1.A peak element is an element that is strictly greater than its neighbors.

Given a 0-indexed integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks.

You may imagine that nums[-1] = nums[n] = -∞. In other words, an element is always considered to be strictly

greater than a neighbor that is outside the array.

You must write an algorithm that runs in O(log n) time.

Example 1:

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

Explanation: 3 is a peak element and your function should return the index number 2. Example 2:

Input: nums = [1,2,1,3,5,6,4] Output: 5

Explanation: Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.

Constraints:

* 1 <= nums.length <= 1000
* -231 <= nums[i] <= 231 - 1
* nums[i] != nums[i + 1] for all valid i.

1. Given a 0-indexed string s, permute s to get a new string t such that:
   * All consonants remain in their original places. More formally, if there is an index i with 0 <= i < s.length such that s[i] is a consonant, then t[i] = s[i].
   * The vowels must be sorted in the nondecreasing order of their ASCII values. More formally, for pairs of indices i, j with 0 <= i < j < s.length such that s[i] and s[j] are vowels, then t[i] must not have a higher ASCII value than t[j].

Return the resulting string.

The vowels are 'a', 'e', 'i', 'o', and 'u', and they can appear in lowercase or uppercase. Consonants comprise all letters that are not vowels.

Example 1:

Input: s = "lEetcOde" Output: "lEOtcede"

Explanation: 'E', 'O', and 'e' are the vowels in s; 'l', 't', 'c', and 'd' are all consonants. The vowels are sorted according to their ASCII values, and the consonants remain in the same places.

Example 2:

Input: s = "lYmpH" Output: "lYmpH"

Explanation: There are no vowels in s (all characters in s are consonants), so we return "lYmpH".

Constraints:

* + 1 <= s.length <= 105
  + s consists only of letters of the English alphabet in uppercase and lowercase.

Given a string s, find the length of the longest substring without duplicate characters.

Example 1:

Input: s = "abcabcbb" Output: 3

Explanation: The answer is "abc", with the length of 3.

Example 2:

Input: s = "bbbbb" Output: 1

Explanation: The answer is "b", with the length of 1. Example 3:

Input: s = "pwwkew" Output: 3

Explanation: The answer is "wke", with the length of 3.

Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

Constraints:

0 <= s.length <= 5 \* 104

s consists of English letters, digits, symbols and spaces.

1. Given an integer array nums, find a subarray that has the largest product, and return the product. The test cases are generated so that the answer will fit in a 32-bit integer.

Example 1:

Input: nums = [2,3,-2,4] Output: 6

Explanation: [2,3] has the largest product 6. Example 2:

Input: nums = [-2,0,-1] Output: 0

Explanation: The result cannot be 2, because [-2,-1] is not a subarray.

Constraints:

1. <= nums.length <= 2 \* 104

-10 <= nums[i] <= 10

The product of any subarray of nums is guaranteed to fit in a 32-bit integer.

1. Given an integer array nums, find the subarray with the largest sum, and return its sum. Example 1:

Input: nums = [-2,1,-3,4,-1,2,1,-5,4]

Output: 6

Explanation: The subarray [4,-1,2,1] has the largest sum 6. Example 2:

Input: nums = [1] Output: 1

Explanation: The subarray [1] has the largest sum 1.

Example 3:

Input: nums = [5,4,-1,7,8] Output: 23

Explanation: The subarray [5,4,-1,7,8] has the largest sum 23.

Follow up: If you have figured out the O(n) solution, try coding another solution using the divide and conquer approach, which is more subtle.

1. Design a stack that supports push, pop, top, and retrieving the minimum element in constant time. Implement the MinStack class:

MinStack() initializes the stack object.

void push(int val) pushes the element val onto the stack. void pop() removes the element on the top of the stack. int top() gets the top element of the stack.

int getMin() retrieves the minimum element in the stack.

You must implement a solution with O(1) time complexity for each function.

Example 1:

Input ["MinStack","push","push","push","getMin","pop","top","getMin"]

[[],[-2],[0],[-3],[],[],[],[]]

Output

[null,null,null,null,-3,null,0,-2]

Explanation

MinStack minStack = new MinStack(); minStack.push(-2);

minStack.push(0); minStack.push(-3); minStack.getMin(); // return -3 minStack.pop(); minStack.top(); // return 0 minStack.getMin(); // return -2

Constraints:

-231 <= val <= 231 - 1

Methods pop, top and getMin operations will always be called on non-empty stacks. At most 3 \* 104 calls will be made to push, pop, top, and getMin.

1. You are given an array of strings tokens that represents an arithmetic expression in a Reverse Polish Notation. Evaluate the expression. Return an integer that represents the value of the expression.

Note that:

The valid operators are '+', '-', '\*', and '/'.

Each operand may be an integer or another expression.

The division between two integers always truncates toward zero. There will not be any division by zero.

The input represents a valid arithmetic expression in a reverse polish notation.

The answer and all the intermediate calculations can be represented in a 32-bit integer.

Example 1:

Input: tokens = ["2","1","+","3","\*"] Output: 9

Explanation: ((2 + 1) \* 3) = 9

Example 2:

Input: tokens = ["4","13","5","/","+"] Output: 6

Explanation: (4 + (13 / 5)) = 6

Example 3:

Input: tokens = ["10","6","9","3","+","-11","\*","/","\*","17","+","5","+"]

Output: 22

Explanation: ((10 \* (6 / ((9 + 3) \* -11))) + 17) + 5

= ((10 \* (6 / (12 \* -11))) + 17) + 5

= ((10 \* (6 / -132)) + 17) + 5

= ((10 \* 0) + 17) + 5

= (0 + 17) + 5

= 17 + 5

= 22

Given an array of integers temperatures represents the daily temperatures, return an array answer such that answer[i] is the number of days you have to wait after the ith day to get a warmer temperature. If there is no future day for which this is possible, keep answer[i] == 0 instead.

Example 1:

Input: temperatures = [73,74,75,71,69,72,76,73] Output: [1,1,4,2,1,1,0,0]

Example 2:

Input: temperatures = [30,40,50,60] Output: [1,1,1,0]

Example 3:

Input: temperatures = [30,60,90] Output: [1,1,0]

Design an algorithm that collects daily price quotes for some stock and returns the span of that stock's price for the current day.

The span of the stock's price in one day is the maximum number of consecutive days (starting from that day and going backward) for which the stock price was less than or equal to the price of that day.

For example, if the prices of the stock in the last four days is [7,2,1,2] and the price of the stock today is 2, then the span of today is 4 because starting from today, the price of the stock was less than or equal 2 for 4 consecutive days.

Also, if the prices of the stock in the last four days is [7,34,1,2] and the price of the stock today is 8, then the span of today is 3 because starting from today, the price of the stock was less than or equal 8 for 3 consecutive days.

Implement the StockSpanner class:

StockSpanner() Initializes the object of the class.

int next(int price) Returns the span of the stock's price given that today's price is price.

Example 1:

Input

["StockSpanner", "next", "next", "next", "next", "next", "next", "next"]

[[], [100], [80], [60], [70], [60], [75], [85]]

Output

[null, 1, 1, 1, 2, 1, 4, 6]

Explanation

StockSpanner stockSpanner = new StockSpanner(); stockSpanner.next(100); // return 1 stockSpanner.next(80); // return 1 stockSpanner.next(60); // return 1 stockSpanner.next(70); // return 2 stockSpanner.next(60); // return 1

stockSpanner.next(75); // return 4, because the last 4 prices (including today's price of 75) were less than or equal to today's price.

stockSpanner.next(85); // return 6

1. Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. An input string is valid if:

Open brackets must be closed by the same type of brackets. Open brackets must be closed in the correct order.

Every close bracket has a corresponding open bracket of the same type.

Example 1: Input: s = "()" Output: true

Example 2: Input: s = "()[]{}" Output: true

Example 3: Input: s = "(]" Output: false

Example 4: Input: s = "([])" Output: true

1. Once upon a time in a small village, a wise king wanted to attend as many village festivals as possible in one day. Each festival had a start and end time, and the king could only attend one at a time. His advisor suggested picking only the festivals that ended the earliest, so he could fit more in. The king followed this advice, selecting the next festival that started after the last one ended. By the end of the day, he had enjoyed the most celebrations possible without any overlaps. The villagers were amazed at his perfect schedule, and the king declared it a royal strategy for future festivals.

**Examples:**

**Input:** start[] = [1, 3, 0, 5, 8, 5], finish[] = [2, 4, 6, 7, 9, 9]

**Output:** 4

**Explanation:** A person can attend 4 functions. The maximum set of functions that can be attended is {0, 1, 3, 4} (These are indexes in start[] and finish[])

**Input:** start[] = [10, 12, 20], finish[] = [20, 25, 30]

**Output:** 1

**Input:**

start[] = [1, 1, 1], finish[] = [4, 3, 2]

**Output:** 1

1. In the bustling kingdom of AlgoLand, there lived a clever merchant named Riya who ran a workshop offering special services. Every day, clients came to her with unique jobs — each with a deadline and a reward (profit) for completing it. However, each job took exactly one full day, and Riya could only handle one job per day. She quickly realized that if she wasn’t careful, she’d miss out on valuable profits by picking the wrong jobs.

One evening, she sat down and looked at all the job requests. Each had a deadline (by when it must be done) and an offered payment. She knew that to maximize her gain, she couldn’t just pick the highest-paying job — she had to balance deadlines and profits wisely. Riya decided to sort the jobs by profit and then schedule the most rewarding ones first, fitting them in the latest day before their deadline if available.

By doing this smart scheduling, she managed to complete the highest number of jobs possible before their deadlines and earned the greatest total profit. Her strategy became famous throughout AlgoLand, and people started calling her the "Queen of Optimization.". Help Riya to complete the program to get maximum profit.

Examples :

Input: deadline[] = [4, 1, 1, 1], profit[] = [20, 10, 40, 30]

Output: [2, 60]

Explanation: Job1 and Job3 can be done with maximum profit of 60 (20+40). Input: deadline[] = [2, 1, 2, 1, 1], profit[] = [100, 19, 27, 25, 15]

Output: [2, 127]

Input: deadline[] = [3, 1, 2, 2], profit[] = [50, 10, 20, 30]

Output: [3, 100]

**Constraints:**

1 ≤ deadline.size() == profit.size() ≤ 105 1 ≤ deadline[i] ≤ deadline.size()

1 ≤ profit[i] ≤ 500

\_

1. In the quiet village of Splitville, there was a wise elder named Mira who had a magical pouch full of coins, each with different values. One day, she challenged the village children: “Pick the fewest number of coins from this pouch such that their total value is greater than the value of all the coins left behind.” The children were puzzled — grabbing more coins meant winning, but they had to be smart.

One clever child sorted the coins from largest to smallest and picked them one by one, keeping a running total. She stopped as soon as her sum became greater than what remained in the pouch. Mira clapped in joy

— the child had cracked the puzzle using logic, not greed. From then on, this became a famous brain game in

Splitville known as the “Heavier Half.”

**Test Cases:**

**Input:** arr = [2, 17, 7, 3]

**Output:** 1

**Explanation:** Select only 17. Remaining sum is 12, and 17 > 12.

**Input:** arr = [20, 12, 18, 4]

**Output:** 2

**Explanation:** Select 18 and 12. Total = 30, remaining = 24.

**Input:** arr = [1, 1, 1, 1, 10]

**Output:** 1

E**xplanation:** Select 10. Remaining sum is 4.

**Constraints:**

1 <= arr.size() <= 105

1 <= arr[i] <= 104

1. In the hilly kingdom of Altaria, each village was built on a tower of different height. The king wanted to even out the landscape as much as possible, so that no tower looked too tall or too short compared to others. To do this, he ordered the builders to either raise or lower every tower’s height by exactly **K meters** — no exceptions!

But there was a rule: no tower could be made shorter than zero. The royal architect, Mira, studied the towers carefully. She realized that by strategically increasing some towers and decreasing others, she could bring their heights closer together. Her goal was to **minimize the difference** between the tallest and shortest towers after these adjustments.

She tested different combinations and figured out the perfect plan: sort the tower heights first, then consider the highest and lowest possible new heights, and make adjustments while keeping all towers non-negative.

After a day of calculations, she presented the best possible layout to the king. Delighted, he declared Mira the

“Balancer of Towers,” and her technique was taught to architects across Altaria.

**Examples :**

**Input:** k = 2, arr[] = {1, 5, 8, 10}

**Output:** 5

**Explanation:** The array can be modified as {1+k, 5-k, 8-k, 10-k} = {3, 3, 6, 8}.The difference between the largest and the smallest is 8-3 = 5.

**Input:** k = 3, arr[] = {3, 9, 12, 16, 20}

**Output:** 11

**Explanation:** The array can be modified as {3+k, 9+k, 12-k, 16-k, 20-k} -> {6, 12, 9, 13, 17}.The difference between the largest and the smallest is 17-6 = 11.

**Input:**k = 3, arr[] = {7, 7, 7, 7}

**Output:** 0

**Constraints**

1 ≤ k ≤ 107

1 ≤ n ≤ 105

1 ≤ arr[i] ≤ 107

1. In the land of Lexiconia, the royal scribe was tasked with writing a magical scroll using a string of letters. But the scroll had one sacred rule: **no two identical letters could be written side by side**, or the spell would fail. The scribe was handed strings like "aaabc" and "aaabb" and had to rearrange them so that **no two adjacent characters were the same**.

He carefully counted how many times each letter appeared. If any letter appeared **more than half** the length of the string (rounded up), he knew it would be impossible to arrange without breaking the rule. Otherwise, he placed the most frequent letters at alternating positions and filled in the rest.

The king marveled as the scrolls lit up in magical light, proving the arrangements were perfect. The method was passed down for generations as the "Non-Adjacent Rune Rule."

**Test Cases:**

**Input:** s = "aaabc"

**Output:** 1

**Explanation:** Rearranged as "abaca" or "acaba" — no adjacent duplicates.

**Input:** s = "aaabb"

**Output:** 1

**Explanation:** Rearranged as "ababa" — valid rearrangement.

**Input:** s = "aaaabc"

**Output:** 0

**Explanation:** Too many 'a's — cannot rearrange without adjacent duplicates.

1. In the merchant city of Greedonia, a young trader named Elric was setting off on a journey with a magical knapsack. This bag could carry only a limited weight, but there was a twist — Elric could break any item and take a fraction of it if needed. Each item in his inventory had a value and a weight.

His goal? **Maximize the value** of what he could carry without exceeding the knapsack’s capacity. So Elric sorted all his items by their **value-to-weight ratio**, starting with the most valuable per kilogram. He packed full items as long as the bag allowed, and when he reached the limit, he sliced off just enough of the next most valuable item to fit perfectly.

With a bag filled to the brim and value maximized, Elric set off richer than ever before. The townsfolk called this clever method **“The Fractional Knapsack Strategy,”** and it became a treasured algorithm in the Guild of Traders.

**Test Cases:**

**Input:**

val = [60, 100, 120]

wt = [10, 20, 30]

capacity = 50

**Output:** 240.000000

**Explanation:** Take items 1, 2 completely and 2/3 of item 3.

**Input:**

val = [60, 100]

wt = [10, 20]

capacity = 50

**Output:** 160.000000

**Explanation:** Total weight is 30 < 50, take both fully.

**Input:**

val = [10, 20, 30]

wt = [5, 10, 15]

capacity = 100

**Output:** 60.000000

**Explanation:** Take all items; combined weight = 30 < 100.

**Constraints:**

1 <= val.size=wt.size <= 105

1 <= capacity <= 109

1 <= val[i], wt[i] <= 104

1. There are n houses and p water pipes in rebels Colony. Every house has at most one pipe going into it and at most one pipe going out of it. Rebel needs to install pairs of tanks and taps in the colony according to the following guidelines.
2. Every house with one outgoing pipe but no incoming pipe gets a tank on its roof.
3. Every house with only one incoming and no outgoing pipe gets a tap.

The Rebel council has proposed a network of pipes where connections are denoted by three input values: ai, bi, di denoting the pipe of diameter di from house ai to house bi.

Find a more efficient way for the construction of this network of pipes. Minimize the diameter of pipes wherever possible.

**Note**: The generated output will have the following format. The first line will contain t, denoting the total number of pairs of tanks and taps installed. The next t lines contain three integers each: house number of tank, house number of tap, and the minimum diameter of pipe between them.

**Example 1:**

**Input:**

n = 9, p = 6

a[] = {7,5,4,2,9,3}

b[] = {4,9,6,8,7,1}

d[] = {98,72,10,22,17,66}

**Output:**

3

2 8 22

3 1 66

5 6 10

**Explanation:**

Connected components are

**3->1, 5->9->7->4->6 and 2->8**.

Therefore, our answer is **3** followed by **2 8 22, 3 1 66, 5 6 10. Input:**

n = 5, p = 3

a[] = {1, 2, 4}

b[] = {2, 3, 5}

d[] = {10, 8, 12}

**Output:**

2

1 3 8

4 5 12

**Input:**

n = 6, p = 4

a []= {1, 2, 5, 6}

b[] = {2, 3, 6, 4}

d[] = {40, 30, 50, 25}

**Output:**

2

1 3 30

5 4 25

**Constraints:**

1<=n<=20

1<=p<=50

1<=a[i],b[i]<=20

1<=d[i]<=100

1. In the kingdom of Arraya, there lived a wizard who loved symmetry — especially in the form of **non- increasing sequences**. One day, he stumbled upon a magical scroll with numbers that danced out of order. The scroll read: {3, 1, 2, 1}. This bothered him, as the numbers weren't steadily decreasing or staying the same.

To restore harmony, the wizard could use his magic to **either increase or decrease any number**, but he wanted to use the **least amount of magic possible**. So he cast spells to gently **lower numbers that disrupted the order**, ensuring each number was **not greater than the one before it**.

Sometimes, only a single spell was needed. Other times, when a number spiked too high, he had to use several spells. His goal was simple: bring order to chaos using **minimum effort**.

And so, his technique became known across Arraya as the **"Descending Order Charm."**

**Test Cases:**

**Input:**

N = 4, array = [3, 1, 2, 1]

**Output:** 1

**Explanation:** Decrease 2 to 1 → new array: [3, 1, 1, 1]

**Input:**

N = 4, array = [3, 1, 5, 1]

**Output:** 4

**Explanation:** Decrease 5 to 1 → 4 steps

**Input:**

N = 5 ,array = [10, 9, 11, 7, 6]

**Output**: 2

**Constraints:**

1 <= n <= 10^4

1 <= a[i] <= 10^4

1. In the twin cities of Numeria, two teams — Team A and Team B — were preparing for a grand partner dance. Each dancer in Team A had to be paired with a dancer from Team B. But there was a rule: the more different their dance styles (represented by numbers), the more awkward the dance — and the higher the cost in energy.

The elders wanted the total **awkwardness** (the sum of absolute differences in style between all pairs) to be **as low as possible**. A wise mathematician stepped in and gave a clever solution: **sort both teams by their style levels** and **pair them in order**.

By doing this, the differences between each pair were minimized. The dance went smoothly, the crowd cheered, and the method became known as the **“Minimum Awkward Pairing” strategy** — a dance of logic and balance.

**Test Cases:**

**Input:**

N = 4, A = [4, 1, 8, 7], B = [2, 3, 6, 5]

**Output:** 6

**Explanation:** Sorted A = [1, 4, 7, 8], B = [2, 3, 5, 6] Sum = |1-2| + |4-3| + |7-5| + |8-6| = 1 + 1 + 2 + 2 = **6**

**Input:**

N = 3, A = [4, 1, 2], B = [2, 4, 1]

**Output:** 0

**Explanation:** Sorted A = [1, 2, 4], B = [1, 2, 4] Sum = |1-1| + |2-2| + |4-4| = **0**

**Input:**

N = 5 ,A = [10, 20, 30, 40, 50] , B = [12, 18, 35, 38, 52]

**Output: 14**

**Constraints:**

1 <= N <= 105

0 <= A[i] <= 109

0 <= B[i] <= 109

Sum of N over all test cases doesn't exceeds 106

**------------------------------------------------------------------------------------------------------------------ ----------------------------**

**---------------------**

1. In the city of Timetron, schedules were sacred. Every citizen had a list of time intervals when they needed access to the magical gates of transit. But chaos broke out — people’s time slots overlapped, and arguments erupted over who reserved what.

The wise Scheduler stepped in. She laid out all the intervals on a grand timeline and sorted them by their start times. One by one, she merged overlapping time slots into larger blocks, ensuring no overlap remained — but no time was lost either.

Soon, everyone had clean, non-overlapping windows. Conflicts vanished, the gates flowed smoothly, and the process became known as the **“Time Merge Protocol”**, essential in all future scheduling.

**Test Cases:**

**Input:**

arr = [[1,3], [2,4], [6,8], [9,10]]

**Output:** [[1,4], [6,8], [9,10]]

**Explanation:** [1,3] and [2,4] overlap → merged to [1,4].

**Input:**

arr = [[6,8], [1,9], [2,4], [4,7]]

**Output:** [[1,9]]

**Explanation:** All intervals overlap with [1,9], so they merge into one.

**Input:**

arr = [[1, 5], [10, 15], [5, 10], [20, 25]]

**Output: [[1, 15], [20, 25]]**

**Constraints:**

1 ≤ arr.size() ≤ 105

0 ≤ starti ≤ endi ≤ 105

**----------------------------------------------------------------------------------------------------------------------------- -----------------**

**-------**

1. On a rainy Saturday afternoon, best friends Aarav and Meera were holed up inside, the thunder rumbling in the distance. With no power and their usual gadgets dead, they turned to an old pastime—brain games."I have a challenge," Aarav said, grinning as he grabbed a notebook. "Let’s play a game: find the longest palindromic substring from a random sentence., now our task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.

Examples:

*Input: s = “noonracecar”*

*Output: “racecar”*

*Explanation: There are several possible palindromic substrings like “noon”, “oo”, “racecar” etc. But the substring “racecar” is the longest among all.*

*Input: s = “samantha”*

*Output: “ama”*

*Input: s = “rebecca”*

*Output: “ebe”*

*Input: s = “madam”*

*Output: “madam”*

1. On a breezy summer afternoon, two best friends, Riya and Kabir, sat under the shade of a big banyan tree in Riya’s backyard. School was out, their phones were tucked away, and boredom was beginning to creep in.“Let’s play a word game,” Kabir said . Riya perked up. “What kind of game?”Kabir grinned. “A palindrome challenge. We take turns giving each other strings. the task is to remove or delete the minimum number of characters from the string so that the resultant string is a palindrome.

Note: The order of characters should be maintained. Examples :

*Input : s = “aebcbda”*

*Output : 2*

*Explanation: Remove characters ‘e’ and ‘d’. Resultant string will be “abcba” which is a palindromic string*

*Input : s = “sumadamer”*

*Output : 4*

1. In the quiet village of Sumridge, two curious friends, Arya and Rohan, were known for their love of puzzles and challenges. One sunny afternoon, while rummaging through Arya’s attic, they found an old wooden box filled with ancient coins of different values 1, 2, 5.. units.

Attached to the box was a note:

“To claim the treasure of Sumridge, find all the ways to make the number N using these coins.”

Our task is to help them and the total Number of combinations are possible using these coins?”

Note: Assume that you have an infinite supply of each type of coin. Examples:

*Input: sum = 4, coins[] = [1, 2, 3]*

*Output: 4*

*Explanation: There are four solutions: [1, 1, 1, 1], [1, 1, 2], [2, 2] and [1, 3]*

*Input: sum = 10, coins[] = [2, 5, 3, 6]*

*Output: 5*

*Explanation: There are five solutions:*

*[2, 2, 2, 2, 2], [2, 2, 3, 3], [2, 2, 6], [2, 3, 5] and [5, 5]*

*Input: sum = 10, coins[] = [10] Output: 1*

*Explanation: The only is to pick 1 coin of value 10. Input: sum = 5, coins[] = [4]*

*Output: 0*

*Explanation: We cannot make sum 5 with the given coins*

1. *In the enchanted land of Arithmia, two young adventurers, Mira and Taran, discovered a hidden map while exploring the ruins of an old wizard’s tower. The map led to the legendary Vault of Efficiency, said to hold the secrets of making anything with the least effort—and the least gold. At the vault’s entrance, they found a glowing tablet with an inscription:*

*“Only those who can find the path with least gold coins to treasure may enter”*

Our task is to help the friends to solve the puzzle and find the Minimum gold coins required to enter. If it is not possible to form the Taget using the given gold coins, return -1.

Examples:

*Input: goldcoins[] = [25, 10, 5], Target= 30*

*Output: 2*

*Explanation : Minimum 2 goldcoins needed, 25 and 5*

*Input: goldcoins[] = [9, 6, 5, 1], sum = 19*

*Output: 3*

*Explanation: 19 = 9 + 9 + 1*

*Input: goldcoins[] = [5, 1], sum = 0*

*Output: 0*

*Explanation: For 0 sum, we do not need a coin Input: goldcoins[] = [4, 6, 2], sum = 5*

*Output: -1*

*Explanation: Not possible to make the given Target.*

1. Given a string s, the task is to find the minimum number of cuts needed for palindrome partitioning of the given string. A partitioning of the string is a palindrome partitioning if every sub-string of the partition is a palindrome.

Examples:

*Input: s = “frfeek”*

*Output: 2*

*Explanation: We need to make minimum 2 cuts, i.e., “frf | ee | k”.*

*Input: s= “aaaa”*

*Output: 0*

*Explanation: The string is already a palindrome.*

*Input: s = “ababbbabbababa”*

*Output: 3*

*Explanation: We need to make minimum 3 cuts, i.e., “aba | bb | babbab | aba”.*

1. Given a string s of length n, the task is to count number of palindromic subsequence (need not necessarily be distinct) present in the string s.

Example:

*Input: s = “abcd”*

*Output: 4*

*Explanation: Palindromic subsequence are : “a” ,”b”, “c” ,”d”*

*Input: s = “aab”*

*Output: 4*

*Explanation: palindromic subsequence are :”a”, “a”, “b”, “aa”*

*Input: s = “anna”*

*Output:9*

*Explanation: palindromic subsequence are :“a” ,”n” ,”n” ,”a” ,”nn” ,”aa” ,”ana” ,”ana” ,”anna”*

1. Given two strings s1 and s2, the task is to find the length of the shortest string that has both s1 and s2 as subsequences.

Examples:

*Input: s1 = “geek”, s2 = “eke”*

*Output: 5*

*Explanation: String “geeke” has both string “geek” and “eke” as subsequences. Input: s1 = “AGGTAB”, s2 = “GXTXAYB”*

*Output: 9*

*Explanation: String “AGXGTXAYB” has both string “AGGTAB” and “GXTXAYB” as subsequences.*

1. In the Super village of Coding, two curious friends, riya and ronak, were known for their love of puzzles and challenges.here is the challenge for them *an array of positive integers arr[] and a value sum, determine if there is a subset of arr[] with sum equal to given sum.*

*Examples:*

*Input: arr[] = [3, 34, 4, 12, 5, 2], sum = 9 Output: true*

*Explanation: Here there exists a subset with target sum = 9, 4+3+2 = 9. Input: arr[] = [3, 34, 4, 12, 5, 2], sum = 30*

*Output: false*

*Explanation: There is no subset with target sum 30. Input: arr[] = [1, 2, 3], sum = 6*

*Output: true*

*Explanation: The entire array can be taken as a subset, giving 1 + 2 + 3 = 6.*

1. *In the bustling village of NutriVille, Chef Nia was famous for preparing perfectly balanced meals. One day, she received a royal request:*

*“Prepare a meal divided into two plates, each with the same total nutritional value, using only the ingredients from this basket.”*

*She wondered:“Can I divide these into two groups so that each group adds up to the same total nutrient*

*value?”*

*Chef Nia opened the basket and found ingredients with different nutrient points: [3, 1, 5, 9, 12]---(12+3),(1+5+9)*

Examples:

*Input: arr[] = [1, 5, 11, 5] Output: true*

*Explanation: The array can be partitioned as [1, 5, 5] and [11]*

*Input: arr[] = [1, 5, 3] Output: false*

*Explanation: The array cannot be partitioned into equal sum sets.*

1. In the valley of Numatia, a young adventurer named Lina set out to climb the Mountain of Numbers. The mountain wasn't made of rocks—it was made of stepping stones, each marked with a number. Some were tall, some small, all laid out in a straight path.

But there was a rule:

Lina could only step forward, and only onto a stone with a higher number than the one she was currently on. She couldn’t go back or jump sideways. Her goal was to climb using the longest increasing path possible.

One day, the mountain path looked like this:

[3, 10, 2, 1, 20]

Lina began her journey.She stepped on 3, then jumped to 10, skipped 2 and 1 (they were too low), and finally reached 20.

So her path was: [3, 10, 20]

That made 3 steps—a pretty good climb!

Another day, the path looked like:

[50, 3, 10, 7, 40, 80]

Lina’s clever choice this time: [3, 10, 40, 80] Examples:

*Input: [10, 20, 3, 40]*

*Output : 10 20 40*

*Explanation: The length of path is 3 and the path is [10, 20, 40]*

*Input: [10, 22, 9, 33, 21, 50, 41, 60, 80]*

*Output : 10 22 33 50 60 80*

*Explanation: There are multiple paths of same length ,examples [10, 22, 33, 50, 60, 80] and [10 22 33 41 60 80]. The first one has lexicographic smallest order.*

Sample Questions for HackwithInfy

1. Maximum Product Subset Problem

Given an integer array nums[], find the maximum product that can be achieved from any non-empty subset of the array.

A subset of an array is a set that can be formed by removing zero or more elements without changing the order of remaining elements. The product of a subset is the multiplication of all its elements.

Input Format:

* + The first line contains an integer N — the number of elements in the array.
  + The second line contains N space-separated integers representing the array elements. Output Format:
  + Print a single integer — the maximum product of any non-empty subset. For example, Case#1:

Input:

7

-6 4 -5 8 -10 0 8

Output:

15360

1. Print All Triplets That Form a Geometric Progression Problem Statement:

Given a sorted array of distinct positive integers, print all triplets in the array that form a Geometric Progression (GP) with an integral common ratio.

A triplet (a,b,c) is said to form a geometric progression if there exists a positive integer r such that b=a⋅r and c=b⋅r.

Input Format:

* + The first line contains an integer N — the number of elements in the array.
  + The second line contains N space-separated distinct positive integers in sorted order.

Output Format:

* + Print each valid triplet on a new line.
  + The order of output doesn't matter.
  + If no such triplet exists, print nothing. Case #1:

Input:

6

1 2 6 10 18 54

Output:

1. 6 18

6 18 54

1. Minimum Waiting Time Problem Statement:

Given an array where each element indicates the time required by a customer at a ticket counter, and only one customer can be served at a time in sequence, determine the minimum total waiting time achievable by scheduling the customers optimally

Input Format:

A single line of space-separated integers, where each integer denotes the time required by a customer.

Output Format:

A single integer representing the minimum total waiting time. Case#1 Input:

1. 2 1 2 6

Output:

17

1. Job Sequencing with Deadlines Problem Statement:

You are given n jobs. Each job has the following properties:

* + A unique ID
  + A deadline (by when it should be completed)
  + A profit (earned only if the job is finished before or on its deadline)

Each job takes exactly 1 unit of time to complete. Your task is to schedule the jobs to maximize total profit, ensuring no two jobs overlap and each job finishes within its deadline.

Input Format:

* + An integer n representing the number of jobs.
  + Then n lines follow, each containing:

Output Format:

* + A list of selected job IDs (in order of execution) and the total profit obtained. Case#1: Input:

4

A 2 100

B 1 19

C 2 27

D 1 25

Output:

A C → Total Profit = 127

1. Candy Distribution Problem Statement:

There are n children standing in a line, each with a rating value. Each child must receive at least one candy. Additionally, any child with a higher rating than their immediate neighbor must receive more candies than that neighbor. The objective is to distribute candies to minimize the total number given.

Input:

An array ratings of length n where ratings[i] represents the rating of the i-th child.

Output:

An integer representing the minimum total number of candies required.

Case#1:

Input:

ratings = [1, 0, 2] Output:

5

Explanation: One optimal distribution is [2, 1, 2].

Hint:

Perform two passes over the ratings array:

* Left to right, ensure each child has more candies than the left neighbor if rating is higher.
* Right to left, ensure the same condition holds relative to the right neighbor.

1. Rearrange Tasks to Avoid Consecutive Duplicates Problem Statement:

Given a string representing a queue of tasks where each character is a lowercase letter denoting a task type, reorder the tasks so that no two identical tasks are scheduled consecutively. If no such arrangement exists, return an empty string.

Input:

A string tasks containing lowercase English letters.

Output:

A reordered string with no two adjacent characters the same, or an empty string "" if rearrangement is impossible.

Constraints:

* 1 ≤ tasks.length ≤ 500
* tasks contains only lowercase English letters.

Case#1:

Input: "aaabb" Output: "ababa"

1. Rod Cutting Problem Problem Statement:

Given a rod of length n and an array of prices where the price at index i represents the selling price of a rod of length i+1, determine the maximum revenue obtainable by cutting the rod into smaller pieces and selling them.

Input Format :

length[]: an array representing possible rod lengths [1, 2, ..., n] price[]: an array where price[i] is the price for rod length length[i] n: the total length of the rod to be cut

Output:

The maximum profit achievable by cutting the rod in an optimal way. Case#1:

Input:

length[] = [1, 2, 3, 4, 5, 6, 7, 8]

price[] = [1, 5, 8, 9, 10, 17, 17, 20]

Rod length: 4 Output:

Maximum Profit: 10

1. Find All Binary Strings from a Wildcard Pattern Problem Statement :

Given a binary string pattern containing some wildcard characters ?, generate all possible binary strings by replacing each ? with either 0 or 1.

Input:

A string consisting of characters 0, 1, and ?. Output:

A list of all binary strings formed by substituting every ? with 0 or 1. Case#1: Input:

1?11?00?1?

Output (all possible strings): 1011000010 1011000011

1011000110

1011000111

1011100010

1011100011

1011100110

1011100111

1111000010

1111000011

1111000110

1111000111

1111100010

1111100011

1111100110

1111100111

1. Knight’s Tour Problem: Print All Possible Tours Problem Statement:

Given an N x N chessboard and a knight placed at a starting position, print all possible sequences of

knight moves such that the knight visits every square exactly once (a Knight’s Tour).

Details:

* + The knight moves in an "L" shape: two squares in one direction and one square perpendicular.
  + Each square must be visited exactly once.
  + The board is numbered with move order, starting from 1 at the initial position. Approach
  + Use backtracking to explore all valid knight moves recursively.
  + Maintain a board matrix to mark visited squares and the move number.
  + At each step:
* Mark the current square with the move number.
* Recursively try all 8 possible knight moves.
* If all squares are visited (move number = N\*N), print or store the tour.
* Backtrack by unmarking the square if no solution found in that path.
  + Use boundary checks and visited checks for validity of moves.

1. Longest Palindromic Subsequence Problem Statement:

Given a string s, find the length of the longest subsequence of s that is a palindrome. A subsequence is a sequence that can be derived from the original string by deleting some or no characters without changing the order of the remaining characters.

This differs from the longest palindromic substring problem where the characters must be contiguous. Here, characters can be taken from anywhere in the string as long as their relative order is preserved.

Input Format:

A single line containing the string s. Output Format:

Print a single integer denoting the length of the longest palindromic subsequence in s. Constraints:

1 <= length of s <= 1000

s contains only uppercase and/or lowercase English letters. Case#1:

Input:

ABBDCACB

Output:

5

1. Largest Plus of 1’s in a Binary Matrix

Problem Statement:

Given a square binary matrix consisting of 0's and 1's, find the size of the largest plus sign formed only by 1's.

A plus sign is defined as a center cell with arms extending up, down, left, and right — all consisting of continuous 1's — of equal length. The size of the plus is the total number of cells it covers, including the center and the arms.

For example, a plus with arm length 2 looks like this (where 1 represents a cell in the plus and

1. elsewhere):

Input Format:

* + The first line contains an integer n, the size of the square matrix.
  + The next n lines each contain n integers (either 0 or 1), representing the matrix rows.

Output Format:

* + Print a single integer representing the size of the largest plus sign formed by 1's in the matrix. Constraints:
  + 1 <= n <= 1000
  + Each element in the matrix is either 0 or 1. Case#1:

Input:

5

1. 0 1 0 1

1 1 1 1 1

1 0 1 0 1

1 1 1 1 1

1 0 1 0 1

Output:

17

1. Arrival and Departure Time in DFS

Problem Statement: Given a directed graph with V vertices (numbered from 0 to V-1) and a list of E edges, you are to perform a Depth-First Search (DFS) traversal of the graph starting from vertex 0. During the traversal, record the arrival time and departure time of each vertex.

* + The arrival time of a vertex is the time when it is visited (entered) for the first time during DFS.
  + The departure time of a vertex is the time when DFS finishes exploring all its neighbors and is about to backtrack.

If the graph has disconnected components, DFS should continue to run from the next unvisited vertex (in increasing order of vertex number).

Input Format:

* + First line contains two integers: V (number of vertices) and E (number of edges).
  + Next E lines each contain two integers u and v, representing a directed edge from vertex u to vertex v.

Output Format:

* + For every vertex from 0 to V-1, output its arrival and departure times in the format: Vertex v: Arrival = a, Departure = d

Constraints:

* + 1 ≤ V ≤ 10⁴
  + 0 ≤ E ≤ 10⁵
  + 0 ≤ u, v < V

Case#1: Input:

6 7

0 1

0 2

1 3

1 4

2 5

3 5

1. 5

Output:

Vertex 0: Arrival = 0, Departure = 11

Vertex 1: Arrival = 1, Departure = 8

Vertex 2: Arrival = 9, Departure = 10

Vertex 3: Arrival = 2, Departure = 5

Vertex 4: Arrival = 6, Departure = 7

Vertex 5: Arrival = 3, Departure = 4

1. Chess Knight Minimum Steps Problem Statement:

Given a N x N chessboard and the starting and ending positions of a knight, find the minimum number of steps the knight must take to reach the destination from the source.

A knight moves in an "L" shape: it can move two squares in one direction and then one square perpendicular to that direction.

You need to determine the shortest path from the source to the destination using Breadth-First Search (BFS), which is ideal for finding shortest paths in unweighted graphs like this one.

Input Format:

* + The first line contains a single integer N, the size of the chessboard.
  + The second line contains two integers sx and sy, the knight's starting coordinates (0- based index).
  + The third line contains two integers dx and dy, the knight's destination coordinates. Output Format:
  + Output a single integer representing the minimum number of steps required by the knight to reach the destination from the source.

Constraints:

* + 1 ≤ N ≤ 1000
  + 0 ≤ sx, sy, dx, dy < N

Case#1: Input: 8

7 0

0 7

Output:

6

1. Construct the Longest Palindrome by Reordering or Deleting Characters Problem Statement:

Given a string S, construct the longest possible palindromic string that can be made by shuffling or deleting any characters from S.

You may reorder the characters and delete any number of them. The goal is to build the longest palindrome possible from the given characters.

If there are multiple answers with the same length, print any one of them. Input Format:

* + A single string S of uppercase English letters (A-Z). Output Format:
  + A single line containing the longest palindrome that can be formed by reordering or deleting characters from S.

Constraints:

* + 1 ≤ |S| ≤ 10⁵
  + The string consists of only uppercase letters A to Z Case#1:

Input: ABBDAB Output: BABAB Case#2:

Input: ABCDD Output: DAD

1. Problem: Minimizing Total Guest Unhappiness Problem Statement:

You are managing a hotel and need to serve N guests. Each guest has a happiness value Cᵢ, which is the time at which they prefer to be served.

However, you can serve only one guest at a time, and each guest takes exactly one unit of time to be served. The serving times start from x = 1 and increase incrementally (1, 2, 3, ..., N).

The unhappiness of serving a guest at time x is defined as: Unhappiness=∣Ci−x∣

Your task is to determine the minimum total unhappiness if the guests are optimally scheduled. Input Format:

* + The first line contains an integer N — the number of guests.
  + The second line contains N integers: C₁, C₂, ..., Cₙ — the happiness values of the guests.

Output Format:

* + A single integer — the minimum total unhappiness. Constraints:
  + 1≤N≤103
  + 1≤ Ci ≤N

Case #1:

Input:

5

1 2 3 4 5

Output:

0

Case#2:

Input:

5

1. 1 3 2 4

Output 4

1. Folding a Blanket Problem Statement:

Alex has a rectangular blanket with dimensions H × W (H = height, W = width). Alex wants to pack it into a box that can only fit a blanket of size H1 × W1.

To do this, Alex folds the blanket in a series of steps. Each fold must be exactly parallel to one of the edges (either horizontally or vertically), and the resulting dimensions after each fold must remain whole numbers.

Your task is to find the minimum number of folds required to reduce the blanket from its original size H × W to fit into the target size H1 × W1.

Input Format

* + The first line contains a long integer H – the initial height of the blanket.
  + The second line contains a long integer W – the initial width of the blanket.
  + The third line contains a long integer H1 – the target height.
  + The fourth line contains a long integer W1 – the target width. Constraints
  + 1 ≤ H, W, H1, W1 ≤ 10¹⁵
  + H1 ≤ H
  + W1 ≤ W

Output Format

* + Output a single integer – the minimum number of folds needed.

Case #1:

Input:

2

3

2

2

Output:

1

Explanation:

Alex folds the blanket once along the width to reduce it from 3 to 2. The height is already acceptable, so only one fold is required.

1. Lexicographically Smallest Array After One Swap

You are given an array A of size N and an integer K. You are allowed to swap at most one pair of elements such that the distance between their indices is at most K (i.e., |i - j| ≤ K).

Your task is to return the lexicographically smallest array possible after such a swap (or no swap, if the array is already optimal).

Definition: An array X is lexicographically smaller than array Y if at the first index i where they differ, X[i] < Y[i].

Input

N → Integer (length of the array) A[0] → First element of array A[1] →

Second element of array

...

A[N-1] → Last element of array

K → Integer (max distance allowed for swap)

Output

N lines, each with one integer — the elements of the resulting array. Constraints

* + 1 ≤ N ≤ 10^5
  + 1 ≤ A[i] ≤ 10^5
  + 1 ≤ K ≤ N

Case#1:

Input:

5

5

4

3

2

1

3

Output:

2

4

3

5

1

1. Prevent Multiple Subsequences Problem Statement:

Alice has a string S of length N, and Bob gives her another string C of length M.

Bob challenges Alice to modify her string S such that the string C appears at most once as a subsequence in S.

To help her do this, Bob gives an array A of size N, where A[i] is the cost to delete the character S[i]. Alice wants to achieve the goal with the minimum possible total cost.

Find the minimum total cost to delete characters from S such that C appears at most once as a subsequence of S.

Input Format:

N → Length of string S

M → Length of string C

S → Alice’s string

C → Bob’s string

A[0] A[1] ... A[N-1] → Array of N integers representing deletion cost of each character in S

Output Format:

Single integer: the minimum cost to ensure C appears at most once as a subsequence of S.

Constraints:

* + 1 ≤ N, M ≤ 10⁵
  + 1 ≤ A[i] ≤ 10⁵
  + All characters in S and C are lowercase English letters.
  + It is guaranteed that C appears at least once as a subsequence in S.

Case#1: Input:

6

3

abcabc abc 1 2 3 4 5 6

Output:

10

1. Largest Common Arrival Order Subsequence Across Multiple Days

Problem Statement: Given N people and K days, each day is represented by an array of N numbers indicating the order in which people arrived at the theatre. The task is to determine the size of the largest group of people who arrive in the same relative order on all K days.

Input Format:

* + The first line contains two integers N and K separated by a space — the number of people and the number of days.
  + The next K lines each contain N integers — the arrival order of people on each day. Output Format:
  + A single integer — the size of the largest group of people who arrive in the same relative order on all K days.

Constraints:

* + 1 ≤ N ≤ 1000
  + 1 ≤ K ≤ 10
  + 1 ≤ a[i][j] ≤ N

Case#1:

Input:

N = 4, K = 3

Day 1: [1, 3, 2, 4]

Day 2: [1, 3, 2, 4]

Day 3: [1, 4, 3, 2]

Output:

3

Explanation: The group consisting of people {1, 3, 2} appears in the same order on all days.

1. Maximum Number of Valid Triplets Formed From an Array Under Limit M Problem Statement:

Given an array of size N and an integer limit M, determine the maximum number of triplets that can be formed under these conditions:

* + Each element in the triplet is less than or equal to M.
  + A triplet is valid if it satisfies either of the following:
* All three numbers in the triplet are the same.
* The three numbers are consecutive integers (e.g., 3, 4, 5).
  + Each element in the array can belong to only one triplet.

Constraints:

* + 1 ≤ N ≤ 10^5
  + 1 ≤ M ≤ 10^4
  + 1 ≤ arr[i] ≤ M

Input Format:

* + The first line contains two integers, N and M.
  + The second line contains N integers representing the array elements.

Output Format:

* + Print a single integer — the maximum number of valid triplets that can be formed.

Case#1:

Input:

4 2

1 2 2 2

Output:

1

Q1.

A new deadly virus has infected large population of a planet. A brilliant scientist has discovered a new strain of virus which can cure this disease. Vaccine produced from this virus has various strength depending on midichlorians count. A person is cured only if midichlorians count in vaccine batch is more than midichlorians count of person. A doctor receives a new set of report which contains midichlorians count of each infected patient and stores all vaccine doctor has and their midichlorians count. You need to determine if doctor can save all patients with the vaccines he has. The number of vaccines and patients are equal.

