# Programs asked in Companies during Technical Rounds

| Sr. | Program |
|---|---|
| 1 | **Dominant Number in an Array**<br><br>A dominant number is a number whose frequency of occurrence in an array is strictly greater than the count of unique elements present in that array.<br>You are given an integer array.<br>Your task is to identify and return all dominant numbers from the array. |
| | **Input:** arr = [1, 2, 3, 1, 1, 4, 5, 1, 1, 1]<br>**Output**: [1]<br><br>**Input:** arr = {1, 1, 1, 2, 2, 2, 3}<br>**Output:** [1, 2] |
| 2 | **Find Maximum, Second Maximum, Minimum, and Second Minimum from an Array**<br><br>This program takes an array of integers as input and identifies:<br>• the **Maximum** element,<br>• the **Second Maximum** element,<br>• the **Minimum** element, and<br>• the **Second Minimum** element.<br>The array must contain **at least four distinct elements** to determine all four values correctly. |
| | **Input:** Array = [10, 5, 20, 8, 15]<br>**Output**: Maximum = 20<br>Second Maximum = 15<br>Minimum = 5<br>Second Minimum = 8<br><br>**Input:** Array = [3, 7, 1, 9, 4]<br>**Output**: Maximum = 9<br>Second Maximum = 7<br>Minimum = 1<br>Second Minimum = 3 |
| 3 | **Time Problem**<br><br>Create a Time class to represent time in **hours, minutes, and seconds**.<br>• Initialize **Hr : Min : Sec** using a **constructor**.<br>• Create an **AddSeconds(seconds)** method to add the given seconds to the initialized time.<br>• Create a **SubtractSeconds(seconds)** method to subtract the given seconds from the initial time.<br>• Ensure proper time adjustment (60 seconds = 1 minute, 60 minutes = 1 hour).<br>• Print the time in **Hr:Min:Sec** format. |
| | **Input:**<br>Initial Time = 02 : 30 : 45<br>Add Seconds = 50<br>Subtract Seconds = 30<br>**Output**:<br>Time after adding seconds = 02 : 31 : 35<br>Time after subtracting seconds = 02 : 31 : 05<br><br>**Input:**<br>Initial Time = 01 : 15 : 20 |

| | |
|---|---|
| | Add Seconds = 100<br>Subtract Seconds = 40<br>**Output**:<br>Time after adding seconds = 01 : 17 : 00<br>Time after subtracting seconds = 01 : 16 : 20 |
| **4** | **Maximum Price Selection Problem**<br><br>You are given a list of items along with their prices and categories.<br>The values **N** and **K** are also provided:<br>• **N** → Maximum number of items that can be selected<br>• **K** → Maximum number of times the same Category can be selected repeatedly<br>Each item is represented in the format:<br>price:category<br>The program should select items such that:<br>• At most **N** items are chosen<br>• Each item can be chosen at most **K** times<br>• The **total price is maximum**<br>Finally, return the **maximum total price**. |
| | **Input:**<br>N = 4<br>K = 2<br>Input Array = ["500:Electronics", "400:Electronics", "300:Electronics","200:Sports", "150:Beauty", "100:Beauty"]<br>**Output**:<br>Maximum Price = 1250<br><br>**Input:**<br>N = 3<br>K = 1<br>Input Array = ["600:Electronics", "350:Sports", "250:Beauty", "150:Books"]<br>**Output**:<br>Maximum Price = 1200 |
| **5** | **Sentence Similarity Ranking Based on a Secret Word**<br><br>You are given a **secret code word** and a list of **sentences**.<br>Your task is to analyse each sentence and calculate how closely it matches the secret word based on the following conditions:<br>**Matching Conditions**<br>For each word in a sentence, compare it with the secret word:<br>1. **Length must be the same** as the secret word<br>2. **Characters must match**<br>3. **Positions of matching characters must be the same**<br>4. Comparison is **case-insensitive**<br>Each matching character at the correct position increases the similarity score by **1**.<br>**Overall Task**<br>• Calculate the **total similarity score** for each sentence<br>• Store all similarity scores in an array<br>• Sort the sentences in **descending order of similarity**<br>• Return the **final decoded string** by joining sentences according to their similarity rank |
| | **Input:**<br>Secret Word = "cat" |

Sentences = [ "I love my pet",
"A car ran fast",
"He bought a hat",
"The cat sat on the mat"]

| Word | Compared with "cat" | Matching Score |
|---|---|---|
| The | No match | 0 |
| Cat | All letters & positions match | 3 |
| Sat | 'a' and 't' match in position | 2 |
| On | Length mismatch | 0 |
| The | No match | 0 |
| Mat | 'a' and 't' match in position | 2 |

Total Similarity = 7
Similarly, similarity scores for all sentences: [7, 3, 2, 1]

**Output**:
Final Decoded String = "The cat sat on the mat A car ran fast He bought a hat I love my pet"

6 **Find All Harmonious Substrings**

**Objective**

The goal of the program is to **generate a collection** of all possible substrings from a given input string that satisfy the specific "Harmonious" criteria.

