6
    - Justification: All the stones are jewels.
2. **Example 2:**
    - Input: Jewels = "aA", Stones = "aAaZz"
    - Expected Output: 3
    - Justification: There are 2 'a' and 1 'A' stones that are jewels.

---

## 42. Unique Number of Occurrences

## Problem Statement

Given an array of integers, determine if the number of times each distinct integer appears in the array is unique.

In other words, the occurrences of each integer in the array should be distinct from the occurrences of every other integer.

**Examples:**

1. 
    - **Input:** [4, 5, 4, 6, 6, 6]
    - **Expected Output:** true
    - **Justification:** The number 4 appears 2 times, 5 appears 1 time, and 6 appears 3 times. All these occurrences (1, 2, 3) are unique.

---

## 43. Longest Substring Without Repeating Characters

## Problem Statement

Given a string, identify the length of its longest segment that contains distinct characters. In other words, find the maximum length of a substring that has no repeating characters.

**Examples:**

1. **Example 1:**
    - Input: "abcdaef"
    - Expected Output: 6
    - Justification: The longest segment with distinct characters is "bcdaef", which has a length of 6.

---

## <u>Heap</u>

## 44. Take Gifts From the Richest Pile

## Problem Statement

You're presented with several piles of gifts, with each pile containing a certain number of gifts. Every second, you'll engage in the following activity:

- Pick the pile that contains the highest number of gifts. If multiple piles share this distinction, you can select any of them.
- Compute the square root of the number of gifts in the selected pile, and then leave behind that many gifts (rounded down). Take all the other gifts from this pile.
- You'll do this for "k" seconds. The objective is to find out how many gifts would still remain after these "k" seconds.

---

## 45. Sort Characters By Frequency

## Problem Statement

Given a string, arrange its characters in descending order based on the frequency of each character. If two characters have the same frequency, their relative order in the output string can be arbitrary.

### Example

1. **Input:** "apple"
    - **Expected Output:** "ppale" or "ppela"
    - **Justification:** The character 'p' appears twice, while 'a', 'l', and 'e' each appear once. Thus, 'p' should appear before the other characters in the output.

---

## 46. Minimum Cost to Connect Sticks

## Problem Statement

Given a collection of sticks with different lengths. To combine any two sticks, there's a cost involved, which is equal to the sum of their lengths.

Connect all the sticks into a single one with the minimum possible cost. Remember, once two sticks are combined, they form a single stick whose length is the sum of the lengths of the two original sticks.

### Examples

1.
    - **Input**: [2, 4, 3]
    - **Expected Output**: 14
    - **Justification**: Combine sticks 2 and 3 for a cost of 5. Now, we have sticks [4,5]. Combine these at a cost of 9. Total cost = 5 + 9 = 14

---

## 47. Find the Median of a Number Stream

## Problem Statement

Design a class to calculate the median of a number stream. The class should have the following two methods:

1. `insertNum(int num)`: stores the number in the class
2. `findMedian()`: returns the median of all numbers inserted in the class

If the count of numbers inserted in the class is even, the median will be the average of the middle two numbers.

**Example 1**:

```
1. insertNum(3)
2. insertNum(1)
3. findMedian() -> output: 2
4. insertNum(5)
5. findMedian() -> output: 3
6. insertNum(4)
7. findMedian() -> output: 3.5
```

---

## Graph

## 48. Find if Path Exists in Graph

## Problem Statement

Given an undirected graph, represented as a list of edges. Each edge is illustrated as a pair of integers `[u, v]`, signifying that there's a mutual connection between node `u` and node `v`.

Given this graph, a starting node `start`, and a destination node `end`, your task is to ascertain if a path exists between the starting node and the destination node.

## Examples

1. **Example 1:**
   - Input: `4, [[0,1],[1,2],[2,3]], 0, 3`
   - Expected Output: `true`
   - Justification: There's a path from node 0 -> 1 -> 2 -> 3.

---

## 49. Number of Provinces

## Problem Statement

Imagine a country with several cities. Some cities have direct roads connecting them, while others might be connected through a sequence of intermediary cities. Using a matrix representation, if `matrix[i][j]` holds the value `1`, it indicates that city `i` is directly linked to city `j` and vice versa. If it holds `0`, then there's no direct link between them.

Determine the number of separate city clusters (or provinces).

Input:

```
matrix = [
  [1, 1, 0],
  [1, 1, 0],
  [0, 0, 1]
]
```

Output:
2

Explanation:
In this example, there are three cities: 0, 1, and 2.
Cities 0 and 1 are directly connected, forming one city cluster.
City 2 is not directly connected to any other city, so it forms its own city cluster.
Hence, the total number of city clusters (provinces) is 2.

Input:

```
matrix = [
  [1, 0, 0, 1],
  [0, 1, 1, 0],
  [0, 1, 1, 1],
  [1, 0, 1, 1]
]
```

Output:
1

Explanation:
In this example, there are four cities: 0, 1, 2, and 3.
All cities are connected to each other either directly or indirectly.
Thus, they form a single city cluster (province).
Hence, the total number of city clusters (provinces) is 1.