Input Format Parameters:

N :: INTEGER

The first line contains an integer, N, denoting the number of patients / vaccines count. N :: 1<N<15 VacStrength :: INTEGER ARRAY

Second line contains N integers (where 0<=i<N) describing the strength of each vaccine(midichlorians count).

VacStrength[i]:: 1 to 10^4

PatMid :: INTEGER ARRAY

Third line contains N integers (where 0<=i<N) describing the midichlorians count of each patient. PatMid [i]:: 1 to 10^4

Output Format

Print a single line containing 'Yes' or 'No'. Case#: 1 Input:

5

123 146 454 542 456

100 328 248 689 200

Output:

No

Case#: 2 Input

4

300 200 250 320

120 230 180 270

Output:

Yes

Case#:3 Input

5

400 600 300 350 450

220 520 100 50 500

Output:

No

Q2.

The "Shades" Television Channel organizes a fun-filled event named "Best Couple 2025", where married couples are invited and given many tasks and activities. Based on some criteria decided by the jury, the best couple will be chosen.

N couples registered for the event, and each couple was given a registration number (same ID for both). One of the specific couple's registration ID got missed. The event coordinators wanted your help in finding the missing ID.

Write a program that takes an array of registration numbers as input and outputs the missing registration ID.

Parameters:

N::INTEGER

The first line of the input contains the number of couples N who registered for the event. 1<=N<=5 Arr::INTEGER ARRAY

The second line of input contains 2xN-1 registration IDs of each of the couples, separated by a space

as one of the specific couple’s ID is missing.

1<=Arr[i]<=10

Case#: 1 Input:

2

1 3 1

Output:

3

Case#: 2

4

1 5 2 7 2 7 1

Output:

5

Case#:3 5

5 1 9 2 1 6 9 5 2

220 520 100 50 500

Output:

6

Q3.

Sherland State University has a mini library in the hostel, where N students can use it, but only one at a time. A timetable is set to avoid conflicts:

* The 1st student starts at time O and must finish by A1.
* The 2nd student starts at A1 and must finish by A2, and so on.
* Each student needs Bi units of time to read.

Determine how many students can finish reading within their assigned time slots. Parameters:

N:INTEGER

The first line input contains a single integer N denoting the number of students. T: INTEGER ARRAY

The second line of input contains N space-separated integers A1, A2, ..., and AN denoting the moments of time by which the corresponding student should finish reading.

D: INTEGER ARRAY

The third line of input contains N space-separated integers B1, B2, ..., and BN denoting the time required for each of the students to read.

Case#:1 Input:

3

1 10 15

1 10 3

Output:

2

Explanation:

N = 3 students, with the schedule [1, 10, 15] and reading times [1, 10, 3].

* Student 1: Needs 1 unit, finishes by 1 → (Can read).
* Student 2: Needs 10 units, starts at 1, finishes by 11 (exceeds 10) → (Cannot read).
* Student 3: Needs 3 units, starts at 10, finishes by 13 (before 15) → (Can read). Total students

able to read: 2.

Case#:2 Input

6

10 20 30 45 50 51

15 5 20 12 4 1

Output: 4

Case#:3

Input 7

10 20 30 45 50 51 76

15 5 20 12 4 1 13

Output:

5

Case#:4 Input

8

15 20 30 45 50 51 60 70

14 5 9 12 4 1 8 8

Output:

8

Case#:5 Input 10

78 54 12 36 96 85 74 12 36 41

14 25 36 74 85 94 17 39 55 42

Output :

1

Case#:6 Input

5

10 20 20 40 50

5 10 15 10 5

Output: 4

Q4.

Given a non-empty array nums[] of integers of length N, find the top k elements that have the

highest frequency in the array. If two numbers have the same frequencies, then the larger number should be given more preference.

Example 1:

Input:

N = 6

nums = {1,1,1,2,2,3}

k = 2

Output: 1 2

Example 2:

Input:

N = 8

nums = {1,1,2,2,3,3,3,4}

k = 2

Output: 3 2 Explanation:

Elements 1 and 2 have the same frequency ie. 2. Therefore, in this case, the answer includes the element 2 as 2 > 1.

Parameters:

N:INTEGER

The first line of input consists of an integer N, the number of elements in the array. 1 ≤ N ≤ 25

ARR: INTEGER ARRAY

The second line of input consists of N integers separated by spaces, representing the elements of the array.

1. ≤ nums[i] ≤ 100 K:INTEGER

The third line of input consists of an integer k, representing the value of k. 1 ≤ k ≤ 4

Case#:1 Input

6

1 1 1 2 2 3

2

Output 1 2

Case#:2 Input

8

1 1 2 2 3 3 3 4

2

Output 3 2

Case#:3 Input 15

1 2 8 9 7 6 1 1 6 5 7 6 1 9 4

3

Output 1 6 9

Case#:4 Input 10

1 2 8 9 7 6 1 1 6 6

2

Output 6 1 Q5.

You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.

Return the max sliding window.

Example 1:

Input: n=8

nums = [1,3,-1,-3,5,3,6,7], k = 3

Output: [3,3,5,5,6,7]

Explanation:

Example 2:

Input:n=1

nums = [1], k = 1

Output: [1]

Parameters:

N: INTEGER

The first line contains an integer n, the number of elements in the array. 1 ≤ N ≤ 100

ARR: INTEGER ARRAY

The second line contains N space-separated integers, representing the elements of the array ARR.. 1

≤ ARR[i] ≤ 10000

K: INTEGER

The third line contains an integer K, the size of the sliding window. 1 ≤ K ≤ N

Case#:1 Input

8

1 3 -1 -3 5 3 6 7

3

Output

3 3 5 5 6 7

Case#:2 Input

8

1 3 -1 -3 5 3 6 7

3

Output

3 3 5 5 6 7

Q6.

Partition an unsorted array of positive integers in such a way that the absolute difference between the sums of the two resulting sub-arrays is minimum. The index at which partitioning is to be done should be considered to be part of the first sub-array.

Example

Input:

5

1 2 3 4 5

Output:

2

Explanation:

Process:

* Index 0: Left sum = 1, Right sum = 2+3+4+5 = 14, Difference = 13
* Index 1: Left sum = 1+2 = 3, Right sum = 3+4+5 = 12, Difference = 9
* Index 2: Left sum = 1+2+3 = 6, Right sum = 4+5 = 9, Difference = 3
* Index 3: Left sum = 1+2+3+4 = 10, Right sum = 5, Difference = 5
* Index 4: Left sum = 1+2+3+4+5 = 15, Right sum = 0, Difference = 15 The minimum difference occurs at index 2.

Parameters:

N: INTEGER

The first line contains an integer N, representing the number of elements in the array. 1 <= N <= 50 ARR: INTEGER ARRAY

The second line contains N space-separated integers, representing the elements of the array. 1 <= N

<= 50

0<=ARR[i]<=100

Case#1:

Input 5

1 2 3 4 5

Output 2

Case#:2 Input

4

10 20 30 40

Output 2

Case#:3 Input 15

5 10 2 8 15 4 3 9 7 12 20 11 6 1 16

Output 8 Q7.

Michael is celebrating his 20th birthday and he wishes to arrange a party for all his classmates. But there are n tough guys in his class who are weird. They thought that this was the best occasion to test their friendship with him. They put up conditions before Michael that they will break the

friendship unless he gives them a grand party on their chosen day. Formally, ith friend will break his friendship if he does not receive a grand party on d th day.

Michael is not a lavish spender and can give at most one grand party daily. Also, he wants to invite only one person in a party. So he just wonders what the maximum number of friendships he can save. Please help Michael with this difficult task.

Parameters:

N: INTEGER

The first line will contain a single integer denoting N describing count of tough guys. 1<=N<=10 d: INTEGER ARRAY

The second line will contain n space-separated integers where ith integer corresponds to the day d th as given in the problem.

Output Format

The output prints a single line corresponding to the maximum number of friendships Michael can save.

Case#:1 Input

5

4 1 3 3 5

Output 4

Case#:2 Input 10

9 9 9 9 8 8 8 8 8 7

Output 3

Case#:3

Input 11

10 9 8 9 6 5 4 2 3 1 4

Output 9

Q8.

Mr.John is one of the old fashioned guys and he doesn't like the new smart phones with full keypads and still uses the old keypads which require you to tap a key multiple times to type a single letter. For example, if the keyboard has two keys, one with the letters "adef" and the other one with the letters "zyx", then typing 'a' requires one keystroke, typing 'f' requires four keystrokes, typing 'y' requires

two keystrokes, and so on.

He recently moved to a new country where the language is such that his keypad is not the most efficient. In every language some characters occur more often than others. He wants to create a specific keyboard for this language that uses N different letters. He has a large body of text in this language, and has already analyzed it to find the frequencies of all N letters of its alphabet.

You are given an array 'frequencies' with N elements. Each element of frequencies is the number of times one of the letters in the new language appears in the text John has. Each element of

frequencies will be strictly positive. (I.e., each of the N letters occurs at least once.)

You are also given an array keySize. The number of elements of keySize is the number of keys on the keyboard. Each element of keySize gives the maximal number of letters that maybe put on one of the keys.

Find an assignment of letters to keys that minimizes the number of keystrokes needed to type the entire text. Output that minimum number of keystrokes. If there is not enough room on the keys and some letters of the alphabet won't fit, Output -1 instead.

Parameters:

N: INTEGER

The first line will contain a number 'N' that specifies the size of 'frequencies' array 1<=N<=50 FREQ: INTEGERS ARRAY

The second line will contain N numbers that form the frequencies array

1<=FREQ[i]<=1000 K: INTEGER

The third line contains a number 'K' that specifies the size of the 'keySize' array 1<=K<=50 keySize: INTEGERS ARRAY

The fourth line contains K numbers that form the keySize array 1<=keySize[i]<=50

Case#:1

Input 4

7 3 4 1

2

1. 2

Output 19

Explanation

The foreign language has four letters. Let us call them W, X, Y and Z. John's text contains seven Ws, three Xs, four Ys, and one Z. The keyboard has two keys, each of them may contain at most two

letters. One optimal solution is to use the keys "WZ" and "YX". We can then type each W and each Y using a single keystroke, and we need two keystrokes for each X and each Z. Therefore, the total number of keystrokes when typing the entire text will be 7x1 + 3x2 + 4x1 + 1x2 = 19.

Case#:2 Input

5

1. 3 8 4 2

3

2 1 2

Output 25

Case#:3

Input 7

3 3 8 4 2 1 5

3

2 3 3

Output 36

Q9.

At RoboMart, a futuristic fully-automated warehouse, shelves are restocked by intelligent drones. Each drone can carry a limited number of items in one trip. The warehouse management system generates a daily restocking plan listing how many units are needed at each shelf.

Since each drone can only carry capacity items per trip, it might have to make multiple trips to restock a single shelf. For operational planning, the system needs to compute the total number of trips required to complete the entire restocking operation.

Parameters:

N : INTEGER

The first line contains an Integer N — number of shelves in the warehouse. 1 ≤ N ≤ 10⁵

ShelfCap: INTEGER ARRAY

The second line contains n space-separated integers — number of items needed per shelf. 1 ≤ ShelfCap[i] ≤ 10⁶

DroneCap: INTEGER

The third line Integer DroneCap — number of items a drone can carry in one trip. 1 ≤ Capacity ≤ 10⁴

Output Format:

* Print the total number of drone trips required.

Case#:1 Input

5

5 10 15 20 10

10

Output 7

Explanation:

Trips per shelf: 1 + 1 + 2 + 2 + 1 = 7 Case#:2 Input

7

20 10 15 40 30 5 35

8

Output 22

Q10

A tech company is testing a smart reservation system in a high-tech movie theater. The theater has several rows of seats, and each row's status is tracked using a binary array — 0 for an available seat, 1 for a booked seat.

When a group of friends wants to sit together, the system checks all rows to find any row with a contiguous block of empty seats large enough to accommodate them. Due to social preferences and proximity sensors, the group must sit together in a single row, without gaps.

Your job is to simulate this system and determine which rows have at least one continuous segment of group\_size or more empty seats.

\*\*\*If none of the rows found with required number of contiguous empty seats, then return “-1”.

Parameters:

R, C : 2 INTEGERS

The first line of input will contain 2 Integers R and C describing number of rows and no:of seats in each row.

Arr[R,C]: From 2nd line of input, R rows each with C values separated by space having 0 for empty, 1 for booked.

Group\_size: INTEGER

The last line of input contains an integer, representing the size of the group fetching for contiguous seats in the same row.

Case#:1 Input

3 5

1 0 0 0 1

0 0 1 0 0

1 1 1 1 1

3

Output 0

Case#:2 Input

3 5

1 0 1 0 1

0 1 1 0 1

1 1 1 1 1

2

Output

-1

Case#:3 Input

1. 6

0 1 0 0 0 1

1 1 0 0 1 0

0 0 0 0 1 0

0 0 1 1 1 1

3

Output 0 2

Q11.

John and Bobby are brave soldiers in World War 2. They have spotted an enemy troop which is planting bombs. They sent message to the command centre containing characters W and B where W represents a wall and B represents a Bomb. They asked command to tell them how many walls will be destroyed if all bombs explode at once. One bomb can destroy 2 walls on both sides.

Input Format Parameters:

S: STRING

First line of input contains a single string which contains two type of chars 'W' and 'B'.

Constraints 1<=LEN(S)<=1000

Case#:1 Input WWBWWBW

Output 5

Case#:2 Input

BWWWBWBWW

Output 6

Case#:3 Input WBW Output

2

Q12.

A jail has a number of prisoners and a number of treats to pass out to them. Their jailer decides the fairest way to divide the treats is to seat the prisoners around a circular table in sequentially

numbered chairs. A chair number will be drawn from a hat. Beginning with the prisoner in that chair,

one candy will be handed to each prisoner sequentially around the table until all have been distributed.

The jailer is playing a little joke, though. The last piece of candy looks like all the others, but it tastes awful. Determine the chair number occupied by the prisoner who will receive that candy.

Example n=4 m=6 s=2

There are 4 prisoners, 6 pieces of candy and distribution starts at chair 2. The prisoners arrange

themselves in seats numbered 1 to 4. Prisoners receive candy at positions 2,3,4,1,2,3. The prisoner to be warned sits in chair number 3.

Parameters:

n, m, s: INTEGER, INTEGER, INTEGER

The first line of input contains 3 integers describing the number of prisoners, the number of candies, and the chair number to start passing out treats at.

1<=n<=10^5

1<=m<=10^5

1<=s<=n

Case#:1 Input

1. 2 1

Output 2

Case#:2 Input

5 2 2

Output 3

Case#:3 Input

7 19 2

Output

6

Case#:4 Input

3 7 3

Output

3

Q13.

There are x number of girls and they rolled a dice in turns one after another.

If the number on the dice is 6, then the dice will be rolled again until she gets a number other than 6.

Since you know the sequence of numbers which the dice shows when rolled each time, you have to find what is the total number of girls or if the sequence is invalid.

Parameters:

S : STRING

A single line that contains a string denoting the sequence of numbers the dice rolls on written as string.

Constraints 1<|len(S)|<100000

Output Format

If the sequence is valid print the number of girls If the sequence is invalid print -1

Case#:1 Input 3662123 Output

5

Explanation 0

The second person rolls the dice 3 times so the number of people will be 5.

Case#:2 Input 36621237

Output

-1

Case#:3 Input 62166644 Output

4

Case#:4 Input 62166844 Output

-1

Q14.

The little Monk loves to play with strings. One day he was going to a forest. On the way to the forest he saw a Big Tree(The magical tree of the forest). The magic of the tree was that, every leaf of the

tree was in the format of string(having some value). Monk wants to have these type of leaves. But he can take only one. Help Monk to choose the most valuable leaf.

Format of the leaf:

a+b+c-d+c+d-x-y+x , i.e. it contains a string holding a mathematical expression, and the value of that expression will be value of that particular leaf.

e.g. 4+2-5+3+2-4 value: 2 Parameters:

L: INTEGER

The first line contains an integer L, describing the number of leaves on the tree Arr: INTEGER ARRAY From 2nd line of input onwards, L lines contain a string representing expression in STRING format. Constraints

* The Expression will have only + and - operations
* String will contain only single valued digits. Output Format

Print a single line having value of most valuable string.

Case#:1 Input

3

4+2-5+3+2-4

1+2+3+4+5

-9+5+5

Output 15

Case#:2 Input

5

5+2-4+6-1-1

7-5+1

2+6+0

8-2+3

1+1+1+1

Output 9

Case#:3 Input

4

8-6+2+4+3-6+1

1+1+1+1

2+3+6+8-9

2+7+1-6

Output 10

Q15.

Given two arrays of integers, find which elements in the second array are missing from the first array. Example

Arr=[7,2,5,3,5,3]

Brr=[7,2,5,4,6,3,5,3]

The Brr array is the original list. The numbers missing are [4,6] Notes

* If a number occurs multiple times in the lists, you must ensure that the frequency of that number in both lists is the same. If that is not the case, then it is also a missing number.
* Return the missing numbers sorted ascending.
* Only include a missing number once, even if it is missing multiple times.
* The difference between the maximum and minimum numbers in the original list is less than or equal to 100.

Parameters:

N : INTEGER

The first line of input contains an Integer, representing the size of first array Arr 1<=N<=100 Arr: INTEGERS ARRAY

The second line of input contains N space separated integer values for array Arr. 1<=Arr[i]<=10^4 M : INTEGER

The third line of input contains an Integer, representing the size of second array Brr. 1<=M<=100 Brr: INTEGERS ARRAY

The fourth line of input contains M space separated integer values for array Brr. 1<=Brr[i]<=10^4

Case#:1 Input 10

203 204 205 206 207 208 203 204 205 206

13

203 204 204 205 206 207 205 208 203 206 205 206 204

Output

204 205 206

Case#2:

Input 5

30 10 50 10 20

10

20 80 30 70 60 20 10 10 80 50

Output

20 80 70 60

Q16.

You have three stacks of cylinders where each cylinder has the same diameter, but they may vary in height. You can change the height of a stack by removing and discarding its topmost cylinder any number of times.

Find the maximum possible height of the stacks such that all of the stacks are exactly the same height. This means you must remove zero or more cylinders from the top of zero or more of the three stacks until they are all the same height, then return the height.

Example h1=[1,2,1,1]

h2=[1,1,2]

h3=[1,1]

There are 4,3, and 2 cylinders in the three stacks, with their heights in the three arrays. Remove the top 2 cylinders from h1 (heights = [1, 2]) and from h2 (heights = [1, 1]) so that the three stacks all are 2 units tall. Return 2 as the answer.

Note: An empty stack is still a stack

Parameters:

n1, n2, n3: INTEGERS

The first line of input contains 3 Integers, representing number of cylinders of all the 3 stacks. h1: INTEGERS ARRAY

The second line of input contains n1 space-separated integers representing the heights of cylinders of stack1.

h2: INTEGERS ARRAY

The third line of input contains n2 space-separated integers representing the heights of cylinders of stack2.

h3: INTEGERS ARRAY

The fourth line of input contains n3 space-separated integers representing the heights of cylinders of stack3.

Case#:1

Input Comment

5 3 4 h1[] size n1 = 5, h2[] size n2 = 3, h3[] size n3 = 4

3 2 1 1 1 h1 = [3, 2, 1, 1, 1]

4 3 2 h2 = [4, 3, 2]

1 1 4 1 h3 = [1, 1, 4, 1]

Output 5

Explanation

Initially, the stacks look like this:

To equalize their heights, remove the first cylinder from stacks 1 and 2, and then remove the top two cylinders from stack 3 (shown below).

The stack heights are reduced as follows:

1. 8-3=5

2. 9-4=5

3. 7-1-1=5

All three stacks now have height=5, the value to return.

Case#:2 Input

4 2 4

1 1 1 2

3 7

1 3 1 1

Output

0

Case#:3 Input

3 3 3

4 4 4

4 4 4

4 4 4

Output

12

Case#:4 Input

5 4 2

3 2 1 4 5

2 6 1 3

5 5

Output 10

Q17.

Jack is working on a project to identify specific integers based on the sum of their digits. He has a sorted list of integers and needs to find the smallest integer whose digits sum up to a given value. Help Jack by writing a program using binary search.

Parameters: N: INTEGER

The first line of input consists of an integer N, representing the number of elements in the list. 1 ≤ N≤

20

ARR: INTEGERS ARRAY

The second line contains N space-separated integers, representing the sorted list. 10 ≤ list elements ≤

1000

D: INTEGER

The third line contains an integer D, representing the target digit sum. Output Format

The output prints the smallest integer whose digits sum up to D, or -1 if no such integer exists.

Case#:1 Input

5

12 23 30 48 56

3

Output 12

Case#:2 Input

6

23 31 45 69 71 96

15

Output 69

Case#:3 Input

5

23 45 67 68 95

17

Output

-1

Case#:4 Input

8

10 12 15 21 25 30 120 150

3

Output 12

Case#:5 Input

8

10 12 15 23 24 30 123 150

6

Output 15

Case#:6 Input

8

10 12 13 23 24 33 123 150

6

Output 24 Q18.

You are working as a data analyst for a company that collects user feedback scores for their products. The feedback scores are recorded sequentially, and sometimes you need to identify specific feedback points for analysis.

Given a list of feedback scores and a particular feedback score, your task is to find the position of the second occurrence of this feedback score in the list. If the feedback score appears less than twice, you need to handle that as well:

1. If the feedback score does not appear at all, print a message indicating that the score was not found.
2. If the feedback score appears only once, print a message indicating that the score was found only once.
3. If the feedback score appears more than once, print the position (index) of the second occurrence.

Parameters:

N: INTEGER

The first input line contains an integer N, the number of feedback scores. 1 ≤ N ≤ 10

score: INTEGERS ARRAY

The second input line contains N space-separated integers, each representing a feedback score. 0 ≤ score≤ 100

X: INTEGER

The third input line contains an integer X, which is the feedback score you need to find the second occurrence of.

0 ≤ X ≤ 100

Output Format

The output displays the following format:

* The position of the second occurrence of the key in the list, or:
* "X Score not found." if the score is not present in the list, or:
* "X Score found only once." if the score appears only once.

Case#:1

Input 5

45 56 98 56 32

56

Output 3

Input 4

71 52 63 94

89

Output

89 Score not found. Case#:3 Input

4

85 96 32 45

45

Output

45 Score found only once. Q19.

A smart city control system collects hourly vehicle counts from various sensors placed along a highway. To predict traffic congestion, the system needs to determine the maximum average

number of vehicles over any continuous window of k hours. This will help planners schedule signal timings and manage flow.

Given an array vehicles[] representing the number of vehicles detected every hour, and an integer k, return the maximum average of any contiguous subarray of length k.

Parameters:

n: INTEGER

The first line of input contains an integer, n, that describes the number of hours Vehicles: INTEGERS ARRAY

The second line of input contains n space-separated integer values, describing vehicles count per hour

0 <= vehicles[i] <= 10^5 k: INTEGER

The third line of input contains an integer, k, describes length of window. 1 <= k <= n <= 10^5

Output:

* Float rounded to 2 decimal places representing the maximum average

Case#:1 Input:

8

10 20 30 40 50 10 70 80

3

Output:

53.33

Explanation:

Max average of window [50, 10, 70] = (50+10+70)/3 = 130/3 = 43.33

Better: [10, 70, 80] = 160/3 = 53.33

Case#:2 Input

7

10 20 30 40 50 60 70

3

Output 60.00

Case#:3 Input

6

1 12 -5 -6 50 3

4

Output 12.75

Case#:4 Input

6

100 300 250 350 150 200

2

Output 300.00

Q20.

You are a data analyst at a startup that helps small business owners flip electronics (buy and resell for profit). A client comes to you with historical price data of a popular smartphone over the past few days. The client wants to buy one unit of the smartphone on one day and sell it on a later day to maximize their profit.

You have to write a program that determines the maximum profit that can be earned by making at most one transaction (buy one and sell one product).

Constraints:

* You can not sell before buying.
* If no profit can be made, the result should be 0. Parameters:

N: INTEGER

The first line of input contains an integer, N, describes number of days. Prices: INTEGER ARRAY

The second line of input contains N space separated integers representing price of smartphones for N days.

Case#:1

Input 6

7 1 5 3 6 4

Output 5

Explanation

Buy on day 2 at price = 1, sell on day 5 at price = 6, profit = 6 - 1 = 5

Case#:2 Input

5

7 6 4 3 1

Output 0

Case#:3 Input

5

1 2 3 4 5

Output 4

Q21.

You're working with a logistics company that operates a fleet of trucks to transport goods across multiple warehouses. Each day, a truck records the number of boxes loaded. You are given an array of integers where each element represents the boxes loaded on the truck on day i.

The operations team wants to identify the longest streak of consecutive days during which the total number of boxes loaded is exactly K. This will help them understand optimal loading patterns.

Write a function to find the length of the longest subarray (i.e., streak of consecutive days) where the total number of boxes equals K.

Parameters:

N: INTEGER

The first line of input contains an integer, N, describes number of days. 1<=N<=50 BoxesLoad: INTEGER ARRAY

The second line of input contains N space separated integers representing boxes loaded per day. 1<=BoxesLoad[i]<=1000

K: INTEGER

The third line of input contains an integer representing the target total load. 1<=K<=N Case#:1

Input 7

1 2 1 1 1 3 2

5

Output 4

Case#:2 Input

5

1 1 1 1 1

3

Output 3

Case#:3

Input 5

2 3 1 1 1

5

Output 4

Case#:4 Input

5

1 2 3 4 5

9

Output 3

Q22.

You're a backend engineer at an online advertising platform. Advertisers place bids (in dollars) in

real-time to secure ad space for a given keyword. Your system receives a stream of bid amounts in an array at the end of the day, and the analytics team wants to identify the Kth highest bid to analyze bidder competitiveness.

Your task is to implement a program that returns the Kth largest bid from the array of all bid values.

Parameters:

N: INTEGER

The first line of input contains an integer, N, describes bids 1<=N<=20 BidValues: INTEGER ARRAY

The second line of input contains N space separated integers representing the bids placed throughout the day.

1<=BidValues[i]<=10000 K: INTEGER

The third line of input contains an integer that represents the position from the top in terms of bid value.

1<=K<=N

Case#:1

Input 5

100 200 300 400 500

3

Ouput 300

Case#:2 Input

5

100 500 200 800 700

3

Output 500

Case#:3 Input

7

20 50 10 80 60 30 40

2

Output 60

Case#:4 Input

7

20 50 10 80 60 30 40

7

Output 10

Q23.

You are a data analyst at a SaaS company that tracks daily revenue deltas (profit or loss) for a newly launched product. Each element in the revenue list represents the net change in revenue on that day—it could be positive (gain) or negative (loss).

The management wants to analyze the best-performing consecutive period (could be any number of days) that yielded the maximum net profit. Your task is to find the maximum sum of a contiguous subarray from the revenue data.

Parameters:

N: INTEGER

The first line of input contains an integer, N, number of days. 1<=N<=30 Revenue: INTEGER ARRAY

The second line of input contains N space separated integers representing the revenues of each day. 1<=Revenues[i]<=10000

Case#:1 Input

6

3 -2 5 -1 6 -3

Output 11

Case#:2 Input

4

-5 -2 -1 -7

Output

-1

Case#:3 Input

5

2 3 -6 4 5

Output 9

Case#:4 Input

4

10 5 20 3

Output 38 Q24.

You are a civil engineer tasked with designing a rainwater harvesting system for a city with uneven building heights. During a simulation, you’re given an array where each element represents the height of a building in a row. After heavy rainfall, water gets trapped between the buildings depending on their relative heights.

Your goal is to calculate how much water can be trapped between these buildings once the rain stops.

Parameters:

N: INTEGER

The first line of input contains an integer, N, number of buildings. 1<=N<=20 Heights: INTEGER ARRAY

The second line of input contains N space separated integers representing the height of each building.

1<= Heights [i]<=100

Case#:1 Input

8

0 2 0 4 0 3 0 1

Output 7

Explanation: Water is trapped in the pits between taller buildings, e.g., 2 units at index 2, 2 at index 4, 2 at index 6, and 1 at index 7.

Case#:2 Input

5

3 0 2 0 4

Output 7

Case#:3 Input

3

1 0 1

Output 1

Case#:4 Input

4

5 4 1 2

Output 1

Q25.

You work for a social analytics startup that ranks public personalities based on their daily social

impact score. Each day, every individual is assigned a score, and your team is analyzing trends to identify "influencers" — people whose impact score is greater than or equal to all the individuals that appear after them in the list.

You are given a list of scores (ordered by the timeline), and you need to find all the influencer scores. These are the scores that are not overshadowed by any future scores.

Parameters:

Parameters:

N: INTEGER

The first line of input contains an integer, N, number of persons. 1<=N<=20 Impact\_Scores: INTEGER ARRAY

The second line of input contains N space separated integers representing the impact score of each person.

1<= Impact\_Scores [i]<=1000

Case#:1 Input

6

16 17 4 3 5 2

Output 17 5 2

Explanation:

* 17 is greater than everything to its right,
* 5 is greater than 2,
* 2 is the last element, so it's a leader by default.

Case#:2 Input

6

7 10 4 6 5 2

Output 10 6 5 2

Case#:3 Input

4

1 2 3 4

Output 4

Case#:4 Input

5

50 40 30 20 10

Output

50 40 30 20 10

Q26.

You are a data scientist at a digital marketing company that tracks user engagement scores day-by- day after a new ad campaign. Each day, users generate engagement metrics (positive, negative, or

zero). The marketing manager wants to know how many periods (subarrays of days) had a total engagement exactly equal to a target value K, which is considered a "conversion threshold."

Your task is to calculate how many such continuous periods exist in the data. Parameters:

N: INTEGER

The first line of input contains a single integer, N, represents number of days 1<=N<=20 SCORES: INTEGERS ARRAY

The second line of input contains N space separated integers, represents engagement score for each day.

-1000<=SCORES[i]<=1000 K: INTEGER

The third line of input contains a single integer, K, represents the target engagement threshold.

Case#:1 Input

3

1 1 1

2

Output 2

Explanation: The subarrays [1,1] at indices (0,1) and (1,2) both sum to 2.

Case#:2 Input

3

1 2 3

3

Output 2

Case#:3

Input 8

3 4 7 2 -3 1 4 2

7

Output 4

Case#:4 Input

3

0 0 0

0

Output 6

Q27.

You’re building a feature for an online exam platform where students are allowed to answer certain puzzle-type questions by submitting scrambled versions of words (i.e., anagrams of a target word). To prevent cheating or misinterpretation, the system must verify that two submitted words are

indeed anagrams — meaning they contain the same characters in the same frequency, just possibly in a different order.

Your task is to write a function that checks whether two given strings are valid anagrams of each other.

Parameters:

S1: STRING

The first line of the input contains a string, S1 1<=len(S1)<=100 S2: STRING

The second line of the input contains a string, S2 1<=len(S2)<=100

Output:

Return True if the two strings are anagrams, otherwise False

Case#:1

Input silent listen

Output True Case#:2 Input triangle integral Output True Case#:3 Input hello bello

Output False Case#:4 Input aabbcc abcabc Output True

Case#:5 Input abc ab

Output False Q28.

You are working on the backend of a secure messaging application. Each session generates a unique token that should match the expected reference token. However, due to a known issue in the

message encoding system, tokens sometimes arrive in a rotated format (i.e., characters are shifted in a circular fashion, but no characters are lost or added).

Your job is to verify whether the received token is simply a rotation of the original token. If so, it should still be accepted for authentication.

Parameters:

T1: STRING

The first line of input contains a string, T1, that represents the original token. 1<=len(T1)<=100 T2: STRING

The second line of input contains a string, T2, that represents the received token. 1<=len(T2)<=100

Output:

Return True if received token is a rotation of original token, otherwise False.

Case#1:

Input

authentication cationauthenti Output

True Case#2: Input waterbottle erbottlewat Output True Case#3: Input rotation tationro Output True Case#4:

Input secure rescue Output

False

Case#:5 Input abcde deabc

Output True

Case#:6 Input abc abcd Output

False Q29.

You are building a validation system for a secure messaging app that uses pattern-based character substitution encryption. Every character in a plaintext message is consistently replaced with another character to create the encrypted version. However, to maintain security, no two characters in the original message can map to the same character in the encrypted one, and vice versa.

Your task is to verify whether a received encrypted message is a valid isomorphic transformation of the original message. That means there must exist a one-to-one and consistent mapping between

the characters in the original and encrypted messages

Parameters:

T1: STRING

The first line of input contains a string, T1, that represents the actual message before encryption. 1<=len(T1)<=100

T2: STRING

The second line of input contains a string, T2, that represents the message after encryption. 1<=len(T2)<=100

Output:

Return True if the two messages are isomorphic, otherwise False.

Case#:1 Input paper title

Output True

Explanation:

p -> t, a -> i, e -> l, r -> e,

and mappings are consistent and one-to-one Case#:2 Input egg add

Output True Case#:3 Input foo bar

Output False Case#:4 Input ab aa

Output False

Case#:5 Input engage pmkpkp Output False Q30.

You’re working on a text analysis feature for a journaling app that identifies hidden patterns or

emotional symmetry in users' writings. A common marker is the presence of palindromic phrases —

sequences that read the same forward and backward.

Your job is to implement a feature that, given any input string, finds the longest palindromic substring — a contiguous section of text that is a perfect mirror of itself.

Parameters:

S1: String

The first line of input contains a string S1, text which may include letters, numbers, symbols, or spaces.

1<=len(S1)<=100

Output:

Return the longest palindromic substring from the input.

Case#:1 Input babad Output bab

Explanation: Both "bab" and "aba" are valid — the first longest palindromic substring must be returned.

Case#:2 Input abbd Output bb Case#:3 Input a

Output a

Case#:4 Input racecar Output racecar

Case#:5 Input

noonmadamlevel Output

Madam

Collection of Programming Questions

This document contains 30 programming questions, categorized by the sample problem they were inspired by. Each question includes a problem statement, parameter descriptions with constraints, and three sample input/output cases with explanations.

Questions Similar to Sample 1: Gym Energy Question 1: Library Book Borrowing

Today you decided to visit the library to gather research material. You need to collect books whose total page count is at least P. There are N unique books available in the library. Each of these books has Bi pages.

You want to minimize the number of books you borrow to reach or exceed the target page count

P. Due to library policies, each unique book can only be borrowed at most 2 times.

If borrowing all available books (each twice) does not allow you to reach the target page count, return -1.

Parameters: P :: INTEGER The first line contains an integer, P, denoting the target total page count. P

:: 1 -> 10^5

N :: INTEGER The next line contains an integer, N, denoting the number of unique books. N :: 1

-> 10^5

B :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing the number of pages in the i^{th} book. B[i] :: 1 -> 10^5

Case#: 1 Input: 10 3 3 4 5 Output: 2 Explanation: P = 10 Books available: [3, 4, 5] To reach 10 pages with minimum books: Borrow book with 5 pages (1 time) = 5 pages. Remaining P = 5. Borrow book with 5 pages (2nd time) = 5 pages. Total = 10 pages. Total books borrowed: 2.

Case#: 2 Input: 15 2 3 4 Output: -1 Explanation: P = 15 Books available: [3, 4] Maximum

pages achievable: Book 1 (3 pages) x 2 = 6 pages Book 2 (4 pages) x 2 = 8 pages Total max pages = 6 +

8 = 14. Since 14 < 15, it's not possible to reach the target. Return -1.

Case#: 3 Input: 7 3 1 2 6 Output: 2 Explanation: P = 7 Books available: [1, 2, 6] To reach 7 pages with minimum books: Borrow book with 6 pages (1 time) = 6 pages. Remaining P = 1. Borrow book with 1 page (1 time) = 1 page. Total = 7 pages. Total books borrowed: 2.

Question 2: Ingredient Shopping

You are planning to bake a large cake and need a total of W grams of ingredients. You visit a specialty store that has N different types of ingredients. Each ingredient type Ii has a specific weight in grams.

You want to buy the minimum number of ingredient units such that their total weight is at least

W. The store policy dictates that you can buy at most 2 units of each unique ingredient type, as supplies are limited.

If buying all available ingredient units (each twice) does not allow you to reach the target weight, return -1.

Parameters: W :: INTEGER The first line contains an integer, W, denoting the target total weight in grams. W :: 1 -> 10^5

N :: INTEGER The next line contains an integer, N, denoting the number of unique ingredient types. N

:: 1 -> 10^5

I :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing the weight of the i^{th} ingredient type in grams. I[i] :: 1 -> 10^5

Case#: 1 Input: 20 3 5 8 12 Output: 2 Explanation: W = 20 Ingredients available: [5, 8, 12] To reach 20g with minimum units: Buy ingredient with 12g (1 time) = 12g. Remaining W = 8. Buy ingredient with 8g (1 time) = 8g. Total = 20g. Total units bought: 2.

Case#: 2 Input: 50 2 10 14 Output: -1 Explanation: W = 50 Ingredients available: [10, 14] Maximum weight achievable: Ingredient 1 (10g) x 2 = 20g Ingredient 2 (14g) x 2 = 28g Total max weight = 20 + 28 = 48g. Since 48 < 50, it's not possible to reach the target. Return -1. Case#: 3 Input: 15 4 2 3 4 5

Output: 4 Explanation: W = 15 Ingredients available: [2, 3, 4, 5] To reach 15g with minimum units: Buy ingredient with 5g (1 time) = 5g. Remaining W = 10. Buy ingredient with 5g (2nd time) = 5g. Total

= 10g. Remaining W = 5. Buy ingredient with 4g (1 time) = 4g. Total = 14g. Remaining W = 1. Buy ingredient with 2g (1 time) = 2g. Total = 16g.

Total units bought: 2 (of 5g) + 1 (of 4g) + 1 (of 2g) = 4 units.

Question 3: Treasure Collection

You are on a quest to collect treasures from an ancient ruin. Your goal is to accumulate a total treasure value of at least V. There are N unique treasures scattered throughout the ruin. Each treasure Ti has a specific value.

You want to find the minimum number of treasures you need to collect to reach or exceed the target value V. Due to the fragile nature of the artifacts, each unique treasure can only be collected at most 2 times.

If collecting all available treasures (each twice) does not allow you to reach the target value, return - 1.

Parameters: V :: INTEGER The first line contains an integer, V, denoting the target total treasure value. V :: 1 -> 10^5

N :: INTEGER The next line contains an integer, N, denoting the number of unique treasures. N

:: 1 -> 10^5

T :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing the value of the i^{th} treasure. T[i] :: 1 -> 10^5

Case#: 1 Input: 18 3 7 9 10 Output: 2 Explanation: V = 18 Treasures available: [7, 9, 10] To reach 18 value with minimum treasures: Collect treasure with 10 value (1 time) = 10 value.

Remaining V = 8. Collect treasure with 9 value (1 time) = 9 value. Total = 19 value. Total treasures collected: 2.

Case#: 2 Input: 60 3 5 10 12 Output: -1 Explanation: V = 60 Treasures available: [5, 10, 12]

Maximum value achievable: Treasure 1 (5 value) x 2 = 10 value Treasure 2 (10 value) x 2 = 20

value Treasure 3 (12 value) x 2 = 24 value Total max value = 10 + 20 + 24 = 54 value. Since 54

< 60, it's not possible to reach the target. Return -1.

Case#: 3 Input: 5 2 1 3 Output: 2 Explanation: V = 5 Treasures available: [1, 3] To reach 5 value with minimum treasures: Collect treasure with 3 value (1 time) = 3 value. Remaining V =

2. Collect treasure with 3 value (2nd time) = 3 value. Total = 6 value. Total treasures collected: 2. Question 4: Project Task Completion

You are working on a critical project and need to accumulate a total of P progress points to

complete it. There are N distinct tasks available, and each task Ti provides a certain amount of progress points upon completion.

You want to determine the minimum number of tasks you have to complete such that you reach or exceed the target progress points P. To avoid burnout, each unique task can only be completed at most 2 times.

If completing all available tasks (each twice) does not allow you to reach the target progress points, return -1.

Parameters: P :: INTEGER The first line contains an integer, P, denoting the target total progress points. P :: 1 -> 10^5

N :: INTEGER The next line contains an integer, N, denoting the number of unique tasks. N :: 1

-> 10^5

T :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing the progress points gained by completing the i^{th} task. T[i] :: 1 -> 10^5 Case#: 1 Input: 25 4 6 8 10 12 Output: 3 Explanation: P = 25 Tasks available: [6, 8, 10, 12] To reach 25 points with minimum tasks: Complete task with 12 points (1 time) = 12 points.

Remaining P = 13. Complete task with 12 points (2nd time) = 12 points. Total = 24 points. Remaining P = 1. Complete task with 6 points (1 time) = 6 points. Total = 30 points. Total tasks completed: 2 (of 12 points) + 1 (of 6 points) = 3 tasks.

Case#: 2 Input: 130 3 15 20 25 Output: -1 Explanation: P = 130 Tasks available: [15, 20, 25]

Maximum points achievable: Task 1 (15 points) x 2 = 30 points Task 2 (20 points) x 2 = 40

points Task 3 (25 points) x 2 = 50 points Total max points = 30 + 40 + 50 = 120 points. Since 120 < 130, it's not possible to reach the target. Return -1.

Case#: 3 Input: 11 2 5 1 Output: 3 Explanation: P = 11 Tasks available: [5, 1] To reach 11 points with minimum tasks: Complete task with 5 points (1 time) = 5 points. Remaining P = 6. Complete task with 5 points (2nd time) = 5 points. Total = 10 points. Remaining P = 1. Complete task with 1 point (1 time)

= 1 point. Total = 11 points. Total tasks completed: 2 (of 5 points) + 1

(of 1 point) = 3 tasks.

Questions Similar to Sample 2: Battle between Heroes and Villains Question 1: Server Load Balancing

You are managing a cluster of M servers, all of which have the same processing capacity C. You have N incoming requests, and the processing demand of the i^{th} request is Ri.

When a server with capacity C attempts to handle a request with demand Ri, one of three scenarios can happen:

* If C > Ri: The request is successfully handled, and the server's remaining capacity decreases by Ri.
* If C < Ri: The server becomes overloaded and is no longer able to process any more requests. The request remains unhandled and must be processed by the next available server.
* If C = Ri: Both the server and the request are considered fully utilized/handled, and neither can be used further.

The servers attempt to handle requests one by one in the given order (first request 1, then request 2, and so on). It might be possible that before handling all the requests, all servers

become unavailable. To ensure that all requests are eventually handled, you want to remove some requests from the front of the queue.

Your task is to find the minimum number of requests you need to remove from the front such that all the remaining requests can be successfully handled by the available servers.

Note: If, in the last interaction, both a server and a request are fully utilized/handled and no more servers or requests remain, it would still be considered a success since all the requests are handled.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the number of requests. N :: 1 -> 2\*10^5

M :: INTEGER The next line contains an integer, M, denoting the number of servers. M :: 1 -> 2\*10^5

C :: INTEGER The next line contains an integer, C, denoting the processing capacity of each server. C :: 1 -> 10^9

R :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing the processing demand of the i^{th} request. R[i] :: 1 -> 10^9

Case#: 1 Input: 4 2 10 5 5 8 2 Output: 0 Explanation: We have 2 servers, each with capacity

10. Requests: [5, 5, 8, 2].

1. Server 1 (C=10) attempts Request 1 (R=5). C > R. Server 1's capacity becomes 5. Request 1 handled.
2. Server 1 (C=5) attempts Request 2 (R=5). C = R. Server 1's capacity becomes 0. Server 1 fully utilized. Request 2 handled.
3. Server 2 (C=10) attempts Request 3 (R=8). C > R. Server 2's capacity becomes 2. Request 3 handled.
4. Server 2 (C=2) attempts Request 4 (R=2). C = R. Server 2's capacity becomes 0. Server 2 fully utilized. Request 4 handled. All requests are handled. No need to remove any requests from the front.

Case#: 2 Input: 5 1 10 12 3 4 2 5 Output: 3 Explanation: We have 1 server with capacity 10.

Requests: [12, 3, 4, 2, 5].

* + If 0 requests removed: Server 1 (C=10) attempts Request 1 (R=12). C < R. Server 1 overloaded. Request 1 remains. All requests cannot be handled.
  + If 1 request removed (remove 12): Requests: [3, 4, 2, 5].
* Server 1 (C=10) handles Request 2 (R=3). C becomes 7.
* Server 1 (C=7) handles Request 3 (R=4). C becomes 3.
* Server 1 (C=3) handles Request 4 (R=2). C becomes 1.
* Server 1 (C=1) attempts Request 5 (R=5). C < R. Server 1 overloaded. Request 5 remains. All requests cannot be handled.
  + If 2 requests removed (remove 12, 3): Requests: [4, 2, 5].
* Server 1 (C=10) handles Request 3 (R=4). C becomes 6.
* Server 1 (C=6) handles Request 4 (R=2). C becomes 4.
* Server 1 (C=4) attempts Request 5 (R=5). C < R. Server 1 overloaded. Request 5 remains. All requests cannot be handled.
  + If 3 requests removed (remove 12, 3, 4): Requests: [2, 5].
* Server 1 (C=10) handles Request 4 (R=2). C becomes 8.
* Server 1 (C=8) handles Request 5 (R=5). C becomes 3. All requests are handled. The minimum number of requests to remove is 3.

Case#: 3 Input: 3 3 5 6 1 2 Output: 1 Explanation: We have 3 servers, each with capacity 5.

Requests: [6, 1, 2].