**The "Harmonious" Criteria**

For any individual substring to be included in the final list, the frequency of its characters must adhere to the following rule:

**The difference between the maximum character frequency and the minimum character frequency within that substring must be 0 or 1.**

**Selection Logic:**

1. Break the input string into every possible contiguous substring.
2. For each substring, calculate Max_Frequency - Min_Frequency.
3. If the result is <=1, **add it to the output list**.

**Input:** String = "ABA"
**Output**: [ "A", "B", "A", "AB", "BA", "ABA" ]

**Input:** String = "AAAB"
**Output**: [ "A", "A", "A", "B", "AA", "AA", "AB", "AAA", "AAB" ]

7 **Simqual String Problem**

Two strings are called **Simqual strings** if they satisfy the following conditions:

1. Both strings must have the **same length**
2. Both strings must contain the **same characters**
3. The **frequency of each character must be equal** in both strings
4. **Order of characters does not matter**

The program should check whether the given two strings are **Simqual** or **Not Simqual**.

**Input:** String 1 = "listen"
String 2 = "silent"
**Output**: Strings are Simqual

**Input:** String 1 = "hello"
String 2 = "world"

| | |
|---|---|
| | **Output:** Strings are NOT Simqual |
| 8 | **Find the Second Largest Number (Without Sorting)**<br><br>Write a program that takes an array of integers as input and finds the second largest number in the array.<br>1. Sorting (built-in or manual) is not allowed.<br>2. Solve using scanning and comparisons only. |
| | **Input:** [5, 3, 8, 1, 2]<br>**Output:** Second largest number is: 5<br><br>**Input:** [7, 3, 8, 1, 5]<br>**Output:** Second largest number is: 7 |
| 9 | **Coin Change Problem**<br><br>Write a program that, given an amount and available coin denominations, calculates the minimum number of coins required to make that amount. |
| | **Input:** Amount = 27<br>Denominations = {1, 2, 5, 10}<br>**Output:** Minimum coins needed: 4 (Breakdown: 10 + 10 + 5 + 2)<br><br>**Input:** Amount = 23<br>Denominations = {1, 2, 7}<br>**Output:** Minimum coins needed = 5(Breakdown = 7 + 7 + 7 + 1 + 1) |
| 10 | **Check for Balanced Brackets**<br><br>Write a program to check whether a given **expression containing brackets** is **balanced** or **not**.<br>**Allowed Brackets**<br>( ) { } [ ]<br>**Balancing Rules**<br>• Every **opening bracket** must have a corresponding **closing bracket**<br>• Brackets must be **closed in the correct order**<br>• Mixed types of brackets are allowed<br>The program should return whether the expression is **Balanced** or **Not Balanced**. |
| | **Input:** Expression = "{[()]}"<br>**Output:** Balanced<br><br>**Input:** Expression = "{[(])}"<br>**Output:** Not Balanced |
| 11 | **Rotate an N×M Matrix by 90 Degrees Clockwise**<br><br>Write a program to rotate a **dynamic matrix of size N×M** (not necessarily a square matrix) by **90 degrees clockwise**.<br>**Rotation Rules**<br>• The number of rows and columns may be different.<br>• After rotation, the matrix size becomes **M×N**.<br>• Elements must be repositioned correctly to represent a **90° clockwise rotation**. |
| | **Input:** 1 2 3<br>4 5 6<br>7 8 9<br>**Output:** 7 4 1 |