---

### 50. Minimum Number of Vertices to Reach All Nodes

### Problem Statement

Given a directed acyclic graph with `n` nodes labeled from `0` to `n-1`, determine the smallest number of initial nodes such that you can access all the nodes by traversing edges. Return these nodes.

### Examples

1. **Input:**
    - `n = 6`
    - `edges = [[0,1],[0,2],[2,5],[3,4],[4,2]]`

   **Expected Output:** `[0,3]`

   **Justification:** Starting from nodes 0 and 3, you can reach all other nodes in the graph. Starting from node 0, you can reach nodes 1, 2, and 5. Starting from node 3, you can reach nodes 4 and 2 (and by extension 5).

---

## Trie (aka Prefix Tree)

### 51. Implement Trie

### Problem Statement

Design and implement a Trie (also known as a Prefix Tree). A trie is a tree-like data structure that stores a dynamic set of strings, and is particularly useful for searching for words with a given prefix.

Implement the `Solution` class:

- `Solution()` Initializes the object.
- `void insert(word)` Inserts `word` into the trie, making it available for future searches.
- `bool search(word)` Checks if the word exists in the trie.
- `bool startsWith(word)` Checks if any word in the trie starts with the given prefix.

---

## 52. Index Pairs of a String

## Problem Statement

Given a string `text` and a list of strings `words`, identify all `[i, j]` index pairs such that the substring `text[i...j]` is in words.

These index pairs should be returned in ascending order, first by the start index, then by the end index. Find every occurrence of each word within the `text`, ensuring that overlapping occurrences are also identified.

### Examples

1. 
    - **Input:** text = `"bluebirdskyscraper"`, words = `["blue", "bird", "sky"]`
    - **Expected Output:** `[[0, 3], [4, 7], [8, 10]]`

---

## 53. Extra Characters in a String

## Problem Statement

Given a string `s` and an array of words `words`. Break string `s` into multiple non-overlapping substrings such that each substring should be part of the `words.` There are some characters left which are not part of any substring.

Return the minimum number of remaining characters in `s`, which are not part of any substring after string break-up.

### Examples

1. **Example 1:**
    - **Input:** `s = "amazingracecar"`, `words = ["race", "car"]`
    - **Expected Output:** `7`

---

## 54. Search Suggestions System

## Problem Statement

Given a list of distinct strings `products` and a string `searchWord`.

Determine a set of `product suggestions` after each character of the search word is typed. Every time a character is typed, return a `list` containing up to three product names from the `products` list that have the same prefix as the typed string.

If there are more than 3 matching products, return 3 lexicographically smallest products. These product names should be returned in lexicographical (alphabetical) order.

### Examples

---

## 55. Design Add and Search Words Data Structure

## Problem Statement

Design a data structure that supports the addition of new words and the ability to check if a string matches any previously added word.

Implement the `Solution` class:

- `Solution()` Initializes the object.
- `void addWord(word)` Inserts `word` into the data structure, making it available for future searches.
- `bool search(word)` Checks if there is any word in the data structure that matches `word`. The method returns `true` if such a match exists, otherwise returns `false`.

---

# Greedy Algorithm

## 56. Valid Palindrome

## Problem Statement

Given string `s`, determine whether it's possible to make a given string palindrome by removing at most one character.

A palindrome is a word or phrase that reads the same backward as forward.

### Examples

1. **Example 1:**
   - Input: `"racecar"`
   - Expected Output: `true`
   - Justification: The string is already a palindrome, so no removals are needed.
2. **Example 2:**

- Input: `"abccdba"`
- Expected Output: `true`
- Justification: Removing the character 'd' forms the palindrome "abccba".

---

## 57. Maximum Length of Pair Chain

## Problem Statement

Given a collection of pairs where each pair contains two elements [a, b], find the maximum length of a chain you can form using pairs.

A pair [a, b] can follow another pair [c, d] in the chain if b < c.

You can select pairs in any order and don't need to use all the given pairs.

### Examples

1. **Example 1**:
    - **Input**: `[[1,2], [3,4], [2,3]]`
    - **Expected Output**: `2`
    - **Justification**: The longest chain is `[1,2] -> [3,4]`. The chain `[1,2] -> [2,3]` is invalid because 2 is not smaller than 2.

---

## 58. Add to Make Parentheses Valid

## Problem Statement

Given a string str containing '(' and ')' characters, find the minimum number of parentheses that need to be added to a string of parentheses to make it valid.

A valid string of parentheses is one where each opening parenthesis '(' has a corresponding closing parenthesis ')' and vice versa. The goal is to determine the least amount of additions needed to achieve this balance.