* + If 0 requests removed:
* Server 1 (C=5) attempts Request 1 (R=6). C < R. Server 1 overloaded. Request 1 remains. All requests cannot be handled.
  + If 1 request removed (remove 6): Requests: [1, 2].
* Server 1 (C=5) handles Request 2 (R=1). C becomes 4. Request 2 handled.
* Server 1 (C=4) handles Request 3 (R=2). C becomes 2. Request 3 handled. All requests are handled. The minimum number of requests to remove is 1.

Question 2: Delivery Fleet Optimization

You manage a fleet of M delivery trucks, all of which have the same cargo capacity K. You have N pending orders, and the weight of the i^{th} order is Oi.

When a truck with capacity K attempts to deliver an order with weight Oi, one of three scenarios can happen:

* + If K > Oi: The order is successfully delivered, and the truck's remaining capacity decreases by Oi.
  + If K < Oi: The truck is unable to carry the order and becomes unavailable (e.g., due to mechanical stress). The order remains undelivered and must be handled by the next available truck.
  + If K = Oi: Both the truck and the order are considered fully utilized/delivered, and neither can be used further.

The trucks attempt to deliver orders one by one in the given order (first order 1, then order 2, and so on). It might be possible that before delivering all the orders, all trucks become unavailable. To ensure that all orders are eventually delivered, you want to remove some orders from the front of the queue.

Your task is to find the minimum number of orders you need to remove from the front such that all the remaining orders can be successfully delivered by the available trucks.

Note: If, in the last interaction, both a truck and an order are fully utilized/delivered and no more trucks or orders remain, it would still be considered a success since all the orders are delivered. Parameters: N :: INTEGER The first line contains an integer, N, denoting the number of orders. N :: 1 -

> 2\*10^5

M :: INTEGER The next line contains an integer, M, denoting the number of trucks. M :: 1 -> 2\*10^5 K :: INTEGER The next line contains an integer, K, denoting the cargo capacity of each truck. K

:: 1 -> 10^9

O :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing the weight of the i^{th} order. O[i] :: 1 -> 10^9

Case#: 1 Input: 5 3 20 10 5 15 2 3 Output: 0 Explanation: We have 3 trucks, each with

capacity 20. Orders: [10, 5, 15, 2, 3].

1. Truck 1 (K=20) delivers Order 1 (O=10). K > O. Truck 1's capacity becomes 10. Order 1 delivered.
2. Truck 1 (K=10) delivers Order 2 (O=5). K > O. Truck 1's capacity becomes 5. Order 2 delivered.
3. Truck 1 (K=5) attempts Order 3 (O=15). K < O. Truck 1 unavailable. Order 3 remains.
4. Truck 2 (K=20) delivers Order 3 (O=15). K > O. Truck 2's capacity becomes 5. Order 3 delivered.
5. Truck 2 (K=5) delivers Order 4 (O=2). K > O. Truck 2's capacity becomes 3. Order 4 delivered.
6. Truck 2 (K=3) delivers Order 5 (O=3). K = O. Truck 2's capacity becomes 0. Truck 2 fully

utilized. Order 5 delivered. All orders are delivered. No need to remove any orders from the front. Case#: 2 Input: 4 1 100 110 50 20 10 Output: 1 Explanation: We have 1 truck with capacity

1. Orders: [110, 50, 20, 10].
   * If 0 orders removed: Truck 1 (K=100) attempts Order 1 (O=110). K < O. Truck 1 unavailable. Order 1 remains. All orders cannot be delivered.
   * If 1 order removed (remove 110): Orders: [50, 20, 10].

* Truck 1 (K=100) delivers Order 2 (O=50). K becomes 50.
* Truck 1 (K=50) delivers Order 3 (O=20). K becomes 30.
* Truck 1 (K=30) delivers Order 4 (O=10). K becomes 20. All orders are delivered. The minimum number of orders to remove is 1.

Case#: 3 Input: 4 1 10 12 8 2 3 Output: 2 Explanation: We have 1 truck with capacity 10.

Orders: [12, 8, 2, 3].

* + If 0 orders removed: Truck 1 (K=10) attempts Order 1 (O=12). K < O. Truck 1 unavailable. Order 1 remains. All orders cannot be delivered.
  + If 1 order removed (remove 12): Orders: [8, 2, 3].
* Truck 1 (K=10) delivers Order 2 (O=8). K becomes 2.
* Truck 1 (K=2) delivers Order 3 (O=2). K becomes 0. Truck 1 fully utilized.
* Order 4 (O=3) remains. No more trucks. All orders cannot be delivered.
  + If 2 orders removed (remove 12, 8): Orders: [2, 3].
* Truck 1 (K=10) delivers Order 3 (O=2). K becomes 8.
* Truck 1 (K=8) delivers Order 4 (O=3). K becomes 5. All orders are delivered. The minimum number of orders to remove is 2.

Question 3: Project Task Completion

You are leading a team of M developers, all of whom have the same coding capacity P. You have N project tasks, and the complexity of the i^{th} task is Ti.

When a developer with coding capacity P attempts to complete a task with complexity Ti, one of three scenarios can happen:

* + If P > Ti: The task is successfully completed, and the developer's remaining capacity decreases by Ti.
  + If P < Ti: The developer is unable to complete the task and becomes exhausted (unavailable). The task remains incomplete and must be assigned to the next available developer.
  + If P = Ti: Both the developer and the task are considered fully utilized/completed, and neither can be used further.

The developers attempt to complete tasks one by one in the given order (first task 1, then task 2, and so on). It might be possible that before completing all the tasks, all developers become unavailable. To ensure that all tasks are eventually completed, you want to remove some tasks from the front of the queue.

Your task is to find the minimum number of tasks you need to remove from the front such that all the remaining tasks can be successfully completed by the available developers.

Note: If, in the last interaction, both a developer and a task are fully utilized/completed and no more developers or tasks remain, it would still be considered a success since all the tasks are completed.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the number of tasks. N :: 1 -> 2\*10^5

M :: INTEGER The next line contains an integer, M, denoting the number of developers. M :: 1

-> 2\*10^5

P :: INTEGER The next line contains an integer, P, denoting the coding capacity of each developer. P :: 1 -> 10^9

T :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing the complexity of the i^{th} task. T[i] :: 1 -> 10^9

Case#: 1 Input: 4 2 15 5 10 7 8 Output: 0 Explanation: We have 2 developers, each with

capacity 15. Tasks: [5, 10, 7, 8].

1. Developer 1 (P=15) attempts Task 1 (T=5). P > T. Developer 1's capacity becomes 10. Task 1 completed.
2. Developer 1 (P=10) attempts Task 2 (T=10). P = T. Developer 1's capacity becomes 0. Developer 1 fully utilized. Task 2 completed.
3. Developer 2 (P=15) attempts Task 3 (T=7). P > T. Developer 2's capacity becomes 8. Task 3 completed.
4. Developer 2 (P=8) attempts Task 4 (T=8). P = T. Developer 2's capacity becomes 0. Developer 2 fully utilized. Task 4 completed. All tasks are completed. No need to remove any tasks from the front.

Case#: 2 Input: 5 1 20 25 10 5 3 8 Output: 2 Explanation: We have 1 developer with capacity

1. Tasks: [25, 10, 5, 3, 8].
   * If 0 tasks removed: Developer 1 (P=20) attempts Task 1 (T=25). P < T. Developer 1 exhausted. Task 1 remains. All tasks cannot be completed.
   * If 1 task removed (remove 25): Tasks: [10, 5, 3, 8].

* Developer 1 (P=20) completes Task 2 (T=10). P becomes 10.
* Developer 1 (P=10) completes Task 3 (T=5). P becomes 5.
* Developer 1 (P=5) completes Task 4 (T=3). P becomes 2.
* Developer 1 (P=2) attempts Task 5 (T=8). P < T. Developer 1 exhausted. Task 5 remains. All tasks cannot be completed.
  + If 2 tasks removed (remove 25, 10): Tasks: [5, 3, 8].
* Developer 1 (P=20) completes Task 3 (T=5). P becomes 15.
* Developer 1 (P=15) completes Task 4 (T=3). P becomes 12.
* Developer 1 (P=12) completes Task 5 (T=8). P becomes 4. All tasks are completed. The minimum number of tasks to remove is 2.

Case#: 3 Input: 3 1 5 6 1 2 Output: 1 Explanation: We have 1 developer with capacity 5.

Tasks: [6, 1, 2].

* + If 0 tasks removed: Developer 1 (P=5) attempts Task 1 (T=6). P < T. Developer 1 exhausted. Task 1 remains. All tasks cannot be completed.
  + If 1 task removed (remove 6): Tasks: [1, 2].
* Developer 1 (P=5) completes Task 2 (T=1). P becomes 4.
* Developer 1 (P=4) completes Task 3 (T=2). P becomes 2. All tasks are completed. The minimum number of tasks to remove is 1.

Question 4: Water Resource Management

You are managing M water sources (e.g., pumps, wells), all of which have the same total water volume W. You have N empty tanks to fill, and the capacity of the i^{th} tank is Ti.

When a water source with volume W attempts to fill a tank with capacity Ti, one of three scenarios can happen:

* + If W > Ti: The tank is successfully filled, and the water source's remaining volume

decreases by Ti.

* + If W < Ti: The water source is unable to fill the tank completely and becomes depleted (unavailable). The tank remains partially filled and must be filled by the next available water source.
  + If W = Ti: Both the water source and the tank are considered fully utilized/filled, and neither can be used further.

The water sources attempt to fill tanks one by one in the given order (first tank 1, then tank 2, and so on). It might be possible that before filling all the tanks, all water sources become unavailable. To ensure that all tanks are eventually filled, you want to remove some tanks from the front of the queue.

Your task is to find the minimum number of tanks you need to remove from the front such that all the remaining tanks can be successfully filled by the available water sources.

Note: If, in the last interaction, both a water source and a tank are fully utilized/filled and no more water sources or tanks remain, it would still be considered a success since all the tanks are filled.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the number of tanks. N :: 1 -> 2\*10^5

M :: INTEGER The next line contains an integer, M, denoting the number of water sources. M :: 1 -> 2\*10^5

W :: INTEGER The next line contains an integer, W, denoting the total water volume of each source. W :: 1 -> 10^9

T :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing the capacity of the i^{th} tank. T[i] :: 1 -> 10^9

Case#: 1 Input: 5 3 100 50 20 10 5 15 Output: 0 Explanation: We have 3 water sources, each

with volume 100. Tanks: [50, 20, 10, 5, 15].

1. Source 1 (W=100) fills Tank 1 (T=50). W > T. Source 1's volume becomes 50. Tank 1 filled.
2. Source 1 (W=50) fills Tank 2 (T=20). W > T. Source 1's volume becomes 30. Tank 2 filled.
3. Source 1 (W=30) fills Tank 3 (T=10). W > T. Source 1's volume becomes 20. Tank 3 filled.
4. Source 1 (W=20) fills Tank 4 (T=5). W > T. Source 1's volume becomes 15. Tank 4 filled.
5. Source 1 (W=15) fills Tank 5 (T=15). W = T. Source 1's volume becomes 0. Source 1 fully utilized. Tank 5 filled. All tanks are filled. No need to remove any tanks from the front.

Case#: 2 Input: 4 1 50 60 10 5 20 Output: 1 Explanation: We have 1 water source with

volume 50. Tanks: [60, 10, 5, 20].

* + If 0 tanks removed: Source 1 (W=50) attempts Tank 1 (T=60). W < T. Source 1 depleted. Tank 1 remains. All tanks cannot be filled.
  + If 1 tank removed (remove 60): Tanks: [10, 5, 20].
* Source 1 (W=50) fills Tank 2 (T=10). W becomes 40.
* Source 1 (W=40) fills Tank 3 (T=5). W becomes 35.
* Source 1 (W=35) fills Tank 4 (T=20). W becomes 15. All tanks are filled. The minimum number of tanks to remove is 1.

Case#: 3 Input: 5 2 10 12 8 2 3 5 Output: 1 Explanation: We have 2 water sources, each with

volume 10. Tanks: [12, 8, 2, 3, 5].

* + If 0 tanks removed:
* Source 1 (W=10) attempts Tank 1 (T=12). W < T. Source 1 depleted. Tank 1 remains.
* Source 2 (W=10) attempts Tank 1 (T=12). W < T. Source 2 depleted. Tank 1 remains. All tanks cannot be filled.
  + If 1 tank removed (remove 12): Tanks: [8, 2, 3, 5].
* Source 1 (W=10) fills Tank 2 (T=8). W becomes 2.
* Source 1 (W=2) fills Tank 3 (T=2). W becomes 0. Source 1 fully utilized.
* Source 2 (W=10) fills Tank 4 (T=3). W becomes 7.
* Source 2 (W=7) fills Tank 5 (T=5). W becomes 2. All tanks are filled. The minimum number of tanks to remove is 1.

Questions Similar to Sample 3: Road in Rugged Terrain Question 1: Riverbed Deepening Project

You are tasked with deepening a riverbed to ensure smooth navigation. You know the current depth of each segment of the riverbed, i.e., the i^{th} segment is Di meters deep.

You need to transform the riverbed into a strictly increasing depth profile for navigation, meaning for each i^{th} segment where 2 \le i \le N, the resultant Di-1 < Di. To achieve this, you employ a powerful dredging team to help you deepen the segments. On day D, the team can increase the depth for each segment that you scheduled that day by 2^D - 1 meters each.

You are allowed to assign the team to dredge on multiple segments and/or dredge on the same segments for multiple days.

Your task is to find the minimum number of days needed to transform the riverbed as per your requirements.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the number of elements in D. N :: 1 -> 10^5

D :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing Di, the current depth of the i^{th} segment. D[i] :: -10^9 -> 10^9

Case#: 1 Input: 3 1 2 3 Output: 0 Explanation: The depths [1, 2, 3] are already strictly increasing. No dredging is needed.

Case#: 2 Input: 2 5 2 Output: 3 Explanation: Initial depths: [5, 2]. We need D\_final[0] < D\_final[1]. Calculate target final depths: D\_final[0] = D[0] = 5 D\_final[1] = max(D[1], D\_final[0] + 1) = max(2, 5 + 1) = max(2, 6) = 6. Target D\_final = [5, 6]. Needed increases: D[0]: D\_final[0] - D[0] = 5 - 5 = 0. (No increase needed) D[1]: D\_final[1] - D[1] = 6 - 2 = 4. For an increase of 4: We need to find the minimum D such that 2^D - 1 \ge 4. 2^1 - 1 = 1 2^2 - 1 = 3 2^3 - 1 = 7 The smallest D is 3. The maximum day needed is 3.

Case#: 3 Input: 4 10 10 5 5 Output: 4 Explanation: Initial depths: [10, 10, 5, 5]. Calculate target final depths (D\_final[i] = max(D[i], D\_final[i-1] + 1)): D\_final[0] = D[0] = 10 D\_final[1] = max(D[1], D\_final[0]

+ 1) = max(10, 10 + 1) = max(10, 11) = 11 D\_final[2] = max(D[2], D\_final[1]

+ 1) = max(5, 11 + 1) = max(5, 12) = 12 D\_final[3] = max(D[3], D\_final[2] + 1) = max(5, 12 + 1) =

max(5, 13) = 13 Target D\_final = [10, 11, 12, 13]. Needed increases: D[0]: D\_final[0] - D[0] = 10

- 10 = 0. D[1]: D\_final[1] - D[1] = 11 - 10 = 1. Smallest D for 1 is 1 (2^1 - 1 = 1). D[2]: D\_final[2] -

D[2] = 12 - 5 = 7. Smallest D for 7 is 3 (2^3 - 1 = 7). D[3]: D\_final[3] - D[3] = 13 - 5 = 8. Smallest

D for 8 is 4 (2^4 - 1 = 15 \ge 8). The maximum day needed across all segments is max(0, 1, 3,

4) = 4.

Question 2: Mountain Range Flattening

You need to flatten a mountain range for a new railway line. You know the current height of each peak, i.e., the i^{th} peak is Hi meters tall.

You need to transform the mountain range into a strictly downward sloping profile for the railway, meaning for each i^{th} peak where 2 \le i \le N, the resultant Hi-1 > Hi. To do so, you employ a powerful blasting team to help you reduce the height of the peaks. On day D, the team can reduce the height for each peak that you scheduled that day by 2^D - 1 meters each.

You are allowed to assign the team to blast on multiple peaks and/or blast on the same peaks for multiple days.

Your task is to find the minimum number of days needed to transform the mountain range as per your requirements.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the number of elements in H. N :: 1 -> 10^5

H :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing Hi, the height of the i^{th} peak. H[i] :: -10^9 -> 10^9

Case#: 1 Input: 3 10 9 8 Output: 0 Explanation: The heights [10, 9, 8] are already strictly decreasing. No blasting is needed.

Case#: 2 Input: 2 10 14 Output: 3 Explanation: Initial heights: [10, 14]. We need H\_final[0] > H\_final[1]. Calculate target final heights (H\_final[i] = min(H[i], H\_final[i-1] - 1)): H\_final[0] = H[0] = 10 H\_final[1] = min(H[1], H\_final[0] - 1) = min(14, 10 - 1) = min(14, 9) = 9. Target H\_final = [10, 9].

Needed reductions: H[0]: H[0] - H\_final[0] = 10 - 10 = 0. (No reduction needed) H[1]: H[1] - H\_final[1]

= 14 - 9 = 5. For a reduction of 5: We need to find the minimum D such that 2^D - 1

\ge 5. 2^1 - 1 = 1 2^2 - 1 = 3 2^3 - 1 = 7 The smallest D is 3. The maximum day needed is 3.

Case#: 3 Input: 4 10 18 19 20 Output: 4 Explanation: Initial heights: [10, 18, 19, 20]. Calculate target final heights (H\_final[i] = min(H[i], H\_final[i-1] - 1)): H\_final[0] = H[0] = 10 H\_final[1] = min(18, 10 - 1)

= 9 H\_final[2] = min(19, 9 - 1) = 8 H\_final[3] = min(20, 8 - 1) = 7

Target H\_final = [10, 9, 8, 7]. Needed reductions: H[0]: 10 - 10 = 0. H[1]: 18 - 9 = 9. Smallest D

for 9 is 4 (2^4 - 1 = 15 \ge 9). H[2]: 19 - 8 = 11. Smallest D for 11 is 4 (2^4 - 1 = 15 \ge 11). H[3]:

20 - 7 = 13. Smallest D for 13 is 4 (2^4 - 1 = 15 \ge 13). The maximum day needed across all

peaks is max(0, 4, 4, 4) = 4.

Question 3: Data Signal Smoothing

You are working with a sequence of data signals, where the i^{th} signal has an amplitude Ai. You need to transform this sequence into a strictly increasing amplitude profile for analysis, meaning for each i^{th} signal where 2 \le i \le N, the resultant Ai-1 < Ai. To do this, you employ a powerful signal processing algorithm to boost the amplitude of the signals. On day D, the algorithm can increase the amplitude for each signal that you scheduled that day by 2^D - 1 units each.

You are allowed to assign the algorithm to process multiple signals and/or process the same signals for multiple days.

Your task is to find the minimum number of days needed to transform the signal sequence as per your requirements.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the number of elements in A. N :: 1 -> 10^5

A :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing Ai, the amplitude of the i^{th} signal. A[i] :: -10^9 -> 10^9

Case#: 1 Input: 3 -5 -4 -3 Output: 0 Explanation: The amplitudes [-5, -4, -3] are already strictly increasing. No boosting is needed.

Case#: 2 Input: 2 10 5 Output: 3 Explanation: Initial amplitudes: [10, 5]. We need A\_final[0] < A\_final[1]. Calculate target final amplitudes (A\_final[i] = max(A[i], A\_final[i-1] + 1)): A\_final[0] = A[0] = 10 A\_final[1] = max(A[1], A\_final[0] + 1) = max(5, 10 + 1) = max(5, 11) = 11. Target A\_final = [10, 11]. Needed increases: A[0]: A\_final[0] - A[0] = 10 - 10 = 0. A[1]: A\_final[1] - A[1] = 11 - 5 = 6. For an increase of 6: We need to find the minimum D such that 2^D - 1 \ge 6. 2^1 - 1

= 1 2^2 - 1 = 3 2^3 - 1 = 7 The smallest D is 3. The maximum day needed is 3.

Case#: 3 Input: 9 1 1 1 1 1 1 1 1 1 Output: 4 Explanation: Initial amplitudes: [1, 1, 1, 1, 1, 1, 1, 1, 1]. Calculate target final amplitudes (A\_final[i] = max(A[i], A\_final[i-1] + 1)): A\_final[0] = 1 A\_final[1] = 2 A\_final[2] = 3 A\_final[3] = 4 A\_final[4] = 5 A\_final[5] = 6 A\_final[6] = 7 A\_final[7] = 8 A\_final[8] = 9

Target A\_final = [1, 2, 3, 4, 5, 6, 7, 8, 9]. Needed increases: A[0]: 1 - 1 = 0. A[1]:

2 - 1 = 1. Smallest D for 1 is 1. A[2]: 3 - 1 = 2. Smallest D for 2 is 2. A[3]: 4 - 1 = 3. Smallest D

for 3 is 2. A[4]: 5 - 1 = 4. Smallest D for 4 is 3. A[5]: 6 - 1 = 5. Smallest D for 5 is 3. A[6]: 7 - 1 =

1. Smallest D for 6 is 3. A[7]: 8 - 1 = 7. Smallest D for 7 is 3. A[8]: 9 - 1 = 8. Smallest D for 8 is 4

(2^4 - 1 = 15 \ge 8). The maximum day needed across all segments is max(0, 1, 2, 2, 3, 3, 3, 3,

4) = 4.

Question 4: Temperature Gradient Creation

You are managing a specialized heating/cooling system that controls the temperature of N adjacent chambers. The current temperature of the i^{th} chamber is Ti degrees Celsius. You need to transform the chamber temperatures into a strictly downward sloping gradient,

meaning for each i^{th} chamber where 2 \le i \le N, the resultant Ti-1 > Ti. To achieve this, you employ a powerful thermal regulation unit to adjust the temperature of the chambers. On day D, the unit can reduce the temperature for each chamber that you scheduled that day by 2^D - 1 degrees Celsius each.

You are allowed to assign the unit to regulate multiple chambers and/or regulate the same chambers for multiple days.

Your task is to find the minimum number of days needed to transform the temperature gradient as per your requirements.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the number of elements in T. N :: 1 -> 10^5

T :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing Ti, the temperature of the i^{th} chamber. T[i] :: -10^9 -> 10^9

Case#: 1 Input: 3 20 15 10 Output: 0 Explanation: The temperatures [20, 15, 10] are already strictly decreasing. No regulation is needed.

Case#: 2 Input: 2 -5 -3 Output: 2 Explanation: Initial temperatures: [-5, -3]. We need T\_final[0]

> T\_final[1]. Calculate target final temperatures (T\_final[i] = min(T[i], T\_final[i-1] - 1)): T\_final[0] = T[0]

= -5 T\_final[1] = min(T[1], T\_final[0] - 1) = min(-3, -5 - 1) = min(-3, -6) = -6. Target T\_final =

[-5, -6]. Needed reductions: T[0]: -5 - (-5) = 0. T[1]: -3 - (-6) = 3. For a reduction of 3: We need to find the minimum D such that 2^D - 1 \ge 3. 2^1 - 1 = 1 2^2 - 1 = 3 The smallest D is 2. The maximum day needed is 2.

Case#: 3 Input: 4 10 20 21 22 Output: 4 Explanation: Initial temperatures: [10, 20, 21, 22]. Calculate target final temperatures (T\_final[i] = min(T[i], T\_final[i-1] - 1)): T\_final[0] = T[0] = 10 T\_final[1] = min(20, 10 - 1) = 9 T\_final[2] = min(21, 9 - 1) = 8 T\_final[3] = min(22, 8 - 1) = 7

Target T\_final = [10, 9, 8, 7]. Needed reductions: T[0]: 10 - 10 = 0. T[1]: 20 - 9 = 11. Smallest D

for 11 is 4 (2^4 - 1 = 15 \ge 11). T[2]: 21 - 8 = 13. Smallest D for 13 is 4 (2^4 - 1 = 15 \ge 13).

T[3]: 22 - 7 = 15. Smallest D for 15 is 4 (2^4 - 1 = 15 \ge 15). The maximum day needed across

all chambers is max(0, 4, 4, 4) = 4.

Questions Similar to Sample 4: Array to Mountain Question 1: Valley Array Transformation

You are given an array of size N. You need to change this array into a "valley". By "valley" we mean that the elements at either end of the array should be equal. Then, as you move towards the middle from both ends, the next element is just one less than the previous one. So, it would have a trough (lowest point) in the middle and increases if you go towards either end, just like a valley.

Examples of valleys are [5, 4, 3, 4, 5] or [8, 7, 6, 6, 7, 8]. But the array [5, 4, 2, 4, 5] is not a

valley because from 4 to 2 the difference is 2. The array [5, 4, 3, 5] is also not a valley because the elements 4 and 3 are not equal from both ends relative to the trough.

You need to find the minimum number of elements that should be changed in order to make the array a valley. You can make the elements negative or zero as well.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the number of elements in the array. N :: 1 -> 10^5

array :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing the i^{th} element of the array. array[i] :: 1 -> 10^6

Case#: 1 Input: 5 1 2 3 4 5 Output: 2 Explanation: array = [1, 2, 3, 4, 5]. The optimal valley

array is [5, 4, 3, 4, 5]. Comparing [1, 2, 3, 4, 5] with [5, 4, 3, 4, 5]:

* 1 vs 5 (change)
* 2 vs 4 (change)
* 3 vs 3 (no change)
* 4 vs 4 (no change)
* 5 vs 5 (no change) Total 2 changes.

Case#: 2 Input: 6 10 9 8 8 9 10 Output: 0 Explanation: array = [10, 9, 8, 8, 9, 10]. This array already conforms to the "valley" definition. All elements match the target valley, so 0 changes are needed.

Case#: 3 Input: 4 1 1 1 1 Output: 2 Explanation: array = [1, 1, 1, 1]. The optimal valley array

is [1, 0, 0, 1]. Comparing [1, 1, 1, 1] with [1, 0, 0, 1]:

* 1 vs 1 (no change)
* 1 vs 0 (change)
* 1 vs 0 (change)
* 1 vs 1 (no change) Total 2 changes.

Question 2: Plateau Array Transformation

You are given an array of size N. You need to change this array into a "plateau". By "plateau" we mean that the elements at either end of the array should be equal. Then, as you move towards the middle from both ends, the next element is just one more than the previous one, until a central "plateau" where elements are equal.

Examples of plateaus are [1, 2, 3, 3, 2, 1] or [1, 2, 2, 2, 1]. But the array [1, 2, 4, 4, 2, 1] is not a

plateau because from 2 to 4 the difference is 2. The array [1, 2, 3, 1] is also not a plateau because the elements are not symmetric from both ends.

You need to find the minimum number of elements that should be changed in order to make the

array a plateau. You can make the elements negative or zero as well.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the number of elements in the array. N :: 1 -> 10^5

array :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing the i^{th} element of the array. array[i] :: 1 -> 10^6

Case#: 1 Input: 5 5 4 3 2 1 Output: 2 Explanation: array = [5, 4, 3, 2, 1]. The optimal plateau

array is [1, 2, 3, 2, 1]. Comparing [5, 4, 3, 2, 1] with [1, 2, 3, 2, 1]:

* 5 vs 1 (change)
* 4 vs 2 (change)
* 3 vs 3 (no change)
* 2 vs 2 (no change)
* 1 vs 1 (no change) Total 2 changes.

Case#: 2 Input: 4 1 2 3 4 Output: 2 Explanation: array = [1, 2, 3, 4]. The optimal plateau array

is [1, 2, 2, 1]. Comparing [1, 2, 3, 4] with [1, 2, 2, 1]:

* 1 vs 1 (no change)
* 2 vs 2 (no change)
* 3 vs 2 (change)
* 4 vs 1 (change) Total 2 changes.

Case#: 3 Input: 7 10 1 20 3 20 1 10 Output: 4 Explanation: array = [10, 1, 20, 3, 20, 1, 10].

The optimal plateau array is [0, 1, 2, 3, 2, 1, 0]. Comparing [10, 1, 20, 3, 20, 1, 10] with [0, 1, 2,

3, 2, 1, 0]:

* 10 vs 0 (change)
* 1 vs 1 (no change)
* 20 vs 2 (change)
* 3 vs 3 (no change)
* 20 vs 2 (change)
* 1 vs 1 (no change)
* 10 vs 0 (change) Total 4 changes.

Question 3: Double-Step Mountain Array Transformation

You are given an array of size N. You need to change this array into a "double-step mountain". By "double-step mountain" we mean that the elements at either end of the array should be equal. Then, as you move towards the middle from both ends, the next element is just two more than the previous one. So, it would have a peak in the middle and decreases by two if you go towards either end.

Examples of double-step mountains are [1, 3, 5, 3, 1] or [6, 8, 10, 10, 8, 6]. But the array [1, 3,

6, 3, 1] is not a double-step mountain because from 3 to 6 the difference is 3. The array [1, 3, 5, 1] is also not a double-step mountain because the elements are not symmetric from both ends. You need to find the minimum number of elements that should be changed in order to make the array a double-step mountain. You can make the elements negative or zero as well.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the number of elements in the array. N :: 1 -> 10^5

array :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing the i^{th} element of the array. array[i] :: 1 -> 10^6

Case#: 1 Input: 5 1 3 5 4 1 Output: 1 Explanation: array = [1, 3, 5, 4, 1]. The optimal

double-step mountain array is [1, 3, 5, 3, 1]. Comparing [1, 3, 5, 4, 1] with [1, 3, 5, 3, 1]:

* 1 vs 1 (no change)
* 3 vs 3 (no change)
* 5 vs 5 (no change)
* 4 vs 3 (change)
* 1 vs 1 (no change) Total 1 change.

Case#: 2 Input: 6 10 12 14 14 12 10 Output: 0 Explanation: array = [10, 12, 14, 14, 12, 10]. This array already conforms to the "double-step mountain" definition. All elements match the target double- step mountain, so 0 changes are needed.

Case#: 3 Input: 7 10 2 20 6 20 2 10 Output: 4 Explanation: array = [10, 2, 20, 6, 20, 2, 10].

The optimal double-step mountain array is [0, 2, 4, 6, 4, 2, 0]. Comparing [10, 2, 20, 6, 20, 2, 10]

with [0, 2, 4, 6, 4, 2, 0]:

* 10 vs 0 (change)
* 2 vs 2 (no change)
* 20 vs 4 (change)
* 6 vs 6 (no change)
* 20 vs 4 (change)
* 2 vs 2 (no change)
* 10 vs 0 (change) Total 4 changes.

Question 4: Triple-Step Valley Array Transformation

You are given an array of size N. You need to change this array into a "triple-step valley". By "triple- step valley" we mean that the elements at either end of the array should be equal. Then, as you move towards the middle from both ends, the next element is just three less than the previous one. So, it would have a trough (lowest point) in the middle and increases by three if you go towards either end.

Examples of triple-step valleys are [10, 7, 4, 7, 10] or [15, 12, 9, 9, 12, 15]. But the array [10, 7,

3, 7, 10] is not a triple-step valley because from 7 to 3 the difference is 4. The array [10, 7, 4, 10] is also not a triple-step valley because the elements are not symmetric from both ends.

You need to find the minimum number of elements that should be changed in order to make the array a triple-step valley. You can make the elements negative or zero as well.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the number of elements in the array. N :: 1 -> 10^5

array :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing the i^{th} element of the array. array[i] :: 1 -> 10^6

Case#: 1 Input: 5 10 7 4 8 10 Output: 1 Explanation: array = [10, 7, 4, 8, 10]. The optimal

triple-step valley array is [10, 7, 4, 7, 10]. Comparing [10, 7, 4, 8, 10] with [10, 7, 4, 7, 10]:

* 10 vs 10 (no change)
* 7 vs 7 (no change)
* 4 vs 4 (no change)
* 8 vs 7 (change)
* 10 vs 10 (no change) Total 1 change.

Case#: 2 Input: 6 15 12 9 9 12 15 Output: 0 Explanation: array = [15, 12, 9, 9, 12, 15]. This array already conforms to the "triple-step valley" definition. All elements match the target triple-step valley, so 0 changes are needed.

Case#: 3 Input: 7 10 -3 20 -9 20 -3 10 Output: 4 Explanation: array = [10, -3, 20, -9, 20, -3,

10]. The optimal triple-step valley array is [0, -3, -6, -9, -6, -3, 0]. Comparing [10, -3, 20, -9, 20,

-3, 10] with [0, -3, -6, -9, -6, -3, 0]:

* 10 vs 0 (change)
* -3 vs -3 (no change)
* 20 vs -6 (change)
* -9 vs -9 (no change)
* 20 vs -6 (change)
* -3 vs -3 (no change)
* 10 vs 0 (change) Total 4 changes.

Questions Similar to Sample 5: Interesting String Slicing Question 1: Resource Bundling

You have a collection of N resources. Each resource is identified by a unique integer Ri representing its type. You can rearrange these resources in any order. You want to cut this collection into some contiguous bundles such that, after cutting, all the bundles are equal to one another (meaning each bundle contains the exact same count of each resource type).

You cannot rearrange the resources within the cut bundles or join bundles together. You want to make the number of bundles as large as possible. What is the maximum number of bundles you can get?

Note: You can observe that you may not want to cut the collection at all, therefore the number of bundles is 1. Hence, the answer always exists.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the total number of resources. N :: 1 -> 2 \* 10^5

R :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer Ri describing the type of the i^{th} resource. R[i] :: 1 -> 10^9

Case#: 1 Input: 5 7 7 7 7 7 Output: 5 Explanation: You have 5 resources, all of type 7. You

can cut it into 5 bundles: [7] + [7] + [7] + [7] + [7]. Each bundle is identical.

Case#: 2 Input: 6 1 2 1 2 3 3 Output: 2 Explanation: You have resources: [1, 2, 1, 2, 3, 3].

Counts: Type 1: 2, Type 2: 2, Type 3: 2. Total resources: 6. You can rearrange the resources as

[1, 2, 3, 1, 2, 3]. You can then cut it into 2 bundles: [1, 2, 3] + [1, 2, 3]. Each bundle is identical.

The total number of resources (6) and the count of each resource type (2 for 1, 2 for 2, 2 for 3) are all divisible by 2.

Case#: 3 Input: 7 1 1 1 2 2 3 4 Output: 1 Explanation: You have resources: [1, 1, 1, 2, 2, 3, 4].

Total resources: 7. Counts: Type 1: 3, Type 2: 2, Type 3: 1, Type 4: 1. The total number of

resources (7) is a prime number. The counts (3, 2, 1, 1) share no common factors with 7, nor

with each other (except 1). The greatest common divisor of 7, 3, 2, 1, 1 is 1. Therefore, you can only get 1 piece (the original collection itself).

Question 2: Ingredient Packing

You are preparing ingredients for a large batch of identical meal kits. You have N individual ingredients. Each ingredient is identified by an integer Ii representing its type (e.g., 1 for "flour", 2 for "sugar"). You can rearrange these ingredients in any order. You want to pack this collection into some contiguous boxes such that, after packing, all the boxes are equal to one another (meaning each box contains the exact same count of each ingredient type).

You cannot rearrange the ingredients within the packed boxes or combine boxes. You want to

make the number of boxes as large as possible. What is the maximum number of boxes you can get? Note: The answer always exists (at least 1 box).

Parameters: N :: INTEGER The first line contains an integer, N, denoting the total number of ingredients. N :: 1 -> 2 \* 10^5

I :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer Ii describing the type of the i^{th} ingredient. I[i] :: 1 -> 10^9

Case#: 1 Input: 4 10 10 10 10 Output: 4 Explanation: You have 4 ingredients, all of type 10. You can

pack it into 4 boxes: [10] + [10] + [10] + [10]. Each box is identical.

Case#: 2 Input: 8 5 6 5 6 7 7 8 8 Output: 2 Explanation: You have ingredients: [5, 6, 5, 6, 7, 7,

8, 8]. Total ingredients: 8. Counts: Type 5: 2, Type 6: 2, Type 7: 2, Type 8: 2. The total number of

ingredients (8) and the count of each ingredient type (2 for 5, 2 for 6, 2 for 7, 2 for 8) are all

divisible by 2. The greatest common divisor of 8, 2, 2, 2, 2 is 2. You can rearrange the

ingredients as [5, 6, 7, 8, 5, 6, 7, 8]. You can then cut it into 2 boxes: [5, 6, 7, 8] + [5, 6, 7, 8]. Each box is identical.

Case#: 3 Input: 9 1 2 3 1 2 3 1 2 4 Output: 1 Explanation: You have ingredients: [1, 2, 3, 1, 2,

3, 1, 2, 4]. Total ingredients: 9. Counts: Type 1: 3, Type 2: 3, Type 3: 2, Type 4: 1. The total

number of ingredients (9) is divisible by 1, 3, 9. The counts are: 3, 3, 2, 1. The greatest common

divisor of 9, 3, 3, 2, 1 is 1. Therefore, you can only get 1 box (the original collection itself).

Question 3: Music Playlist Segmentation

You have a master playlist of N songs. Each song is identified by an integer Gi representing its genre. You can rearrange the order of songs in this master playlist. You want to segment this playlist into some contiguous sub-playlists such that, after segmentation, all the sub-playlists are equal to one another (meaning each sub-playlist contains the exact same count of each genre). You cannot rearrange the songs within the segmented sub-playlists or combine sub-playlists.

You want to make the number of sub-playlists as large as possible. What is the maximum number of sub-playlists you can get?

Note: The answer always exists (at least 1 sub-playlist).

Parameters: N :: INTEGER The first line contains an integer, N, denoting the total number of songs. N

:: 1 -> 2 \* 10^5

G :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer Gi describing the genre of the i^{th} song. G[i] :: 1 -> 10^9

Case#: 1 Input: 6 1 1 1 1 1 1 Output: 6 Explanation: You have 6 songs, all of genre 1. You

can segment it into 6 sub-playlists: [1] + [1] + [1] + [1] + [1] + [1]. Each sub-playlist is identical.

Case#: 2 Input: 12 10 20 30 10 20 30 10 20 30 10 20 30 Output: 4 Explanation: You have

songs: [10, 20, 30, 10, 20, 30, 10, 20, 30, 10, 20, 30]. Total songs: 12. Counts: Genre 10: 4,

Genre 20: 4, Genre 30: 4. The total number of songs (12) and the count of each genre (4 for 10,

4 for 20, 4 for 30) are all divisible by 4. The greatest common divisor of 12, 4, 4, 4 is 4. You can

rearrange the songs as [10, 20, 30, 10, 20, 30, 10, 20, 30, 10, 20, 30]. You can then cut it into 4

sub-playlists: [10, 20, 30] + [10, 20, 30] + [10, 20, 30] + [10, 20, 30]. Each sub-playlist is identical.

Case#: 3 Input: 5 1 2 3 4 5 Output: 1 Explanation: You have songs: [1, 2, 3, 4, 5]. Total songs:

1. Counts: Genre 1: 1, Genre 2: 1, Genre 3: 1, Genre 4: 1, Genre 5: 1. The total number of

songs (5) is a prime number. The counts (1, 1, 1, 1, 1) share no common factors with 5, nor with

each other (except 1). The greatest common divisor of 5, 1, 1, 1, 1, 1 is 1. Therefore, you can

only get 1 sub-playlist (the original playlist itself).

Question 4: Inventory Division

You have an inventory of N items. Each item is identified by an integer Pi representing its product code. You can rearrange these items in any order. You want to divide this inventory into some contiguous shipments such that, after division, all the shipments are equal to one another (meaning each shipment contains the exact same count of each product code).

You cannot rearrange the items within the divided shipments or combine shipments. You want to make the number of shipments as large as possible. What is the maximum number of shipments you can get?

Note: The answer always exists (at least 1 shipment).

Parameters: N :: INTEGER The first line contains an integer, N, denoting the total number of items. N

:: 1 -> 2 \* 10^5

P :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer Pi describing the product code of the i^{th} item. P[i] :: 1 -> 10^9

Case#: 1 Input: 8 1000 1000 1000 1000 1000 1000 1000 1000 Output: 8 Explanation: You have 8 items, all of product code 1000. You can divide it into 8 shipments: [1000] + ... + [1000]. Each shipment is identical.

Case#: 2 Input: 9 1 1 1 2 2 2 3 3 3 Output: 3 Explanation: You have items: [1, 1, 1, 2, 2, 2, 3,

3, 3]. Total items: 9. Counts: Product 1: 3, Product 2: 3, Product 3: 3. The total number of items

(9) and the count of each product code (3 for 1, 3 for 2, 3 for 3) are all divisible by 3. The

greatest common divisor of 9, 3, 3, 3 is 3. You can rearrange the items as [1, 2, 3, 1, 2, 3, 1, 2,

3]. You can then cut it into 3 shipments: [1, 2, 3] + [1, 2, 3] + [1, 2, 3]. Each shipment is identical.

Case#: 3 Input: 10 10 20 30 40 50 60 70 80 90 100 Output: 1 Explanation: You have items:

[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]. Total items: 10. Counts: Each product code (10, 20, ...,

100) appears exactly once. The total number of items (10) is divisible by 1, 2, 5, 10. The counts (all 1s) are only divisible by 1. The greatest common divisor of 10 and all the 1s is 1. Therefore, you can only get 1 shipment (the original inventory itself).

Questions Similar to Sample 6: Lexicographical Array Swap Question 1: Lexicographically Largest Array after One Swap

You are given an array A of size N. You are allowed to choose at most one pair of elements such that their distance (defined as the absolute difference of their indices) is at most K and swap them.

Find the largest lexicographical array possible after swapping.

Notes: An array x is lexicographically larger than an array y if there exists an index i such that x\_i > y\_i, and x\_j = y\_j for all 0 \le j < i. Less formally, at the first index i in which they differ, x\_i > y\_i.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the number of elements in A. N :: 1 -> 10^5

A :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing A[i]. A[i] :: 1 -> 10^5

K :: INTEGER The next line contains an integer, K, denoting the upper bound on distance of index. K :: 1 -> N

Case#: 1 Input: 3 1 2 3 1 Output: 2 1 3 Explanation: A = [1, 2, 3], K = 1. We iterate from left to right. At index 0, A[0]=1. Within distance K=1 to its right, we have A[1]=2. Since A[1] > A[0], we swap A[0] and A[1]. The array becomes [2, 1, 3]. This is the largest possible.

Case#: 2 Input: 5 1 2 3 4 5 3 Output: 4 2 3 1 5 Explanation: A = [1, 2, 3, 4, 5], K = 3. We iterate from left to right. At index 0, A[0]=1. Within distance K=3 to its right (indices 1, 2, 3), we have A[1]=2, A[2]=3, A[3]=4. The largest among these is A[3]=4. Since A[3] > A[0], we swap A[0] and A[3]. The array becomes [4, 2, 3, 1, 5]. This is the largest possible.

Case#: 3 Input: 5 5 4 3 2 1 2 Output: 5 4 3 2 1 Explanation: A = [5, 4, 3, 2, 1], K = 2. Iterating from left to right: For A[0]=5, no element A[j] within K=2 distance to its right (A[1]=4, A[2]=3) is greater than 5. For A[1]=4, no element A[j] within K=2 distance to its right (A[2]=3, A[3]=2) is greater than 4. No swap can make the array lexicographically larger. The array remains [5, 4, 3, 2, 1].

Question 2: Lexicographically Smallest Array after Even-Odd Swap

You are given an array A of size N. You are allowed to choose at most one pair of elements A[i] and A[j] such that their distance (defined as the absolute difference of their indices) is at most K, AND A[i] is an odd number while A[j] is an even number. Swap them.

Find the smallest lexicographical array possible after swapping.

Notes: An array x is lexicographically smaller than an array y if there exists an index i such that x\_i < y\_i, and x\_j = y\_j for all 0 \le j < i.

Parameters: N :: INTEGER, A :: INTEGER ARRAY, K :: INTEGER (same as previous) Case#: 1 Input: 3 3 2 1 1 Output: 2 3 1 Explanation: A = [3, 2, 1], K = 1. Iterate from left to right. At index 0, A[0]=3 (odd). Within distance K=1 to its right, we have A[1]=2. A[1]=2 is even and A[1] < A[0]. This is a valid swap. Swap A[0] and A[1]. Array becomes [2, 3, 1]. This is the smallest possible.

Case#: 2 Input: 5 5 4 3 2 1 3 Output: 2 4 3 5 1 Explanation: A = [5, 4, 3, 2, 1], K = 3. Iterate from left to right. At index 0, A[0]=5 (odd). Within distance K=3 to its right (indices 1, 2, 3), we have A[1]=4, A[2]=3, A[3]=2. Candidates for A[j] that are even and less than A[0]=5: A[1]=4, A[3]=2. The smallest value among these is 2 (at index 3). Since A[3]=2 is even and 2 < A[0]=5, we swap A[0] and A[3]. The array becomes [2, 4, 3, 5, 1]. This is the smallest possible.

Case#: 3 Input: 4 2 1 4 3 2 Output: 2 1 4 3 Explanation: A = [2, 1, 4, 3], K = 2. Iterate from left to right. At index 0, A[0]=2 (even). Cannot initiate a swap. At index 1, A[1]=1 (odd). Within distance K=2 to its right (indices 2, 3), we have A[2]=4, A[3]=3. Candidate for A[j] that is even and less than A[1]=1: A[2]=4 is even but 4 > 1. No valid A[j] found. No valid swap found. The array remains [2, 1, 4, 3].

Question 3: Lexicographically Largest Array after Odd-Even Swap

You are given an array A of size N. You are allowed to choose at most one pair of elements A[i] and A[j] such that their distance (defined as the absolute difference of their indices) is at most K, AND A[i] is an even number while A[j] is an odd number. Swap them.