| | |
|---|---|
| | 8 5 2<br>9 6 3<br>**Input:** 1 2 3 4<br>5 6 7 8<br>**Output:** 5 1<br>6 2<br>7 3<br>8 4 |
| **12** | **Form the Largest Number from an Array**<br><br>You are given a list of **non-negative integers**.<br>Arrange them in such an order that they form the **largest possible number**.<br>**Important Rules**<br>• Numbers must be **arranged, not added**<br>• The final result can be **very large**, so return it as a **string**<br>• Concatenate numbers in an order that gives the **maximum value** |
| | **Input:** nums = [10, 2]<br>**Output:** "210"<br><br>**Input:** nums = [3, 30, 34, 5, 9]<br>**Output:** "9534330" |
| **13** | **Jump Game – Reach the Last Index**<br><br>You are given an integer array **nums** where each element represents the **maximum jump length** from that position.<br>• You start at the **first index (index 0)**.<br>• From any index, you can jump forward up to the number of steps given by the value at that index.<br>• Determine whether it is possible to **reach the last index** of the array.<br>The program should return:<br>• **True** if the last index can be reached<br>• **False** otherwise |
| | **Input:** nums = [2, 3, 1, 1, 4]<br>**Explanation**<br>• Jump from index 0, Value (2) → index 2, Value(1)<br>• Jump from index 2, Value (1) → index 3, Value(1)<br>• Jump from index 3, Value (1) → index 4, **(Last Index)**<br>**Output:** True<br><br>**Input:** nums = [3, 2, 1, 0, 4]<br>**Explanation**<br>• You reach index 3, but the value is 0<br>• No further jump is possible<br>**Output:** False |
| **14** | **Best Time to Buy and Sell Stock**<br><br>You are given an integer array **prices**, where prices[i] represents the price of a stock on the **i$^{th}$ day**.<br>**Rules**<br>• You may **buy and/or sell** the stock on any day.<br>• You can hold **at most one share** of the stock at any time.<br>• You are allowed to **buy and sell on the same day**. |

- You may complete **multiple transactions**.

The task is to calculate and return the **maximum profit** that can be achieved.

**Input:** Prices = [7, 1, 5, 3, 6, 4]
**Explanation**

- Buy on day 2 at price 1, sell on day 3 at price 5 → profit = 4
- Buy on day 4 at price 3, sell on day 5 at price 6 → profit = 3
- Total profit = 4 + 3 = 7

**Output:** 7

**Input: P**rices = [1, 2, 3, 4, 5]
**Explanation**

- Buy on day 1 and sell on day 5
- Total profit = (2−1) + (3−2) + (4−3) + (5−4) = 4

**Output:** 4

**15** **Maximum Students Taking Exam**

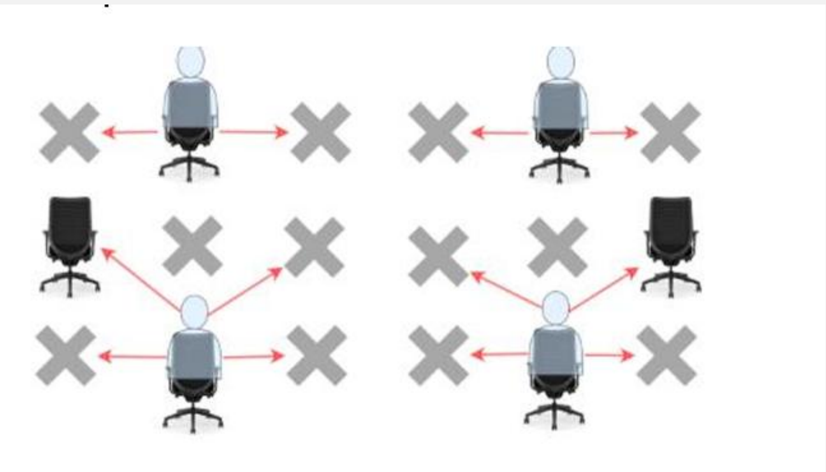
Given an **m × n matrix** seats representing a classroom:

- A seat is **broken** → '#'
- A seat is **available** → '.'

**Rules for placing students:**

1. Students can **see the answers** of those sitting:
   - Left and right
   - Upper-left and upper-right
2. Students **cannot see** the answers of those directly in front or behind.
3. Students can only be placed on **available seats** ('.').

Return the **maximum number of students** that can take the exam **without cheating**.



**Input:** seats = [ ["#",".","#","#",".","#"],
["." ,"#","#","#","#","."],
["#",".","#","#",".","#"] ]
**Output:** 4

**Input:** seats = [ [".","#"],
["#","#"],
["#","."],
["#","#"],
[".","#"] ]

**Output:** 3

**Input:** seats = [ ["#",".",".",".","#"],
```
                     [".","#",".","#","."],
                     [".",".","#",".","."],
                     [".","#",".","#","."],
                     ["#",".",".",".","#"] ]
```

**Output:** 10

### 16 Find the Repeated Number

You are given an integer array **nums** containing **n + 1** elements.
Each element in the array is in the range **[1, n]**.

- There is **exactly one number** that is repeated in the array.
- The repeated number may appear **more than once**.
- Your task is to **identify and return** the repeated number.