### Examples

1. **Example 1:**
    - **Input:** "(()"
    - **Expected Output:** 1
    - **Justification:** The string has two opening parentheses and one closing parenthesis

---

## 59. Remove Duplicate Letters

## Problem Statement

Given a string `s`, remove all duplicate letters from the input string while maintaining the original order of the letters.

Additionally, the returned string should be the smallest in lexicographical order among all possible results.

**Examples:**

1. 
   - **Input:** "bbaac"
   - **Expected Output:** "bac"
   - **Justification:** Removing the extra 'b' and one 'a' from the original string gives 'bac', which is the smallest lexicographical string without duplicate letters.

---

## 60. Largest Palindromic Number

## Problem Statement

Given a string `s` containing 0 to 9 digits, create the largest possible palindromic number using the string characters.

A palindromic number reads the same backward as forward.

If it's not possible to form such a number using all digits of the given string, you can skip some of them.

### Examples

1. 
   - **Input:** "323211444"
   - **Expected Output:** "432141234"
   - **Justification:** This is the largest palindromic number that can be formed from the given digits.
2. 
   - **Input:** "998877"
   - **Expected Output:** "987789"

---

## 61. Removing Minimum and Maximum From Array

## Problem Statement

Determine the `minimum` number of `deletions` required to remove the smallest and the largest elements from an array of integers.

In each `deletion`, you are allowed to remove either the `first (leftmost)` or the `last (rightmost)` element of the array.

### Examples

1. **Example 1:**
   - Input: `[3, 2, 5, 1, 4]`
   - Expected Output: `3`
   - Justification: The smallest element is `1` and the largest is `5`. Removing `4`, `1`, and then `5` (or `5`, `4`, and then `1`) in three moves is the most efficient strategy.

---

## Divide and Conquer Algorithm

## 62. Majority Element

## Problem Statement

Given an array `nums` having an `n` elements, identify the element that appears the majority of the time, meaning more than n/2 times.

### Examples

1. **Example 1:**
   - Input: `[1, 2, 2, 3, 2]`
   - Expected Output: `2`
   - Justification: Here, '2' appears 3 times in a 5-element array, making it the majority element.
2. **Example 2:**
   - Input: `[4, 4, 4, 4, 7, 4, 4]`
   - Expected Output: `4`
   - Justification: '4' is the majority element as it appears 5 out of 7 times.
3. **Example 3:**
   - Input: `[9, 9, 1, 1, 9, 1, 9, 9]`
   - Expected Output: `9`

---

## 63. Longest Nice Substring

## Problem Statement

Given a string `str`, return the longest `nice` substring of a given string.

A substring is considered nice if for every lowercase letter in the substring, its uppercase counterpart is also present, and vice versa.

If no such string exists, return an empty string.

### Examples

1. **Example 1:**
   - Input: `"BbCcXxY"`
   - Expected Output: `"BbCcXx"`
   - Justification: Here, `"BbCcXx"` is the longest substring where each letter's uppercase and lowercase forms are present.

---

## 64. Sort List

## Problem Statement

Given a `head` of the linked list, return the list after sorting it in ascending order.

### Examples

1. **Example 1:**
   - **Input:** `[3, 1, 2]`
   - **Expected Output:** `[1, 2, 3]`
   - **Justification:** The list is sorted in ascending order, with `1` coming before `2`, and `2` before `3`.
2. **Example 2:**
   - **Input:** `[4]`
   - **Expected Output:** `[4]`
   - **Justification:** A list with a single element is already sorted.
3. **Example 3:**
   - **Input:** `[9, 8, 7, 6, 5, 4, 3, 2, 1]`
   - **Expected Output:** `[1, 2, 3, 4, 5, 6, 7, 8, 9]`

---

## 65. Beautiful Array

## Problem Statement

Given a positive integer `N`, construct a beautiful array of size `N` containing the number `[1, N]` .

An array is beneficial if it follows the conditions below.

- If for any three indices `i`, `j,` `k` (with `i < j < k`), `A[j] * 2` is not equal to `A[i] + A[k].`

### Examples

#### Example 1

- **Input:** `4`
- **Expected Output:** `[2, 1, 4, 3]`
- **Justification:** In this array, no combination of `i`, `j`, `k` (where `i < j < k`) exists such that `2 * A[j] = A[i] + A[k]`.

#### Example 2

- **Input:** `3`
- **Expected Output:** `[1, 3, 2]`

---

## 66. Median of Two Sorted Arrays

## Problem Statement

Given two sorted arrays, `nums1` and `nums2` of different sizes in the ascending order, return the median of two sorted arrays after merging them.

The median is the middle value in an ordered set, or the average of the two middle values if the set has an even number of elements.

### Examples

1. **Example 1:**
    - **Input:** `[1, 3, 5]` and `[2, 4, 6]`
    - **Expected Output:** `3.5`
    - **Justification:** When merged, the array becomes `[1, 2, 3, 4, 5, 6]`. The median is the average of the middle two values, `(3 + 4) / 2 = 3.5`.

---

---