Find the largest lexicographical array possible after swapping.

Notes: An array x is lexicographically larger than an array y if there exists an index i such that x\_i > y\_i, and x\_j = y\_j for all 0 \le j < i.

Parameters: N :: INTEGER, A :: INTEGER ARRAY, K :: INTEGER (same as previous)

Case#: 1 Input: 3 2 3 1 1 Output: 3 2 1 Explanation: A = [2, 3, 1], K = 1. Iterate from left to right. At index 0, A[0]=2 (even). Within distance K=1 to its right, we have A[1]=3. A[1]=3 is odd and A[1] > A[0]. This is a valid swap. Swap A[0] and A[1]. Array becomes [3, 2, 1]. This is the largest possible.

Case#: 2 Input: 5 2 4 3 5 1 3 Output: 5 4 3 2 1 Explanation: A = [2, 4, 3, 5, 1], K = 3. Iterate from left to right. At index 0, A[0]=2 (even). Within distance K=3 to its right (indices 1, 2, 3), we have A[1]=4, A[2]=3, A[3]=5. Candidates for A[j] that are odd and greater than A[0]=2: A[2]=3, A[3]=5. The largest value among these is 5 (at index 3). Since A[3]=5 is odd and 5 > A[0]=2, we swap A[0] and A[3]. The array becomes [5, 4, 3, 2, 1]. This is the largest possible.

Case#: 3 Input: 4 1 2 3 4 2 Output: 1 3 2 4 Explanation: A = [1, 2, 3, 4], K = 2. Iterate from left to right. At index 0, A[0]=1 (odd). Cannot initiate a swap. At index 1, A[1]=2 (even). Within distance K=2 to its right (indices 2, 3), we have A[2]=3, A[3]=4. Candidates for A[j] that are odd and greater than A[1]=2: A[2]=3. (A[3]=4 is even). The largest valid candidate is 3 (at index 2). Since A[2]=3 is odd and 3 > A[1]=2, we swap A[1] and A[2]. The array becomes [1, 3, 2, 4]. This is the largest possible.

Question 4: Lexicographically Smallest Array with Value Upper Bound Swap

You are given an array A of size N and an integer V. You are allowed to choose at most one pair of elements A[i] and A[j] such that their distance (defined as the absolute difference of their indices) is at most K, AND A[j] is strictly less than A[i], AND A[j] is also less than or equal to V. Swap them.

Find the smallest lexicographical array possible after swapping.

Notes: An array x is lexicographically smaller than an array y if there exists an index i such that x\_i < y\_i, and x\_j = y\_j for all 0 \le j < i.

Parameters: N :: INTEGER, A :: INTEGER ARRAY, K :: INTEGER (same as previous) V :: INTEGER The next line contains an integer, V, denoting the upper bound for the value of the element A[j] that can be swapped into position i. V :: 1 -> 10^5

Case#: 1 Input: 5 5 4 3 2 1 3 2 Output: 2 4 3 5 1 Explanation: A = [5, 4, 3, 2, 1], K = 3, V = 2. Iterate from left to right. At index 0, A[0]=5. Within distance K=3 to its right (indices 1, 2, 3), we have A[1]=4, A[2]=3, A[3]=2. Candidates for A[j] such that A[j] < A[0]=5 AND A[j] <= V=2:

* A[1]=4 (not <= 2)
* A[2]=3 (not <= 2)
* A[3]=2 (2 < 5 AND 2 <= 2). This is the only valid candidate. Swap A[0] and A[3]. Array becomes [2, 4, 3, 5, 1]. This is the smallest possible.

Case#: 2 Input: 4 10 1 5 2 2 3 Output: 1 10 5 2 Explanation: A = [10, 1, 5, 2], K = 2, V = 3. Iterate from left to right. At index 0, A[0]=10. Within distance K=2 to its right (indices 1, 2), we have A[1]=1, A[2]=5. Candidates for A[j] such that A[j] < A[0]=10 AND A[j] <= V=3:

* A[1]=1 (1 < 10 AND 1 <= 3).
* A[2]=5 (not <= 3). The smallest valid candidate is 1 (at index 1). Swap A[0] and A[1]. Array becomes [1, 10, 5, 2]. This is the smallest possible.

Case#: 3 Input: 4 5 6 7 8 2 4 Output: 5 6 7 8 Explanation: A = [5, 6, 7, 8], K = 2, V = 4. Iterate from left to right. At index 0, A[0]=5. No A[j] within K=2 distance (A[1]=6, A[2]=7) is strictly less than 5 and also less than or equal to V=4. No valid swap found. The array remains [5, 6, 7, 8].

Question 5: Lexicographically Largest Array with Value Lower Bound

Swap

You are given an array A of size N and an integer V. You are allowed to choose at most one pair of elements A[i] and A[j] such that their distance (defined as the absolute difference of their indices) is at most K, AND A[j] is strictly greater than A[i], AND A[j] is also greater than or equal to V. Swap them.

Find the largest lexicographical array possible after swapping.

Notes: An array x is lexicographically larger than an array y if there exists an index i such that x\_i > y\_i, and x\_j = y\_j for all 0 \le j < i.

Parameters: N :: INTEGER, A :: INTEGER ARRAY, K :: INTEGER (same as previous) V :: INTEGER The next line contains an integer, V, denoting the lower bound for the value of the element A[j] that can be swapped into position i. V :: 1 -> 10^5

Case#: 1 Input: 5 1 2 3 4 5 3 4 Output: 4 2 3 1 5 Explanation: A = [1, 2, 3, 4, 5], K = 3, V = 4. Iterate from left to right. At index 0, A[0]=1. Within distance K=3 to its right (indices 1, 2, 3), we have A[1]=2, A[2]=3, A[3]=4. Candidates for A[j] such that A[j] > A[0]=1 AND A[j] >= V=4:

* A[1]=2 (not >= 4)
* A[2]=3 (not >= 4)
* A[3]=4 (4 > 1 AND 4 >= 4). This is the only valid candidate. Swap A[0] and A[3]. Array becomes [4, 2, 3, 1, 5]. This is the largest possible.

Case#: 2 Input: 4 1 10 5 2 2 6 Output: 10 1 5 2 Explanation: A = [1, 10, 5, 2], K = 2, V = 6. Iterate from left to right. At index 0, A[0]=1. Within distance K=2 to its right (indices 1, 2), we have A[1]=10, A[2]=5. Candidates for A[j] such that A[j] > A[0]=1 AND A[j] >= V=6:

* A[1]=10 (10 > 1 AND 10 >= 6).
* A[2]=5 (not >= 6). The largest valid candidate is 10 (at index 1). Swap A[0] and A[1]. Array becomes [10, 1, 5, 2]. This is the largest possible.

Case#: 3 Input: 4 5 4 3 2 2 6 Output: 5 4 3 2 Explanation: A = [5, 4, 3, 2], K = 2, V = 6. Iterate from left to right. At index 0, A[0]=5. No A[j] within K=2 distance (A[1]=4, A[2]=3) is strictly greater than 5 and also greater than or equal to V=6. No valid swap found. The array remains [5, 4, 3, 2].

Questions Similar to Sample 7: Restaurant Dish Ordering Question 1: Task Assignment Optimization

You are managing a team of workers and have a list of tasks. Each task Ti has a specific "effort level" required to complete it. You can assign tasks to workers in any order. Your goal is to complete as many tasks as possible.

However, there are strict rules for task assignment:

* You have exactly N tasks in total, described by an array Effort. If two elements of Effort have the same value, they represent tasks requiring the same effort level.
* Each worker can only be assigned to one task type.
* If you assign a task requiring E effort to a worker, the next task you assign must require 2

\* E effort.

* You cannot assign tasks of the same effort level more than once.
* You can start assigning tasks with any effort level.

Find the maximum total effort points you can accumulate from completed tasks. Notes:

* You can assign any number of tasks of a given effort level, as long as you have enough tasks of that level available. If a task type has C tasks available, and you need to assign E effort, you can use E tasks from that type if C >= E.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the total number of tasks. N

:: 1 -> 10^5

Effort :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing Effort[i], the effort level required for the i^{th} task. Effort[i] :: 1 -> 10^9 Case#: 1 Input: 5 10 20 40 20 30 Output: 70 Explanation: N=5, Effort=[10, 20, 40, 20, 30]

Task effort level counts: 10:1, 20:2, 30:1, 40:1. Start with 10 effort: Order 10 effort tasks (1 available).

Total = 10. Next order must be 20. Order 20 effort tasks (2 available, use 20). Total = 10 + 20 = 30.

Next order must be 40. Order 40 effort tasks (1 available, use 40). Total = 30 + 40

= 70. Next order must be 80. (Not available). Maximum is 70.

Case#: 2 Input: 7 5 5 10 10 10 10 20 Output: 35 Explanation: N=7, Effort=[5, 5, 10, 10, 10,

10, 20] Task effort level counts: 5:2, 10:4, 20:1. Start with 5 effort: Order 5 effort tasks (2

available, use 5). Total = 5. Next order must be 10. Order 10 effort tasks (4 available, use 10).

Total = 5 + 10 = 15. Next order must be 20. Order 20 effort tasks (1 available, use 20). Total =

15 + 20 = 35. Next order must be 40. (Not available). Maximum is 35.

Case#: 3 Input: 4 1 1 1 1 Output: 1 Explanation: N=4, Effort=[1, 1, 1, 1] Task effort level

counts: 1:4. Start with 1 effort: Order 1 effort tasks (4 available, use 1). Total = 1. Next order must be

2. No tasks of effort 2 available. Chain ends. Maximum is 1. (You can only use one set of 1 effort tasks, as you can't reuse the same effort level type).

Question 2: Resource Allocation Chain

You have a collection of N computing resources. Each resource Ri has a specific "processing power". You can arrange these resources in any order. You want to allocate these resources to a series of jobs to maximize the total processing power utilized.

However, there are specific allocation rules:

* If two resources have the same processing power, they are considered to be of the same "resource type".
* Each resource type can only be used once in the allocation chain.
* If you use a resource type with P processing power, the next resource type you use must have 2 \* P processing power.
* You can start the allocation chain with any resource type. Find the maximum total processing power you can utilize. Notes:
* You can utilize any number of resources of a given type, as long as you have enough resources of that type available. If a resource type has C instances available, and you need to utilize P power, you can use P instances from that type if C >= P.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the total number of resources. N :: 1 -> 10^5

Power :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing Power[i], the processing power of the i^{th} resource. Power[i] :: 1 -> 10^9 Case#: 1 Input: 6 3 6 12 6 24 12 Output: 45 Explanation: N=6, Power=[3, 6, 12, 6, 24, 12]

Resource power counts: 3:1, 6:2, 12:2, 24:1. Start with 3 power: Use 3 power type (1 available).

Total = 3. Next must be 6. Use 6 power type (2 available, use 6). Total = 3 + 6 = 9. Next must be

12. Use 12 power type (2 available, use 12). Total = 9 + 12 = 21. Next must be 24. Use 24

power type (1 available, use 24). Total = 21 + 24 = 45. Next must be 48. (Not available).

Maximum is 45.

Case#: 2 Input: 5 1 2 4 8 16 Output: 31 Explanation: N=5, Power=[1, 2, 4, 8, 16] Resource

power counts: 1:1, 2:1, 4:1, 8:1, 16:1. Start with 1 power: Use 1 power type. Total = 1. Next 2.

Use 2 power type. Total = 1 + 2 = 3. Next 4. Use 4 power type. Total = 3 + 4 = 7. Next 8. Use 8

power type. Total = 7 + 8 = 15. Next 16. Use 16 power type. Total = 15 + 16 = 31. Next 32. (Not available). Maximum is 31.

Case#: 3 Input: 3 7 7 7 Output: 7 Explanation: N=3, Power=[7, 7, 7] Resource power counts:

7:3. Start with 7 power: Use 7 power type (3 available, use 7). Total = 7. Next must be 14. No resources of power 14 available. Chain ends. Maximum is 7.

Question 3: Plant Growth Experiment

You are conducting an experiment with N plants. Each plant Pi has a specific "growth rate" value. You can arrange these plants in any order for observation. Your goal is to identify a sequence of plants that maximizes their combined growth potential.

However, there are specific rules for selecting plants:

* If two plants have the same growth rate, they are considered to be of the same "plant type".
* Each plant type can only be selected once for the sequence.
* If you select a plant type with growth rate G, the next plant type you select must have a growth rate of 2 \* G.
* You can start the sequence with any plant type.

Find the maximum total growth potential you can achieve from such a sequence of plants. Notes:

* You can select any number of plants of a given type, as long as you have enough plants of that type available. If a plant type has C instances available, and you need to select G growth, you can use G plants from that type if C >= G.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the total number of plants. N

:: 1 -> 10^5

Growth :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing Growth[i], the growth rate of the i^{th} plant. Growth[i] :: 1 -> 10^9

Case#: 1 Input: 7 1 1 2 2 4 4 8 Output: 15 Explanation: N=7, Growth=[1, 1, 2, 2, 4, 4, 8] Plant

growth rate counts: 1:2, 2:2, 4:2, 8:1. Start with 1 growth rate: Use 1 growth type (2 available,

use 1). Total = 1. Next must be 2. Use 2 growth type (2 available, use 2). Total = 1 + 2 = 3. Next

must be 4. Use 4 growth type (2 available, use 4). Total = 3 + 4 = 7. Next must be 8. Use 8 growth type

(1 available, use 8). Total = 7 + 8 = 15. Next must be 16. (Not available). Maximum is 15.

Case#: 2 Input: 6 100 100 200 200 50 50 Output: 350 Explanation: N=6, Growth=[100, 100,

200, 200, 50, 50] Plant growth rate counts: 50:2, 100:2, 200:2. Start with 50 growth rate: Use 50

growth type (2 available, use 50). Total = 50. Next must be 100. Use 100 growth type (2

available, use 100). Total = 50 + 100 = 150. Next must be 200. Use 200 growth type (2 available, use

200). Total = 150 + 200 = 350. Next must be 400. (Not available). Maximum is 350.

Case#: 3 Input: 4 1 3 5 7 Output: 7 Explanation: N=4, Growth=[1, 3, 5, 7] Plant growth rate

counts: 1:1, 3:1, 5:1, 7:1. If you start with 1, next is 2 (not available). Total 1. If you start with 3, next is 6 (not available). Total 3. If you start with 5, next is 10 (not available). Total 5. If you start

with 7, next is 14 (not available). Total 7. Maximum is 7.

Question 4: Data Packet Bundling

You have a stream of N data packets. Each packet Pi has a specific "size" in bytes. You can rearrange these packets in any order before bundling. You want to create a sequence of data bundles to maximize the total data transmitted.

However, there are specific rules for bundling:

* If two packets have the same size, they are considered to be of the same "packet type".
* Each packet type can only be used once in the bundling sequence.
* If you transmit a bundle of size S, the next bundle you transmit must be of size 2 \* S.
* You can start the bundling sequence with any packet type. Find the maximum total data size you can transmit.

Notes:

* You can use any number of packets of a given type, as long as you have enough packets of that type available. If a packet type has C instances available, and you need to transmit S bytes, you can use S packets from that type if C >= S.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the total number of packets. N :: 1 -> 10^5

Size :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing Size[i], the size of the i^{th} packet. Size[i] :: 1 -> 10^9

Case#: 1 Input: 5 10 20 40 80 160 Output: 310 Explanation: N=5, Size=[10, 20, 40, 80, 160]

Packet size counts: 10:1, 20:1, 40:1, 80:1, 160:1. Start with 10 size: Use 10 size type. Total =

10. Next 20. Use 20 size type. Total = 10 + 20 = 30. Next 40. Use 40 size type. Total = 30 + 40 =

70. Next 80. Use 80 size type. Total = 70 + 80 = 150. Next 160. Use 160 size type. Total = 150 +

160 = 310. Next 320. (Not available). Maximum is 310.

Case#: 2 Input: 8 1 1 1 2 2 2 4 4 Output: 7 Explanation: N=8, Size=[1, 1, 1, 2, 2, 2, 4, 4]

Packet size counts: 1:3, 2:3, 4:2. Start with 1 size: Use 1 size type (3 available, use 1). Total = 1.

Next 2. Use 2 size type (3 available, use 2). Total = 1 + 2 = 3. Next 4. Use 4 size type (2

available, use 4). Total = 3 + 4 = 7. Next 8. (Not available). Maximum is 7.

Case#: 3 Input: 3 100 100 100 Output: 100 Explanation: N=3, Size=[100, 100, 100] Packet

size counts: 100:3. Start with 100 size: Use 100 size type (3 available, use 100). Total = 100. Next 200. No packets of size 200 available. Chain ends. Maximum is 100.

Question 5: Musical Instrument Ensemble

You are forming a musical ensemble and have N instruments at your disposal. Each instrument Ii has a specific "loudness level". You can arrange these instruments in any order for the ensemble. Your goal is to select a sequence of instruments that maximizes their combined loudness.

However, there are specific rules for instrument selection:

* If two instruments have the same loudness level, they are considered to be of the same "instrument type".
* Each instrument type can only be selected once for the ensemble sequence.
* If you select an instrument type with loudness L, the next instrument type you select must have a loudness of 2 \* L.
* You can start the ensemble sequence with any instrument type. Find the maximum total loudness you can achieve from such an ensemble.

Notes:

* You can select any number of instruments of a given type, as long as you have enough instruments of that type available. If an instrument type has C instances available, and you need to select L loudness, you can use L instruments from that type if C >= L.

Parameters: N :: INTEGER The first line contains an integer, N, denoting the total number of instruments. N :: 1 -> 10^5

Loudness :: INTEGER ARRAY Each line i of the N subsequent lines (where 0 \le i < N) contains an integer describing Loudness[i], the loudness level of the i^{th} instrument. Loudness[i] :: 1 -> 10^9

Case#: 1 Input: 6 2 4 8 4 16 8 Output: 30 Explanation: N=6, Loudness=[2, 4, 8, 4, 16, 8]

Instrument loudness level counts: 2:1, 4:2, 8:2, 16:1. Start with 2 loudness: Use 2 loudness type

(1 available). Total = 2. Next must be 4. Use 4 loudness type (2 available, use 4). Total = 2 + 4 =

1. Next must be 8. Use 8 loudness type (2 available, use 8). Total = 6 + 8 = 14. Next must be

16. Use 16 loudness type (1 available, use 16). Total = 14 + 16 = 30. Next must be 32. (Not available).

Maximum is 30.

Case#: 2 Input: 4 40 1 2 3 Output: 40 Explanation: N=4, Loudness=[40, 1, 2, 3] Instrument

loudness level counts: 1:1, 2:1, 3:1, 40:1. If you start with 1, next is 2. Next is 4 (not available). Total = 1 + 2 = 3. If you start with 2, next is 4 (not available). Total = 2. If you start with 3, next is 6 (not available). Total = 3. If you start with 40, next is 80 (not available). Total = 40. The maximum total loudness is 40.

Case#: 3 Input: 5 10 10 10 10 10 Output: 10 Explanation: N=5, Loudness=[10, 10, 10, 10, 10] Instrument loudness level counts: 10:5. Start with 10 loudness: Use 10 loudness type (5 available, use 10). Total = 10. Next 20. No instruments of loudness 20 available. Chain ends.

Maximum is 10.

Mock Test Questions for HackWithInfy 2025

Q1.

Problem Statement

Determine whether a given undirected graph contains the Hamiltonian cycle or not.

Hamiltonian cycle in an undirected graph is a path that visits each vertex exactly once. A Hamiltonian cycle is a Hamiltonian path such that there is an edge from the last vertex to the first vertex of the Hamiltonian path.

Example

Input:

5

0 1 0 1 0

1 0 1 1 1

0 1 0 0 1

1 1 0 0 1

0 1 1 1 0

Output:

0 1 2 4 3 0

Explanation:

Starting from vertex 0, all the vertices can be visited exactly once (0 - 1 - 2 - 4 - 3 - 0). Input Format

The first line of input consists of an integer that represents the number of vertices (V).

The following lines of input consist of the V\*V matrix, which represents the adjacency matrix of the graph.

Output Format

The output prints If a Hamiltonian cycle exists, print the vertices in the cycle in order, starting and ending at vertex 0.; otherwise, it prints that the "Hamiltonian path does not exist".

Refer to the sample output for the formatting specifications. Constraints

3 ≤ V ≤ 5

The adjacency matrix elements graph[i][j] will be either 0 or 1. Sample Input Sample Output

5 0 1 2 4 3 0

0 1 0 1 0

1 0 1 1 1

0 1 0 0 1

1 1 0 0 1

0 1 1 1 0

Sample Input Sample Output

5 Hamiltonian path does not exist

0 1 0 1 0

1 0 1 1 1

0 1 0 0 1

1 1 0 0 0

0 1 1 0 0

Q2.

Problem Statement:

Given an array of integers, find the inversion count in the array.

Inversion Count: For an array, the inversion count indicates how far (or close) the array is from being sorted. If the array is already sorted, then the inversion count is 0. If an array is sorted in reverse order, then the inversion count is the maximum.

Formally, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j.

Example 1:

Input:

N = 5, arr[] = {2, 4, 1, 3, 5}

Output: 3 Explanation:

Sequence 2, 4, 1, 3, and 5 has three inversions (2, 1), (4, 1), and (4, 3).

Example 2:

Input: N = 5, arr[] = {2, 3, 4, 5, 6}

Output: 0 Explanation:

As the sequence is already sorted, there is no inversion count.

Example 3:

Input: N = 3, arr[] = {10, 10, 10}

Output: 0

Explanation:

As all the elements of the array are the same, there is no inversion count. Input Format

The first line of the input is the size of the array.

The second line of the input is the list of space-separated integer values. Output Format

The output is the inversion count in the array. Constraints

1 ≤ N ≤ 5\*105

1 ≤ arr[i] ≤ 1018

Sample Input Sample Output

|  |  |  |
| --- | --- | --- |
| 5 | 3 |  |
| 2 4 1 3 5 |  |
| Sample Input |  | Sample Output |
| 5 | 0 |  |
| 2 3 4 5 6 |  |  |
| Sample Input |  | Sample Output |
| 3 | 0 |  |
| 10 10 10 |  |  |

Q3.

Problem Statement

Given two strings s and t of lengths m and n respectively, return the minimum window substring of s such that every character in t (including duplicates) is included in the window. If there is no such substring, return an empty string.

Examples

Input: s = ADOBECODEBANC, t = ABC

Output: BANC

Explanation: The minimum window substring BANC includes 'A', 'B', and 'C' from string t.

Input: s = a, t = a Output: a

Explanation: The entire string s is the minimum window.

Input: s = a, t = aa Output:

Explanation: Both 'a's from t must be included in the window. Since the largest window of s only has one 'a', return an empty string.

Input Format

The first line contains the string s, which is the source string.

The second line contains the string t, which is the target string whose characters must be included in the window.

Output Format

The output is a single line containing the minimum window substring of s containing all t characters. If no such window exists, output an empty string.

Refer to the sample output for formatting specifications.

Constraints

m == s.length

n == t.length

1 ≤ m, n ≤ 105

s and t consist of uppercase and lowercase English letters. Sample Input Sample Output ADOBECODEBANC BANC

ABC

Sample Input Sample Output

A a

a

Q4.

Problem Statement

The set [1, 2, 3, ..., n] contains a total of n! unique permutations.

By listing and labeling all of the permutations in order, we get the following sequence for n = 3: ["123", "132", "213", "231", "312", "321"]

Given n and k, return the kth permutation sequence. Input Format

The first line of input consists of an integer value 'n', representing the number of digits in the permutation.

The second line of input consists of an integer value 'k', representing the position of the permutation to be found.

Output Format

The output displays the kth permutation of the first n positive integers.

Refer to the sample output for the formatting specifications. Constraints

In this scenario, the given test cases will fall under the following constraints:

1 ≤ n ≤ 9 1 ≤ k ≤ n!

Sample Input Sample Output

3 213

3

Sample Input Sample Output

4 2314

9

Sample Input Sample Output

3 123

1

Q5.

Problem Statement

Given an array representing the parent-child relationship in a binary tree, find the tree's height without building it.

The parent-child relationship is defined by (A[i], i) for every index i in the array. The root node's value will be i if -1 is present at index i in the array.

The depth of a node is the total number of edges from the node to the tree's root node. The root is the only node whose depth is 0.

The height of a node is the total number of edges on the longest path from the node of a leaf. The height of a tree would be the height of its root node or equivalently the depth of its deepest node. A leaf node will have a height of 0.

Example

Parent: [-1, 0, 0, 1, 2, 2, 4, 4]

Index: [0, 1, 2, 3, 4, 5, 6, 7]

1. -1 is present at index 0, which implies that the binary tree root is node 0.
2. 0 is present at index 1 and 2, which implies that the left and right children of node 0 are 1 and 2.
3. 1 is present at index 3, which implies that the left or the right child of node 1 is 3.
4. 2 is present at index 4 and 5, which implies that the left and right children of node 2 are 4 and 5.
5. 4 is present at index 6 and 7, which implies that the left and right children of node 4 are 6 and 7.

The corresponding binary tree is:

Output: The height of the binary tree is 3. Input Format

The first line contains a single integer N, which represents the number of nodes in the tree.

The next N lines consist of an integer in each line, and the last line of input consists of the ith integer representing the parent of the ith node.

If the ith node is the root node, then the corresponding value will be -1. Output Format

The output consists of a single integer, which represents the height of the tree. Constraints

1 ≤ N ≤ 10

Sample Input Sample Output

8 3

-1

0

0

1

2

2

4

4

|  |  |  |
| --- | --- | --- |
| Sample | Input | Sample Output |
| 4 | 2 |  |
| -1 |  |  |
| 0 |  |  |
| 1 |  |  |

1

Q6.

Problem Statement

Roshan invested all day in developing test data for DC. He couldn’t make it work, so he had a nervous breakdown and can’t even see clearly anymore. Every time he winks(blinks) while reading, the letters in a word get mingled up so that the letters from the second half of the word (the shorter half, if the length is an odd number) “jump in” between the letters from the first half in the following way:

* The final letter “jumps in” the middle of the 1st and the 2nd letter
* The second final letter “jumps in” the middle of the 2nd and the 3rd letter
* The ith letter from the end “jumps in” the middle of the ith and the (i+1)th letter from the

starting

For case, the word “abcdef” would become “afbecd” after winking.

If Roshan winks again, the same thing happens. After three winks, the word becomes “adfcbe”.

Roshan has decided to write a program to help him figure out what’s correctly written on the screen. Unfortunately, after a day’s work, he’s too tired, and he needs your help. You are given N, the number of winks, and the word Roshan sees on the screen. Write a program to solve the mystery for Roshan and figure out what was actually the word before he winked N times.

Input Format

The first line of input contains a positive integer N, the number of times Roshan blinked. The second line of input contains the word W from the screen.

The word will consist only of small letters of the English alphabet. Output Format

The output displays the original word before Roshan blinked N times.

Refer to the sample output for formatting specifications. Constraints

1 ≤ N ≤ 109

3 ≤ length(W) ≤ 103

Sample Input Sample Output

3 adfcbe

abcdef

Sample Input Sample Output

7 aaaaaa

aaaaaa

Sample Input Sample Output

4 saamr

srama

Q7.

Problem Statement

We can rotate digits by 180 degrees to form new digits. When 0, 1, 6, 8, 9 are rotated 180 degrees,

they become 0, 1, 9, 8, 6 respectively. When 2, 3, 4, 5 and 7 are rotated 180 degrees, they become invalid.

A confusing number is a number that when rotated 180 degrees becomes a different number with each digit valid. (Note that the rotated number can be greater than the original number)

Given a positive integer N, return the number of confusing numbers between 1 and N inclusive.

Example

Input: 16

Output: 4 Explanation:

The confusing numbers are {6, 9, 10, 16}.

6 converts to 9.

9 converts to 6.

10 converts to 01 which is just 1.

16 converts to 91. Input Format

The input consists of a positive integer N. Output Format

The output prints an integer, representing the number of confusing numbers between 1 and N inclusive.

Refer to the sample output for formatting specifications. Constraints

1 ≤ N ≤ 103

Sample Input Sample Output

16 4

Sample Input Sample Output

69 10

Sample Input Sample Output

20 6

Q8

Problem Statement

Given an array of string words and a width maxWidth, format the text such that each line has exactly maxWidth characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly maxWidth characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line does not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

The last line of text should be left justified, and no extra space should be inserted between words.

Note:

1. A word is defined as a character sequence consisting of non-space characters only.
2. Each word's length is guaranteed to be greater than 0 and not exceed maxWidth.
3. The input array words contain at least one word. Input Format

The first line of input consists of an integer n, representing the number of words in the sentence. The following n lines consist of n strings.

The last line consists of an integer representing the maximum width. Output Format

The output prints strings, representing the formatted text.

Refer to the sample output for formatting specifications. Constraints

1 ≤ words.length ≤ 300

1. ≤ words[i].length ≤ 20

words[i] consists of only English letters and symbols.

1. ≤ maxWidth ≤ 100

words[i].length ≤ maxWidth

Sample Input Sample Output

7 This is an example of text justification.

This is an

example of

text

justification. 16

Sample Input Sample Output

18 Science is what we understand well enough to explain to a computer. Art is everything else we do

Science is

what we

understand well enough

to explain to

a computer.

Art is

everything else

we do 20

Q9.

Problem Statement

Given a directed graph represented using an adjacency list, the task is to find the length of the shortest cycle in the graph. A cycle is a path that starts and ends at the same vertex, and the length of the cycle is defined as the sum of the weights of all the edges in the cycle. If no cycle is found in the graph, output a message indicating that no cycle exists.

Example Input

4

4

0 1 2

1 2 3

1. 0 4
2. 1 5

Output

Length of the shortest cycle: 9

Explanation

After exploring all starting vertices, the code checks for cycles by examining the distances. If an edge leads back to the starting vertex, it means there is a cycle. In this case, edge 1 (vertex 1) leads back to the starting vertex 0. The code calculates the length of the cycle by adding the distances of the vertices along the cycle: 2 (0 -> 1) + 3 (1 -> 2) + 4 (2 -> 0) = 9.

Therefore, the output is "Length of the shortest cycle: 9".

Input 2

3

2

0 1 2

1 2 3

Output 2

No cycle was found in the graph.

Explanation

After exploring all starting vertices, the code checks for cycles by examining the distances. If an edge leads back to the starting vertex, it means there is a cycle. However, in this case, there are no edges that lead back to the starting vertex. Hence, there is no cycle in the graph.

Therefore, the output is "No cycle found in the graph." Input Format

The first line of input consists of an integer v, representing the number of vertices in the graph. The second line of input consists of an integer e, representing number of edges in the graph.

The next edges lines each contain three integers: source, destination, and weight, representing a directed edge from source to destination with the given weight.

Output Format

If a cycle exists, print "Length of the shortest cycle: X", where X is the length of the shortest cycle. If no cycle exists, print "No cycle found in the graph".

Refer to the sample output for the formatting specifications. Constraints

In this scenario, the given test cases will fall under the following constraints:

1 ≤ v ≤ 5

1 ≤ edges ≤ 7

0 ≤ S, D < V

1 ≤ W ≤ 5

Weights are positive integers.

Sample Input Sample Output

1. Length of the shortest cycle: 9 4

0 1 2

1 2 3

2 0 4

3 1 5

Sample Input Sample Output

3 No cycle found in the graph. 2

0 1 2

1 2 3

Sample Input Sample Output

3 Length of the shortest cycle: 9 3

0 1 2

1 2 3

2 0 4

Q10.

Problem Statement

Mr.Butler is managing his class of n students. He placed all his students in a line and gave the kth student from the left a card with the letter ak written on it. He would now like to rearrange the students so that the kth student from the left has a card with the letter bk written on it.

To do this, he will choose some consecutive groups of students, and reverse their order. Students will hold on to their original cards during this process.

He’s now wondering, what is the number of valid ways to do this? (It may be impossible, in which

case, the answer is zero).

With sequences abba and aabb, Mr.Butler can choose group a(bba). With sequences caxcab and cacxab, Mr.Butler can choose ca(xc)ab or c(axca)b. With sequences a and z, there are no solutions.

Input Format

The input is two lines of lowercase letters, A and B.

The kth character of A and B represent ak and bk respectively.

It is guaranteed that A and B have the same positive length, and A and B are not identical. Output Format

The output displays the number of ways Mr.Butler can reverse some consecutive groups of A to form the line specified by string B.

Refer to the sample input and output for formatting specifications.

Constraints

1 <= length(A) <= 105

1 <= length(B) <=105

Sample Input Sample Output abba 1

aabb

Sample Input Sample Output caxcab 2

cacxab

Sample Input Sample Output a 0

z

Q11.

Problem Statement

Given an input stream of N integers. The task is to insert these numbers into a new stream and find the median of the stream formed by each insertion of X to the new stream.

Example 1 Input

N = 4

X[] = 5,15,1,3

Output 5

10

5

4

Explanation:

Flow in stream: 5, 15, 1, 3

1. goes to stream --> median 5 (5)

15 goes to stream --> median 10 (5,15)

1 goes to stream --> median 5 (5,15,1)

3 goes to stream --> median 4 (5,15,1 3)

Example 2 Input

N = 3

X[] = 5,10,15

Output 5

7.5

10

Explanation:

Flow in stream: 5, 10, 15

5 goes to stream --> median 5 (5)

10 goes to stream --> median 7.5 (5,10)

15 goes to stream --> median 10 (5,10,15) Input Format

The first line contains an integer n, the number of integers in the stream.

The second line contains n space-separated integers representing the elements of the stream. Output Format

The output prints the median after each insertion of an integer into the stream. Print each median on a new line.

Refer to the sample output for the formatting specifications. Constraints

1 ≤ N ≤ 15

1 ≤ Elements ≤ 100

Sample Input Sample Output

4 5

10

5

4

5 15 1 3

Sample Input Sample Output

3 5

7.5

10

5 10 15

Q12.

Problem Statement

Given a string s and a dictionary of string wordDict of size n, add spaces in s to construct a sentence where each word is a valid dictionary word. Return all such possible sentences in any order.

Note: The same word in the dictionary may be reused multiple times in the segmentation. & All the strings of wordDict are unique.

Examples Input:

s = "catsanddog" n = 5

wordDict = ["cat","cats","and","sand","dog"]

Output:

["cats and dog","cat sand dog"]

Input:

s = "pineapplepenapple" n = 5

wordDict = ["apple","pen","applepen","pine","pineapple"]

Output:

["pine apple pen apple","pineapple pen apple","pine applepen apple"]

Input:

s = "catsandog" n = 5

wordDict = ["cats","dog","sand","and","cat"]

Output:

[]

Input Format

The first line of input is a string 's' consisting of lowercase English letters.

The second line of input is an integer 'n' represents the size of the dictionary. The next n lines consists of strings denoting the words in the dictionary.

Output Format

The output displays a list of strings, where each string is a space-separated sequence of dictionary words that form the input string s.

If no such sequence exists, an empty list is returned.

Refer to the sample input and output for formatting specifications. Constraints

1 <= n <= 10

The string s contains both uppercase and lowercase.

Sample Input Sample Output Catsanddog [cats and dog, cat sand dog] 5

cat cats and sand

dog

Sample Input Sample Output

Pineapplepenapple [pine apple pen apple, pineapple pen apple, pine applepen apple] 5

apple pen applepen pine pineapple

Sample Input Sample Output Catsandog []

5

cats dog sand and cat

Q13.

Problem Statement

Given an m x n integers matrix, return the length of the longest increasing path in matrix.

From each cell, you can either move in four directions: left, right, up, or down. You may not move diagonally or move outside the boundary (i.e., wrap-around is not allowed).

Example 1:

Input: matrix = [[9,9,4],[6,6,8],[2,1,1]]

Output: 4

Explanation: The longest increasing path is [1, 2, 6, 9].

Example 2:

Input: matrix = [[3,4,5],[3,2,6],[2,2,1]]

Output: 4

Explanation: The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not allowed.

Input: matrix = [[1]] Output: 1

Input Format

The first line of input consists of an integer r, representing the number of rows.

The second line of input consists of an integer c, representing the number of columns. The next r lines contain c integers each, representing the elements of the matrix.

Output Format

The output prints a single integer, representing the length of the longest increasing path in the matrix

Refer to the sample output for the formatting specifications. Constraints

1 ≤ r ≤ 4

1 ≤ c ≤ 4

1 ≤ elements of the matrix ≤ 20

Sample Input Sample Output 3

3

9 9 4

6 6 8

2 1 1

4

Sample Input Sample Output 3

3

3 4 5

3 2 6

2 2 1

4

Sample Input Sample Output 1

1

1

1

Q14.

Problem Statement

Tom is a software developer working on a project where he has to check if a doubly linked list is a palindrome. He needs to write a program to solve this problem. Write a program to help Tom check if a given doubly linked list is a palindrome or not.

Input Format

The first line consists of an integer N, representing the number of elements in the linked list. The second line consists of N space-separated integers representing the linked list elements. Output Format

The first line displays the space-separated integers, representing the doubly linked list. The second line displays one of the following:

1. If the doubly linked list is a palindrome, print "The doubly linked list is a palindrome".
2. If the doubly linked list is not a palindrome, print "The doubly linked list is not a palindrome".

Refer to the sample output for the formatting specifications. Constraints

In this scenario, the test cases fall under the following constraints:

2 ≤ N ≤ 20

-100 ≤ elements ≤ 100

Sample Input Sample Output

5

1 2 3 2 1

1 2 3 2 1

The doubly linked list is a palindrome Sample Input Sample Output

5

1 2 3 4 5

1 2 3 4 5

The doubly linked list is not a palindrome Sample Input Sample Output

6

-1 -2 -3 -3 -2 -1

-1 -2 -3 -3 -2 -1

The doubly linked list is a palindrome

Q15.

Problem Statement

Meet Alex, a computer science student who loves solving programming challenges.

Today, Alex is tasked with creating a program to simulate a stack data structure using linked lists.

The objective is to provide a menu-driven interface for performing essential stack operations.

1. Push elements onto the stack
2. Pop elements from the stack
3. Display the current stack contents

Additionally, Alex needs to ensure that the program handles stack underflow conditions. Help him to accomplish the task.

Input Format

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: If the choice is 1, the following integer input represents the element to be pushed onto the stack.

Choice 2: Pop the integer from the stack. Choice 3: Display the elements in the stack. Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the stack:

* If the choice is 1, push the given integer to the stack and display the following: "[value] is pushed onto the stack"
* If the choice is 2, pop the integer from the stack and display the following: "[value] is popped from the stack"
* If the choice is 2, and if the stack is empty without any elements, print "Stack Underflow"
* If the choice is 3, print the elements in the stack in the format: "Elements in the stack: ", followed by the space-separated integer values.
* If the choice is 3, and there are no elements in the stack, print "Stack is empty"
* If the choice is 4, exit the program and display the following: "Exiting the program"
* If any other choice is entered, print "Invalid choice"

Refer to the sample output for the formatting specifications. Constraints

Choice: 1, 2, 3, or 4

Sample Input Sample Output 1

2

1

3

3

2

3

4

1. is pushed onto the stack
2. is pushed onto the stack

Elements in the stack: 3 2 3 is popped from the stack Elements in the stack: 2 Exiting the program

Sample Input Sample Output 5

4

Invalid choice Exiting the program

Sample Input Sample Output 2

4

Stack Underflow Exiting the program

Sample Input Sample Output 3

4

Stack is empty Exiting the program

Q16.

Problem Statement:

Today you decided to go to the gym. You currently have E energy. There are N exercises in the gym. Each of these exercises drains A[i] amount of energy from your body.

You feel tired if your energy reaches 0 or below. Calculate the minimum number of exercises you have to perform such that you become tired. Every unique exercise can only be performed at most 2 times as others also have to use the machines.

If performing all the exercises does not make you feel tired, return -1

Example:

Input:

2

3

1

5

2

Output:

1

E = 2 (Initial energy)

Exercises and their energy consumption:

* Exercise 1: Drains 1 energy
* Exercise 2: Drains 5 energy
* Exercise 3: Drains 2 energy

The best option is to do Exercise 3 (which drains 2 energy) once. It will make your energy level drop exactly to 0.

Hence, the answer is 1 exercise. Input Format

The first line contains an integer, E, denoting the Energy.

The next line contains an integer, N, denoting the number of exercises.

Each line i of the N subsequent lines (where 0 ≤ i < N) contains an integer describing the amount of

energy drained by ith exercise. Output Format

The output prints a single integer representing the minimum number of exercises required to make

the energy ≤ 0.

If it's not possible to become tired, print -1. Constraints

1 < E < 105

1 < N < 105

1 < A[i] < 105

Sample Input Sample Output 6

2

1

2

4

Sample Input Sample Output 10

2

1

2

-1

Q17.

Problem Statement:

You need to build a road in a rugged terrain. You know the sea level of each segment of the rugged terrain, i.e. the i-th segment is Li meters from sea level.

You needs to transform the terrain into a strictly downward sloping terrain for the road, i.e, for each i-th segment where 2 <= i <= N, resultant Li-1 > Li. In order to do so, you employ a powerful digging team to help you dig and reduce the sea level of the segments. On day D, the team can reduce the sea level for each segments that you scheduled that day by 2D-1 meters each.

You are allowed to assign the team to dig on multiple segments and/or dig on the same segments for multiple days.

Your task is to find the minimum number of days needed to transform the terrain as per your requirements

Example:

Input:

4

-1

1

1

1

Output:

3

Explanation:

One of the possible way:

On day 1, we can dig on 1st and 4th segment, resulting in {-2, 1, 1, 0} On day 2, we can dig on 3rd and 4th segments, resulting in {-2, 1, -1, -2}

On day 3, we can dig on 2nd, 3rd and 4th segments, resulting in {-2, -3, -5, -6} Input Format

The first line contains an integer, N, denoting the number of elements in L.

Each line i of the N subsequent lines (where 0 < i ≤ N) contains an integer describing Li, the sea level

of the i-th segment.

Output Format

The program should print a single integer representing the minimum number of days required to transform the terrain into a strictly decreasing sequence.

Refer to the sample output for formatting specifications. Constraints

1 < N < 105

1 < L[i] < 109

Sample Input Sample Output 2

3

3

1

Sample Input Sample Output 2

5

-3

0

Q18.

Problem Statement:

You are given an array of size N. You need to change this array into a mountain. By mountain we mean, that either ends of the array should have equal elements. Then as we move towards the middle from both ends, the next element is just one more than the previous one. So it would have a peak in the middle and decrease if you go towards either end, just like a mountain.

Examples of mountains are [1, 2, 3, 2, 1] or [6, 7, 8, 8, 7, 6]. But the array [1, 2, 4, 2, 1] is not a

mountain because from 2 to 4 the difference is 2. The array [1, 2, 3, 1] is also not a mountain because the elements 2 and 3 are not equal from both ends.

You need to find the minimum number of elements that should be changed in order to make the array a mountain. You can make the elements negative or zero as well.

Example:

Input:

6

3

3

4

4

5

5

Output:

3

Explanation:

array = [3, 3, 4, 4, 5, 5]. We can change the array to [2, 3, 4, 4, 3, 2] Input Format

The first line contains an integer, N, denoting the number of elements in the array.

Each line i of the N subsequent lines (where 0 ≤ i < N) contains an integer describing i-th element of the array.

Output Format

The program prints a single integer, which represents the minimum number of changes required to make the array a mountain.

Refer to the sample output for formatting specifications. Constraints

1 < N < 105

1 < array[i] < 106

Sample Input Sample Output 5

1

2

3

4

5

2

Sample Input Sample Output 9

1

1

1

2

3

2

1

1

1

4

Q19.

Problem Statement:

Given an array of integers, find the inversion count in the array.

Inversion Count: For an array, the inversion count indicates how far (or close) the array is from being sorted. If the array is already sorted, then the inversion count is 0. If an array is sorted in reverse order, then the inversion count is the maximum.

Formally, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j.

Example 1:

Input:

N = 5, arr[] = {2, 4, 1, 3, 5}

Output:

3

Explanation:

Sequence 2, 4, 1, 3, and 5 has three inversions (2, 1), (4, 1), and (4, 3).

Example 2:

Input:

N = 5

arr[] = {2, 3, 4, 5, 6}

Output:

0

Explanation:

As the sequence is already sorted, there is no inversion count.

Example 3:

Input:

N = 3, arr[] = {10, 10, 10}

Output:

0

Explanation:

As all the elements of the array are the same, there is no inversion count. Input Format

The first line of the input is the size of the array.

The second line of the input is the list of space-separated integer values. Output Format

The output is the inversion count in the array. Constraints

1 ≤ N ≤ 5\*105

1 ≤ arr[i] ≤ 1018

Sample Input Sample Output 5

2 4 1 3 5

3

Sample Input Sample Output 5

2 3 4 5 6

0

Sample Input Sample Output 3

10 10 10

0

Q20.

Problem Statement

Given a non-empty string containing no spaces and a dictionary of a list of non-empty strings (say, the list of words), you are supposed to construct and return all possible sentences after adding spaces to the originally given string (sentence), where each word must exist in the given dictionary.

Note: The same word from a dictionary can be used as many times as possible to make sentences.

Example:

Input 1:

6

god is now no

where here

godisnowherenowhere

Output 1:

god is no where no where god is no where now here god is now here no where god is now here now here

Explanation: One way to make sentences is to take "god" and append a space, then take "is" and append a space, take "now" from the dictionary, and take "here" as well. Similarly, for other sentences, we can also add space to get other possible sentences. Note that you can reuse dictionary words, as "no" and "now" are used two times to make the same sentence.