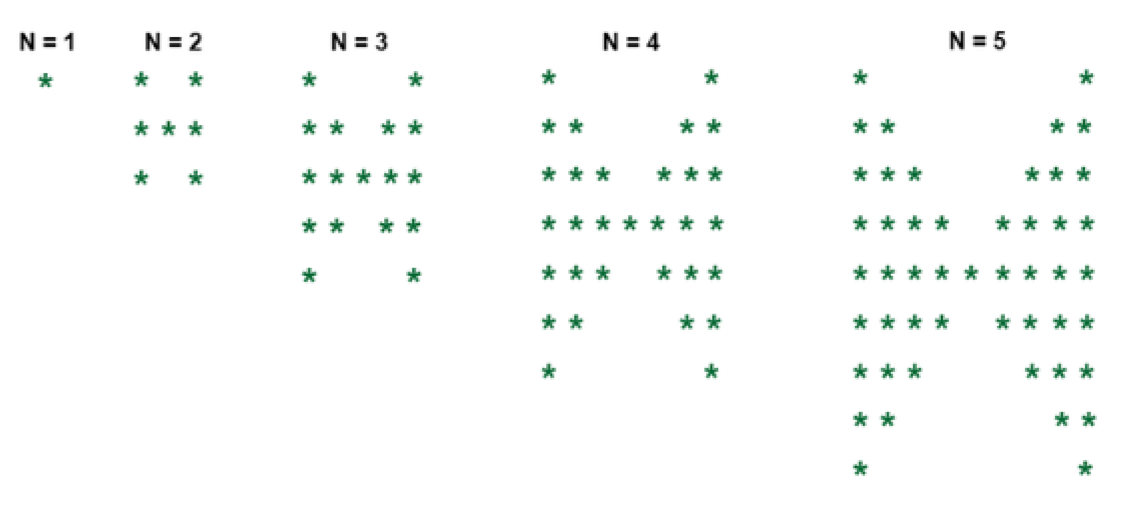
The solution should run in **O(n) time complexity**.

**Input:** nums = [1, 3, 4, 2, 2]
**Output:** 2

**Input:** nums = [3, 1, 3, 4, 2]
**Output:** 3

### 17 Butterfly Pattern

The butterfly pattern is a **symmetric star (*) pattern** that looks like the shape of a butterfly.
It is usually printed using **two mirrored triangles**.



```
N = 1    N = 2        N = 3              N = 4                        N = 5
  *      *   *      *       *      *           *          *                   *
         * * *      * *   * *      * *       * *          * *               * *
         *   *      * * * * *      * * *   * * *          * * *           * * *
                    * *   * *      * * * * * * *          * * * *       * * * *
                    *       *      * * *   * * *          * * * * * * * * * * *
                                   * *       * *          * * * *       * * * *
                                   *           *          * * *           * * *
                                                          * *               * *
                                                          *                   *
```

### 18 Check Perfect Number

A **perfect number** is a positive integer that is **equal to the sum of its positive divisors**, excluding the number itself.
Given an integer **num**, determine whether it is a **perfect number**.

- Return **true** if num is a perfect number
- Return **false** otherwise

The solution should run in **O(√n) time complexity** and **O(1) Space Complexity**

| | |
|---|---|
| | **Input:** num = 28<br>**Output**: true<br>**Input:** num = 7<br>**Output**: false |
| **19** | **Square Root of a Number (Sqrt(x))**<br><br>Given a **non-negative integer** x, write a program to compute and return the **square root of x**, rounded **down to the nearest integer**.<br>**Constraints**<br>• Do **not** use any built-in square root or exponent functions/operators.<br>• The result must be the **floor value** of √x.<br>The solution should run in **O(log x) time complexity** and **O(1) Space Complexity** |
| | **Input:** x = 4<br>**Output:** 2<br><br>**Input:** x = 8<br>**Output:** 2<br>Explanation: sqrt(8) = 2.828..., round down to 2 |
| **20** | **Check Prime Number**<br><br>A **prime number** is a natural number greater than **1** that has **exactly two positive divisors**: **1** and the number itself.<br>Given an integer **x**, write a program to determine whether the number is **prime** or **not**.<br>• Return **true** if x is a prime number<br>• Return **false** otherwise<br>The solution should run in **O(n) time complexity** and **O(1) Space Complexity** |
| | **Input:** x = 7<br>**Output:** true<br><br>**Input:** x = 10<br>**Output:** false |
| **21** | **Factorial Trailing Zeroes**<br><br>Given a non-negative integer **n**, write a program to calculate the **number of trailing zeroes** in **n!** (n factorial).<br>The factorial of a number is defined as:<br>n! = n × (n − 1) × (n − 2) × ... × 2 × 1<br>Trailing zeroes are the **continuous zeros at the end** of the factorial result.<br>The solution should run in **O(n) time complexity** and **O(1) Space Complexity** |
| | **Input:** n = 3<br>**Output:** 0<br>Explanation: 3!=6<br><br>**Input:** n = 5<br>**Output:** 1<br>Explanation: 5! = 120, one trailing zero<br><br>**Input:** n = 0<br>**Output:** 0 |
| **22** | **Next Greater Element** |

Given an array of integers **arr[]**, find the **Next Greater Element (NGE)** for each element in the array, in the **order of their appearance**.
The **Next Greater Element** for an element is defined as:

- The **first element to the right** of it that is **greater** than the current element.
- If no such element exists, return **-1** for that position.

**Input:** arr[] = [1, 3, 2, 4]
**Output:** [3, 4, 4, -1]
**Explanation**

- Next greater of 1 → 3
- Next greater of 3 → 4
- Next greater of 2 → 4
- No greater element after 4 → -1

**Input:** arr[] = [6, 8, 0, 1, 3]
**Output:** [8, -1, 1, 3, -1]
**Explanation**

- Next greater of 6 → 8
- No greater element after 8 → -1
- Next greater of 0 → 1
- Next greater of 1 → 3
- No greater element after 3 → -1

23 **Display Fibonacci Sequence**

Write a C program to display the **Fibonacci sequence** up to **n terms**.
The Fibonacci sequence is defined as a series of numbers where:

- The first two terms are **0** and **1**
- Each subsequent term is the **sum of the previous two terms**

**Input:** n = 5
**Output:** 0 1 1 2 3

24 **Find Nth Factor of a Number**

Given a positive integer **n** and an integer **k**, write a program to find the **$k^{th}$ factor** of **n**.

- Factors of a number are the **positive integers that divide the number exactly**.
- Factors are considered in **ascending order**.
- If the **$k^{th}$ factor does not exist**, return **-1**.

**Input:** n = 12, k = 1
**Output:** 1

**Input:** n = 12, k = 5
**Output:** 6

**Input:** n = 12, k = 3
**Output:** 3

25 **Arrange Array Elements in Ascending and Descending Order**

Given an array of integers, write a program to **sort the array elements** in:

- **Ascending order** (smallest to largest)
- **Descending order** (largest to smallest)

The program should display the sorted array in both orders.

**Input:** Array = [4, 2, 7, 1]
**Output:** Ascending Order: [1, 2, 4, 7]

| | |
|---|---|
| | Descending Order: [7, 4, 2, 1]<br><br>**Input:** Array = [10, 5, 3, 8, 2]<br>**Output:** Ascending Order: [2, 3, 5, 8, 10]<br>Descending Order: [10, 8, 5, 3, 2] |
| **26** | **Simple Calculator**<br><br>This program acts like a **basic calculator**, similar to the one on a mobile phone.<br>It is designed in such a way that **anyone—even a person who does not know coding—can understand how it works**.<br>The calculator:<br>• Takes **two numbers** from the user<br>• Asks the user **what calculation to perform**<br>• Performs the calculation<br>• Shows the **final answer** |
| | **Input:** First number: 8<br>Second number: 4<br>Operation: *<br>**Output:** 32 |
| **27** | **Find Two Elements with Given Sum**<br><br>Given an array of numbers and a **target value**, find **two numbers in the array** whose **sum equals the target**.<br>• Return the **indices (positions) of these two numbers**.<br>• Indices start from **0**.<br>• Assume there is **exactly one solution**. |
| | **Input:** arr = [2, 4, 10, 6, 7]<br>target = 6<br>**Output:** [0, 1]<br><br>**Input:** arr = [3, 5, 1, 7]<br>target = 8<br>**Output:** [2, 3] |
| **28** | **Rearrange Array Alternately**<br><br>Given an array of **even length**, rearrange the elements such that:<br>• Elements from the **first half** and **second half** are placed **alternately**.<br>• The **first element** comes from the **first half**, the next from the **second half**, and so on. |
| | **Input:** [1, 2, 3, 4, 5, 6]<br>**Output:** [1, 4, 2, 5, 3, 6]<br><br>**Input:** [10, 20, 30, 40]<br>**Output:** [10, 30, 20, 40] |
| **29** | **Happy Number**<br><br>A **Happy Number** is a positive integer that eventually reaches **1** when repeatedly replaced by the **sum of the squares of its digits**.<br>• If the process reaches **1**, the number is **Happy**.<br>• If the process enters a **loop** that does not include 1, the number is **not Happy**.<br>Write a program to **determine whether a given number is a Happy Number**. |
| | **Input:** 19 |