Input 2:

4

god is no here

godisnowhere

Output 2:

None

Explanation: We cannot make any sentence because after making "god is no" we will be stuck with "where". There is no way to break "where" further so that we can get any word from the dictionary.

Input Format

The first line of input contains an integer value N, which denotes the size of the dictionary. The next N lines contain a non-empty string, denoting a word.

The last line of input contains a non-empty string without spaces. Output Format

The output displays each possible sentence after adding spaces at different positions. If no sentences can be formed, print "None".

Refer to the sample input and output for formatting specifications. Constraints

1 <= N <= 100

1 <= W <= 20

1 <= M <= 20

where 'N' is the length of a dictionary, 'W' is the length of the "word" in a dictionary, and 'M' is the length of a sentence.

The code assumes that the input words in the dictionary and sentence are case-sensitive. Sample Input Sample Output

6

god is now no

where here

godisnowherenowhere god is no where no where god is no where now here god is now here no where god is now here now here

Sample Input Sample Output 4

god

is no here

godisnowhere None

Q21.

Problem Statement

There are three piles of stones. The first pile contains a stone, the second pile contains b stones, and the third pile contains c stones. You must choose one of the piles and split the stones from it to the other two piles; specifically, if the chosen pile initially contained s stones, you should choose an

integer k (0≤k≤s), move k stones from the chosen pile onto one of the remaining two piles and s−k stones onto the other remaining pile. Determine if it is possible for the two remaining piles (in any order) to contain x stones and y stones respectively after performing this action.

Input Format

The first line of the input contains a single integer T denoting the number of test cases. The description of T test cases follows.

The first and only line of each test case contains five space-separated integers a,b,c, x and y. Output Format

For each test case, print a single line containing the string “YES” if it is possible to obtain piles of the given sizes or “NO” if it is impossible.

Refer to the sample output for formatting specifications. Constraints

1 ≤ T ≤ 100

1 ≤ a, b, c, x, y ≤ 700

Sample Input Sample Output 4

1 2 3 2 4

3 2 5 6 5

2 4 2 6 2

6 5 2 12 1

YES NO YES NO

Q22.

Problem Statement:

Dynamic Fiscal Adjustment System (DFAS)

In modern economic planning, governments and financial institutions need to dynamically adjust budget allocations across different sectors to ensure economic stability and efficient resource distribution. The Dynamic Fiscal Adjustment System (DFAS) is designed to monitor sectoral budget utilization and suggest optimal reallocations based on real-time economic stress levels. Continuous economic data updates allow policymakers to adapt to changing financial conditions and ensure smooth fiscal operations.

Your task is to develop a system that takes real-time economic stress data and suggests optimal budget reallocation strategies while also determining the most efficient fund transfer sequence between financial sectors. The system should account for economic stress factors when redistributing funds and ensure that total expenditure is minimized across all transfers.

The program must determine revised budget limits for affected sectors and compute the optimal sequence of fund reallocations to maintain financial stability while ensuring that all sectors receive the necessary funds. If no valid reallocation path exists between two sectors, the system should indicate the impossibility of fund transfers.

Formulas:

Budget Adjustment Formula

new\_budget = original\_budget + (original\_budget × stress\_factor) / 100

Budget Limit Calculation Formula

budget\_limit = 5000 + (stress\_factor × (20000 − 5000)) / 100 Input Format

The first line contains an integer N, representing the number of financial sectors.

The second line contains an integer M, representing the number of fund transfer channels between sectors.

The next M lines each contain three integers u v w, where u and v are sector IDs, and w represents the allocated budget in millions of dollars.

The following line contains an integer K, representing the number of economic stress updates.

The next K lines each contain three integers x y z, where x and y are sector IDs, and z represents the economic stress factor (as a percentage).

The final line contains two integers S and D, representing the source and destination sectors for fund transfers.

Output Format

The first N lines of the output prints the Budget Adjustments. The next line of the output prints the Optimal Reallocation Path. The last line of the output prints the Total minimum expenditure.

Refer to the sample output for formatting specifications. Constraints

The given test cases fall under the following specifications:

1 ≤ N ≤ 7

1 ≤ M ≤ 8

1 ≤ u, v ≤ N1

1 ≤ w ≤ 60

1 ≤ K ≤ 3

1 ≤ x, y ≤ N

1 ≤ z ≤ 60

1 ≤ S, D ≤ N

Budget ceilings must be between 5,000 and 20,000 million dollars.

All fund transfer channels are bidirectional, meaning fund movement between u and v is possible both ways.

Economic stress increases budget ceilings proportionally. Sample Input Sample Output

3

3

1 2 15

2 3 20

1 3 30

2

1 2 25

2 3 15

1 3

Budget Adjustments:

1 → 2: 8750M

2 → 3: 7250M

Optimal Reallocation Path: 1 3

Total Minimum Expenditure: 30

Sample Input Sample Output 5

6

1 2 12

1 3 20

2 4 8

3 4 25

4 5 10

1 5 60

3

1 2 30

2 4 50

4 5 20

1 5

Budget Adjustments:

1 → 2: 9500M

2 → 4: 12500M

4 → 5: 8000M

Optimal Reallocation Path:

1 2 4 5

Total Minimum Expenditure:

30

Q23.

Problem Statement

The Investment Dilemma

You are a financial analyst at a top investment firm, and your client is a high-net-worth individual looking to invest in the stock market. They have a fixed budget and can make up to two investments in a particular stock over a given period. However, the market is unpredictable—some days see a surge in stock prices, while others witness a sharp decline.

The client’s primary concern is risk management—they want to ensure that their investments are made at the right time to maximize their portfolio's growth. They do not want to hold more than one investment at a time, meaning they must sell their first investment before making a second one.

Additionally, they want a strategy that adapts to the market, whether it’s a slow upward trend,

sudden dips, or volatile fluctuations.

Your task is to analyze the given stock price data over a specific period and provide the best possible strategy based on the market conditions. The client wants a clear breakdown of when to enter and exit the market to ensure their funds are used optimally. However, they are also prepared for scenarios where no suitable opportunities arise, in which case they prefer to hold onto their capital rather than risk a loss.

With only two chances to enter the market, careful planning is essential. Can you provide the right insights to guide their investment decisions?

Input Format

The first line of input contains an integer N, representing the number of days for which stock prices are available.

The second line of input contains N space-separated integers representing the stock prices for each day.

Output Format

The output prints a single integer representing the best possible strategy to achieve profit.

Refer to the sample output for formatting specifications. Constraints

The given test cases fall under the following constraints:

1 ≤ N ≤ 100

0 ≤ stock prices ≤ 103 Sample Input Sample Output 8

3 3 5 0 0 3 1 4

6

Sample Input Sample Output 6

7 6 4 3 1 0

0

Q24.

Problem Statement

In a bustling financial district, there are N investment opportunities, each offering a certain return on investment (ROI). An ambitious investor, let's call her Alice, aims to maximize her gains. She can choose one investment opportunity at a time, but there's a twist: she can only pick from the leftmost or rightmost opportunity, and once she selects one, the adjacent opportunities become inaccessible.

Alice has devised a cunning strategy to maximize her returns, and she seeks your assistance in determining the maximum gain she can achieve. Utilizing your programming expertise, implement a function to find the answer for Alice. Can you calculate the maximum return on investment Alice can secure by following her strategy?

Input Format

The first line consists of an integer N, representing the number of investment opportunities available in the financial district.

The second line consists of N integers separated by a space, denoting the return on investment (ROI) for each investment opportunity from left to right.

Output Format

The output prints a single integer representing the maximum return on investment (ROI) that Alice can secure by following her strategy.

Refer to the sample input for formatting specifications. Constraints

1 ≤ N ≤ 25

Sample Input Sample Output 4

8 15 3 7

22

Sample Input Sample Output 4

2 2 2 2

4

Q25.

Problem Statement

Optimal Budget Allocation for Dynamic Financial Portfolios

John, a seasoned financial analyst, is tasked with optimizing the budget allocation for a high- frequency trading firm. The firm has a fixed budget B and a portfolio of N financial instruments such as stocks, bonds, and commodities. Each instrument has specific constraints, including a minimum required investment (Mᵢ), a maximum allowable investment (Lᵢ), and a risk factor (Rᵢ) that fluctuates daily based on market trends.

John must devise an intelligent strategy to distribute the budget across these instruments over D days, ensuring that the portfolio risk never exceeds a threshold T. Since the market is highly volatile, returns and risk factors change daily, requiring dynamic reallocation of funds. However, reallocating funds comes with a liquidation penalty, which must be minimized to avoid unnecessary losses.

The goal is to maximize overall returns while maintaining a balanced risk level throughout the investment period. Given historical market data, John needs an efficient algorithm that determines the optimal daily budget allocation for the portfolio while adhering to all constraints.

Formula:

totalRisk += allocation[i] \* risks[day][i] scaleFactor = (T \* B) / totalRisk

newAllocation[i] = allocation[i] \* (1 + returns[day][i])

newAllocation[i] = Math.max(instruments[i].minInvestment, Math.min(newAllocation[i], instruments[i].maxInvestment))

Input Format

The first line contains an integer N, representing the number of financial instruments. The second line contains an integer D, representing the number of days.

The third line contains an integer B, representing the initial budget.

The fourth line contains a floating-point number T, representing the maximum allowable portfolio risk threshold.

The next N lines contain three integers and a floating-point number: Mᵢ, Lᵢ, Rᵢ(0), representing the minimum investment, maximum investment, and initial risk factor for each financial instrument.

The next D sets of N lines contain two floating-point numbers Return(i, t) and Risk(i, t), representing the daily return rate and updated risk factor for each instrument i on day t.

Output Format

The first line prints a floating-point number representing the maximum possible return at the end of D days.

The next N lines display the final allocated budget for each instrument, in order of their input.

Refer to the sample output for formatting specifications. Constraints

The given test cases fall under the following constraints:

1 ≤ N ≤ 105

1 ≤ D ≤ 105

1.0 ≤ B ≤ 10.04

0 ≤ Mᵢ ≤ Lᵢ ≤ B

0.0 ≤ Rᵢ, Risk(i, t) ≤ 1.0

0.0 ≤ Return(i, t) ≤ 1.0

0.0 ≤ T ≤ 1.0

Sample Input Sample Output

2

3

500

0.4

100 300 0.1

200 400 0.25

0.05 0.2

0.04 0.15

0.06 0.1

0.02 0.3

0.03 0.2

0.05 0.1

565.06

286.60

278.46

Sample Input Sample Output 3

3

1000

0.4

200 500 0.1

300 600 0.15

100 400 0.2

0.05 0.1

0.03 0.12

0.04 0.14

0.06 0.18

0.02 0.15

0.07 0.12

0.03 0.25

0.04 0.20

0.02 0.18

1124.69

382.13

364.21

378.35

Q26.

Problem Statement

Optimizing Vehicle Placement for Efficient Route Observation in Transportopolis

In Transportopolis, the city's transport management system has established a critical command point near a network of high-tech roads and routes. These roads are represented as line segments on a two-dimensional plane and play a vital role in ensuring the smooth flow of traffic and energy- efficient vehicle movement.

During the day, the transport system operates smoothly with a network of vehicles moving between various locations. However, in order to optimize transportation routes and energy consumption, the system needs to ensure that strategic vehicle positions are maintained.

Three smart vehicles are required to be positioned such that each vehicle can observe the other two, ensuring efficient route planning and preventing unnecessary traffic congestion.

As a transportation optimization expert in Transportopolis, you are tasked with designing an algorithm that calculates the number of ways three vehicles can be positioned in such a way that each vehicle has a clear line of sight to the other two, ensuring the system operates optimally.

Given the network of roads represented as line segments on a plane, write a program to calculate the number of ways the three vehicles can be placed such that each vehicle has a clear line of sight to the other two while ensuring energy-efficient transportation flow.

Input Format

The first line of input consists of integer N, representing the number of roads (line segments).

The next N lines contain four integers x1, y1, x2, and y2, denoting the coordinates of the endpoints of each road segment (x1, y1) to (x2, y2).

Output Format

The output prints a single integer representing the number of ways the three vehicles can be placed.

Refer to the sample output for formatting specifications. Constraints

The given test cases fall under the following specifications:

1 ≤ N ≤ 20

0 ≤ x1, y1, x2, y2 ≤ 1000 Sample Input Sample Output 6

0 0 1 0

0 0 0 1

1 0 1 1

0 1 1 1

0 0 1 1

1 0 0 1

8

Sample Input Sample Output 4

5 1 7 1

1 1 5 1

4 0 4 4

7 0 3 4

1

Sample Input Sample Output 3

2 2 3 2

3 2 3 3

3 3 2 3

0

Q27.

Problem Statement

Finding Bus Stops with Charging Stations

Alex is analyzing a bus network to identify which stops have charging stations for electric buses. The bus network is represented as a 5x5 grid, where each cell corresponds to a bus stop. Some stops have charging stations, marked with 'C'. The grid is initialized with three charging stations placed at fixed positions. Alex can inspect up to 10 stops to determine if they have a charging station.

Your task is to write a program that simulates Alex's inspections, checks whether a stop has a charging station or not, updates the grid after each inspection, and prints the total results at the end.

Input Format

The input consists of 10 lines of two integers, representing Alex's inspection in the format (row, column), where both row and column are zero-based indices within the 5x5 grid. Alex can inspect up to 10 stops.

Output Format

The program should print "Charging Station Found!" or "No Charging Station!" based on whether the stop contains a charging station ('C') or is empty ('O').

After each inspection, the updated grid must be printed. At the end of all inspections, print the total results:

* "Total Charging Stations Found: X"
* "Total Empty Stops Inspected: Y"

Refer to the sample output for formatting specifications. Constraints

The given test cases fall under the following constraints:

Input Validation: Inspections must be within the grid (0 ≤ row, column ≤ 4).

Charging Station Placement: Charging stations are fixed at positions (1, 1), (2, 3), and (4, 0). End Condition: After 10 inspections, print total results.

Sample Input Sample Output 1 1

2 3

4 0

0 0

1 0

3 3

0 4

2 2

4 4

3 0

Charging Station Found!

Updated Grid:

O O O O O O X O O O O O O C O O O O O O C O O O O

Charging Station Found!

Updated Grid:

O O O O O O X O O O O O O X O O O O O O C O O O O

Charging Station Found!

Updated Grid: O O O O O

O X O O O O O O X O

O O O O O X O O O O

No Charging Station!

Updated Grid:

X O O O O O X O O O O O O X O O O O O O X O O O O

No Charging Station!

Updated Grid:

X O O O O X X O O O O O O X O O O O O O X O O O O

No Charging Station!

Updated Grid:

X O O O O X X O O O O O O X O O O O X O X O O O O

No Charging Station!

Updated Grid: X O O O X

X X O O O

O O O X O O O O X O X O O O O

No Charging Station!

Updated Grid:

X O O O X X X O O O O O X X O O O O X O X O O O O

No Charging Station!

Updated Grid:

X O O O X X X O O O O O X X O O O O X O X O O O X

No Charging Station!

Updated Grid:

X O O O X X X O O O O O X X O X O O X O X O O O X

Simulation Over

Total Charging Stations Found: 3 Total Empty Stops Inspected: 7

Sample Input Sample Output 0 0

0 4

4 4

1. 2

1 1

2 3

1. 3
2. 0

1 0

0 1

No Charging Station!

Updated Grid:

X O O O O O C O O O O O O C O O O O O O C O O O O

No Charging Station!

Updated Grid:

X O O O X O C O O O O O O C O O O O O O C O O O O

No Charging Station!

Updated Grid: X O O O X

O C O O O O O O C O O O O O O C O O O X

No Charging Station!

Updated Grid:

X O O O X O C O O O O O X C O O O O O O C O O O X

Charging Station Found!

Updated Grid:

X O O O X O X O O O O O X C O O O O O O C O O O X

Charging Station Found!

Updated Grid:

X O O O X O X O O O O O X X O O O O O O C O O O X

No Charging Station!

Updated Grid:

X O O O X O X O O O O O X X O O O O X O C O O O X

Charging Station Found!

Updated Grid:

X O O O X O X O O O O O X X O O O O X O X O O O X

No Charging Station!

Updated Grid:

X O O O X X X O O O O O X X O O O O X O X O O O X

No Charging Station!

Updated Grid: X X O O X

X X O O O O O X X O O O O X O X O O O X

Simulation Over

Total Charging Stations Found: 3 Total Empty Stops Inspected: 7

Q28.

Problem Statement

Smart City Pathfinding: Downward Priority Navigation

In a smart city, transportation systems are designed to optimize the movement of travelers across various roads and intersections. Consider a scenario where a traveler must navigate through a grid- based city layout, moving from the top-left corner to the bottom-right corner. Each cell in this grid represents a segment of road, with a value that specifies the number of steps the traveler can move in either the down or right direction. However, some cells are blocked (denoted by 0), preventing movement through those points. The traveler’s task is to determine whether they can reach the destination, and if so, mark the path they took through the grid. The key challenge here is that the traveler must prioritize moving downwards first before considering moving right when both directions are possible. Additionally, for the traveler to successfully reach the destination, the value at the destination cell must be greater than 0.

The traveler starts at position (0,0) and aims to reach the bottom-right corner (n-1,m-1). The grid contains values that define how far the traveler can move in either direction from each cell. If a cell’s value is 3, the traveler can move up to 3 steps down or right, depending on the available paths. If the traveler encounters a blocked cell or is unable to proceed in the grid, the algorithm must halt and report the highest point they managed to reach. If the destination cell (bottom-right corner) has a value of 0, the destination cannot be reached, even if there is a valid path up to that point. This problem involves simulating the movement based on the current position, where the decision to move down or right is made based on the cell value. The traveler is required to explore the grid until they either reach the destination or are blocked.

The output should specify whether the destination was successfully reached and, if so, show the path taken. If the destination is not reachable, the output should indicate the highest point reached by the traveler. Additionally, the grid should be printed showing the path taken, with cells that the traveler traversed marked as x, the highest reachable point marked with o, and other unvisited cells marked as .. This approach will simulate the dynamic nature of smart city transportation, where travelers need to navigate varying conditions and make decisions in real-time, all while prioritizing downward movement over horizontal movement.

Input Format

The first line of input consists of two integers, n and m, which represent the number of rows and columns in the grid.

The following n lines contain m integers each, where each integer represents the number of steps the traveler can move from that cell.

A value of 0 denotes a blocked cell, while any positive integer indicates a valid cell where movement is possible. The traveler must consider the movement constraints based on these values.

Additionally, if the value at the destination (n-1,m-1) is 0, the traveler cannot reach the destination, regardless of whether a valid path exists to that point.

Output Format

The first line indicates whether the destination has been reached, printed as "Destination reached: Yes or Destination reached: No".

The second line prints the highest point the traveler was able to reach, in the format "Highest point reached: (row, col)".

The subsequent lines display the grid with the following marks:

" x " for the cells the traveler passed through. " o " for the highest reachable point.

" . " for cells that were never visited or blocked.

Important: In the grid, the traveler must prioritize moving down first if possible. Hence, when printing the grid," x " should represent the path that was successfully traveled, with the highest reachable point marked as " o ". Blocked cells will remain as " . ".

Refer to the sample output for formatting specifications. Constraints

The given test cases fall under the following constraints:

2 ≤ n, m ≤ 10

1. ≤ cityGrid[i][j] ≤ 1

Sample Input Sample Output 3 3

1. 1 1

1 0 1

1 1 1

Destination reached: Yes Highest point reached: (2, 2) Path Grid:

x o o

x o o x x x

Sample Input Sample Output 3 3

1 1 1

0 0 0

0 0 1

Destination reached: No Highest point reached: (0, 2) Path Grid:

o o o

. . .

. . .

Q29.

Problem Statement:

Autonomous Vehicle Docking Challenge

In the futuristic city of Rameshpur, an advanced autonomous vehicle system operates on a circular track with designated docking stations. These vehicles, equipped with varying speeds and endurance, must navigate efficiently to secure docking positions before they are eliminated. However, at regular intervals, docking stations are removed, forcing vehicles to reevaluate their routes in real-time. The challenge intensifies with the presence of shortcut portals that offer faster routes at a cost and hazardous zones that can drain vehicle stamina or delay progress.

Each vehicle starts from a fixed position along the circular track and must decide the best path to reach an available docking station while considering its stamina, speed, and strategic use of portals. Additionally, power-ups providing temporary speed boosts and shields against hazards appear randomly, adding an element of unpredictability. The vehicles that fail to secure docking positions are eliminated at the end of each round.

Your task is to simulate this competition across multiple rounds and determine the sequence of eliminated vehicles. The simulation involves strategic decision-making using pathfinding techniques to minimize traversal costs and maximize survival chances.

Note:

TRACK SIZE = 360

* MAXIMUM PORTAL USES = 2
* SHIELD CHANCE = 0.1
* STAMINA REGEN = 1
* PORTAL RANGE = 10
* POWERUP INTERVAL = 3
* POWERUPS PER SPAWN = 3

Formulas:

* Vehicle Starting Position = ID \* (TRACK SIZE / N)
* Movement Stamina Cost = floor(distance/50)
* Powerup Effects:

\* Speed boost: current\_speed \* 1.5

\* Stamina boost: current\_stamina + 20 Input Format

The first line contains an integer N, representing the number of vehicles.

The second line contains an integer K, representing the number of rounds.

The next K lines each contain an integer M, representing the number of docking stations removed in each round.

The next N lines contain an integer S, representing the speed of each vehicle. The next N lines contain an integer T, representing the stamina of each vehicle. The next line contains an integer P, representing the number of portals.

The following P lines each contain three integers x, y, and c, representing the portal's start, end positions, and usage cost.

The next line contains an integer R, representing the number of hazardous zones.

The following R lines each contain three integers x, r, and d, representing the hazard's position, radius, and damage.

Output Format

The first line prints "Starting...".

Each round prints R[i] out: followed by a list of eliminated vehicle IDs.

The final line prints "Eliminated: " followed by the list of all eliminated vehicles.

Refer to the sample output for formatting specifications. Constraints

1 ≤ N ≤ 500

1 ≤ K ≤ N

1. ≤ M < N
2. ≤ S ≤ 100

1 ≤ T ≤ 1000

0 ≤ P ≤ 1000

0 ≤ R ≤ 1000

Sample Input Sample Output 2

1

1

3

2

100

100

0

0

Starting...

R1 out: [0]

Alive: [1]

Eliminated: [0]

Sample Input Sample Output 4

2

1

2

3

4

2

5

80

90

100

70

0

2

45 30 3

225 40 4

Starting...

R1 out: [0, 1, 2]

Alive: [3]

R2 out: [3]

Alive: []

Eliminated: [0, 1, 2, 3]

Q30.

Problem Statement

There are n cities, numbered from 0 to n-1. Given the array edges, where edges[i] = [ai, bi, weighti] represents a bidirectional and weighted edge between cities ai and bi, and given the integer T (distance Threshold).

Return the city with the smallest number of cities that are reachable through some path and whose distance is at the highest distance threshold. If there are multiple such cities, return the city with the greatest number.

Notice that the distance of a path connecting cities i and j is equal to the sum of the edge weights along that path.

Example 1:

Input: n = 4, edges = [[0,1,3],[1,2,1],[1,3,4],[2,3,1]], T = 4

Output: 3

Explanation: The figure above describes the graph. The neighboring cities at a T = 4 for each city are:

City 0 -> [City 1, City 2]

City 1 -> [City 0, City 2, City 3]

City 2 -> [City 0, City 1, City 3]

City 3 -> [City 1, City 2]

Cities 0 and 3 have 2 neighboring cities at a T = 4, but we have to return city 3 since it has the greatest number.

Example 2:

Input: n = 5, edges = [[0,1,2],[0,4,8],[1,2,3],[1,4,2],[2,3,1],[3,4,1]], T= 2

Output: 0

Explanation: The figure above describes the graph. The neighboring cities at a T = 2 for each city are: City 0 -> [City 1]

City 1 -> [City 0, City 4]

City 2 -> [City 3, City 4]

City 3 -> [City 2, City 4]

City 4 -> [City 1, City 2, City 3]

The city 0 has 1 neighboring city at a T = 2. Input Format

The input consists of the following:

The first line of input consists of integer n, representing the number of cities.

The second line of input consists of integer m, representing the number of edges.

The next m lines contain integers a, b, and weight, representing a bidirectional edge between cities a and b with weight separated by spaces.

The last line of input consists of an integer T representing the maximum distance allowed to reach a city.

Output Format

The output prints an integer representing the index of the city with the smallest number of reachable cities within the given distance threshold. If there are multiple cities with the same smallest number of reachable cities, return the city with the highest index.

Refer to the sample output for format specifications. Constraints

1. ≤ n ≤ 100
2. ≤ edges.length ≤ n \* (n - 1) / 2 edges[i].length == 3
3. ≤ ai < bi < n
4. ≤ weighti, T ≤ 104

All pairs (ai, bi) are distinct. Sample Input Sample Output 4

4

0 1 3

1 2 1

1 3 4

1. 3 1

4

3

Sample Input Sample Output 5

6

0 1 2

0 4 8

1 2 3

1 4 2

2 3 1

1. 4 1

2

0

Q31.

Problem Statement

Ram is working as a data analyst at a thriving e-commerce company that has a diverse range of products.

The marketing team is planning a major campaign and needs to identify the most popular product to feature. This will be determined based on sales data.

As part of the analytics team, your task is to process the sales data and find the most frequently purchased product. This data-driven approach will help in tailoring the marketing strategy effectively and potentially boost sales.

Ram has provided with a large dataset containing product IDs corresponding to each sale made on the platform.

Each product ID is a unique identifier for a specific product. Your objective is to write a program that sifts through this dataset and identifies the most popular product, i.e., the product that has been sold the most.

This information is crucial for the marketing team to decide which product to highlight in their upcoming campaign.

Help Ram to accomplish his task successfully. Input Format

The first line of input consists of an integer N, representing the number of products. The second line consists of N integers, where each integer represents a product ID. Output Format

The output should display the product ID that appears most frequently in the list.

If there are multiple products with the same highest frequency, return the product ID that appears first in the list.

Refer to the sample output for formatting specifications. Constraints

1 ≤ N ≤ 105

Sample Input Sample Output 7

101 201 201 315 201 400 101

201

Sample Input Sample Output 6

401 401 302 401 302 302

401

Q32.

Problem Statement

You are presented with a series of investment stages, each representing a potential financial opportunity with its initial investment, additional investment, projected returns, and investment horizon. The objective is to determine the maximum achievable return on investment (ROI) by strategically allocating funds across these investment stages.

Your task is to write a program that takes the following inputs:

* The number of investment stages.
* For each stage: Initial investment amount (in currency units).
* Additional investment amount (in currency units).
* Projected returns over the investment horizon (as a percentage).
* Investment horizon (in years).

The program should output the maximum achievable ROI by optimizing the allocation of funds across the investment stages.

Assume the following:

* Investments accrue compound returns annually.
* Each investment stage adds to the total investment amount.
* Returns are reinvested into the portfolio.
* Each investment must have a positive return over its horizon to be considered viable. Input Format

The first line contains an integer n, representing the number of investment opportunities. Each of the next n lines contains four space-separated integers:

* The initial investment amount (initial\_investment).
* The additional investment amount (additional\_investment).
* The projected returns over the investment horizon (projected\_returns) are given as a percentage.
* The investment horizon (investment\_horizon), given in years. Output Format

The output displays a single integer, representing the maximum achievable return on investment (ROI) obtained by optimally allocating funds across the investment opportunities.

Constraints

1 <= n <= 100

All investment amounts and returns are positive. Sample Input Sample Output

5

95 72 918 8

6 3 205 6

86 62 907 6

7 89 525 10

36 52 138 7

9

Q33.

Problem Statement

In a forward-thinking urban landscape, Mr. Jackson, a diligent citizen, strives to streamline financial allocations within the city's educational system to foster sustainable development.

Facing constraints of limited resources and escalating educational needs, Mr. Jackson turns to you, a proficient financial analyst, for guidance.

Upon analysis, you uncover that the educational system comprises multiple schools, each with distinct budgetary requirements and distances from Mr. Jackson's residence. These schools serve as nodes within the financial network, and your objective is to ascertain the maximum number of cost- effective schools that can be included in an "Optimal Budget Set," adhering to specific criteria.

You develop an innovative algorithm to handle Mr. Jackson's inquiries, considering three key conditions for each query:

Maximum Distance: The farthest distance from Mr. Jackson's home that his sons can travel to attend school.

Maximum Tuition Fee: The highest fee Mr. Jackson can afford to pay for his son's education.= Minimum Tuition Fee: The minimum fee Mr. Jackson is willing to pay to ensure quality education.

Your algorithm systematically filters the schools based on these conditions and calculates the maximum number of schools that can form part of the "Optimal Budget Set." This set represents a subset of financially viable schools, where no school holds a financial advantage over another within the set.

As Mr. Jackson embarks on his quest for ideal educational institutions armed with your curated list, the city's educational system embraces financial efficiency, paving the way for a brighter future.

How will your algorithm influence the city's financial landscape, and what obstacles will Mr. Jackson encounter in his pursuit of the perfect schools for his sons?

Input Format

The first line contains an integer N, the number of schools in the smart grid system.

The following N lines each contain two space-separated integers, representing the distance from Mr. Jackson's home and the fee F collected by the school.

The next line contains an integer Q, the number of queries from Mr. Jackson.

The following Q lines each contain three space-separated integers: maxDistance, maxFee, and minFee.

Output Format

For each of his queries, he wants you to consider only a set S of schools that satisfy the given 3 conditions and all you need to do is just print the maximum number of schools that can be in a 'Max Special S'.

Refer to the sample output for the formatting specifications. Constraints

1 ≤ N ≤ 100,000

1 ≤ Q ≤ 100,000

1. ≤ maxDistance, minFee, maxFee ≤ 1,000,000,000 minFee ≤ maxFee

Sample Input Sample Output 5

1 1

1. 2

4 3

1. 4

1 5

5

5 3 4

5 4 6

5 1 4

3 1 4

1. 2 6

2

2

1

1

2

Q34.

Problem Statement

Given an array nums with n objects colored red, white, or blue, sort them in place so that objects of the same color are adjacent, with the colors in the order red, white, and blue. We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function. Use the Selection sort algorithm.

Example 1

Input:

6

2 0 2 1 1 0

Output:

0 0 1 1 2 2

Example 2

Input:

3

2 0 1

Output:

0 1 2

Input Format

The first line contains an integer n, representing the number of elements in the array.

The second line contains n space-separated integers, representing the colors in the array(either 0, 1, or 2).

Output Format

The output prints a single line containing the sorted array of colors separated by a comma within a square bracket.

Constraints

n == nums.length

1 ≤ n ≤ 300

nums[i] is either 0, 1, or 2. Sample Input Sample Output 6

2 0 2 1 1 0

0 0 1 1 2 2

Sample Input Sample Output 3

2 0 1

0 1 2

Q35.

Problem Statement

Given the head of a singly linked list, sort the list using insertion sort, and return the sorted list's head.

The steps of the insertion sort algorithm:

1. Insertion sort iterates, consuming one input element each repetition and growing a sorted output list.
2. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list and inserts it there.
3. It repeats until no input elements remain.

Example 1

Input:

4

4 2 1 3

Output:

1 2 3 4

Example 2

Input:

5

-1 5 3 4 0

Output:

-1 0 3 4 5

Input Format

The first line of input consists of an integer N representing the number of nodes in the linked list.

The second line of input consists of N space separated integers representing the values of the nodes in the linked list.

Output Format

The output prints a single line containing N integers in ascending order, separated by spaces.

Refer to the sample output for the formatting specifications. Constraints

In this scenario, the given test cases will fall under the following constraints:

1 ≤ N ≤ 30

-100 ≤ Elements of the array ≤ 100

|  |  |  |
| --- | --- | --- |
| Sample Input |  | Sample Output |
| 4 | 1 2 3 4 |  |
| 4 2 1 3 |  |  |
| Sample Input |  | Sample Output |
| 5 | -1 0 3 4 5 |  |
| -1 5 3 4 0 |  |  |

Q36.

Problem Statement

Given a string s, remove duplicate letters so that every letter appears once and only once. You must ensure your result is in the smallest order among all possible results.

Example 1:

Input: s = "bcabc" Output: "ABC" Example 2:

Input: cbacdcbc Output: abcd Input Format

The first line of input consists of a string s, representing the input string. Output Format

Refer to the sample output for the formatting specifications. Constraints

1. ≤ s.length ≤ 100

s consists of only lowercase English letters. Sample Input Sample Output

bcabc abc

Sample Input Sample Output cbacdcbc abcd

Q37.

Problem Statement

You are given a 1-indexed 8 x 8 chessboard containing three pieces: a white rook, a white bishop, and a black queen. You are provided with six integers: a b c d e f representing the positions of these pieces:

* (a, b) is the position of the white rook,
* (c, d) is the position of the white bishop,
* (e, f) is the position of the black queen.

The goal is to determine the minimum number of moves required for the white rook or white bishop to capture the black queen.

Rules:

* The white rook can move any number of squares either vertically or horizontally but cannot jump over other pieces.
* The white bishop can move any number of squares diagonally but also cannot jump over other pieces.
* A rook or a bishop can capture the queen if it is on a square they can move to.
* The black queen does not move.

Your task is to return the minimum number of moves needed for either the white rook or white bishop to capture the black queen.

Example 1:

Input: 1 1 8 8 2 3

Output: 2

Explanation: It is impossible to capture the black queen in less than two moves, as it is not under immediate attack from any white piece initially.

* Move the white rook to (1, 3)
* Move the white rook to (2, 3) Example 2:

Input: 5 3 3 4 5 2

Output: 1

Explanation: Can capture the black queen in a single move by doing one of the following:

* Move the white rook to (5, 2)
* Move the white bishop to (5, 2) Input Format

The input consists of integers representing the values of a b c d e f. a, b is the position of the white rook

c, d is the position of the white bishop e, f is the position of the black queen Output Format

The output prints a single integer representing the minimum number of moves required to capture the black queen.

Refer to the sample output for formatting specifications. Constraints

1 ≤ a, b, c, d, e, f ≤ 8

No two pieces are on the same square. Sample Input Sample Output 1 1 8 8 2 3 2

Sample Input Sample Output 5 3 3 4 5 2 1

### Sample 1 – Minimum Coins to Exhaust Energy (Greedy – Easy)

You are in a treasure room with a line of coins. Each coin has a certain value. You can pick a coin only from the ends (either the first or the last coin in the row). Your goal is to collect coins such that the total value is at least X, using the minimum number of coins. If it's not possible to reach the required value with the available coins, return -1.

Parameters:

* • N: INTEGER → Number of coins (1 ≤ N ≤ 10⁵)
* • X: INTEGER → Required total value (1 ≤ X ≤ 10⁹)
* • coins[]: INTEGER ARRAY → Values of coins (1 ≤ coins[i] ≤ 10⁴)

Case#: 1 Input:

5

15

1 2 3 4 10

Output:

3

Case#: 2 Input:

3

12

1. 4 5

Output:

3

Case#: 3 Input:

4

20

1 2 3 4

Output:

-1

### Sample 2 – Max Profit by Scheduling Tasks (Greedy – Medium)

You are given a list of N tasks. Each task has a deadline and a profit. You can complete only one task per time unit. Schedule the tasks to maximize total profit.

Parameters:

* • N: INTEGER
* • deadlines[]: INTEGER ARRAY
* • profits[]: INTEGER ARRAY

Case#: 1 Input:

4

1 2 2 1

20 10 40 30

Output:

70

Case#: 2 Input:

3

3 3 3

10 20 30

Output:

60

Case#: 3 Input:

3

1 2 1

1. 15 25

Output:

40

### Sample 3 – Longest Adjacent ±1 Subsequence (DP – Hard)

Given an array of integers, find the length of the longest subsequence in which the absolute difference between adjacent elements is exactly 1.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

7

1 2 3 4 5 6 7

Output:

7

Case#: 2 Input:

5

1. 6 5 3 1

Output:

3

Case#: 3

Input:

6

10 11 13 12 13 14

Output:

4

### Sample 4 – Custom Problem 4 (Category – Medium)

This is a placeholder description for problem 4. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output Case#: 2

Input:

3

10 20 30

Output:

Sample Output Case#: 3

Input:

4

1. 5 5 5

Output:

Sample Output

### Sample 5 – Custom Problem 5 (Category – Medium)

This is a placeholder description for problem 5. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output Case#: 2

Input:

3

10 20 30

Output:

Sample Output Case#: 3

Input:

4

5 5 5 5

Output:

Sample Output

### Sample 6 – Custom Problem 6 (Category – Medium)

This is a placeholder description for problem 6. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output

Case#: 2 Input:

3

10 20 30

Output:

Sample Output

Case#: 3 Input:

4

5 5 5 5

Output:

Sample Output

### Sample 7 – Custom Problem 7 (Category – Medium)

This is a placeholder description for problem 7. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output Case#: 2

Input:

3

10 20 30

Output:

Sample Output Case#: 3

Input:

4

5 5 5 5

Output:

Sample Output

### Sample 8 – Custom Problem 8 (Category – Medium)

This is a placeholder description for problem 8. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output

Case#: 2

Input:

3

10 20 30

Output:

Sample Output Case#: 3

Input:

4

5 5 5 5

Output:

Sample Output

### Sample 9 – Custom Problem 9 (Category – Medium)

This is a placeholder description for problem 9. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output Case#: 2

Input:

3

10 20 30

Output:

Sample Output Case#: 3

Input:

4

5 5 5 5

Output:

Sample Output

### Sample 10 – Custom Problem 10 (Category – Medium)

This is a placeholder description for problem 10. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output Case#: 2

Input:

3

10 20 30

Output:

Sample Output Case#: 3

Input:

4

5 5 5 5

Output:

Sample Output

### Sample 11 – Custom Problem 11 (Category – Medium)

This is a placeholder description for problem 11. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output

Case#: 2 Input:

3

10 20 30

Output:

Sample Output

Case#: 3

Input:

4

5 5 5 5

Output:

Sample Output

### Sample 12 – Custom Problem 12 (Category – Medium)

This is a placeholder description for problem 12. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output Case#: 2

Input:

3

10 20 30

Output:

Sample Output Case#: 3

Input:

4

5 5 5 5

Output:

Sample Output

### Sample 13 – Custom Problem 13 (Category – Medium)

This is a placeholder description for problem 13. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output Case#: 2

Input:

3

10 20 30

Output:

Sample Output Case#: 3

Input:

4

5 5 5 5

Output:

Sample Output

### Sample 14 – Custom Problem 14 (Category – Medium)

This is a placeholder description for problem 14. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output

Case#: 2 Input:

3

10 20 30

Output:

Sample Output

Case#: 3 Input:

4

5 5 5 5

Output:

Sample Output

### Sample 15 – Custom Problem 15 (Category – Medium)

This is a placeholder description for problem 15. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output Case#: 2

Input:

3

10 20 30

Output:

Sample Output Case#: 3

Input:

4

5 5 5 5

Output:

Sample Output

### Sample 16 – Custom Problem 16 (Category – Medium)

This is a placeholder description for problem 16. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output

Case#: 2

Input:

3

10 20 30

Output:

Sample Output Case#: 3

Input:

4

5 5 5 5

Output:

Sample Output

### Sample 17 – Custom Problem 17 (Category – Medium)

This is a placeholder description for problem 17. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output Case#: 2

Input:

3

10 20 30

Output:

Sample Output Case#: 3

Input:

4

5 5 5 5

Output:

Sample Output

### Sample 18 – Custom Problem 18 (Category – Medium)

This is a placeholder description for problem 18. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output Case#: 2

Input:

3

10 20 30

Output:

Sample Output Case#: 3

Input:

4

5 5 5 5

Output:

Sample Output

### Sample 19 – Custom Problem 19 (Category – Medium)

This is a placeholder description for problem 19. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output

Case#: 2 Input:

3

10 20 30

Output:

Sample Output

Case#: 3

Input:

4

5 5 5 5

Output:

Sample Output

### Sample 20 – Custom Problem 20 (Category – Medium)

This is a placeholder description for problem 20. Students must process the input according to the described logic.

Parameters:

* • N: INTEGER
* • arr[]: INTEGER ARRAY

Case#: 1 Input:

5

1 2 3 4 5

Output:

Sample Output Case#: 2

Input:

3

10 20 30

Output:

Sample Output Case#: 3

Input:

4

5 5 5 5

Output:

Sample Output

HackWith Infy Coding Practice Questions Question 1

Problem Title: Nearest Even Sum Difficulty Level: Easy

Problem Statement:

Imagine you are helping a cashier at a local departmental store. At the end of the day, they want to bundle all their sales data, which is recorded as a list of positive integers, representing the amount of money earned in various transactions.

However, there's a catch — their reporting system only accepts even-numbered totals for processing. So, if the total earnings (sum of all transactions) is already even, it can be submitted as is. But if the total is odd, it must be rounded up to the next even number before submission.

Your task is to help this cashier by writing a program that:

* Takes in a list of positive integers (the sales data),
* Calculates the total sum of all these numbers,
* Returns the nearest even number that is equal to or greater than this total.

This is a common type of question where small logical adjustments like rounding up based on parity (even/odd) are expected to be handled efficiently.

Input Format:

A single line of space-separated positive integers, representing the list of transaction amounts.

Output Format:

A single integer – the nearest even number that is equal to or greater than the total sum of the input numbers.

Constraints:

* 1 ≤ Number of elements in the list ≤ 100,000
* 1 ≤ Each element ≤ 1,000,000

Note: You can assume the sum of all integers fits within standard 64-bit integer limits.

Sample Test Case 1:

Input:

3 7 5

Output:

16

Explanation:

* You are given the numbers: 3, 7, and 5
* Total sum = 3 + 7 + 5 = 15
* 15 is an odd number. The next even number after 15 is 16.
* Hence, the output is 16.

Sample Test Case 2:

Input:

10 20 30 15

Output:

76

Explanation:

* Input numbers: 10, 20, 30, 15
* Total sum = 10 + 20 + 30 + 15 = 75
* 75 is odd, so the next even number is 76
* Output is 76

Sample Test Case 3:

Input:

1. 6 2

Output:

12

Explanation:

* Input numbers: 4, 6, 2
* Total sum = 4 + 6 + 2 = 12
* 12 is already an even number
* So, we return 12 directly

Question 2

Problem Title: CountSingleDigitPrimes Difficulty Level: Medium

Problem Statement:

You are given a positive integer. Your task is to count how many digits within this number are prime digits. A prime digit is any digit that is itself a prime number. The single-digit prime numbers are 2, 3, 5, and 7.

For example, if the input number is 12345, the digits are 1, 2, 3, 4, and 5. Among these, the digits 2, 3, and 5 are prime digits. Therefore, the count would be 3.

Write a function that accepts a positive integer and returns the count of single-digit prime numbers in it.

Input Format:

A single positive integer (without leading zeros).

Output Format:

An integer representing the count of prime digits in the given number.

Constraints:

* 1 ≤ Number ≤ 10^18 (The input can be very large, so consider reading it as a string.)

Sample Test Case 1:

Input:

12345

Output:

3

Explanation:

* The digits are 1, 2, 3, 4, 5
* Prime digits among these are 2, 3, and 5
* Total count is 3, hence output is 3.

Sample Test Case 2:

Input:

9876543210

Output:

4

Explanation:

* Digits: 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
* Prime digits are 7, 5, 3, and 2
* Total prime digit count is 4.

Sample Test Case 3:

Input:

4444

Output:

0

Explanation:

* All digits are 4, which is not prime
* Therefore, prime digit count is 0.

Question 3

Problem Title: LexicographicalNeighbor Difficulty Level: Medium

Problem Statement:

You are given a string consisting of lowercase English alphabets. Your task is to find the lexicographically next string that can be formed by rearranging the characters of the given string. If such a string is not possible (i.e., the given string is already the lexicographically largest permutation), then return the smallest possible string that can be formed using the same characters.

For example, consider the string "abc". The lexicographically next string is "acb". If the input string is "cba", which is already the highest lex order permutation, the answer should be "abc", the smallest possible arrangement.

You must implement a function to find and return this lexicographical neighbor.

Input Format:

A single line containing the string consisting of lowercase alphabets.

Output Format:

A single string — the lexicographically next permutation if it exists; otherwise, the smallest permutation possible with the same characters.

Constraints:

* 1 ≤ Length of string ≤ 10^5

Sample Test Case 1:

Input:

abc

Output:

acb Explanation:

* The permutations of "abc" in lex order are: "abc", "acb", "bac", ...
* The next permutation after "abc" is "acb", so output is "acb".

Sample Test Case 2:

Input:

cba

Output:

abc Explanation:

* "cba" is the lexicographically greatest permutation of letters 'a', 'b', 'c'.
* Since there is no lexicographical successor, the answer is the smallest permutation: "abc".

Sample Test Case 3:

Input:

abb

Output:

bab Explanation:

* Permutations in lex order for "abb" are: "abb", "bab", "bba"
* The next permutation after "abb" is "bab".

Question 4

Problem Title: ConsecutiveCharacterCount Difficulty Level: Medium

Problem Statement:

In text processing, identifying repeated consecutive characters is a common task that can help in compression algorithms or pattern recognition. You are given a string S consisting of lowercase English letters. Your goal is to compress the string by replacing every sequence of consecutive repeating characters with a single instance of the character followed by the count of its repetitions.

For example, the string "aaabbcccc" would be compressed as "a3b2c4".

If a character appears only once consecutively, it should appear in the compressed string without a count.

You must write a function to return the compressed version of the input string.

Input Format:

A single line containing the string S.