| | |
|---|---|
| | **Explanation:** 19 → $1^2 + 9^2 = 82$<br>82 → $8^2 + 2^2 = 68$<br>68 → $6^2 + 8^2 = 100$<br>100 → $1^2 + 0^2 + 0^2 = \mathbf{1}$<br>**Output:** Happy Number<br><br>**Input:** n = 4<br>**Explanation:** 4 → $4^2 = 16$<br>16 → $1^2 + 6^2 = 37$<br>37 → $3^2 + 7^2 = 58$<br>58 → $5^2 + 8^2 = 89$<br>89 → $8^2 + 9^2 = 145$<br>145 → $1^2 + 4^2 + 5^2 = 42$<br>42 → $4^2 + 2^2 = 20$<br>20 → $2^2 + 0^2 = 4$<br>**Output:** Not a Happy Number |
| **30** | **Largest 5-Digit Prime Containing a Specific Digit Twice**<br>**Program Definition**<br>Write a program to find the **largest 5-digit prime number** that contains a given digit **exactly twice**.<br>**Constraints:**<br>• The digit n is between **0 and 9**.<br>• Consider numbers in **descending order** from 99999 to 10000.<br>• For each number:<br>1. Check if the number is **prime**.<br>2. Check if the number contains the digit n **exactly two times**.<br>• Print the **first number** that satisfies both conditions. |
| | **Input:** n = 3<br>**Output:** 99833<br><br>**Input:** n = 7<br>**Output:** 99787 |
| **31** | **LEGO Tower (Alternate Color Stack)**<br>**Program Definition:**<br>You are given two arrays red[] and blue[]. Each index represents one LEGO block position, where red[i] is the height of a red block and blue[i] is the height of a blue block at the same position.<br>Your task is to find the **maximum height of a tower** that can be built by stacking blocks using the following rules:<br>1. You can start the tower with **either red or blue** block.<br>2. The colors of the blocks must **alternate** (red → blue → red → … or blue → red → blue → …).<br>3. From each index, **only one block** can be used (either red[i] or blue[i]), and each block can be used **only once**. |
| | **Input:** red = [3, 5, 2]<br>blue = [4, 1, 6]<br>**Output:** 14<br>**Explanation**<br>• Start with blue (4), then red (5), then blue (6).<br>• otal height = 4 + 5 + 6 = **14**<br><br>**Input:** red = [2, 7, 4, 1]<br>blue = [3, 5, 6, 2] |

| | |
|---|---|
| | **Output: 18**<br>**Explanation**<br>• Start with **red** and alternate colors:<br>• Red (2)<br>• Blue (5)<br>• Red (4)<br>• Blue (2)<br>• Total height = 2 + 5 + 4 + 2 = **18** |
| **32** | **Pivot Index**<br><br>Given an array of integers, write a program to find the **pivot index**.<br>A **pivot index** is an index i in the array such that:<br>1. The **sum of all elements to the left** of i is **equal** to the sum of all elements to the right of i.<br>2. The **element at the pivot index itself** is **not included** in either sum.<br>• If such an index exists, return the **first pivot index** found.<br>• If no pivot index exists, print a message indicating that it was **not found**. |
| | **Input:** arr = [1, 7, 3, 6, 5, 6]<br>**Output:** 3<br>**Explanation**<br>• Pivot index = 3<br>• Left sum = 1 + 7 + 3 = 11<br>• Right sum = 5 + 6 = 11<br>• Both sums are equal, and pivot element 6 is excluded.<br><br>**Input:** arr = [1, 2, 3]<br>**Output:** Pivot index not found<br>**Explanation**<br>• No index satisfies the left and right sum condition. |
| **33** | **Electricity Bill Calculator**<br><br>Write a program to calculate the **electricity bill** based on the number of units consumed, using the following **slab rates**:<br>1. **First 30 units:** ₹2.70 per unit<br>2. **Next 70 units (31–100):** ₹4.00 per unit<br>3. **Next 100 units (101–200):** ₹5.40 per unit<br>4. **Above 200 units:** ₹6.40 per unit<br>**Additional Rule:**<br>• If the total calculated bill is **less than ₹125**, set the bill to a **minimum charge of ₹125**.<br>The program should:<br>• Take the **number of units consumed** as input.<br>• Calculate the **total bill** according to the slab rates.<br>• Apply the **minimum bill rule**.<br>• Print the **final amount** to be paid. |
| | **Input:** Units consumed = 50<br>**Output:** Bill amount = ₹161.0<br><br>**Input:** Units consumed = 20<br>**Output:** Bill amount = ₹125.0 |
| **34** | **Print a Pyramid Star Pattern**<br><br>Write a program to **print a pyramid pattern of stars (*)** using a for loop. |

**Input:** 5
**Output:**

```
    *
   * *
  * * *
 * * * *
* * * * *
```

35 **Maximum Sum of Contiguous Subarray of Size k**

**Program Definition**

Given an array of integers arr[] and an integer k, find the **maximum sum** of any **contiguous subarray** of size k.

- A **contiguous subarray** is a subarray with consecutive elements.
- The program should return the **largest sum** possible among all subarrays of length k.

**Input:** arr = [2, 1, 5, 1, 3, 2]
k = 3
**Output:** Maximum sum = 9
**Explanation**

- Subarray [5, 1, 3] has the **highest sum** of 9.

**Input:** arr = [1, 2, 3, 4, 5, 6]
k = 2
**Output:** Maximum sum = 11
**Explanation**

- Subarray [5, 6] has the **highest sum** of 11.

36 **Longest Common Prefix**

**Program Definition**

Given an array of strings, write a function to find the **longest common prefix** shared among all the strings.

- A **common prefix** is a sequence of characters that appears at the **beginning of every string** in the array.
- If **no common prefix exists**, return an **empty string ("")**.

**Input:** str = ["flowers", "flow", "fly", "flight"]
**Output:** "fl"
**Input:** str = ["dog", "cat", "animal", "monkey"]
**Output:** ""

37 **Count Duplicate Numbers in an Array**

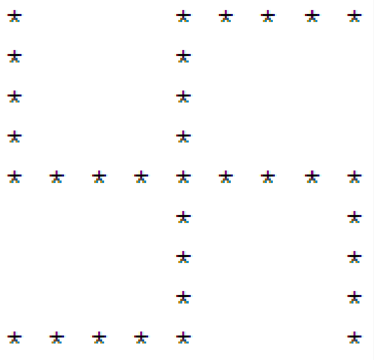
A program was written to identify and count the number of **duplicate elements** present in a given array of integers.
An element was considered a duplicate if it appeared **more than once** in the array.
Each duplicate element was counted **only once**, regardless of how many times it appeared.

**Input:** [1, 2, 3, 2, 4, 1, 5, 1]
**Output:** 2

38 **Sum of Largest and Second Largest Elements in an Array**

A program was written to find the **sum of the largest and the second largest elements** in a given array of integers.

- The array must be processed **without sorting** its elements.
- The solution must run in **O(n) time complexity**, where n is the number of elements in the array.

| | |
|---|---|
| | • The program scans the array only once to identify the two largest values and then computes their sum. |
| | **Input:** [4, 1, 9, 3, 7]<br>**Output:** 16<br><br>**Input:** [10, 5, 8, 20, 15]<br>**Output:** 35 |
| **39** | **Minimum Training Hours to Equalize Skills**<br><br>You are given an array representing the skill levels of students.<br>Each student has an integer skill value.<br>• In one hour of training, a student's skill level can be increased by 1.<br>• Skill levels cannot be decreased.<br>• Training can be given to any student for any number of hours.<br>Your task is to find the minimum total number of training hours required to make the skill level of any two students equal. |
| | **Input:** skills = [1, 3, 6]<br>**Output:** 2<br>**Input:** skills = [4, 10, 7, 6]<br>**Output:** 1<br>**Input:** skills = [5, 3, 5, 8]<br>**Output:** 0 |
| **40** | Given two sorted arrays of sizes m and n respectively, the task is to find the element that would be at the k-th position in the final sorted array formed by merging these two arrays. |
| | **Input: a[] = [2, 3, 6, 7, 9], b[] = [1, 4, 8, 10], k = 5**<br>**Output: 6**<br>**Explanation: The final sorted array is [1, 2, 3, 4, 6, 7, 8, 9, 10] The 5th element is 6.** |
| **41** | Given the number of rows and columns, print the corresponding **swastika pattern using loops**.<br>**Note:** The number of rows and columns should be the same and an odd number. This will generate a perfect swastika pattern. |
| |  |
| **42** | **Write** a program to print **Pascal triangle.** |
| | **Input : N = 5**<br>**Output:**<br>**1**<br>**1 1**<br>**1 2 1**<br>**1 3 3 1**<br>**1 4 6 4 1** |
| **43** | Find the difference between the second largest element and the second smallest element of an array**.** |
| | **Input:** Enter the size of array: 7 |

```
*       * * * * *
*       *
*       *
*       *
* * * * * * * * *
        *       *
        *       *
        *       *
* * * * *       *
```