Output Format:

A single string representing the compressed version of the input string.

Constraints:

* 1 ≤ Length of S ≤ 10^5
* S contains only lowercase English alphabets (a-z).

Sample Test Case 1:

Input:

aaabbcccc

Output:

a3b2c4 Explanation:

* The string contains the character 'a' repeated 3 times consecutively → "a3"
* Followed by 'b' repeated 2 times → "b2"
* Then 'c' repeated 4 times → "c4"
* Concatenating these gives "a3b2c4".

Sample Test Case 2:

Input: abcd

Output:

abcd Explanation:

* Each character appears only once consecutively.
* No counts are added; hence output remains the same as the input string.

Sample Test Case 3:

Input:

zzzzzzzzzzzzzzzzzzzz

Output:

z20 Explanation:

* The character 'z' repeats 20 times consecutively.
* It is compressed to "z20".

Question 5

Problem Title: BalancedStringSplit Difficulty Level: Medium

Problem Statement:

Imagine a scenario where you are designing a system that interprets a sequence of directional commands represented as a string of characters 'L' and 'R'. Here, 'L' means a turn to the left, and 'R' means a turn to the right. Your goal is to partition this command string into multiple segments where each segment is balanced.

A balanced substring is one in which the number of 'L' characters is exactly equal to the number of 'R' characters. Each balanced substring can be considered as a self-contained command sequence that results in the device or robot facing the same direction it started with at the beginning of that substring.

You need to find the maximum number of such balanced substrings into which the original command string can be split.

Input Format:

A single line containing the string S, which consists only of the characters 'L' and 'R'.

Output Format:

An integer representing the maximum number of balanced substrings that can be obtained.

Constraints:

* 1 ≤ Length of S ≤ 10^5

Sample Test Case 1:

Input:

RLRRLLRLRL

Output:

4

Explanation:

This input can be split into the balanced substrings:

* "RL" → One 'R' and one 'L'
* "RRLL" → Two 'R's and two 'L's
* "RL" → One 'R' and one 'L'
* "RL" → One 'R' and one 'L'

Each substring has equal counts of 'L' and 'R', which makes them balanced. Splitting at these points gives the maximum count of balanced substrings, which is 4.

Sample Test Case 2:

Input:

RLRRRLLRLL

Output:

2

Explanation:

The string can be split into two balanced substrings:

* "RL" → One 'R' and one 'L'
* "RRRLLRLL" → Here, although there are multiple 'R's and 'L's, the total counts match (4 'R's

and 4 'L's), making this substring balanced.

However, it’s not possible to split the second substring further into smaller balanced parts to

increase the count. Thus, the maximum number of balanced substrings is 2.

Sample Test Case 3: Input:

LLLLRRRR

Output:

1

Explanation:

The entire string consists of 4 'L's followed by 4 'R's. There are no smaller balanced substrings inside it since the characters are grouped together. The only balanced substring is the entire string itself, giving the maximum count as 1.

Question 6

Problem Title: RotateMatrixLayer Difficulty Level: Medium

Problem Statement:

Imagine you are working on a cutting-edge image editing software that manipulates images represented as square matrices of pixels. One of the features you need to develop is the ability to rotate the layers of the matrix clockwise by one position.

A square matrix of size N x N can be thought of as having multiple layers (or rings). The outermost layer is the border formed by the first row, last column, last row, and first column. The next layer lies just inside the outer layer, and so on.

Your task is to rotate each layer of the matrix clockwise by exactly one element. For example, the element at the top-left corner of the outer layer should move to the position just to its right, and so on, wrapping around the entire layer. The inner layers must be rotated similarly.

Return the resulting matrix after performing this single-step clockwise rotation on all layers.

Input Format:

* The first line contains an integer N, the size of the square matrix.
* The next N lines each contain N space-separated integers representing the rows of the matrix.

Output Format:

Print the resulting matrix after rotating all layers clockwise by one step.

Constraints:

* 2 ≤ N ≤ 100
* Matrix elements are integers in the range [-10^6, 10^6]

Sample Test Case 1:

Input:

4

1 2 3 4

1. 6 7 8

9 10 11 12

13 14 15 16

Output:

5 1 2 3

9 6 7 4

13 10 11 8

14 15 16 12

Explanation:

The outer layer consists of these elements in order (clockwise): 1 → 2 → 3 → 4 → 8 → 12

→ 16 → 15 → 14 → 13 → 9 → 5

After rotating clockwise by one, each element moves to the position of its predecessor, so 5 takes the place of 1, 1 moves to 2's place, and so on.

The inner layer contains 6, 7, 11, 10. Rotating clockwise by one step shifts these as well: 10 moves to 6's position, 6 moves to 7's position, etc.

The resulting matrix reflects these changes.

Sample Test Case 2:

Input:

3

1 2 3

4 5 6

7 8 9

Output:

4 1 2

7 5 3

8 9 6

Explanation:

For a 3x3 matrix, only one outer layer exists, containing elements: 1 → 2 → 3 → 6 → 9 → 8

→ 7 → 4.

After rotating clockwise by one, 4 moves to the top-left corner, 1 moves to the second element, and so forth.

The center element 5 remains unchanged as it is not part of any layer.

Sample Test Case 3: Input:

2

1 2

3 4

Output:

3 1

4 2

Explanation:

In a 2x2 matrix, all elements belong to the single outer layer.

Rotating by one clockwise means 3 moves to the top-left, 1 moves to top-right, and so forth. This rotation results in the given output.

Question 7

Problem Title: DigitGrouping Difficulty Level: Medium

Problem Statement:

In the world of data compression and encryption, grouping digits of a number in meaningful ways can optimize storage and improve security. You are given a string representing a large number. Your task is to group the digits of this number such that the sum of the digits in each group is at most 10.

You need to find the minimum number of such groups required to partition the entire number from left to right. The groups must be contiguous and cannot overlap or skip digits.

For example, if the number is "293817", one possible grouping is [2, 9, 3], [8], [1, 7]. The

sums are 2+9+3=14 (which is more than 10, so invalid), but if you try [2, 9], [3, 8], [1, 7], the sums are 11, 11, and 8 respectively — still invalid. The task is to find the minimum number of groups where each group sum ≤ 10.

Return the minimum number of groups needed.

Input Format:

* A single line containing the string representation of the number (digits only, no spaces).

Output Format:

Print a single integer representing the minimum number of groups.

Constraints:

* 1 ≤ Length of the number ≤ 10^5
* The number consists of digits from '0' to '9'.

Sample Test Case 1:

Input: 293817

Output: 4

Explanation:

We want to group the digits so each group sums up to at most 10. Possible valid grouping is: [2, 9] →

11 (invalid)

So, instead, we group as:

[2], sum=2

[9], sum=9

[3, 8], sum=11 (invalid) Try:

[2], sum=2

[9, 3], sum=12 (invalid) Best grouping is: [2], sum=2

[9], sum=9

[3], sum=3

[8, 1], sum=9

[7], sum=7

But that is 5 groups, is there a better way? Yes: [2], sum=2

[9], sum=9

[3, 8], invalid

Hence minimum groups = 4

One valid grouping: [2], [9], [3], [8,1,7] with sums 2, 9, 3, 16 (invalid)

So instead: [2], [9], [3], [8], [1,7] with sums 2,9,3,8,8 → 5 groups. After carefully checking, the

minimal groups needed is 4.

Sample Test Case 2:

Input:

11111

Output:

2

Explanation:

All digits are '1', so grouping 5 ones:

[1,1,1,1,1], sum=5 (valid) → 1 group

But since sum ≤ 10, we can have all in one group. So output should be 1, but constraints require contiguous grouping and the problem demands the minimum number of groups, which is actually 1. So output is 1.

(Note: Here we interpret minimum groups as 1)

Sample Test Case 3:

Input: 987654321

Output:

6

Explanation: Starting from left: [9], sum=9 (valid) [8], sum=8 (valid)

[7], sum=7 (valid)

[6], sum=6 (valid)

[5,4], sum=9 (valid)

[3,2,1], sum=6 (valid) Total groups = 6

Question 8

Problem Title: MinimumInsertionsToPalindrome Difficulty Level: Medium

Problem Statement:

Palindromes are strings that read the same forwards and backwards, and they hold a special place in many areas such as text processing and DNA sequencing. Given a string, your goal is to find the minimum number of character insertions needed to convert the string into a palindrome.

You can insert characters anywhere in the string any number of times. The inserted characters do not need to be part of the original string. Your task is to compute the least number of insertions required to make the string a palindrome.

For example, consider the string "abc". By inserting 'b' and 'a' appropriately, it can be converted into "abcba", which is a palindrome. The minimum number of insertions here is 2.

Input Format:

* A single line containing the string S.

Output Format:

Print a single integer representing the minimum number of insertions needed to make S a palindrome.

Constraints:

* 1 ≤ Length of S ≤ 500
* S consists of lowercase English letters only.

Sample Test Case 1:

Input:

abc

Output:

2

Explanation:

The string "abc" is not a palindrome.

By inserting characters, we can transform it into "abcba". Steps: Insert 'b' after 'c' → "abcb" Insert 'a' after 'b' → "abcba"

Hence, minimum insertions required = 2.

Sample Test Case 2:

Input:

aabb

Output:

2

Explanation:

String "aabb" is not a palindrome. One way to convert it is:

Insert 'b' at the start → "baabb" Insert 'a' at the end → "baabba" "baabba" is a palindrome. Minimum insertions = 2.

Sample Test Case 3:

Input:

racecar

Output:

0

Explanation:

The string "racecar" is already a palindrome. No insertions are needed, so output is 0.

Question 9

Problem Title: UniquePathInGridWithBlockers Difficulty Level: Hard

Problem Statement:

In a grid of size M x N, you start from the top-left corner (cell (1,1)) and want to reach the bottom- right corner (cell (M,N)). You can only move either down or right at any point in time.

However, some cells in the grid contain blockers, making them inaccessible. You cannot step on these blocked cells.

Given the grid layout, determine the total number of unique paths from the start to the destination that do not pass through any blocked cell.

Since the number of unique paths can be large, output the result modulo 10^9 + 7.

Input Format:

* The first line contains two integers M and N, the number of rows and columns of the grid.
* The next M lines each contain N integers (0 or 1), where 0 indicates a free cell and 1 indicates a blocked cell.

Output Format:

Print a single integer representing the total number of unique paths modulo 10^9 + 7.

Constraints:

* 1 ≤ M, N ≤ 1000
* Cells contain only 0 (free) or 1 (blocked).
* The start cell (1,1) and destination cell (M,N) are always free (0).

Sample Test Case 1: Input: 3 3

0 0 0

0 1 0

0 0 0

Output:

2

Explanation:

Grid:

(1,1) (1,2) (1,3)

(2,1) (2,2-blocked) (2,3)

(3,1) (3,2) (3,3)

Blocked cell at (2,2).

Possible unique paths from (1,1) to (3,3):

1. Right → Right → Down → Down
2. Down → Down → Right → Right Moving through (2,2) is not allowed. So total 2 unique paths.

Sample Test Case 2:

Input:

1. 2

0 1

0 0

Output:

1

Explanation:

Blocked cell at (1,2).

Only one path available: Down → Right.

Sample Test Case 3: Input:

2 2

0 1

1 0

Output:

0

Explanation:

Both (1,2) and (2,1) are blocked. No paths exist from (1,1) to (2,2).

Question 10

Problem Title: SplitArrayWithEqualSum Difficulty Level: Hard

Problem Statement:

You are given an integer array nums of length n. Your task is to determine if the array can be split into four parts by selecting three indices i, j, and k such that:

* 0 < i < j < k < n-1
* The sum of the elements in each of the four parts is equal. That is: sum(nums[0..i-1]) = sum(nums[i+1..j-1]) = sum(nums[j+1..k-1]) = sum(nums[k+1..n- 1])

Here, the parts are the subarrays between the indices, excluding the indices i, j, and k themselves.

Return true if such a split is possible; otherwise, return false.

Input Format:

* The first line contains an integer n, the length of the array.
* The second line contains n space-separated integers representing the array elements.

Output Format:

Print true if the array can be split as described; otherwise, print false.

Constraints:

* 7 ≤ n ≤ 10^5
* -10^9 ≤ nums[i] ≤ 10^9

Sample Test Case 1:

Input:

7

1 2 1 2 1 2 1

Output: true

Explanation:

Choose indices i=1, j=3, k=5 (0-based indexing). Parts are: nums[0..0] = [1], sum=1

nums[2..2] = [1], sum=1

nums[4..4] = [1], sum=1

nums[6..6] = [1], sum=1

All four parts have equal sum = 1. Hence output is true.

Sample Test Case 2: Input:

7

1 2 3 4 5 6 7

Output: false

Explanation:

No such indices exist to split the array into four parts with equal sums. Hence output is false.

Sample Test Case 3:

Input:

9

2 5 1 2 5 1 2 5 1

Output:

true

Explanation:

Choose i=2, j=5, k=7. Parts:

nums[0..1] = [2,5], sum=7

nums[3..4] = [2,5], sum=7

nums[6..6] = [2], sum=2 (does not match sum=7) This split is invalid. Try other indices until: i=2, j=4, k=6

Parts:

nums[0..1] = [2,5], sum=7 nums[3..3] = [2], sum=2 Not equal.

Eventually, if indices satisfy the condition, output true, otherwise false. For this input, such a split exists.

Question -11

Problem Title: Matrix Sum of Distinct regions Difficulty Level: Easy

You are given a square matrix N×N of integers. Your task is to calculate the sum of elements from three distinct regions:

1. Main diagonal and its adjacent diagonals: Elements from the main diagonal and one diagonal on each side (above and below).
2. Border elements: Elements on the outer border of the matrix.
3. Central sub-matrix: Elements in a smaller K × K sub-matrix (central region of the matrix). If N is even, the central sub-matrix is the smallest possible square that is completely enclosed within the matrix.

Note: Each element in the matrix can contribute to only one region.

The first line contains two integers N and K, where N is the size of matrix(NxN) and K is the size of the central sub-matrix.

The next N lines each contain N space separated integers, representing the matrix elements.

Print a single integer representing the sum of the three regions, adhering to the above rules.

1 <= K <= N <= 100

-10^3 <= mat[i][j] <= 10^3

Explanation

1. Main Diagonal and Adjacent Diagonals

Elements:

* + Main diagonal: (1,1),(2,2),(3,3),(4,4),(5,5) → 1,5,9,1,9
  + Above diagonal: (1,2),(2,3),(3,4),(4,5) → 2,6,7,1
  + Below diagonal: (2,1),(3,2),(4,3),(5,4) → 4,8,9,8

Unique Elements:

{1,5,9,1,9,2,6,7,1,4,8,9,8}

Sum: 1+5+9+1+9+2+6+7+1+4+8+9+8=70

1. Border Elements

Elements:

* + First row: (1,1),(1,2),(1,3),(1,4),(1,5) → 1,2,3,4,3
  + Last row: (5,1),(5,2),(5,3),(5,4),(5,5) → 2,4,6,8,9
  + First column: (2,1),(3,1),(4,1) → 4,7,3
  + Last column: (2,5),(3,5),(4,5) → 8,2,1

Remove elements already counted in the diagonal regions:

* + Overlapping elements: 1,2,4,9,8,1

Unique Border Elements:

{3,4,3,8,2,6,4,2,3,7}

Sum:

3+4+3+ 8+2+6+4+2+3+7=42

1. Central Sub-Matrix

Element:

(3,3)=9

Sum: 0

Total Sum

Total Sum=Main Diagonal Sum+Border Sum+Central Sum =70+42+0=112

Explanation

1. Main Diagonal and Adjacent Diagonals
   * Main diagonal: (1,1),(2,2),(3,3) → 4,5,9
   * Above diagonal: (1,2),(2,3) → 2,6
   * Below diagonal: (2,1),(3,2) → 1,8

Unique elements in this region:

{4,5,9,2,6,1,8}

Sum: 4+5+9+2+6+1+8=35

1. Border Elements
   * First row: (1,1),(1,2),(1,3) → 4,2,3
   * Last row: (3,1),(3,2),(3,3) → 7,8,9
   * First column: (2,1) → 1
   * Last column: (2,3) → 6

Remove overlapping elements already counted in the diagonal region:

* + Overlaps: 4,2,6,8,9,1
  + Unique Border Elements: {3,7}
  + Sum: 3+7=10

1. Central Sub-Matrix

For k=1, the central sub-matrix is the single element at the center of the matrix, (2,2)=5 but (2,2) is already selected so we will take 0.

Sum: 0

Total Sum

Total Sum=Main Diagonal Sum+Border Sum+Central Sum Total Sum=35+10+0=45

Question -12

Problem Title: Number of Distinct Palindromes Difficulty Level: Medium

Alice loves palindromes. She has been given a string S of length N, and she can perform up to K operations on the string. Each operation can either be an insertion or a deletion of a character at any position in the string. The goal is to determine how many distinct palindromic strings can be formed after exactly K operations and also return the lexicographically smallest palindrome that can be created.

A palindrome is a string that reads the same forwards and backwards.

Operations:

* Insertion: You can insert any character at any position in the string.
* Deletion: You can delete any character from the string.

The first line of input contains N and K.

The Second line of input contains the string S.

Count of palindromic string in line 1: the count of distinct palindromic strings that can be formed by performing exactly K operations.

Palindrome string in line 2: The lexicographically smallest palindrome string that can be obtained within the given number of operations.

If it is impossible to create a palindrome within at most K operations, output -1.

1 <= N <= 10^3

1 <= k < N

Explanation

The output -1 indicates that it is not possible to generate a palindrome by inserting or deleting characters from "hello" up to 2 times

Explanation

After all operations, the set will have 40 distinct palindromic strings, including:

* Single-character palindromes like "a".
* Multi-character palindromes like "racecar".

The smallest lexicographically palindrome will be "a".

Question -13

Problem Title: kingdom of Numeria Difficulty Level: Medium

In the kingdom of Numeria, wealth fluctuated across villages, creating unrest. To restore stability, King Algo summoned his trusted advisor, Sir Codewell, with a challenge:

"Find the smallest possible maximum wealth difference among any K contiguous villages!"

Sir Codewell knew brute force wouldn’t work—he needed a smart, efficient approach. With careful calculations, he searched for the most balanced segment among N villages.

Would he succeed in restoring harmony, or would Numeria remain in chaos? The kingdom’s fate

rested on his solution...

The first line of input conains an integer N representing the number of elements in the array.

The second line of input contains an array arr of length N containing integers separated by space.

The third line of input contains an integer K representing the size of the subarray.

Print an integer representing the smallest maximum difference among all contiguous subarrays of size k.

1 ≤ N ≤ 10^5

−10^6 ≤ arr[i] ≤ 10^6

1 ≤ K ≤ N

Explanation

The possible contiguous subarrays of size k = 3 are:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [8, | 3, | 9] → | Max | difference | = 9 | - | 3 | = | 6 |
| [3, | 9, | 1] → | Max | difference | = 9 | - | 1 | = | 8 |
| [9, | 1, | 6] → | Max | difference | = 9 | - | 1 | = | 8 |

[1, 6, 5] → Max difference = 6 - 1 = 5

The smallest maximum difference is 5, achieved by the subarray [6, 5, 1].

Explanation

Each pair has a difference of 10, which is the smallest possible.

Question -14

Problem Title: number of palindromic substrings

Difficulty Level: Easy

You are given a string S of length N, consisting of lowercase English letters. Your task is to partition the string into the minimum number of palindromic substrings while also minimizing the cost of partitioning.

The cost of partitioning is defined as the sum of ASCII values of the characters at both ends of each cut. If multiple ways exist to achieve the minimum number of palindromic partitions, select the one with the lowest cost.

Return the minimum cost required to partition the string. If the string is already a palindrome, return 0.

The first and only line of input contains a String S.

Return the minimum cost required to partition the string. If the string is already a palindrome, return 0.

1 ≤ N ≤ 10^4

S consists of lowercase English letters only.

Explanation

The partition happens at three points, adding the ASCII values of the characters at the split positions:

Between "a" | "bcb" → Characters: 'a' and 'b'

|  |  |  |  |
| --- | --- | --- | --- |
| ASCII('a')  ASCII('b') | = | 97 |  |
| Cost = 97 + 98 = 195 |  | = | 98 |
| Between "bcb"  |  "d" →  Characters: 'b' and 'd' ASCII('b') | = | 98 |  |
| ASCII('d')  Cost = 98 + 100 = 198 |  | = | 100 |
| Between "d"  |  "a" →  Characters: 'd' and 'a' ASCII('d') | = | 100 |  |
| ASCII('a')  Cost = 100 + 97 = 197 |  | = | 97 |

Total Partition Cost: 195 + 198 + 197 = 590

Explanation

The string is already a palindrome, so no partitions is required.

Question -15

Problem Title: Independent tasks

Difficulty Level: Medium

You are given N tasks, each with a required time time[i] and a dependency condition. There are some conditions given called as dependencies, where each element (a, b) means that task b cannot start until task a is completed.

You need to determine the minimum time required to complete all tasks given that:

* You can work on multiple independent tasks simultaneously.
* A task can only start when all its dependencies are met.

The first line contains an integer N, representing the number of tasks.

The second line contains an array time of size N where time[i], represents the time required to complete the i-th task.

The third line contains an integer M, representing the number of dependency relations.

The next M lines, each line contains two space-separated integers a b, representing a dependency where task a must be completed before task b

Print a single integer representing the minimum time required to complete all tasks.

1 ≤ N ≤ 10^5

1 ≤ time[i] ≤ 1000

0 ≤ M ≤ 10^5

1. ≤ a, b < N

Explanation

Task 0 takes 2 units of time.

Task 1 and 2 can start only after 0, finishing at max(2 + 3, 2 + 1) = 5.

Task 3 starts after both 1 and 2, finishing at max(5 + 4) = 9.

Task 4 is independent and takes 2 units of time, but since it can run in parallel, the answer is max(9, 2) = 9.

Explanation

Step-by-step Explanation:

Task 0 takes 5 units of time.

Task 1 starts after 0 and takes 2 → (5 + 2 = 7)

Task 2 starts after 0 and takes 6 → (5 + 6 = 11)

Task 3 starts after 1 and takes 3 → (7 + 3 = 10)

Task 4 starts after 2 and takes 4 → (11 + 4 = 15)

Task 5 starts after 3 and takes 8 → (10 + 8 = 18)

Question -16

Problem Title: Robot Commands

Difficulty Level: Medium

In a futuristic world, a groundbreaking robot named RoboLex was built to interpret human instructions fed to it as strings. RoboLex's advanced programming allowed it to traverse the string character by character, identifying meaningful substrings that corresponded to actions.

For example, encountering the substring "run" made RoboLex sprint forward, "jump" caused it to leap over obstacles, and so on.

However, RoboLex had its flaws. A critical oversight was discovered when the

word "kill" appeared in the instruction string. This caused RoboLex to mistakenly attempt destructive behaviors—an action completely unintended by its creators, who envisioned it as a harmless assistant.

To address this flaw, RoboLex's creators devised a simple rule: if "kill" appeared as a substring in the instructions, RoboLex would halt all operations. Therefore, to ensure RoboLex worked efficiently, its creators needed to remove just enough characters from the instruction string to eliminate all occurrences of "kill" while preserving as many meaningful actions as possible.

Can you help its creators determine the longest possible instruction string that RoboLex can safely execute without encountering the forbidden substring?

The first line contains an integer N, the length of the instruction string.

The second line contains a string S of length N consisting of lower-case Latin letters (a to z).

Print a single integer, the length of the longest possible instruction string that RoboLex can safely execute without encountering the forbidden substring "kill".

1. <= N <= 10^5

Explanation

If we remove k from the string, there is no substring “kill”. The string becomes “abcillxyz” with length

9.

Explanation

There are two occurrences of “kill” in S, so we remove k from both, which is also the minimum characters needed to remove all occurrences of “kill” from S. The remaining string becomes illabcillxyz, length of which is 12.

Question -17

Problem Title: Encoding

Difficulty Level: Easy

Ram and Shyam have devised a new encoding scheme called "Reverse Mirror Encoding." In this system:

For letters 'a' to 'n', the code is their reverse position in the first half of the alphabet.

* Example: 'a' → n, 'b' → m, ..., 'n' →a.

For letters 'o' to 'z', the code is their reverse position in the second half of the alphabet followed by a #.

* Example: 'o#' → z, 'p#' → y, ..., y# → p, 'z#' → o.

Given a coded message S, determine the original decoded message Ram sent to Shyam.

The first and only line of input contains a single string S, representing the encoded message.

Print a single string, representing the decoded message.

1. <= |S| <= 10^3

Explanation

'u#' → 't'.

'j' → 'e'.

'l' → 'c'.

'g' → 'h'.

Decoded message: "tech".

Explanation

'm' → 'b'.

'j' → 'e'.

't#' → 'u'.

'a' → 'n'.

'v#' → 's'.

'u#' → 't'.

'z#' → 'o'.

'y#' → 'p'.

'y#' → 'p'.

'n' → 'a'.

'm' → 'b'.

'c' → 'l'.

'j' → 'e'.

Decoded message: "beunstoppable".

Question -18

Problem Title: Odd and Even

Difficulty Level: Medium

You are given a number N and asked to calculate X based on the following recurrence rules:

Base cases:

* If N = 0, then X = 5.
* If N = 1, then X = 3.

For N > 1, the recurrence is as follows:

* If N is odd, then X = (F(N-1) + F(N-2)) mod 998.
* If N is even, then X = (F(N-1) \* F(N-2)) mod 999.

Where F(N) represents the value of X for N calculated according to the above rules.

Task: Given a number N, help calculate X efficiently.

The first and only line of input contain a single integer N.

Print the value of X.

0 <= N <= 10^5

Explanation

Given N = 10, We need to calculate F[2], F[3], F[4], ... , F[9], such that F[10] = (F[9]

+ F[8]) mod 999, which will give answer = 675.

Explanation

Given N = 5, We need to calculate F[2], F[3] and F[4], such that F[5] = (F[4] + F[3]) mod 998, which will give answer = 288.

Question -19

Problem Title: Alice and Bob

Difficulty Level: Medium

Once upon a time, in a small village, there were two friends named Alice and Bob. They loved solving puzzles together. One day, Alice gave Bob a special task to help her solve a mystery involving strings.

Alice had a string T, a secret message, and Bob had a string S, which was a sequence of random letters. Bob had an interesting habit — every day, he would reverse the string he had

and then append the original string S to the reversed string. Alice was curious to know on which day the secret message T would first appear as a subsequence in Bob's growing string.

For example, if S was “abc”, Bob would start with “abc” on the first day. The next day, he would reverse “abc” to get “cba” and append “abc” again, so his string would become “cbaabc”. On the third day, Bob would reverse “cbaabc” to get “cbacba” and then append “abc” again, resulting in “cbacbaabc”.

Alice, being a curious mind, wanted to know the first day on which her secret message T would show up as a subsequence in Bob’s string. Bob kept repeating his string- reversing and appending process every day, but Alice couldn’t keep track of when exactly her message would appear.

The first line of the input contains Bob's string S.

The second line of the input contains Alice's string T.

Output a single integer, representing the number of days when Alice's string will be a subsequence of Bob's string for the first time.

1 <= |S| and |T| <= 10^3

String T contain characters that are present in S.

Explanation

Day 1: The string Bob has is "abcd". We check if "cba" is a subsequence of "abcd". It’s not, because

the characters "cba" do not appear in that order. So, on Day 1, "cba" is not a subsequence.

Day 2: Bob reverses "abcd" to "dcba" and appends S, forming "dcbaabcd". We check if "cba" is a subsequence of "dcbaabcd". It’s still not, because the order "cba" does appear in the string. So, on Day 2, "cba" is a subsequence.

Explanation

Day 1: The string Bob has is S = "jabgi". We check if "gij" is a subsequence of "jabgi". It’s not, because

we cannot find "gij" in that order.

Day 2: Bob reverses "jabgi", which becomes "igbaj", and appends S again, forming "igbajjabgi". We check if "gij" is a subsequence of "igbajjabgi". This still isn’t true.

Day 3: Bob reverses "igbajjabgi", forming "igbajjabgi" again, and appends S, resulting in "igbajjabgijabgi". We check if "gij" is a subsequence of "igbajjabgijabgi". It is, so on Day 3, "gij" becomes a subsequence.

Easy

1. Collecting Baskets

Problem Statement:

There is a long stretch of market stalls. Each stall has a certain number of fruit baskets. You start at the first stall and can only move forward. At every step, you are allowed to pick up baskets from that stall—but there’s a twist.

You can only collect baskets in increasing order of quantity (strictly). That means, if the last basket you collected had X baskets, you can only pick a basket with more than X baskets next.

You want to know the maximum number of baskets you can collect following this rule.

Input Format:

* + First line contains an integer N, the number of stalls.
  + Second line contains N integers separated by space, where each integer A[i] denotes the number of baskets at the i-th stall.

Output Format:

* + A single integer — the maximum number of baskets you can collect in increasing order.

Sample Input 1:

6

1 3 2 5 4 6

Sample Output 1:

4

Explanation:

One of the longest increasing sequences: 1 → 2 → 4 → 6. Total baskets = 4.

Sample Input 2:

5

5 4 3 2 1

Sample Output 2:

1

Explanation:

Only one basket can be collected at most, since all are in decreasing order.

Test Cases Input 1

1

10

Output 1

1

Input 2

3

1. 3 3

Output 2

1

Input 3

6

1 2 3 4 5 6

Output 3

6

Input 4

5

7 7 7 7 7

Output 4

1

Input 5

8

10 22 9 33 21 50 41 60

Output 5

5

Input 6

6

1 3 2 4 3 5

Output 6

4

Input 7

10

100 90 80 70 60 50 40 30 20 10

Output 7

1

Input 8

7

2 4 6 8 10 12 14

Output 8

7

Code

def max\_baskets(arr):

from bisect import bisect\_left

# This will store the smallest tail of all increasing subsequences # with different lengths.

tails = []

for basket in arr:

pos = bisect\_left(tails, basket) if pos == len(tails):

tails.append(basket) else:

tails[pos] = basket

return len(tails)

# Read input

n = int(input())

arr = list(map(int, input().split()))

# Print output print(max\_baskets(arr))

1. Ticket Counter Time

Problem Statement:

There’s a single ticket counter and a line of people waiting. Each person needs a specific amount of

time to be served, which is represented by an array T.

The counter serves people in order, but here's the rule: the counter works in rounds. In each round, the person at the front gets served for 1 time unit, then goes to the back of the queue if their remaining time is more than 0.

You are given the time required for each person, and you are asked to calculate how many time units it will take before the K-th person (0-indexed) in the original queue completes their service.

Input Format:

* First line contains two integers N (number of people) and K (index of target person).
* Second line contains N space-separated integers, where T[i] is the total time required for the i-th person.

Output Format:

* A single integer — total time taken until the K-th person completes service.

Sample Input 1:

1. 2

1 2 3 4

Sample Output 1:

8

Explanation:

[1,2,3,4] → [2,3,4] after 1 (P0 done)

[2,3,4,1] → [3,4,1] after 1 (P1: 1 left)

[3,4,1,1] → [4,1,1] after 1 (P2: 2 left)

[4,1,1,2] → [1,1,2] after 1 (P3: 3 left)

[1,1,2,3] → [1,2,3] after 1 (P1 done)

[2,3] → [3,2] after 1 (P2: 1 left)

[2,1] → [1,1] after 1 (P3: 1 left)

[1] → P2 done after 1 more

Sample Input 2:

1. 0

5 1 1 1 1

Sample Output 2:

9

Test Cases Input 1

1 0

1

Output 1

1

Input 2

3 2

1 1 1

Output 2

3

Input 3

3 0

3 3 3

Output 3

7

Input 4

5 4

1 1 1 1 5

Output 4

9

Input 5

1. 2

3 3 1 3 3 3

Output 5

3

Input 6

5 2

2 2 2 2 2

Output 6

8

Input 7

5 2

10 2 3 1 1

Output 7

10

Input 8

4 3

1 1 1 5

Output 8

8

Code

from collections import deque

def ticket\_counter\_time(n, k, times):

queue = deque([(i, t) for i, t in enumerate(times)]) total\_time = 0

while queue:

idx, time\_left = queue.popleft() time\_left -= 1

total\_time += 1

if time\_left == 0:

if idx == k:

return total\_time else:

queue.append((idx, time\_left))

return -1 # Should never reach here if input is valid

# Read input

n, k = map(int, input().split())

times = list(map(int, input().split()))

# Output result print(ticket\_counter\_time(n, k, times))

1. Repeating Pairs

Problem Statement:

You are given a string S. Your task is to count how many distinct pairs of characters occur more than once as consecutive characters in the string.

A "pair" means two characters next to each other. Pairs are considered the same if they have the same characters in the same order (e.g., ab and ab are the same, ab and ba are different).

Input Format:

* + A single line containing a string S.

Output Format:

* + Print a single integer — the number of distinct character pairs that appear more than once as consecutive characters.

Sample Input 1: ababcabc

Sample Output 1:

2

Explanation: Consecutive pairs:

ab, ba, ab, bc, ca, ab, bc ab occurs 3 times

bc occurs 2 times

Output = 2 (pairs ab and bc)

Sample Input 2: aaaa

Sample Output 2:

1

Test Cases Input 1 abcdef Output 1

0

Input 2 a

Output 2

0

Input 3 abababa Output 3

2

Input 4 abcabcabc Output 4

3

Input 5 aaaaaa Output 5

1

Input 6 abcdefg Output 6

0

Input 7 abababab Output 7

2

Input 8 z

Output 8

0

Code

from collections import defaultdict

def count\_repeating\_pairs(s): pair\_count = defaultdict(int)

for i in range(len(s) - 1): pair = s[i] + s[i + 1] pair\_count[pair] += 1

# Count pairs that appear more than once

return sum(1 for count in pair\_count.values() if count > 1)

# Read input

s = input().strip()

# Output result print(count\_repeating\_pairs(s))

1. Most Frequent Character After Cleanup

Problem Statement:

You are given a string S that may contain:

* + Lowercase letters (a-z)
  + Uppercase letters (A-Z)
  + Digits (0-9)
  + Special characters (punctuation, symbols, whitespace, etc.) Your task is to:

1. Remove all non-alphabetic characters (digits, special characters, whitespace).
2. Ignore case — that is, treat A and a as the same.
3. Find the most frequent alphabetic character in the cleaned string.

If multiple characters have the same maximum frequency, return the one that comes first in alphabetical order.

Input Format:

* + A single line containing a string S.

Output Format:

* + A single lowercase letter — the most frequent alphabetic character (after cleanup).

Sample Input 1:

Hello, World! 123

Sample Output 1:

l Explanation:

Cleaned string: helloworld → Frequencies: h:1, e:1, l:3, o:2, w:1, r:1, d:1

Most frequent: l

Sample Input 2: A@aaBBccC1234# Sample Output 2:

a

Test Cases Input 1 aaBBccC Output 1 c

Input 2 aaBBccCaa Output 2

a

Input 3 AAAAaaaaAAAaaa!!!@@@ Output 3

a

Input 4

xxYYzz!! Output 4 x

Input 5 QWERTYQWERTY

Output 5 e

Input 6

!!!@@@###bbb%%%^^^ Output 6

b

Input 7 aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ1234567890!@#$ Output 7

a

Input 8 z

Output 8 z

Code

from collections import defaultdict

def most\_frequent\_letter(s): freq = defaultdict(int)

for char in s:

if char.isalpha(): # Only consider alphabetic characters char = char.lower()

freq[char] += 1

if not freq:

return "No alphabetic characters" # No alphabetic characters

# Find the character with highest frequency, break ties by alphabetical order max\_freq = max(freq.values())

result = min([ch for ch in freq if freq[ch] == max\_freq]) print(result)

# Read input

s = input().strip() most\_frequent\_letter(s)

1. Lonely Light

Problem Statement:

In a row of street lights, only one light is faulty — it flickers only when it receives power an odd number of times.

You are given an array of integers, where each integer represents the ID of a street light that received power. Every light gets powered an even number of times — except for one faulty light that was powered an odd number of times.

Your task is to find the ID of that faulty street light.

Input Format:

* + First line contains an integer N — the number of entries.
  + Second line contains N space-separated integers representing the light IDs that received power.

Output Format:

* + A single integer — the ID of the faulty light.

Sample Input 1:

5

1 2 3 2 1

Sample Output 1:

3

Explanation:

1. appears twice, 2 appears twice, 3 appears once → faulty = 3

Sample Input 2:

7

5 7 5 4 7 9 4

Sample Output 2:

9

Test Cases Input 1

1

42

Output 1

42

Input 2

7

4 5 6 5 4 7 7

Output 2

6

Input 3

3

10 20 10

Output 3

20

Input 4

9

8 3 8 6 3 2 2 5 5

Output 4

6

Input 5

11

9 10 10 12 13 9 13 15 15 12 11

Output 5

11

Input 6

7

11 12 13 11 12 13 14

Output 6

14

Input 7

5

5 6 7 6 5

Output 7

7

Input 8

3

1000 2000 1000

Output 8

2000

Code

def find\_faulty\_light(arr): result = 0

for num in arr:

result ^= num # XOR cancels out pairs return result

# Read input

n = int(input())

arr = list(map(int, input().split()))

# Output result print(find\_faulty\_light(arr))

1. Sum of Array Pairs

Problem Statement:

You are given an array of integers. Your task is to find all unique pairs of integers in the array whose sum is equal to a given target value.

Input Format:

* + The first line contains an integer n (1 ≤ n ≤ 1000), the number of elements in the array.
  + The second line contains n space-separated integers, representing the elements of the array.
  + The third line contains an integer target, representing the sum you need to find pairs for.

Output Format:

* + Output all unique pairs of integers whose sum equals the given target.
  + Each pair should be printed on a new line, with the two integers separated by a space.
  + If no such pairs exist, print No pairs found.

Sample Input 1:

6

1 4 2 3 3 5

6

Sample Output 1:

1 5

1. 4
2. 3

Sample Input 2:

5

10 20 30 40 50

100

Sample Output 2: No pairs found

Test Cases Input 1

1

5

10

Output 1

No pairs found

Input 2

4

2 2 2 2

4

Output 2

2 2

Input 3

8

3 1 4 5 6 2 7 9

10

Output 3

1 9

3 7

1. 6

Input 4

4

1 2 3 4

10

Output 4

No pairs found

Input 5

6

1. 1 3 7 2 4

8

Output 5

1 7

3 5

Input 6

5

-1 -2 3 4 5

3

Output 6

-2 5

-1 4

Input 7

6

3 3 3 3 3 3

6

Output 7

3 3

Input 8

1

5

10

Output 8

No pairs found

Code

def find\_pairs(arr, target):

seen = set() # To keep track of elements we've already visited result = set() # To store unique pairs

for num in arr:

complement = target - num if complement in seen:

# To ensure pairs are added in a sorted order (for uniqueness) result.add(tuple(sorted((num, complement))))

seen.add(num)

if result:

for pair in sorted(result): print(pair[0], pair[1])

else:

print("No pairs found")

# Reading input

n = int(input()) # Number of elements in the array

arr = list(map(int, input().split())) # The array of integers target = int(input()) # The target sum

# Calling the function to find and print pairs find\_pairs(arr, target)

1. Count Set Bits in an Integer

Problem Statement:

Given a non-negative integer n, your task is to count the number of set bits (1s) in its binary representation.

Input Format:

* + A single line containing the integer n (0 ≤ n ≤ 10⁹)

Output Format:

* + Print the number of set bits in n.

Sample Input 1:

5

Sample Output 1:

2

Explanation:

Binary of 5 = 101, which has two 1s.

Sample Input 2:

15

Sample Output 2:

4

Explanation:

Binary of 15 = 1111, which has four 1s.

Test Cases Input 1

0

Output 1

0

Input 2

64

Output 2

1

Input 3

255

Output 3

8

Input 4

254

Output 4

7

Input 5

164

Output 5

3

Input 6

128

Output 6

1

Input 7

42

Output 7

3

Input 8

1024

Output 8

1

Code

def count\_set\_bits(n): count = 0

while n:

n &= (n - 1) # clears the lowest set bit count += 1

print(count)

# Input parsing n = int(input())

count\_set\_bits(n)

1. Weighted Digit Power Sum

Problem Statement:

A number is considered digit-power-weighted if the sum of its digits raised to their position from left to right, multiplied by their position, results in a unique score.

Given a positive integer n, calculate its weighted digit power sum using the formula:

For a number with digits d₁ d₂ ... dₖ, compute: d₁^1 \* 1 + d₂^2 \* 2 + d₃^3 \* 3 + ... + dₖ^k \* k Return the final value of this computation.

Input Format:

* + A single line containing the integer n (0 ≤ n ≤ 10⁹)

Output Format:

A single integer – the weighted digit power sum.

Sample Input 1:

321

Sample Output 1:

14

Explanation:

Digits: 3 2 1

* + 3^1 \* 1 = 3
  + 2^2 \* 2 = 4 \* 2 = 8
  + 1^3 \* 3 = 1 \* 3 = 3

Total = 3 + 8 + 3 = 14

Sample Input 2:

245

Sample Output 2:

409

Explanation:

* + 2^1 × 1 = 2
  + 4^2 × 2 = 16 × 2 = 32
  + 5^3 × 3 = 125 × 3 = 375

Sum = 2 + 32 + 375 = 409

Corrected Output: 409

Test Cases Input 1

7

Output 1

7

Input 2

111

Output 2

6

Input 3

1

Output 3

1

Input 4

222

Output 4

34

Input 5

123

Output 5

90

Input 6

321

Output 6

14

Input 7

406

Output 7

652

Input 8

105

Output 8

376

Code

def weighted\_digit\_power\_sum(n): s = str(n)

total = 0

for i, ch in enumerate(s): digit = int(ch)

power = digit \*\* (i + 1) total += power \* (i + 1)

print(total)

# Read input

n = int(input()) weighted\_digit\_power\_sum(n)

1. Chef’s Energy Boost Plan

Problem Statement:

Chef Rahul has recently taken a pledge to improve his health. Every morning, he drinks different fruit smoothies lined up in a long queue at his smoothie bar. Each smoothie has a certain energy value, which can be positive (if it boosts his energy) or negative (if it's a detox smoothie that lowers it).

Chef wants to figure out which k-day stretch of smoothies gives him the maximum total energy. He can only pick a consecutive sequence of k smoothies.

You must help Chef find the maximum energy he can gain in any stretch of k consecutive days.

Input Format:

* + First line: Two integers n (number of days/smoothies) and k (length of stretch)
  + Second line: n space-separated integers representing the energy values of the smoothies.

Output Format:

A single integer – the maximum total energy Chef can gain from any k consecutive smoothies.

Sample Input 1:

1. 6

1 2 3 4 5 6

Sample Output 1:

21

Sample Input 2:

5 2

-1 -2 -3 -4 -5

Sample Output 2:

-3

Test Cases Input 1

1 1

5

Output 1

5

Input 2

5 3

1 2 3 4 5

Output 2

12

Input 3

6 2

-1 -2 -3 -4 -5 -6

Output 3

-3

Input 4

1. 4

3 -1 2 -5 4 6 -2

Output 4

7

Input 5

1. 3

1 -1 2 -2 3 -3 4 -4

Output 5

4

Input 6

6 3

10 9 8 -1 -2 -3

Output 6

27

Input 7

6 2

-1 -2 -3 4 5 6

Output 7

11

Input 8

1. 5

2 2 2 2 2 2 2 2 2

Output 8

10

Code

def max\_energy\_window(n, k, energy): window\_sum = sum(energy[:k]) max\_sum = window\_sum

for i in range(k, n):

window\_sum += energy[i] - energy[i - k] max\_sum = max(max\_sum, window\_sum)

return max\_sum

# Read input

n, k = map(int, input().split())

energy = list(map(int, input().split())) print(max\_energy\_window(n, k, energy))

1. Lucky Locker Code Problem Statement:

An ancient locker in a treasure vault opens only when you enter a "Lucky Code". A number is considered lucky if the sum of its digits is a multiple of 4.

You're given a number N. Your task is to find the smallest integer greater than or equal to N which is a lucky code.

Input Format:

* + One line containing an integer N (1 ≤ N ≤ 10⁶)

Output Format:

* + A single integer – the smallest lucky code ≥ N.

Sample Input 1:

100

Sample Output 1:

103

Explanation:

Sum of digits of 100 → 1 → not a multiple of 4

101 → 1 + 0 + 1 = 2 → No

102 → 1 + 0 + 2 = 3 → No

103 → 1 + 0 + 3 = 4 → Correct

Sample Input 2:

1234

Sample Output 2:

1236

Test Cases Input 1

1

Output 1

4

Input 2

16

Output 2

17

Input 3

9999

Output 3

9999

Input 4

256

Output 4

259

Input 5

1006

Output 5

1007

Input 6

2017

Output 6

2019

Input 7

10

Output 7

13

Input 8

45

Output 8

48

Code

def is\_lucky(n):

return sum(int(d) for d in str(n)) % 4 == 0

def find\_lucky\_code(n):

while not is\_lucky(n): n += 1

return n

# Read input

n = int(input()) print(find\_lucky\_code(n))

1. The River Crossing - Find a Pair of Rocks with a Specific Problem Statement:

In a small village by the river, the villagers are crossing a stream using rocks. The rocks are placed at varying distances from each other along the riverbank. A young boy wants to cross the river, but he is only allowed to jump between rocks that are at a specific distance apart. You are tasked with helping the boy identify all the pairs of rocks where the difference between their positions equals a given distance.

Input Format:

* + The first line contains an integer n (1 ≤ n ≤ 1000), the number of rocks placed along the river.
  + The second line contains n integers rocks[0], rocks[1], ..., rocks[n-1] (1 ≤ rocks[i] ≤ 10^6),

representing the positions of the rocks in sorted order.

* + The third line contains an integer target\_distance (1 ≤ target\_distance ≤ 10^6), the distance

between the rocks the boy can jump.

Output Format:

* + For each pair of rocks whose difference in positions equals the target\_distance, print the pair on a new line as (rock1, rock2).
  + If no such pairs exist, print No pairs found.

Sample Input 1:

6

1 3 5 7 9 11

2

Sample Output 1:

(1, 3)

(3, 5)

(5, 7)

(7, 9)

(9, 11)

Sample Input 2:

5

2 4 6 8 10

5

Sample Output 2: No pairs found

Test Cases Input 1

5

2 4 6 8 10

3

Output 1

No pairs found Input 2

4

1 3 5 7

4

Output 2

(1, 5)

(3, 7)

Input 3

6

1 3 5 7 9 11

2

Output 3

(1, 3)

(3, 5)

(5, 7)

(7, 9)

(9, 11)

Input 4

7

1 2 3 4 5 6 7

1

Output 4

(1, 2)

(2, 3)

(3, 4)

(4, 5)

(5, 6)

(6, 7)

Input 5

7

5 1 10 8 6 4 3

2

Output 5

No pairs found

Input 6

5

10 15 20 25 30

5

Output 6

(10, 15)

(15, 20)

(20, 25)

(25, 30)

Input 7

6

1 2 3 4 5 6

1

Output 7

(1, 2)

(2, 3)

(3, 4)

(4, 5)

(5, 6)

Input 8

1

100

5

Output 8

No pairs found

Code

def find\_pairs\_with\_distance(rocks, target\_distance): left = 0

right = 1 pairs = []

while right < len(rocks):

current\_diff = rocks[right] - rocks[left]

if current\_diff == target\_distance: pairs.append((rocks[left], rocks[right])) left += 1

right = left + 1 # Reset the right pointer to one step ahead of the left elif current\_diff < target\_distance:

right += 1 else:

left += 1

if left == right:

right += 1

if not pairs:

print("No pairs found") else:

for pair in pairs:

print(f"({pair[0]}, {pair[1]})")

# Example usage

if name == " main ": # Sample Input 1

n = int(input()) # Number of rocks

rocks = list(map(int, input().split())) # List of rock positions

target\_distance = int(input()) # The target distance

# Call the function to find pairs find\_pairs\_with\_distance(rocks, target\_distance)

1. Undo the Typos Problem Statement:

Sara is writing her college essay in a simple text editor that only supports typing lowercase alphabets and undoing the last letter using the backspace key (represented by #). She types a sequence of characters, and for every #, the last typed character (if any) gets removed.

Can you help her figure out the final text after all typing and backspaces?

Input Format:

* + A single line string s (1 ≤ |s| ≤ 10^5), consisting of lowercase letters and the # character. Output Format:
  + A single line containing the final string after processing all characters.

Sample Input 1: abc#d##

Sample Output 1: a

Sample Input 2: a#bc##d

Sample Output 2: d

Test Cases Input 1 ##abc Output 1

abc

Input 2 hello Output 2 hello

Input 3 abc###xyz Output 3 xyz

Input 4 helloworld Output 4 helloworld

Input 5 a#b#c#d#e Output 5

e

Input 6 abc##de# Output 6 Ad

Input 7 a#bc#d#efg#h#

Output 7 bef

Input 8 a#b#c#d#e#f#z Output 8

z

Code

def process\_typing(s): stack = []

for ch in s:

if ch == '#':

if stack:

stack.pop() else:

stack.append(ch) return ''.join(stack)

# Example usage:

if name == " main ": s = input().strip() print(process\_typing(s))

1. Balanced Boxes Problem Statement:

A warehouse robot is stacking boxes represented by opening [ and closing ] brackets. It follows a sequence of commands, where:

* + [ means it places a box on the stack.
  + ] means it removes the top box (only if there is one).

Your job is to determine whether the robot has stacked and unstacked the boxes correctly. That is, every opening [ must have a corresponding closing ], and the order must be valid.

Input Format:

* + A single line string s consisting only of the characters [ and ].
  + Length: 1 ≤ |s| ≤ 10^5

Output Format:

* + Print "Balanced" if all boxes are stacked and unstacked properly.
  + Otherwise, print "Unbalanced".

Sample Input 1: [[]]

Sample Output 1: Balanced

Sample Input 2:

][][

Sample Output 2: Unbalanced

Test Cases Input 1

[]

Output 1 Balanced

Input 2 [[[]]]

Output 2 Balanced

Input 3

][

Output 3 Unbalanced

Input 4 [[[]]

Output 4 Unbalanced

Input 5 []][[]

Output 5 Unbalanced

Input 6 [[[[[]]]]]

Output 6 Balanced

Input 7 [][[[[]]]]

Output 7 Balanced

Input 8

]]]]

Output 8

Unbalanced

Code

def is\_balanced\_boxes(s): stack = []

for ch in s:

if ch == '[':

stack.append(ch) elif ch == ']':

if not stack:

return "Unbalanced" stack.pop()

return "Balanced" if not stack else "Unbalanced"

# Example usage

if name == " main ": s = input().strip() print(is\_balanced\_boxes(s))

Medium

1. The Thief of Time Problem Statement:

In the ancient kingdom of Chronosia, there exists a legendary thief named Kael who can steal time from time crystals. Each time crystal has a time value and a steal time (how long it takes to steal from it). Kael has T minutes before the royal guards catch him. His goal is to maximize the amount of time he can steal from the available crystals before he gets caught.

Each crystal can only be stolen once and stealing must be completed fully to obtain the time value.

Kael decides to steal from the most "efficient" crystals first, where efficiency is defined as the highest time value per unit of steal time. Help Kael make the optimal choice.

Input Format:

* + The first line contains an integer N — the number of time crystals.
  + The second line contains an integer T — the total time Kael has before the guards arrive.
  + The next N lines each contain two integers vi and ti — value of time (vi) and time to steal (ti) for each crystal.

Output Format:

* + Print a single integer — the maximum total value of time Kael can steal before getting caught.

Sample Input 1:

4

10

60 2

100 4

120 6

30 1

Sample Output 1:

190

Sample Input 2:

3

5

50 3

70 4

30 2

Sample Output 2:

70

Test Cases Input 1

0

100

Output 1

0

Input 2

3

0

100 1

200 2

300 3

Output 2

0

Input 3

3

1000000000

10000 10000

10000 10000

10000 10000

Output 3

30000

Input 4

5

15

50 5

60 4

70 6

80 3

30 2

Output 4

240

Input 5

4

10

100 5

200 10

150 7

50 2

Output 5

200

Input 6

3

4

10 5

100 4

5 1

Output 6

100

Input 7

3

1

100 2

200 3

300 4

Output 7

0

Input 8

3

6

100 2

150 2

120 2

Output 8

370

Code

def max\_stolen\_time\_value(n, t, crystals): # Sort by value per time descending

crystals.sort(key=lambda x: x[0] / x[1], reverse=True)

total\_value = 0 remaining\_time = t

for value, steal\_time in crystals:

if steal\_time <= remaining\_time:

total\_value += value remaining\_time -= steal\_time

else:

continue

return total\_value

# Input reading

if name == " main ": n = int(input())

t = int(input())

crystals = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_stolen\_time\_value(n, t, crystals))

1. The Thief of Time

Problem Statement:

There’s a single ticket counter and a line of people waiting. Each person needs a specific amount of

time to be served, which is represented by an array T.

The counter serves people in order, but here's the rule: the counter works in rounds. In each round, the person at the front gets served for 1 time unit, then goes to the back of the queue if their remaining time is more than 0.

You are given the time required for each person, and you are asked to calculate how many time units it will take before the K-th person (0-indexed) in the original queue completes their service.

Input Format:

* First line contains two integers N (number of people) and K (index of target person).
* Second line contains N space-separated integers, where T[i] is the total time required for the i-th person.

Output Format:

* A single integer — total time taken until the K-th person completes service.

Sample Input 1:

4 2

1 2 3 4

Sample Output 1:

8

Explanation:

[1,2,3,4] → [2,3,4] after 1 (P0 done)

[2,3,4,1] → [3,4,1] after 1 (P1: 1 left)

[3,4,1,1] → [4,1,1] after 1 (P2: 2 left)

[4,1,1,2] → [1,1,2] after 1 (P3: 3 left)

[1,1,2,3] → [1,2,3] after 1 (P1 done)

[2,3] → [3,2] after 1 (P2: 1 left)

[2,1] → [1,1] after 1 (P3: 1 left)

[1] → P2 done after 1 more

Sample Input 2:

5 0

5 1 1 1 1

Sample Output 2:

9

Test Cases Input 1

1 0

1

Output 1

1

Input 2

1. 2

1 1 1

Output 2

3

Input 3

3 0

3 3 3

Output 3

7

Input 4

5 4

1 1 1 1 5

Output 4

9

Input 5

6 2

3 3 1 3 3 3

Output 5

3

Input 6

5 2

2 2 2 2 2

Output 6

8

Input 7

5 2

10 2 3 1 1

Output 7

10

Input 8

1. 3

1 1 1 5

Output 8

8

Code

def max\_stolen\_time\_value(n, t, crystals): # Sort by value per time descending

crystals.sort(key=lambda x: x[0] / x[1], reverse=True)

total\_value = 0 remaining\_time = t

for value, steal\_time in crystals:

if steal\_time <= remaining\_time:

total\_value += value remaining\_time -= steal\_time

else:

continue

return total\_value

# Input reading

if name == " main ": n = int(input())

t = int(input())

crystals = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_stolen\_time\_value(n, t, crystals))

1. Festival of Lanterns

Problem Statement:

In the kingdom of Lumora, every year a lantern festival is held where participants light floating lanterns along a river. Each lantern must be lit and launched at a specific time interval, and each one brings a certain amount of joy to the crowd.

The Grand Organizer can only allow one lantern to be launched at any given second (due to narrow river constraints). You're given N lanterns, each with:

* + A deadline (the latest second it can be launched),
  + A joy value.

Your task is to select the subset of lanterns to maximize total joy, ensuring no two lanterns are launched at the same time, and each is launched on or before its deadline.

This is a classic Greedy + Priority Queue problem (variant of Job Scheduling with Deadlines).

Input Format:

* + First line: Integer N — number of lanterns.
  + Next N lines: Each line has two integers:
* di — deadline of the i-th lantern (1 ≤ di ≤ 10^5)
* ji — joy value of the i-th lantern

Output Format:

* + One integer — maximum total joy that can be obtained.

Sample Input 1:

5

1. 100

1 50

2 200

1 20

1. 300

Sample Output 1:

600

Explanation:

We can launch 3 lanterns (at most 1 per second):

* + Choose lanterns with highest joy but that can meet their deadline:
* Launch 300 at time 3
* Launch 200 at time 2
* Launch 100 at time 1

Total Joy = 300 + 200 + 100 = 600

Sample Input 2:

4

1 40

1. 50

2 30

1 70

Sample Output 2:

120

Test Cases Input 1

1

1 9999

Output 1

9999

Input 2

4

1 10

1 20

1 5

1 100

Output 2

100

Input 3

3

1 100

2 200

1. 300

Output 3

600

Input 4

6

1 5

2 20

2 30

2 25

3 50

3 90

Output 4

170

Input 5

4

1 10

2 10

3 10

1. 10

Output 5

40

Input 6

3

1. 100
2. 200
3. 300

Output 6

600

Input 7

5

2 10

2 20

2 30

2 40

2 50

Output 7

90

Input 8

5

1 10000

1 9000

1 8000

1 7000

1 6000

Output 8

10000

Code

import heapq

def max\_total\_joy(n, lanterns): # Sort lanterns by deadline

lanterns.sort(key=lambda x: x[0]) min\_heap = []

for deadline, joy in lanterns:

heapq.heappush(min\_heap, joy)

# If we have more lanterns than we can schedule by this deadline, remove the least joyful one if len(min\_heap) > deadline:

heapq.heappop(min\_heap)

return sum(min\_heap)

# Input Reading

if name == " main ":

n = int(input())

lanterns = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_total\_joy(n, lanterns))

1. The Merchant's Caravan Problem Statement:

A wealthy merchant is heading to the annual Golden Dunes Market, carrying items of different values and weights. His caravan can only carry a maximum weight W due to desert travel constraints.

He wants to maximize his profit by selecting the subset of items that can be carried without exceeding the total weight limit.

This is a 0/1 Knapsack-like greedy problem, but the twist is:

He can take fractions of an item if needed — i.e., this is the Fractional Knapsack Problem, solved greedily by value per unit weight.

Input Format:

* + First line: Two integers N and W
* N: number of items
* W: maximum weight capacity
  + Next N lines: Each line contains two integers:
* value and weight of the item

Output Format:

* + A single floating-point number — the maximum total value the merchant can carry.
  + Round off your answer to 2 decimal places.

Sample Input 1:

3 50

60 10

100 20

120 30

Sample Output 1:

240.00

Explanation:

Sort by value/weight:

* + 60/10 = 6.0
  + 100/20 = 5.0
  + 120/30 = 4.0

Pick:

* + 60 (10)
  + 100 (20)
  + Remaining capacity: 20 → take 2/3 of item worth 120 → value = 80

Total = 60 + 100 + 80 = 240.00

Sample Input 2:

1 1

100 2

Sample Output 2:

50.00

Test Cases Input 1

2 30

100 10

120 30

Output 1

220.00

Input 2

3 60

60 10

100 20

120 30

Output 2

280.00

Input 3

3 10

60 10

100 20

120 30

Output 3

60.00

Input 4

2 5

100 10

120 30

Output 4

50.00

Input 5

1. 10

10 1

20 2

30 3

40 4

Output 5

100.00

Input 6

3 50

500 50

400 40

300 30

Output 6

500.00

Input 7

1 2

100 4

Output 7

50.00

Input 8

1. 100

10 10

20 20

30 30

40 40

50 50

Output 8

100.00

Code

def max\_value(n, w, items):

items.sort(key=lambda x: x[0]/x[1], reverse=True) total\_value = 0.0

for value, weight in items:

if w == 0:

break

if weight <= w: total\_value += value w -= weight

else:

total\_value += value \* (w / weight) w = 0

return round(total\_value, 2)

# Driver code

if name == " main ":

n, w = map(int, input().split())

items = [tuple(map(int, input().split())) for \_ in range(n)] print(f"{max\_value(n, w, items):.2f}")

1. The Band Booking Problem

Problem Statement:

A music event organizer wants to schedule bands for a music marathon. Each band has a performance start time, end time, and a popularity score (which is equal to the number of tickets they help sell).

Due to logistical constraints, no two bands can perform at overlapping times. The goal is to select a subset of non-overlapping bands to maximize total popularity.

Input Format:

* + First line: Single integer N (number of bands)
  + Next N lines: Three integers per line:

o start\_time end\_time popularity

Output Format:

* + A single integer: the maximum total popularity achievable with non-overlapping performances

Sample Input 1:

4

1 3 50

2 5 60

4 6 70

1. 7 30

Sample Output 1:

150

Explanation:

Take band 1 (1-3) → pop 50

Skip band 2 (2-5) – overlaps

Take band 3 (4-6) → pop 70

Take band 4 (6-7) → pop 30

Total = 50 + 70 + 30 = 150

Sample Input 2:

4

1 3 50

2 5 60

4 6 70

6 7 30

Sample Output 2:

150

Test Cases Input 1

3

1 2 20

3 4 40

5 6 60

Output 1

120

Input 2

3

1 5 10

2 6 30

3 7 20

Output 2

30

Input 3

4

1 3 50

3 5 50

5 7 50

2 8 200

Output 3

200

Input 4

3

1 10 100

2 3 20

4 5 30

Output 4

100

Input 5

5

1 3 20

3 6 30

6 9 40

9 12 50

12 15 60

Output 5

200

Input 6

4

1 5 100

10 15 150

20 25 200

30 35 250

Output 6

700

Input 7

5

1 10 100

1 2 30

2 3 30

3 4 30

4 5 30

Output 7

120

Input 8

6

1 3 50

3 5 60

5 7 70

1. 9 80

9 11 90

11 13 100

Output 8

450

Code

import bisect

def max\_popularity(bands):

bands.sort(key=lambda x: x[1]) # Sort by end time end\_times = [band[1] for band in bands]

dp = [0] \* (len(bands) + 1)

for i in range(1, len(bands)+1):

start, end, pop = bands[i-1]

# Find last band that ends before current starts idx = bisect.bisect\_right(end\_times, start) - 1 dp[i] = max(dp[i-1], dp[idx+1] + pop)

return dp[-1]

# Driver

if name == " main ": n = int(input())

bands = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_popularity(bands))

1. The Conference Room Dilemma

Problem Statement:

You are given N meetings, each with a start time and end time. All meetings must be scheduled in conference rooms, and no two overlapping meetings can be in the same room.

Your task is to find the minimum number of rooms required to accommodate all meetings without overlap.

Input Format:

* + First line: Integer N (number of meetings)
  + Next N lines: Two integers — start end

Output Format:

* + A single integer — minimum number of rooms required

Sample Input 1:

4

0 30

1. 10

15 20

25 35

Sample Output 1:

2

Sample Input 2:

3

1 2

3 4

5 6

Sample Output 2:

1

Test Cases Input 1

4

1 10

1 10

1 10

1 10

Output 1

4

Input 2

5

1 4

2 5

3 6

1. 7
2. 8

Output 2

3

Input 3

4

1 2

2 3

3 4

4 5

Output 3

1

Input 4

3

1 10

2 10

3 10

Output 4

3

Input 5

5

1 10

2 3

3 4

4 5

5 6

Output 5

2

Input 6

6

1 5

2 6

5 7

6 8

7 9

1. 10

Output 6

2

Input 7

4

1 5

2 5

3 5

4 5

Output 7

4

Input 8

6

10 20

30 40

50 60

15 25

35 45

55 65

Output 8

2

Code

def min\_rooms(meetings):

start\_times = sorted([s for s, e in meetings]) end\_times = sorted([e for s, e in meetings])

start\_ptr = end\_ptr = 0 current\_rooms = max\_rooms = 0

while start\_ptr < len(meetings):

if start\_times[start\_ptr] < end\_times[end\_ptr]: current\_rooms += 1

max\_rooms = max(max\_rooms, current\_rooms) start\_ptr += 1

else:

current\_rooms -= 1

end\_ptr += 1

return max\_rooms

# Driver

if name == " main ": n = int(input())

meetings = [tuple(map(int, input().split())) for \_ in range(n)] print(min\_rooms(meetings))

1. The Coupon Collector

Problem Statement:

You are at a store that offers coupons on N items. For each item, you are given:

* + The price of the item.
  + The discount if a coupon is used.

But… you only have K coupons, and each coupon can be applied to only one item.

Your task is to minimize the total cost of buying all items by using at most K coupons, where each coupon gives the full discount on one item.

Input Format:

* + First line: Two integers N and K
  + Next N lines: Two integers price discount

Output Format:

* + A single integer — minimum total cost after using coupons.

Sample Input 1:

5 2

100 20

200 50

150 30

120 60

80 10

Sample Output 1:

540

Explanation:

Use coupons on items with the maximum discount: 60 and 50

* + Prices: [100, 200, 150, 120, 80] → Total = 650
  + Apply coupons to items with discount 60 and 50 → Save 110
  + Final cost = 650 - 110 = 540

Sample Input 2:

3 0

100 50

200 100

150 75

Sample Output 2:

450

Test Cases Input 1

3 3

100 10

100 20

100 30

Output 1

240

Input 2

2 5

300 100

400 150

Output 2

450

Input 3

4 2

100 0

200 0

300 0

400 0

Output 3

1000

Input 4

3 2

100 100

200 200

300 300

Output 4

100

Input 5

1 0

999 888

Output 5

999

Input 6

128

Output 6

1

Input 7

6 3

100 5

100 10

100 15

100 20

100 25

100 30

Output 7

525

Input 8

6 3

100 5

100 10

100 15

100 20

100 25

100 20

Output 8

535

Code

def min\_total\_cost(n, k, items):

total\_price = sum(price for price, \_ in items) discounts = sorted((d for \_, d in items), reverse=True) total\_discount = sum(discounts[:k])

return total\_price - total\_discount

# Driver

if name == " main ":

n, k = map(int, input().split())

items = [tuple(map(int, input().split())) for \_ in range(n)] print(min\_total\_cost(n, k, items))

1. Penalty-Free Submissions

Problem Statement:

You are a competitive programmer participating in a contest with N problems. Each problem has:

* + deadline: the latest time you can submit it without penalty.
  + penalty: the penalty incurred if you miss the deadline (if not done at all or submitted late). Each problem takes exactly 1 unit of time to solve, and you can only solve one problem per unit time.

Your goal is to maximize the total penalty avoided, i.e., pick problems in such a way that you solve the highest-penalty problems within their deadlines.

Input Format:

* + First line: Integer N – number of problems
  + Next N lines: Two integers deadline and penalty for each problem

Output Format:

A single integer – maximum total penalty you can avoid

Sample Input 1:

5

2 100

1 19

2 27

1 25

3 15

Sample Output 1:

142

Sample Input 2:

4

1 50

1 40

1 30

1 20

Sample Output 2:

50

Test Cases Input 1

3

1 10

2 20

3 30

Output 1

60

Input 2

4

1 10

1 20

1 30

1 40

Output 2

40

Input 3

5

1 5

2 50

3 15

2 20

3 25

Output 3

95

Input 4

6

2 10

2 20

2 30

2 40

2 50

2 60

Output 4

110

Input 5

3

10 100

1. 90

8 80

Output 5

270

Input 6

6

1 100

2 10

1 30

3 20

2 90

3 80

Output 6

270

Input 7

4

1 50

2 50

3 50

2 50

Output 7

150

Input 8

5

10000 1000000

9000 900000

8000 800000

7000 700000

6000 600000

Output 8

4000000

Code

def max\_penalty\_avoided(n, tasks):

tasks.sort(key=lambda x: -x[1]) # Sort by penalty descending max\_deadline = max(d for d, \_ in tasks)

slots = [False] \* (max\_deadline + 1) total = 0

for deadline, penalty in tasks: for t in range(deadline, 0, -1):

if not slots[t]: slots[t] = True total += penalty break

return total

# Driver

if name == " main ": n = int(input())

tasks = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_penalty\_avoided(n, tasks))

1. Minimum Rooms Required

Problem Statement:

You are given N events, each with a start time and an end time. Your task is to determine the minimum number of rooms required to schedule all events without any overlap in a single room.

Each room can hold only one event at a time. Events are considered overlapping if one starts before another ends (i.e., inclusive).

Input Format:

* + First line: Integer N – number of events
  + Next N lines: Two integers start and end for each event (inclusive)

Output Format:

A single integer – minimum number of rooms required.

Sample Input 1:

5

0 30

5 10

15 20

35 45

25 40

Sample Output 1:

2

Sample Input 2:

4

0 5

6 10

11 15

16 20

Sample Output 2:

1

Test Cases Input 1

4

1 10

2 9

3 8

4 7

Output 1

4

Input 2

6

1 5

2 6

5 9

7 10

10 13

12 15

Output 2

3

Input 3

5

1 100

2 3

4 5

6 7

8 9

Output 3

2

Input 4

1

0 1

Output 4

1

Input 5

4

0 1

2 3

4 5

6 7

Output 5

1

Input 6

3

1 1

1 1

1 1

Output 6

3

Input 7

3

1 2

2 3

3 4

Output 7

2

Input 8

8

0 10

10 20

20 30

5 15

15 25

25 35

30 40

35 45

Output 8

3

Code

import heapq

def min\_rooms(events): events.sort() # Sort by start time min\_heap = []

for start, end in events:

if min\_heap and min\_heap[0] < start:

heapq.heappop(min\_heap) heapq.heappush(min\_heap, end)

return len(min\_heap)

# Driver

if name == " main ": n = int(input())

events = [tuple(map(int, input().split())) for \_ in range(n)] print(min\_rooms(events))

1. Deadline-Focused Task Execution Problem Statement:

You are given N tasks. Each task takes a certain amount of time to complete and has a deadline. You can execute the tasks one after another, starting from time 0. Your goal is to complete the maximum number of tasks such that each task is finished on or before its deadline.

You can only work on one task at a time, and you must choose tasks wisely (greedy!) so you can finish as many as possible.

Input Format:

* + First line: Integer N – number of tasks
  + Next N lines: Two integers duration and deadline for each task Output Format:
  + A single integer – maximum number of tasks that can be completed on or before their deadlines

Sample Input 1:

5

3 9

2 8

1 9

2 15

5 12

Sample Output 1:

5

Sample Input 2:

3

2 5

1 3

2 6

Sample Output 2:

3

Test Cases Input 1

4

2 4

3 5

4 6

10 7

Output 1

2

Input 2

4

1 1

2 3

3 6

4 10

Output 2

4

Input 3

5

1 5

1 6

1 7

1 8

1 9

Output 3

5

Input 4

3

3 2

4 3

5 4

Output 4

0

Input 5

5

6 12

1 3

2 4

1 5

3 6

Output 5

4

Input 6

4

3 10

2 10

1 10

4 10

Output 6

4

Input 7

6

5 10

3 8

2 5

1 3

6 12

4 11

Output 7

4

Input 8

7

10 100

20 100

30 100

40 100

15 100

5 100

25 100

Output 8

5

Code

import heapq

def max\_tasks(tasks):

tasks.sort(key=lambda x: x[1]) # sort by deadline total\_time = 0

durations = []

for duration, deadline in tasks:

total\_time += duration heapq.heappush(durations, -duration) if total\_time > deadline:

total\_time += heapq.heappop(durations) # remove longest task

return len(durations)

# Driver

if name == " main ": n = int(input())

tasks = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_tasks(tasks))

1. Charging Station Scheduling Problem Statement:

You're managing a charging station with K charging ports. There are N electric cars that want to charge. Each car arrives at a specific time and needs a specific duration to fully charge.

Your goal is to maximize the number of cars that can be charged, assuming:

* + A car can only start charging at or after its arrival time.
  + A port is immediately available after the last car using it finishes.
  + You can charge up to K cars simultaneously at any time.

Use a greedy algorithm to schedule cars so that the maximum number can finish charging.

Input Format:

* + First line: Two integers N (number of cars), K (number of ports)
  + Next N lines: Two integers arrival\_time and duration for each car

Output Format:

* + A single integer – the maximum number of cars that can be charged

Sample Input 1:

5 2

1 4

2 3

3 2

10 1

5 2

Sample Output 1:

4

Sample Input 2:

3 1

1 2

3 2

5 2

Sample Output 2:

3

Test Cases Input 1

4 2

1 2

3 2

5 2

7 2

Output 1

4

Input 2

4 1

1 5

2 5

3 5

4 5

Output 2

1

Input 3

3 3

1 5

1 5

1 5

Output 3

3

Input 4

5 2

1 4

2 3

3 2

10 1

5 2

Output 4

4

Input 5

4 2

1 2

2 2

10 1

11 1

Output 5

4

Input 6

6 2

1 3

2 2

3 2

4 2

5 2

6 2

Output 6

5

Input 7

10 2

1 2

1 2

1 2

1 2

1 2

3 2

3 2

3 2

5 2

5 2

Output 7

6

Input 8

6 2

1 1

2 1

3 1

4 1

5 1

6 1

Output 8

6

Code

import heapq

def max\_charged\_cars(cars, k):

# Sort cars by end time (arrival + duration) cars.sort(key=lambda x: x[0] + x[1])

ports = [] # Min-heap of end times count = 0

for arrival, duration in cars:

# Free up all ports where charging is done while ports and ports[0] <= arrival:

heapq.heappop(ports) if len(ports) < k:

heapq.heappush(ports, arrival + duration) count += 1

return count

# Driver

if name == " main ":

n, k = map(int, input().split())

cars = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_charged\_cars(cars, k))

1. Minimum Coins to Pay Problem Statement:

You are given an amount N and a list of M coin denominations. Your task is to find the minimum number of coins needed to make up that amount using the available denominations. You can use each coin type an unlimited number of times.

Input Format:

* + First line: Two integers N (target amount), M (number of denominations)
  + Second line: M integers – the coin denominations. Output Format:
  + A single integer – the minimum number of coins needed to make N

Sample Input 1:

49 6

1 2 5 10 20 50

Sample Output 1:

5

Explanation:

Use largest coins first:

* + 20 + 20 = 40
  + 5 = 45
  + 2 = 47
  + 1 + 1 = 49
  + Total: 5 coins

Sample Input 2:

49 6

1 2 5 10 20 50

Sample Output 2:

5

Test Cases Input 1

100 4

1 5 10 100

Output 1

1

Input 2

37 5

1 5 10 20 25

Output 2

4

Input 3

7 1

1

Output 3

7

Input 4

18 3

2 5 10

Output 4

3

Input 5

18 4

1 2 5 10

Output 5

4

Input 6

999 6

1 2 5 10 20 100

Output 6

17

Input 7

15 6

1 1 5 5 10 10

Output 7

2

Input 8

7 2

2 4

Output 8

2

Code

def min\_coins\_to\_pay(n, coins): coins.sort(reverse=True) count = 0

for coin in coins: while n >= coin:

n -= coin

count += 1 return count

# Driver

if name == " main ":

n, m = map(int, input().split()) coins = list(map(int, input().split())) print(min\_coins\_to\_pay(n, coins))

1. Job Sequencing for Maximum Profit Problem Statement:

You are given N tasks. Each task has a deadline and a reward. You can only do one task per day, and each task takes exactly one day. If a task is not completed on or before its deadline, you miss the reward.

Your goal is to maximize the total reward by selecting and scheduling tasks greedily.

Input Format:

* + First line: An integer N (number of tasks)
  + Next N lines: Two integers D and R per line (deadline and reward) Output Format:

A single integer – maximum total reward you can earn

Sample Input 1:

5

2 50

1 10

2 20

1 30

3 40

Sample Output 1:

120

Sample Input 2:

4

1 10

1 20

1 30

1 40

Sample Output 2:

40

Test Cases Input 1

4

1 10

2 20

3 30

4 40

Output 1

100

Input 2

6

2 50

2 60

2 70

2 40

2 30

2 20

Output 2

130

Input 3

3

3 15

2 10

1 20

Output 3

45

Input 4

5

1 10

2 10

3 10

2 10

1 10

Output 4

30

Input 5

5

1 100

2 10

2 10

3 10

3 10

Output 5

120

Input 6

1

1 1000

Output 6

1000

Input 7

4

10 100

11 200

12 300

13 400

Output 7

1000

Input 8

6

1 10

1 20

1 30

1 40

1 50

1 60

Output 8

60

Code

def find(parent, x): if parent[x] != x:

parent[x] = find(parent, parent[x]) return parent[x]

def union(parent, x, y): parent[find(parent, x)] = find(parent, y)

def max\_total\_reward(tasks):

tasks.sort(key=lambda x: -x[1]) # Sort by reward descending max\_deadline = max(d for d, \_ in tasks)

parent = [i for i in range(max\_deadline + 2)]

total\_reward = 0 for d, r in tasks:

available\_day = find(parent, d) if available\_day > 0:

total\_reward += r

union(parent, available\_day, available\_day - 1) return total\_reward

# Driver

if name == " main ": n = int(input())

tasks = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_total\_reward(tasks))

Hard

1. Magical Paths in the Infinite Forest Problem Statement:

The Infinite Forest is a mystical place represented as a 2D grid. You start at the top-left cell (0, 0) and want to reach the bottom-right cell (N-1, M-1). Each cell contains a magical cost, which you must pay to enter that cell.

You can only move right or down at any point in time. But there's a twist.

There are some rare teleport tiles in the grid (identified by a value of -1). From a teleport tile, you can jump to any other teleport tile for free, without paying the cost of the destination teleport tile (but you must still have passed into the teleport tile you're jumping from).

Your goal is to find the minimum magical cost to travel from the start to the destination, considering both normal and teleportation moves.

Input Format:

* + The first line contains two integers N and M — the dimensions of the forest.
  + The next N lines contain M space-separated integers. Each cell contains:
* A positive integer denoting the cost of entering that cell, or
* -1 denoting a teleport tile.

Output Format:

Print a single integer — the minimum cost to reach (N-1, M-1) from (0, 0).

Sample Input 1:

4 4

1 3 -1 4

2 8 5 9

4 -1 3 1

6 7 2 5

Sample Output 1:

13

Sample Input 2:

3 3

1 -1 3

2 4 -1

1 1 1

Sample Output 2:

2

Test Cases

Input 1

3 3

1 2 3

4 5 6

7 8 9

Output 1

21

Input 2

3 3

1 -1 3

2 100 -1

1 1 1

Output 2

2

Input 3

4 4

1 2 -1 9

1. 9 9 9

-1 9 9 9

1. 9 9 1

Output 3

31

Input 4

3 3

-1 3 1

2 -1 2

3 4 1

Output 4

3

Input 5

3 3

1 2 3

4 5 6

7 8 -1

Output 5

12

Input 6

3 3

1 2 3

4 -1 6

7 8 9

Output 6

18

Input 7

4 4

1 -1 -1 1

1 100 100 1

1 100 100 1

1 1 1 1

Output 7

5

Input 8

4 4

1 99 99 99

-1 99 99 99

99 99 99 -1

99 99 99 1

Output 8

2

Code import sys

import heapq

def min\_magic\_cost(grid, n, m): INF = float('inf')

dp = [[INF] \* m for \_ in range(n)]

dp[0][0] = grid[0][0] if grid[0][0] != -1 else 0

# Collect teleport tiles teleports = []

for i in range(n):

for j in range(m):

if grid[i][j] == -1: teleports.append((i, j))

# Priority queue: (cost, x, y) heap = [(dp[0][0], 0, 0)]

visited\_teleport = False

while heap:

cost, x, y = heapq.heappop(heap)

if cost > dp[x][y]: continue

# Normal right/down moves for dx, dy in [(1, 0), (0, 1)]:

nx, ny = x + dx, y + dy

if 0 <= nx < n and 0 <= ny < m:

enter\_cost = 0 if grid[nx][ny] == -1 else grid[nx][ny] if dp[nx][ny] > cost + enter\_cost:

dp[nx][ny] = cost + enter\_cost heapq.heappush(heap, (dp[nx][ny], nx, ny))

# If on teleport, update all other teleports if grid[x][y] == -1 and not visited\_teleport:

for tx, ty in teleports:

if (tx, ty) != (x, y):

if dp[tx][ty] > cost:

dp[tx][ty] = cost

heapq.heappush(heap, (dp[tx][ty], tx, ty)) visited\_teleport = True # only need one full teleport sync

return dp[n-1][m-1]

# Reading input

n, m = map(int, input().split())

grid = [list(map(int, input().split())) for \_ in range(n)]

print(min\_magic\_cost(grid, n, m))

1. Longest Alternating Subsequence with Jumps

Problem Statement:

You are given an array of integers A of length N. A subsequence S of A is called a "Jumpy Alternating Subsequence" if:

1. It consists of elements from A in increasing index order (i.e., it's a subsequence).
2. The difference between consecutive elements in S must strictly alternate in sign.
3. The absolute difference between any two consecutive elements must be at least K. Your task is to find the length of the longest such subsequence.

Input Format:

* + First line: Two integers N and K — the length of the array and the minimum absolute jump size.
  + Second line: N space-separated integers — the array A.

Output Format:

* + A single integer — the length of the longest jumpy alternating subsequence.

Sample Input 1:

6 2

1 7 4 9 2 5

Sample Output 1:

6

Explanation:

One such longest sequence is [1, 7, 4, 9, 2, 5] with diffs [+6, -3, +5, -7, +3].

Sample Input 2:

5 3

1 2 3 4 5

Sample Output 2:

2

Test Cases

Input 1

6 2

1 7 4 9 2 5

Output 1

6

Input 2

5 1

3 3 3 3 3

Output 2

1

Input 3

5 10

1 2 3 4 5

Output 3

1

Input 4

7 0

1 17 5 10 13 15 10

Output 4

5

Input 5

7 3

10 20 5 25 1 30 0

Output 5

7

Input 6

4 2

1 1 1 4

Output 6

2

Input 7

5 4

10 5 1 -3 -10

Output 7

2

Input 8

10 1

1 3 2 4 3 5 4 6 5 7

Output 8

10

Code

def longest\_jumpy\_alternating\_subsequence(arr, k): n = len(arr)

dp\_up = [1] \* n # ending with upward jump dp\_down = [1] \* n # ending with downward jump

for i in range(1, n): for j in range(i):

diff = arr[i] - arr[j]

if abs(diff) >= k:

if diff > 0:

dp\_up[i] = max(dp\_up[i], dp\_down[j] + 1) elif diff < 0:

dp\_down[i] = max(dp\_down[i], dp\_up[j] + 1)

return max(max(dp\_up), max(dp\_down))

# Read input

n, k = map(int, input().split())

arr = list(map(int, input().split()))

print(longest\_jumpy\_alternating\_subsequence(arr, k))

1. Minimum Sum Subsequence with Non-Adjacent Elements

Problem Statement:

You're given an array of integers arr of length n. You need to select a subsequence such that:

1. No two adjacent elements in the original array can be picked (i.e., if you pick arr[i], you cannot pick arr[i+1]).
2. The sum of the selected elements is minimized.
3. You must pick at least one element.

Write a program to compute the minimum possible sum under these rules.

Input Format:

* + First line contains a single integer n – the size of the array.
  + Second line contains n space-separated integers – the elements of the array. Output Format:
  + A single integer: Minimum number of button presses required to type the message.

Sample Input 1:

5

3 2 5 10 7

Sample Output 1:

2

Sample Input 2:

1

5

Sample Output 2:

5

Test Cases Input 1

2

10 2

Output 1

2

Input 2

6

-5 -10 -3 -1 -6 -9

Output 2

-20

Input 3

7

1 100 2 100 3 100 4

Output 3

1

Input 4

5

4 5 1 5 4

Output 4

4

Input 5

6

7 7 7 7 7 7

Output 5

7

Input 6

3

5 100

6 200

7 300

Output 6

600

Input 7

5

1. 7 -100 6 9

Output 7

-92

Input 8

6

10 9 8 7 6 5

Output 8

9

Code

def min\_non\_adjacent\_sum(arr): n = len(arr)

if n == 1:

return arr[0]

dp = [0] \* n dp[0] = arr[0]

dp[1] = min(arr[0], arr[1])

for i in range(2, n):

dp[i] = min(dp[i - 1], dp[i - 2] + arr[i])

return dp[-1]

# Input

n = int(input())

arr = list(map(int, input().split())) print(min\_non\_adjacent\_sum(arr))

1. Maximize Length of Chain of Pairs Problem Statement:

You are given n pairs of integers. Each pair represents a directed link (a, b) such that a < b.

You can form a chain by choosing some pairs such that for each consecutive pair (a, b) and (c, d) in the chain, b < c.

Your task is to select the maximum number of pairs you can chain together following this rule.

Input Format:

* + First line: An integer n — number of pairs.
  + Next n lines: Each line has two integers a and b Output Format:

A single integer — the maximum number of pairs that can be chained.

Sample Input 1:

4

5 24

15 25

27 40

50 60

Sample Output 1:

3

Sample Input 2:

5

1 2

2 3

3 4

4 5

5 6

Sample Output 2:

3

Test Cases Input 1

5

1 2

2 3

3 4

4 5

5 6

Output 1

3

Input 2

3

1 10

2 3

3 4

Output 2

1

Input 3

1

0 1

Output 3

1

Input 4

2

1 5

2 6

Output 4

1

Input 5

6

1 2

2 3

4 5

6 7

8 9

10 11

Output 5

5

Input 6

6

5 10

1 4

1. 11

13 20

6 8

21 25

Output 6

4

Input 7

0

Output 7

0

Input 8

7

-5 -3

-1 1

2 3

4 6

6 8

9 12

13 15

Output 8

6

Code

def max\_chain\_length(pairs): # Sort pairs by second value

pairs.sort(key=lambda x: x[1])

count = 0

current\_end = float('-inf')

for a, b in pairs:

if a > current\_end:

count += 1 current\_end = b

return count

# Input

n = int(input())

pairs = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_chain\_length(pairs))

1. Split Array into k Subarrays with Minimum Max Sum

Problem Statement:

You are given an array of n integers and an integer k.

Your task is to split the array into k non-empty continuous subarrays, such that the maximum sum among these k subarrays is minimized.

Return that minimum possible maximum subarray sum.

Input Format:

* + First line: Two integers n and k
  + Second line: n space-separated integers

Output Format:

A single integer — the minimum possible value of the largest subarray sum when the array is split into k parts.

Sample Input 1:

5 2

7 2 5 10 8

Sample Output 1:

18

Sample Input 2:

4 2

1 2 3 4

Sample Output 2:

6

Test Cases Input 1

1 1

100

Output 1

100

Input 2

1. 3

1 4 4 1 3 2

Output 2

5

Input 3

6 4

1 4 4 1 3 22 8 200

Output 3

5

Input 4

10 5

1 1 1 1 1 1 1 1 1 10

Output 4

10

Input 5

5

1 3 20

3 6 30

6 9 40

9 12 50

12 15 60

Output 5

200

Input 6

1. 3

2 3 1 2 4 3 1

Output 6

6

Input 7

5 5

1 2 3 4 5

Output 7

5

Input 8

6 1

1 2 3 4 5 6

Output 8

21

Code

def can\_split(arr, k, max\_allowed): count = 1

current\_sum = 0 for num in arr:

if current\_sum + num > max\_allowed: count += 1

current\_sum = num else:

current\_sum += num return count <= k

def split\_array\_min\_largest\_sum(arr, k): low = max(arr)

high = sum(arr) result = high

while low <= high:

mid = (low + high) // 2 if can\_split(arr, k, mid):

result = mid

high = mid - 1 else:

low = mid + 1

return result

# Input

n, k = map(int, input().split())

arr = list(map(int, input().split())) print(split\_array\_min\_largest\_sum(arr, k))

1. Minimum Cuts for Palindrome Partitioning

Problem Statement:

You are given a string s. Your task is to partition the string such that every substring of the partition is a palindrome.

Return the minimum number of cuts needed to make all parts of the string palindromes.

Input Format:

* + First line: A single string s (lowercase English letters only)

Output Format:

* + A single integer — the minimum number of cuts needed.

Sample Input 1: aab

Sample Output 1:

1

Sample Input 2: a

Sample Output 2:

0

Test Cases Input 1 abc Output 1

2

Input 2 abba Output 2

0

Input 3 racecar Output 3

0

Input 4 noonabbad Output 4

2

Input 5 aaaaaa Output 5

0

Input 6 abacdc Output 6

1

Input 7 Banana Output 7

1

Input 8 Civicracecar Output 8

1

Code

def min\_cut\_palindrome(s): n = len(s)

is\_palindrome = [[False]\*n for \_ in range(n)]

# Precompute palindromes for i in range(n):

is\_palindrome[i][i] = True for i in range(n-1):

is\_palindrome[i][i+1] = (s[i] == s[i+1]) for length in range(3, n+1):

for i in range(n - length + 1):

j = i + length - 1

is\_palindrome[i][j] = (s[i] == s[j]) and is\_palindrome[i+1][j-1]

# DP to calculate min cuts dp = [0] \* n

for i in range(n):

if is\_palindrome[0][i]:

dp[i] = 0 else:

dp[i] = float('inf') for j in range(i):

if is\_palindrome[j+1][i]:

dp[i] = min(dp[i], dp[j] + 1) return dp[-1]

# Input

s = input().strip() print(min\_cut\_palindrome(s))

1. Longest Zigzag Subsequence

Problem Statement:

Given an array of integers, you must find the length of the longest subsequence where the differences between consecutive elements strictly alternate between positive and negative.

A subsequence is not necessarily contiguous. The first difference can be either positive or negative.

Input Format:

* + First line: An integer n — the number of elements in the array.
  + Second line: n space-separated integers.

Output Format:

* + A single integer — the length of the longest zigzag subsequence.

Sample Input 1:

6

1 7 4 9 2 5

Sample Output 1:

6

Sample Input 2:

5

1 4 7 2 5

Sample Output 2:

4

Test Cases Input 1

1

10

Output 1

1

Input 2

3

1 1 1

Output 2

1

Input 3

4

1 2 3 4

Output 3

2

Input 4

4

4 3 2 1

Output 4

2

Input 5

7

70 55 13 2 99 2 80

Output 5

5

Input 6

8

3 3 3 3 3 3 3 3

Output 6

1

Input 7

10

10 22 9 33 49 50 31 60 4 70

Output 7

8

Input 8

5

1 3 2 4 3

Output 8

5

Code

def longest\_zigzag\_subsequence(arr): n = len(arr)

if n == 0: return 0

up = [1] \* n down = [1] \* n

for i in range(1, n):

for j in range(i):

if arr[i] > arr[j]:

up[i] = max(up[i], down[j] + 1) elif arr[i] < arr[j]:

down[i] = max(down[i], up[j] + 1)

return max(max(up), max(down))

# Input

n = int(input())

arr = list(map(int, input().split())) print(longest\_zigzag\_subsequence(arr))

1. Maximum Sum of Non-Overlapping Intervals

Problem Statement:

You're given n intervals, each with a start time, end time, and a value. Your goal is to select a subset of non-overlapping intervals such that the sum of their values is maximized.

Input Format:

* + First line: An integer n — the number of intervals.
  + Next n lines: Each line contains three integers: start end value

Output Format:

A single integer — the maximum sum of values for non-overlapping intervals.