| | |
|---|---|
| | Enter 7 elements: 5 1 9 7 1 5 3<br>**Output:** Difference: 4 |
| **44** | Given the principal amount (P), rate of interest per annum (R), and loan tenure in months (N), write a program to calculate the EMI (Equated Monthly Installment) for a loan. |
| | **Input :** Enter principal amount (P): 500000<br>Enter annual interest rate (R): 7.5<br>Enter loan tenure in months (N): 60<br>**Output:** The EMI per month is: 10013.05 |
| **45** | Given a number n then print n terms of fibonacci series in reverse order. |
| | **Input : n =** 5<br>**Output :** 3 2 1 1 0 |
| **46** | Given an array arr[], the task is to reverse the array. Reversing an array means rearranging the elements such that the first element becomes the last, the second element becomes second last and so on. |
| | **Input**: arr[] = {1, 4, 3, 2, 6, 5}<br>**Output**: {5, 6, 2, 3, 4, 1} |
| **47** | Given a string S and a character 'c', the task is to count the occurrence of the given character in the string. |
| | **Input:** S = "geeksforgeeks" and c = 'e'<br>**Output:** 4<br>**Explanation:** 'e' appears four times in str. |
| **48** | Find if a given string can be represented from a substring by iterating the substring "n" times. |
| | **Input:** str = "abcabcabc"<br>**Output:** true |
| **49** | Find the length of the maximum number of consecutive numbers jumbled up in an array. |
| | **Input:** arr[] = {1, 94, 93, 1000, 5, 92, 78};<br>**Output :** 3<br>**Explanation:** The largest set of consecutive elements is 92, 93, 94. |
| **50** | Given an array arr[], the task is to find the subarray that has the maximum sum and return its sum. |
| | **Input:** arr[] = {2, 3, -8, 7, -1, 2, 3}<br>**Output:** 11<br>**Explanation:** The subarray {7, -1, 2, 3} has the largest sum 11. |
| **51** | Write a program to merge two arrays into one, but if any element is repeated in the arrays, it should appear only once, and the duplicates should be replaced with zeros. |
| | **Input:** Enter the size of first array: 4<br>Enter the elements of first array: 7 2 5 9<br>Enter the size of second array: 5<br>Enter the elements of second array: 5 9 3 7 8<br>**Output:** Merged array: 7 2 5 9 0 3 8 0 0 |
| **52** | Given a 2D square matrix, find the sum of elements in Principal and Secondary diagonals. For example, consider the following 4 X 4 input matrix. |
| | **Input:** 4<br>1 2 3 4<br>4 3 2 1<br>7 8 9 6<br>6 5 4 3<br>**Output:**<br>Principal Diagonal: 16<br>Secondary Diagonal: 20 |
| **53** | Given a string and write a C program to count the number of vowels and consonants in this string. |
| | **Input:** str = "geeks for geeks"<br>**Output:** |

| | |
|---|---|
| | Vowels: 5<br>Consonants: 8 |
| **54** | Given a string that contains both upper and lower case characters in it. The task is to count a number of upper and lower case characters in it |
| | **Input:** Introduction to Python<br>**Output**: Lower Case characters : 18<br>Upper case characters : 2 |
| **55** | Write a program to sort the numbers in a string expression where numbers are separated by the '+' sign. The program should rearrange the numbers in nondecreasing order while keeping them separated by '+'. |
| | **Input:** 1+1+3+1+3<br>**Output:** 1+1+1+3+3 |
| **56** | Given two arrays a[] and b[], the task is to find the intersection of the two arrays. Intersection of two arrays is said to be elements that are common in both arrays. The intersection should not count duplicate elements and the result should contain items in any order |
| | **Input:** a[] = {1, 2, 1, 3, 1}, b[] = {3, 1, 3, 4, 1}<br>**Output:** {1, 3}<br>**Explanation:** 1 and 3 are the only common elements and we need to print only one occurrence of common elements. |
| **57** | Write a program to generate very big (at least 11 digits) and calculate the total of all those numbers.<br>In the program, User will input the first digit i.e. "D", input count i.e. "N" which need to generate automatically sequential bases from 1 to N, other remaining digits should be 0<br>Input Validation<br>0 ≤ D ≤ 9<br>1 < N ≤ 1000<br>Explanation<br>Let's assume the inputs from the user for D is 5 and N is 50<br>● So the generated numbers will be 50000000001, 50000000002, 50000000003, 50000000004.... 50000000050.<br>● The calculated sum of above generated numbers is 2500000001275.<br>● So output of this program should be 2500000001275. |
| | **Input**: D: 3 N: 10<br>**Output**: 300000000055 |
| **58** | Given a time in the format of hh:mm (12-hour format) 0 < hh < 12, 0 <= mm < 60. The task is to convert it into words as shown: |
| | **Input**: h = 5, m = 0<br>**Output**: five o' clock<br>**Input**: h = 6, m = 24<br>**Output**: twenty-four minutes past six<br>5:00 → five o’ clock<br>5:01 → one minute past five<br>5:10 → ten minutes past five<br>5:15 → quarter past five<br>5:30 → half past five<br>5:40 → twenty minutes to six<br>5:45 → quarter to six<br>5:47 → thirteen minutes to six<br>5:28 → twenty-eight minutes past five |