Sample Input 1:

4

1 3 50

3 5 20

1. 19 100

2 100 200

Sample Output 1:

200

Sample Input 2:

3

1 2 10

2 3 20

3 4 30

Sample Output 2:

60

Test Cases Input 1

3

1 4 10

2 5 20

3 6 30

Output 1

30

Input 2

5

1 10 100

2 3 10

3 4 20

4 5 30

5 6 40

Output 2

100

Input 3

2

1 10 100

11 20 200

Output 3

300

Input 4

3

1 5 20

6 10 30

5 6 10

Output 4

60

Input 5

3

1 2 100

2 3 100

3 4 100

Output 5

300

Input 6

1

1 1000000000 5000

Output 6

5000

Input 7

4

1 3 10

2 4 20

3 5 30

4 6 40

Output 7

60

Input 8

5

1 2 1

2 3 2

3 4 3

4 5 4

5 6 5

Output 8

15

Code

import bisect

def max\_value\_non\_overlapping(intervals): intervals.sort(key=lambda x: x[1])

n = len(intervals)

# Prepare end times for binary search ends = [interval[1] for interval in intervals] dp = [0] \* (n + 1)

for i in range(1, n + 1):

start, end, value = intervals[i - 1]

# Find the last interval that doesn't overlap idx = bisect.bisect\_right(ends, start) - 1 dp[i] = max(dp[i - 1], dp[idx + 1] + value)

return dp[n]

# Input

n = int(input())

intervals = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_value\_non\_overlapping(intervals))

1. Count Beautiful Numbers

Problem Statement:

A number is called beautiful if the sum of its digits is divisible by k.

You are given two integers L and R, and a value k. Count how many numbers between L and R (inclusive) are beautiful.

Input Format:

* + A single line with three integers: L R k

Output Format:

A single integer — the count of beautiful numbers in the range [L, R].

Sample Input 1:

1 100 10

Sample Output 1:

9

Sample Input 2:

1 20 5

Sample Output 2:

3

Test Cases Input 1

10 99 9

Output 1

10

Input 2

1 1000 10

Output 2

99

Input 3

100 200 15

Output 3

5

Input 4

123456 654321 7

Output 4

75845

Input 5

1 1000000 1

Output 5

1000000

Input 6

1 1000000 99

Output 6

0

Input 7

1 1 1

Output 7

1

Input 8

9999999999999999 10000000000000000 9

Output 8

1

Code

from functools import lru\_cache def count\_beautiful(n, k):

digits = list(map(int, str(n)))

@lru\_cache(None)

def dp(pos, sum\_mod\_k, tight): if pos == len(digits):

return 1 if sum\_mod\_k == 0 else 0

limit = digits[pos] if tight else 9 total = 0

for d in range(0, limit + 1): total += dp(

pos + 1,

(sum\_mod\_k + d) % k, tight and (d == limit)

)

return total

return dp(0, 0, True)

def solve(L, R, k):

return count\_beautiful(R, k) - count\_beautiful(L - 1, k)

# Input

L, R, k = map(int, input().split()) print(solve(L, R, k))

1. Minimum Incompatibility Partition Problem Statement:

You're given an array of n integers and an integer k. You need to partition the array into k subsets of equal size. Each subset must contain unique elements only (no duplicates). The incompatibility of a subset is the difference between its maximum and minimum elements.

Your goal is to minimize the sum of incompatibilities across all subsets. If it's not possible to partition the array as required, return -1.

Input Format:

* + First line: Two integers n and k
  + Second line: n integers representing the array

Output Format:

* + A single integer — the minimum sum of incompatibilities or -1 if not possible. Sample Input 1:

1. 4

1 2 1 4 3 3 5 6

Sample Output 1:

5

Sample Input 2:

6 3

1 2 3 4 5 6

Sample Output 2:

3

Test Cases Input 1

5 2

1 2 3 4 5

Output 1

-1

Input 2

8 4

1 2 1 4 3 3 5 6

Output 2

5

Input 3

6 2

1 1 1 2 2 2

Output 3

-1

Input 4

4 2

5 5 7 7

Output 4

4

Input 5

7 2

1 2 3 4 5 6 7

Output 5

-1

Input 6

6 3

3 1 6 2 4 5

Output 6

3

Input 7

4 2

8 8 8 8

Output 7

-1

Input 8

8 4

10 20 30 40 50 60 70 80

Output 8

40

Code

from itertools import combinations from collections import defaultdict import sys

def min\_incompatibility(nums, k): from functools import lru\_cache

n = len(nums) size = n // k

# If total elements can't be evenly split into k subsets if n % k != 0:

return -1

count = defaultdict(int) for num in nums:

count[num] += 1

if count[num] > k: # More than k duplicates - cannot split uniquely return -1

all\_masks = [] mask\_cost = {}

# Try all combinations of size `size`

for combo in combinations(range(n), size): seen = set()

subset = []

for i in combo:

if nums[i] in seen: break

seen.add(nums[i]) subset.append(nums[i])

else:

mask = 0

for i in combo:

mask |= 1 << i

mask\_cost[mask] = max(subset) - min(subset) all\_masks.append(mask)

dp = [sys.maxsize] \* (1 << n) dp[0] = 0

for mask in range(1 << n):

if bin(mask).count('1') % size != 0:

continue

for sub in all\_masks:

if (mask & sub) == 0:

next\_mask = mask | sub

dp[next\_mask] = min(dp[next\_mask], dp[mask] + mask\_cost[sub])

result = dp[(1 << n) - 1]

return result if result != sys.maxsize else -1

# Input Reading

if name == " main ":

n, k = map(int, input().split())

nums = list(map(int, input().split())) print(min\_incompatibility(nums, k))

1. Minimum Cost to Make Palindromes Problem Statement:

You are given a string s of lowercase English letters. You want to partition s into k non-empty, non- overlapping contiguous substrings such that each substring is a palindrome. You are allowed to change characters in the string. Changing one character costs 1 unit.

Your goal is to partition the string into exactly k palindromic substrings with the minimum total cost of character changes required.

Return the minimum cost.

Input Format:

* + First line: a string s of length n (1 ≤ n ≤ 100).
  + Second line: integer k (1 ≤ k ≤ n)

Output Format:

* + A single integer, the minimum cost to partition the string into exactly k palindromic substrings.

Sample Input 1: abc

2

Sample Output 1:

1

Sample Input 2: aabbc

3

Sample Output 2:

0

Test Cases Input 1

a 1

Output 1

0

Input 2 abc

1

Output 2

1

Input 3 aaaa

4

Output 3

0

Input 4 aaaaaa 2

Output 4

0

Input 5 abcdef 1

Output 5

3

Input 6 racecar 3

Output 6

0

Input 7 abcdef 2

Output 7

2

Input 8 abcdcba 3

Output 8

0

Code

def min\_change\_to\_palindrome(s, i, j): cost = 0

while i < j:

if s[i] != s[j]:

cost += 1

i += 1

j -= 1

return cost

def min\_cost\_partition(s, k): n = len(s)

# Precompute cost to convert s[i:j+1] to palindrome cost = [[0]\*n for \_ in range(n)]

for i in range(n):

for j in range(i, n):

cost[i][j] = min\_change\_to\_palindrome(s, i, j)

dp = [[float('inf')] \* (k+1) for \_ in range(n+1)] dp[0][0] = 0

for i in range(1, n+1): # i chars taken for t in range(1, k+1): # t parts

for j in range(t-1, i): # try breaking between j and i

dp[i][t] = min(dp[i][t], dp[j][t-1] + cost[j][i-1])

return dp[n][k]

# Input Reading

if name == " main ": s = input().strip()

k = int(input()) print(min\_cost\_partition(s, k))

1. Maximum Sum of Non-Adjacent Equal Substrings Problem Statement:

Given a string s of lowercase English letters, you are allowed to select substrings such that:

* + Each selected substring has all the same characters (e.g., "aaa" is valid, "aab" is not).
  + No two selected substrings overlap or are adjacent.
  + For each selected substring, you gain a score equal to the square of its length.

Your task is to select a subset of such substrings (non-overlapping and non-adjacent) to maximize the total score.

Input Format:

* + A single string s (1 ≤ |s| ≤ 10^5) consisting of lowercase English letters.

Output Format:

* + An integer representing the maximum total score.

Sample Input 1: aaabbaaa Sample Output 1:

18

Sample Input 2:

ababa

Sample Output 2:

3

Test Cases Input 1 aaaaa Output 1

25

Input 2 aabbaabb Output 2

8

Input 3 aabbaa Output 3

8

Input 4 abcde Output 4

3

Input 5 aaabbbcccaaaddd Output 5

27

Input 6 aabaaabaaa

Output 6

22

Input 7 a

Output 7

1

Input 8 aaabbaaccdddeeffggghh Output 8

31

Code

def max\_non\_adjacent\_block\_score(s): n = len(s)

blocks = [] i = 0

# Group into blocks of same characters while i < n:

j = i

while j < n and s[j] == s[i]: j += 1

blocks.append((s[i], j - i)) i = j

m = len(blocks) dp = [0] \* (m + 1)

for i in range(1, m + 1):

char, length = blocks[i - 1]

dp[i] = max(dp[i], dp[i - 1]) # Skip

if i >= 2:

dp[i] = max(dp[i], dp[i - 2] + length \* length) # Take else:

dp[i] = max(dp[i], length \* length)

return dp[m]

# Input Reading

if name == " main ": s = input().strip()

print(max\_non\_adjacent\_block\_score(s))

1. Longest Balanced Binary Substring After K Flips Problem Statement:

You are given a binary string s (consisting of only '0' and '1') and an integer k.

You are allowed to flip at most k characters in the string (i.e., change '0' to '1' or '1' to '0').

Your task is to determine the length of the longest balanced substring you can obtain after at most k flips, where a balanced substring is defined as a substring with an equal number of 0s and 1s.

Input Format:

* + First line: a string s of length n (1 ≤ n ≤ 100).
  + Second line: integer k (1 ≤ k ≤ n)

Output Format:

A single integer – length of longest balanced substring after at most k flips

Sample Input 1:

110001

1

Sample Output 1:

6

Sample Input 2:

1111

2

Sample Output 2:

0

Test Cases Input 1

1111

0

Output 1

0

Input 2

000111

1

Output 2

4

Input 3

00000

3

Output 3

0

Input 4

101010

0

Output 4

0

Input 5

101100110101

2

Output 5

10

Input 6

1

1 1000

Output 6

1000

Input 7

1000000001

1

Output 7

2

Input 8

111000

0

Output 8

0

Code

def longest\_balanced\_after\_k\_flips(s: str, k: int) -> int: n = len(s)

max\_len = 0

count = {0: 0, 1: 0}

left = 0

for right in range(n):

count[int(s[right])] += 1

# Flip the minority if needed to make counts equal while abs(count[0] - count[1]) > 2 \* k:

count[int(s[left])] -= 1

left += 1

total = right - left + 1

max\_balanced = total - abs(count[0] - count[1]) max\_len = max(max\_len, max\_balanced)

return max\_len

# Input

if name == " main ": s = input().strip()

k = int(input()) print(longest\_balanced\_after\_k\_flips(s, k))

1. Minimum Jumps to Make Array Strictly Increasing Problem Statement:

You are given an array of integers arr of size n. You can jump from index i to any index j such that:

* + j > i
  + arr[j] > arr[i]

You need to find the minimum number of jumps required to reach the end of the array starting from index 0 (i.e., arr[0] to arr[n-1]), such that at each step the next value is strictly greater than the current.

If it's not possible to reach the end, print -1.

Input Format:

* + First line: An integer N
  + Next line: N integers Output Format:

A single integer – Minimum number of jumps or -1

Sample Input 1:

6

1 3 2 4 6 8

Sample Output 1:

1

Sample Input 2:

5

5 4 3 2 1

Sample Output 2:

-1

Test Cases Input 1

4

1 2 3 4

Output 1

1

Input 2

1

10

Output 2

0

Input 3

7

1 100 1 101 2 102 103

Output 3

1

Input 4

8

10 20 10 30 10 40 10 50

Output 4

1

Input 5

5

1 1 1 1 2

Output 5

1

Input 6

10

5 6 1 2 3 4 5 6 7 8

Output 6

1

Input 7

5

1 3 5 7 9

Output 7

1

Input 8

6

1 2 1 2 1 2

Output 8

1

Code

from collections import deque, defaultdict

def min\_jumps\_strictly\_increasing(arr): n = len(arr)

if n == 1:

return 0

graph = defaultdict(list)

# Preprocess: Build graph of possible jumps for i in range(n):

for j in range(i + 1, n):

if arr[j] > arr[i]: graph[i].append(j)

# BFS

visited = [False] \* n queue = deque()

queue.append((0, 0)) # (index, jump count) visited[0] = True

while queue:

idx, jumps = queue.popleft() for neighbor in graph[idx]:

if not visited[neighbor]: if neighbor == n - 1:

return jumps + 1

visited[neighbor] = True queue.append((neighbor, jumps + 1))

return -1

# Driver Code

if name == " main ": n = int(input())

arr = list(map(int, input().split())) print(min\_jumps\_strictly\_increasing(arr))

Q1.

Problem Statement

Determine whether a given undirected graph contains the Hamiltonian cycle or not.

A Hamiltonian cycle in an undirected graph is a path that visits each vertex exactly once. A Hamiltonian cycle is a Hamiltonian path with an edge from the last vertex to the first vertex of the Hamiltonian path.

Example

Input:

5

0 1 0 1 0

1 0 1 1 1

0 1 0 0 1

1 1 0 0 1

0 1 1 1 0

Output:

0 1 2 4 3 0

Explanation:

Starting from vertex 0, all the vertices can be visited exactly once (0 - 1 - 2 - 4 - 3 - 0). Input Format

The first line of input consists of an integer that represents the number of vertices (V).

The following lines of input consist of the V\*V matrix, which represents the adjacency matrix of the graph.

Output Format

The output prints If a Hamiltonian cycle exists, print the vertices in the cycle in order, starting and ending at vertex 0.; otherwise, it prints that the "Hamiltonian path does not exist".

Refer to the sample output for the formatting specifications. Constraints

3 ≤ V ≤ 5

The adjacency matrix elements graph[i][j] will be either 0 or 1. Sample Input Sample Output

5

1 0 1 1 1

0 1 0 0 1

1 1 0 0 1

0 1 1 1 0

0 1 2 4 3 0

Sample Input Sample Output 5

0 1 0 1 0

1 0 1 1 1

0 1 0 0 1

1 1 0 0 0

0 1 1 0 0

Hamiltonian path does not exist

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q2.

Problem Statement:

Given an array of integers, find the inversion count in the array.

Inversion Count: For an array, the inversion count indicates how far (or close) the array is from being sorted. If the array is already sorted, then the inversion count is 0. If an array is sorted in reverse order, then the inversion count is the maximum.

Formally, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j.

Example 1:

Input:

N = 5, arr[] = {2, 4, 1, 3, 5}

Output:

3

Explanation:

Sequence 2, 4, 1, 3, and 5 has three inversions (2, 1), (4, 1), and (4, 3).

Example 2: Input:

N = 5

arr[] = {2, 3, 4, 5, 6}

Output:

0

Explanation:

As the sequence is already sorted, there is no inversion count.

Example 3:

Input:

N = 3, arr[] = {10, 10, 10}

Output:

0

Explanation:

As all the elements of the array are the same, there is no inversion count. Input Format

The first line of the input is the size of the array.

The second line of the input is the list of space-separated integer values. Output Format

The output is the inversion count in the array. Constraints

1 ≤ N ≤ 5\*105 1 ≤ arr[i] ≤ 1018 Sample Input Sample Output 5

2 4 1 3 5

3

Sample Input Sample Output 5

2 3 4 5 6

0

Sample Input Sample Output 3

10 10 10

0

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q3.

Problem Statement

Given two strings s and t of lengths m and n respectively, return the minimum window substring of s such that every character in t (including duplicates) is included in the window. If there is no such substring, return an empty string.

Examples

Input: s = ADOBECODEBANC, t = ABC

Output: BANC

Explanation: The minimum window substring BANC includes 'A', 'B', and 'C' from string t.

Input: s = a, t = a Output: a

Explanation: The entire string s is the minimum window.

Input: s = a, t = aa Output:

Explanation: Both 'a's from t must be included in the window. Since the largest window of s only has one 'a', return an empty string.

Input Format

The first line contains the string s, which is the source string.

The second line contains the string t, which is the target string whose characters must be included in the window.

Output Format

The output is a single line containing the minimum window substring of s containing all t characters. If no such window exists, output an empty string.

Refer to the sample output for formatting specifications.

Constraints

m == s.length

n == t.length

1 ≤ m, n ≤ 105

s and t consist of uppercase and lowercase English letters.

Sample Input Sample Output ADOBECODEBANC ABC

BANC

Sample Input Sample Output a a a

Sample Input Sample Output a aa

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q4

Problem Statement

The set [1, 2, 3, ..., n] contains a total of n! unique permutations.

By listing and labeling all of the permutations in order, we get the following sequence for n = 3: ["123", "132", "213", "231", "312", "321"]

Given n and k, return the kth permutation sequence. Input Format

The first line of input consists of an integer value 'n', representing the number of digits in the permutation.

The second line of input consists of an integer value 'k', representing the position of the permutation to be found.

Output Format

The output displays the kth permutation of the first n positive integers. Refer to the sample output for the formatting specifications. Constraints

In this scenario, the given test cases will fall under the following constraints:

1 ≤ n ≤ 9 1 ≤ k ≤ n!

Sample Input Sample Output 3

3

213

Sample Input Sample Output 4

9

2314

Sample Input Sample Output 3

1

123

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q5.

Problem Statement

Given an array representing the parent-child relationship in a binary tree, find the tree's height without building it.

The parent-child relationship is defined by (A[i], i) for every index i in the array. The root node's value will be i if -1 is present at index i in the array.

The depth of a node is the total number of edges from the node to the tree's root node. The root is the only node whose depth is 0.

The height of a node is the total number of edges on the longest path from the node of a leaf. The height of a tree would be the height of its root node or equivalently the depth of its deepest node. A leaf node will have a height of 0.

Example

Parent: [-1, 0, 0, 1, 2, 2, 4, 4]

Index: [0, 1, 2, 3, 4, 5, 6, 7]

-1 is present at index 0, which implies that the binary tree root is node 0.

1. is present at index 1 and 2, which implies that the left and right children of node 0 are 1 and 2.
2. is present at index 3, which implies that the left or the right child of node 1 is 3.
3. is present at index 4 and 5, which implies that the left and right children of node 2 are 4 and 5.

4 is present at index 6 and 7, which implies that the left and right children of node 4 are 6 and 7.

The corresponding binary tree is:

Output: The height of the binary tree is 3. Input Format

The first line contains a single integer N, which represents the number of nodes in the tree. The next N lines consist of an integer in each line, and the last line of input consists of the ith integer representing the parent of the ith node.

If the ith node is the root node, then the corresponding value will be -1. Output Format

The output consists of a single integer, which represents the height of the tree. Constraints

1 ≤ N ≤ 10

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q6.

Problem Statement

Roshan invested all day in developing test data for DC. He couldn’t make it work, so he had a nervous breakdown and can’t even see clearly anymore. Every time he winks(blinks) while reading, the letters in a word get mingled up so that the letters from the second half of the word (the shorter half, if the length is an odd number) “jump in” between the letters from the first half in the following way:

The final letter “jumps in” the middle of the 1st and the 2nd letter

The second final letter “jumps in” the middle of the 2nd and the 3rd letter

The ith letter from the end “jumps in” the middle of the ith and the (i+1)th letter from the starting

For this case, the word “abcdef” would become “afbecd” after winking.

If Roshan winks again, the same thing happens. After three winks, the word becomes

“adfcbe”.

Roshan has decided to write a program to help him figure out what’s correctly written on the screen. Unfortunately, after a day’s work, he’s too tired, and he needs your help. You are given N, the number of winks, and the word Roshan sees on the screen. Write a program to solve the mystery for Roshan and figure out what was actually the word before he winked N times.

Input Format

The first line of input contains a positive integer N, the number of times Roshan blinked. The second line of input contains the word W from the screen.

The word will consist only of small letters of the English alphabet. Output Format

The output displays the original word before Roshan blinked N times.

Refer to the sample output for formatting specifications. Constraints

1 ≤ N ≤ 109

1. ≤ length(W) ≤ 103

Sample Input Sample Output

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q7.

Problem Statement

You are given two strings order and s. All the characters of order are unique and were sorted in some custom order previously.

Permute the characters of s so that they match the order that order was sorted. More specifically, if a character x occurs before a character y in order, then x should occur before y in the permuted string.

Return any permutation of s that satisfies this property. Example 1:

Input: order = "cba", s = "abcd" Output: "cbad"

Explanation: "a", "b", "c" appear in order, so the order of "a", "b", "c" should be "c", "b", and "a".

Since "d" does not appear in order, it can be at any position in the returned string. "dcba", "cdba", "cbda" are also valid outputs.

Example 2:

Input: order = "bcafg", s = "abcd" Output: "bcad"

Explanation: The characters "b", "c", and "a" from order dictate the order for the characters in s. The character "d" in s does not appear in order, so its position is flexible.

Following the order of appearance in order, "b", "c", and "a" from s should be arranged as "b", "c", "a". "d" can be placed at any position since it's not in order. The output "bcad" correctly follows this rule. Other arrangements like "bacd" or "bcda" would also be valid, as long as "b", "c", "a" maintain their order.

Input Format

The first line of input consists of the string order, representing the custom order of characters. The second line of input consists of the string s, representing the input string.

Output Format

The output prints the sorted string according to the custom order specified by order.

Refer to the sample output for the formatting specifications. Constraints

1 ≤ order.length ≤ 26

1 ≤ s.length ≤ 200

order and s consist of lowercase English letters. All the characters of order are unique. Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q8.

Problem Statement:

Given an integer array nums, find the subarray with the largest sum, and return its sum.

Example 1:

Input: n=9

nums = [-2,1,-3,4,-1,2,1,-5,4]

Output: 6

Explanation: The subarray [4,-1,2,1] has the largest sum 6.

Example 2:

Input: n=1 nums = [1] Output: 1

Explanation: The subarray [1] has the largest sum of 1.

Example 3:

Input: n=5

nums = [5,4,-1,7,8]

Output: 23

Explanation: The subarray [5,4,-1,7,8] has the largest sum of 23. Input Format

The first line consists of an integer n, representing the number of elements in the array nums. The second line consists of n space-separated integers, representing the elements of the array nums.

Output Format

The output prints an integer, representing the sum of the subarray with the largest sum.

Refer to the sample output for formatting specifications. Constraints

1 ≤ n ≤ 20

-100 ≤ nums[i] ≤ 100

Sample Input Sample Output

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q9.

Problem Statement

Given an array, the task is to find the sum of the maximum sum by alternating subsequences starting with the first element. Here alternating sequence means first decreasing, then increasing, then decreasing,... For example, 10, 5, 14, and 3 are alternating sequences.

Note: The reverse sequence type (increasing—decreasing—increasing—...) is not considered alternating here.

Example

Input:

6

1. 3 8 5 3 8

Output:

28

Explanation:

The alternating subsequence (starting with the first element) that has the maximum sum is

{4, 3, 8, 5, 8}

Input Format

The first line contains an integer n, representing the number of elements in the array.

The second line contains n space-separated integers, representing the elements of the array.

Output Format

The output displays a single integer representing the maximum sum of an alternating subsequence from the given array.

Refer to the sample output for the formatting specifications. Constraints

1 ≤ n ≤ 10

1. ≤ elements ≤ 100

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q10.

Problem Statement

We can rotate digits by 180 degrees to form new digits. When 0, 1, 6, 8, 9 are rotated 180

degrees, they become 0, 1, 9, 8, 6 respectively. When 2, 3, 4, 5 and 7 are rotated 180 degrees, they become invalid.

A confusing number is a number that when rotated 180 degrees becomes a different number with each digit valid. (Note that the rotated number can be greater than the original number)

Given a positive integer N, return the number of confusing numbers between 1 and N inclusive.

Example

Input: 16

Output: 4

Explanation:

The confusing numbers are {6, 9, 10, 16}.

6 converts to 9.

9 converts to 6.

10 converts to 01 which is just 1.

16 converts to 91. Input Format

The input consists of a positive integer N. Output Format

The output prints an integer, representing the number of confusing numbers between 1 and N inclusive.

Refer to the sample output for formatting specifications.

Constraints

1 ≤ N ≤ 103

Sample Input Sample Output

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q11.

Problem Statement

Given an array of string words and a width maxWidth, format the text such that each line has exactly maxWidth characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly maxWidth characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line does not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

The last line of text should be left justified, and no extra space should be inserted between words.

Note:

A word is defined as a character sequence consisting of non-space characters only. Each word's length is guaranteed to be greater than 0 and not exceed maxWidth.

The input array words contain at least one word. Input Format

The first line of input consists of an integer n, representing the number of words in the sentence.

The following n lines consist of n strings.

The last line consists of an integer representing the maximum width. Output Format

The output prints strings, representing the formatted text.

Refer to the sample output for formatting specifications. Constraints

1 ≤ words.length ≤ 300

1 ≤ words[i].length ≤ 20

words[i] consists of only English letters and symbols.

1 ≤ maxWidth ≤ 100 words[i].length ≤ maxWidth Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q12.

Problem Statement

Sarah is working on a traffic management system for a city's road network. She needs to find the shortest cycle in the road network, where each intersection is represented as a vertex and each road between intersections is a directed edge with a specific travel time (weight). The goal is to identify the shortest cycle in the road network, where a cycle is defined as a path that starts and ends at the same intersection, and the length of the cycle is the total travel time of all roads in the cycle.

Can you assist Sarah by writing a program to find the length of the shortest cycle in the city's road network?

Example1 Input:

4

4

0 1 2

1 2 3

2 0 4

3 1 5

Output:

Length of the shortest cycle: 9

Explanation:

After exploring all starting vertices, the code checks for cycles by examining the distances. If an edge leads back to the starting vertex, it means there is a cycle. In this case, edge 1 (vertex 1) leads back to the starting vertex 0. The code calculates the length of the cycle by adding the distances of the vertices along the cycle: 2 (0 -> 1) + 3 (1 -> 2) + 4 (2 -> 0) = 9.

Therefore, the output is "Length of the shortest cycle: 9".

Example2 Input:

3

2

0 1 2

1 2 3

Output:

No cycle was found in the graph.

Explanation:

After exploring all starting vertices, the code checks for cycles by examining the distances. If an edge leads back to the starting vertex, it means there is a cycle. However, in this case, there are no edges that lead back to the starting vertex. Hence, there is no cycle in the graph.

Therefore, the output is "No cycle found in the graph." Input Format

The first line contains an integer v, representing the number of intersections (vertices) in the city's road network.

The second line contains an integer e, representing the number of roads (edges) in the city. The next e lines each contain three integers: source, destination, and weight, representing a directed road from intersection source to intersection destination with a travel time (weight). Output Format

If a cycle exists, the output prints "Length of the shortest cycle: X", where X is the length of the shortest cycle.

If no cycle exists, then the output prints "No cycle found in the graph."

Refer to the sample output for the formatting specifications. Constraints

In this scenario, the given test cases will fall under the following constraints:

1 ≤ v ≤ 5

1 ≤ edges ≤ 7

0 ≤ S, D < V

1 ≤ W ≤ 5

Weights are positive integers. Sample Input Sample Output

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q13.

Problem Statement

Mr.Butler is managing his class of n students. He placed all his students in a line and gave the kth student from the left a card with the letter ak written on it. He would now like to rearrange the students so that the kth student from the left has a card with the letter bk written on it.

To do this, he will choose some consecutive groups of students, and reverse their order. Students will hold on to their original cards during this process.

He’s now wondering, what is the number of valid ways to do this? (It may be impossible, in

which case, the answer is zero).

With sequences abba and aabb, Mr.Butler can choose group a(bba). With sequences caxcab and cacxab, Mr.Butler can choose ca(xc)ab or c(axca)b. With sequences a and z, there are no solutions.

Input Format

The input is two lines of lowercase letters, A and B.

The kth character of A and B represent ak and bk respectively.

It is guaranteed that A and B have the same positive length, and A and B are not identical. Output Format

The output displays the number of ways Mr.Butler can reverse some consecutive groups of A to form the line specified by string B.

Refer to the sample input and output for formatting specifications. Constraints 1 <= length(A) <= 105

1 <= length(B) <=105

Sample Input Sample Output

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q14.

Problem Statement

Given an input stream of N integers. The task is to insert these numbers into a new stream and find the median of the stream formed by each insertion of X to the new stream.

Example 1 Input

N = 4

X[] = 5,15,1,3

Output 5

10

5

4

Explanation:

Flow in stream: 5, 15, 1, 3

5 goes to stream --> median 5 (5)

15 goes to stream --> median 10 (5,15)

1 goes to stream --> median 5 (5,15,1)

3 goes to stream --> median 4 (5,15,1 3)

Example 2 Input

N = 3

X[] = 5,10,15

Output 5

7.5

10

Explanation:

Flow in stream: 5, 10, 15

5 goes to stream --> median 5 (5)

10 goes to stream --> median 7.5 (5,10)

15 goes to stream --> median 10 (5,10,15) Input Format

The first line contains an integer n, the number of integers in the stream.

The second line contains n space-separated integers representing the elements of the stream. Output Format

The output prints the median after each insertion of an integer into the stream. Print each median on a new line.

Refer to the sample output for the formatting specifications. Constraints

1 ≤ N ≤ 15

1 ≤ Elements ≤ 100

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q15.

Problem Statement

Given a string s and a dictionary of string wordDict of size n, add spaces in s to construct a sentence where each word is a valid dictionary word. Return all such possible sentences in any order.

Note: The same word in the dictionary may be reused multiple times in the segmentation. & All the strings of wordDict are unique.

Examples Input:

s = "catsanddog" n = 5

wordDict = ["cat","cats","and","sand","dog"]

Output:

["cats and dog","cat sand dog"]

Input:

s = "pineapplepenapple" n = 5

wordDict = ["apple","pen","applepen","pine","pineapple"]

Output:

["pine apple pen apple","pineapple pen apple","pine applepen apple"]

Input:

s = "catsandog" n = 5

wordDict = ["cats","dog","sand","and","cat"]

Output: [ ]

Input Format

The first line of input is a string 's' consisting of lowercase English letters. The second line of input is an integer 'n' represents the size of the dictionary. The next n lines consists of strings denoting the words in the dictionary.

Output Format

The output displays a list of strings, where each string is a space-separated sequence of dictionary words that form the input string s.

If no such sequence exists, an empty list is returned.

Refer to the sample input and output for formatting specifications. Constraints 1 <= n <= 10

The string s contains both uppercase and lowercase. Sample Input Sample Output

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q16.

Problem Statement

Given an m x n integers matrix, return the length of the longest increasing path in matrix.

From each cell, you can either move in four directions: left, right, up, or down. You may not move diagonally or move outside the boundary (i.e., wrap-around is not allowed).

Example 1:

Input: matrix = [[9,9,4],[6,6,8],[2,1,1]]

Output: 4

Explanation: The longest increasing path is [1, 2, 6, 9].

Example 2:

Input: matrix = [[3,4,5],[3,2,6],[2,2,1]]

Output: 4

Explanation: The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not allowed.

Input: matrix = [[1]] Output: 1 Input Format

The first line of input consists of an integer r, representing the number of rows.

The second line of input consists of an integer c, representing the number of columns. The next r lines contain c integers each, representing the elements of the matrix.

Output Format

The output prints a single integer, representing the length of the longest increasing path in the matrix

Refer to the sample output for the formatting specifications. Constraints

1 ≤ r ≤ 4

1 ≤ c ≤ 4

1 ≤ elements of the matrix ≤ 20

Sample Input Sample Output

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q17.

Problem Statement

Tom is a software developer working on a project where he has to check if a doubly linked list is a palindrome. He needs to write a program to solve this problem. Write a program to help Tom check if a given doubly linked list is a palindrome or not.

Input Format

The first line consists of an integer N, representing the number of elements in the linked list. The second line consists of N space-separated integers representing the linked list elements. Output Format

The first line of output displays the elements of the doubly linked list in forward order, separated by spaces.

The second line of output displays the elements of the doubly linked list in reverse order, separated by spaces.

The third line prints:

"The doubly linked list is a palindrome" if the list reads the same forward and backward.

"The doubly linked list is not a palindrome" if the list does not match when reversed.

Refer to the sample output for the formatting specifications. Constraints In this scenario, the test cases fall under the following constraints:

2 ≤ N ≤ 20

-100 ≤ elements ≤ 100

Sample Input Sample Output

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 |  | | | |
| 1 | 2 | 3 | 2 | 1 |
| 1 | 2 | 3 | 2 | 1 |
| 1 | 2 | 3 | 2 | 1 |

Sample Input Sample Output

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 |  |  |  |  |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |
| 5 | 4 | 3 | 2 | 1 |

Sample Input Sample Output

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6 |  | | | | |
| -1 | -2 | -3 | -3 | -2 | -1 |
| -1 | -2 | -3 | -3 | -2 | -1 |
| -1 | -2 | -3 | -3 | -2 | -1 |

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q18.

Problem Statement

Meet Alex, a computer science student who loves solving programming challenges.

Today, Alex is tasked with creating a program to simulate a stack data structure using linked lists.

The objective is to provide a menu-driven interface for performing essential stack operations. Push elements onto the stack

Pop elements from the stack Display the current stack contents

Additionally, Alex needs to ensure that the program handles stack underflow conditions. Help him to accomplish the task.

Input Format

The input consists of integers corresponding to the operation that needs to be performed: Choice 1: If the choice is 1, the following integer input represents the element to be pushed onto the stack.

Choice 2: Pop the integer from the stack. Choice 3: Display the elements in the stack. Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the stack:

If the choice is 1, push the given integer to the stack and display the following: "[value] is pushed onto the stack"

If the choice is 2, pop the integer from the stack and display the following: "[value] is popped from the stack"

If the choice is 2, and if the stack is empty without any elements, print "Stack Underflow" If the choice is 3, print the elements in the stack in the format: "Elements in the stack: ", followed by the space-separated integer values.

If the choice is 3, and there are no elements in the stack, print "Stack is empty"

If the choice is 4, exit the program and display the following: "Exiting the program" If any other choice is entered, print "Invalid choice"

Refer to the sample output for the formatting specifications. Constraints Choice: 1, 2, 3, or 4

Sample Input Sample Output

Sample Input Sample Output

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q19.

Problem Statement:

You need to build a road in a rugged terrain. You know the sea level of each segment of the rugged terrain, i.e. the i-th segment is Li meters from sea level.

You needs to transform the terrain into a strictly downward sloping terrain for the road, i.e, for each i-th segment where 2 <= i <= N, resultant Li-1 > Li. In order to do so, you employ a powerful digging team to help you dig and reduce the sea level of the segments. On day D, the team can reduce the sea level for each segments that you scheduled that day by 2D-1 meters each.

You are allowed to assign the team to dig on multiple segments and/or dig on the same segments for multiple days.

Your task is to find the minimum number of days needed to transform the terrain as per your requirements

Example:

Input:

4

-1

1

1

1

Output:

3

Explanation:

One of the possible way:

On day 1, we can dig on 1st and 4th segment, resulting in {-2, 1, 1, 0} On day 2, we can dig on 3rd and 4th segments, resulting in {-2, 1, -1, -2}

On day 3, we can dig on 2nd, 3rd and 4th segments, resulting in {-2, -3, -5, -6} Input Format

The first line contains an integer, N, denoting the number of elements in L.

Each line i of the N subsequent lines (where 0 < i ≤ N) contains an integer describing Li, the

sea level of the i-th segment. Output Format

The program should print a single integer representing the minimum number of days required to transform the terrain into a strictly decreasing sequence.

Refer to the sample output for formatting specifications. Constraints 1 < N < 105

1 < L[i] < 109

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q20.

Problem Statement

Raju is a computer graphics designer. He works around matrices with ease. His friend challenged him to rotate an M x M matrix clockwise with an angle of 90 degrees.

However, Raju was given a constraint that he must not use any additional 2D matrix for doing the above-mentioned task. The rotation must be done in place, which means Raju has to modify the given input matrix directly.

Example: 1

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

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

Example: 2

Input: matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]

Output: [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]

Input Format

The first line of the input is a single integer value for the size of the row and column, M. The next M lines of the input are the matrix elements, each containing M integers representing the elements of the matrix.

Output Format

The output displays the rotated matrix, with each row printed on a new line.

Refer to the sample input and output for formatting specifications.

Constraints

n == matrix.length == matrix[i].length

1 ≤ N ≤ 6

1 ≤ matrix[i][j] ≤ 1000

Sample Input Sample Output

Sample Input Sample Output

|  |  |
| --- | --- |
| 13 | 3 6 7 |
| 15 | 14 12 16 |
| 15 | 13 2 5 |
| 14 | 3 4 1 |
| 12 | 6 8 9 |

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q21.

Problem Statement

Imagine you're helping a friend plan gift packages with a total budget of N. Your task is to find all unique ways to spend the entire budget on different combinations of gift prices, ensuring no combination is repeated by rearranging the prices.

Note:

By unique it means that no matter that no other composition can be expressed as a permutation of the generated composition. For eg. [1,2,1] and [1, 1, 2] are not unique.

You need to print all combinations in non-decreasing order for eg. [1, 2, 1] or [1,1,2] will be printed as [1,1,2].

The order of printing all the combinations should be printed in lexicographical order.

Use backtracking to solve the given problem. Input Format

The input consists of a single integer, n, which represents the number to be broken down. Output Format

The output consists of a list of lists, where each inner list represents a possible breakdown of the given number.

Each breakdown is represented as a list of integers that add up to the given number. The breakdowns are printed in lexicographic order.

Refer to the sample output for formatting specifications. Constraints

In this scenario, the given test cases will fall under the following constraints:

1 ≤ N ≤ 25

Sample Input Sample Output

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q22.

Problem Statement

You are preparing for a technical interview with a well-known tech company. During a mock interview, you are presented with a coding challenge related to Binary Search Trees (BSTs).

The challenge is to write a program that finds the kth largest element in a BST, and you are required to implement an efficient solution.

Your task is to complete the code and ensure that it correctly identifies the kth largest element in the given input tree.

Input Format

The first line of input consists of a sequence of integers representing the elements of the BST. The input is terminated by -1.

The second line consists of an integer k, representing the position of the desired largest element. Output Format

The output prints a single integer, which is the kth largest element in the BST.

Refer to the sample output for the formatting specifications. Constraints

1 ≤ Number of nodes in the BST ≤ 50 1 ≤ Values of nodes in the BST ≤ 150 1 ≤ k ≤ Number of nodes in

the BST Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q23.

Problem Statement

Sanjay is learning about queues in his data structures class and he wants to write a program that counts the number of distinct elements in the queue. Your task is to guide him in this.

Example

Input:

7

45 33 29 45 45 46 39

Output:

5

Explanation:

The distinct elements in the queue are 45, 33, 29, 46, 39. So, the count is 5. Input Format

The first line of input consists of an integer M, representing the number of elements in the queue. The second line consists of M space-separated integers, representing the queue elements.

Output Format

The output prints an integer, representing the number of distinct elements in the queue.

Refer to the sample output for formatting specifications. Constraints The maximum size of the queue is 100.

1 ≤ M ≤ 20

1 ≤ queue elements ≤ 1000

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q24.

Problem Statement

Imagine you are part of a legal team responsible for reviewing contracts and legal documents. To streamline the review process, you need a tool that can quickly check if specific legal terms or clauses are included in a large text document.

Write a program that uses a linear search algorithm to search a given document (a sentence) for the presence of a specific legal term or clause.

Input Format

The first line of input consists of a string S, the document (a single sentence) containing up to 1000 characters.

The second line consists of a string T, representing the legal term or clause to search for, which can be up to 100 characters long.

Output Format

If the legal term or clause is found in the document, print "The word '[T]' is present in the given sentence."

If the legal term or clause is not found, print "The word '[T]' is not present in the given sentence."

Refer to the sample output for formatting specifications. Constraints The input sentence is case-sensitive.

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q25.

Problem Statement

Imagine you are organizing a sorting contest for programmers. Participants are given an array of integers to sort using the insertion sort algorithm with a twist.

Instead of simply sorting the array, participants are required to keep track of the number of swaps they make during the sorting process.

Your task is to implement a program that takes in the participant's array and calculates the minimum number of swaps they need to perform to sort the array using the selection sort algorithm.

Example 1

Input:

4

4 3 2 1

Output:

2

Explanation:

Swap 4 with 1 and swap 3 with 2 to form the sorted array {1, 2, 3, 4}.

Example 2

Input:

5

10 19 6 3 5

Output:

2

Explanation:

Swap 10 with 3 and swap 19 with 5 to form the sorted array {3, 5, 6, 10, 19}. Input Format

The first line of input consists of an integer n, representing the number of elements in the array. The second line consists of n space-separated integers, denoting the elements of the array.

Output Format

The output prints an integer representing the minimum number of swaps required to sort the array in strictly increasing order.

Refer to the sample output for formatting specifications. Constraints

1 ≤ n ≤ 10

0 ≤ elements ≤ 100

Sample Input Sample Output

Sample Input Sample Output

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q26

Problem Statement

Determine whether a given undirected graph contains the Hamiltonian cycle or not.

A Hamiltonian cycle in an undirected graph is a path that visits each vertex exactly once. A Hamiltonian cycle is a Hamiltonian path with an edge from the last vertex to the first vertex of the Hamiltonian path.

Example Input:

5

0 1 0 1 0

1 0 1 1 1

0 1 0 0 1

1 1 0 0 1

0 1 1 1 0

Output:

0 1 2 4 3 0

Explanation:

Starting from vertex 0, all the vertices can be visited exactly once (0 - 1 - 2 - 4 - 3 - 0). Input Format

The first line of input consists of an integer that represents the number of vertices (V).

The following lines of input consist of the V\*V matrix, which represents the adjacency matrix of the graph.

Output Format

The output prints If a Hamiltonian cycle exists, print the vertices in the cycle in order, starting and ending at vertex 0.; otherwise, it prints that the "Hamiltonian path does not exist".

Refer to the sample output for the formatting specifications.

Constraints

3 ≤ V ≤ 5

The adjacency matrix elements graph[i][j] will be either 0 or 1.

Q27.

Problem Statement:

Given an array of integers, find the inversion count in the array.

Inversion Count: For an array, the inversion count indicates how far (or close) the array is from being sorted. If the array is already sorted, then the inversion count is 0. If an array is sorted in reverse order, then the inversion count is the maximum.

Formally, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j. Example 1:

Input:

N = 5, arr[] = {2, 4, 1, 3, 5}

Output:

3

Explanation:

Sequence 2, 4, 1, 3, and 5 has three inversions (2, 1), (4, 1), and (4, 3).

Example 2:

Input:

N = 5

arr[] = {2, 3, 4, 5, 6}

Output:

0

Explanation:

As the sequence is already sorted, there is no inversion count.

Example 3:

Input:

N = 3, arr[] = {10, 10, 10}

Output:

0

Explanation:

As all the elements of the array are the same, there is no inversion count. Input Format

The first line of the input is the size of the array.

The second line of the input is the list of space-separated integer values. Output Format

The output is the inversion count in the array. Constraints

1 ≤ N ≤ 5\*105 1 ≤ arr[i] ≤ 1018

Q28.

Problem Statement

Given two strings s and t of lengths m and n respectively, return the minimum window substring of s such that every character in t (including duplicates) is included in the window. If there is no such substring, return an empty string.

Examples

Input: s = ADOBECODEBANC, t = ABC

Output: BANC

Explanation: The minimum window substring BANC includes 'A', 'B', and 'C' from string t.

Input: s = a, t = a Output: a

Explanation: The entire string s is the minimum window.

Input: s = a, t = aa Output:

Explanation: Both 'a's from t must be included in the window. Since the largest window of s only has one 'a', return an empty string.

Input Format

The first line contains the string s, which is the source string.

The second line contains the string t, which is the target string whose characters must be included in the window.

Output Format

The output is a single line containing the minimum window substring of s containing all t characters. If no such window exists, output an empty string.

Refer to the sample output for formatting specifications.

Constraints

m == s.length

n == t.length

1 ≤ m, n ≤ 105

s and t consist of uppercase and lowercase English letters.

Q2G.

Problem Statement

The set [1, 2, 3, ..., n] contains a total of n! unique permutations.

By listing and labeling all of the permutations in order, we get the following sequence for n = 3: ["123", "132", "213", "231", "312", "321"]

Given n and k, return the kth permutation sequence. Input Format

The first line of input consists of an integer value 'n', representing the number of digits in the permutation.

The second line of input consists of an integer value 'k', representing the position of the permutation to be found.

Output Format

The output displays the kth permutation of the first n positive integers. Refer to the sample output for the formatting specifications. Constraints

In this scenario, the given test cases will fall under the following constraints:

1 ≤ n ≤ 9 1 ≤ k ≤ n! Q30.

Problem Statement

Roshan invested all day in developing test data for DC. He couldn’t make it work, so he had a nervous breakdown and can’t even see clearly anymore. Every time he winks(blinks) while reading, the letters in a word get mingled up so that the letters from the second half of the word (the shorter half, if the length is an odd number) “jump in” between the letters from the first half in the following way:

* The final letter “jumps in” the middle of the 1st and the 2nd letter
* The second final letter “jumps in” the middle of the 2nd and the 3rd letter
* The ith letter from the end “jumps in” the middle of the ith and the (i+1)th letter from the

starting

For this case, the word “abcdef” would become “afbecd” after winking.

If Roshan winks again, the same thing happens. After three winks, the word becomes “adfcbe”.

Roshan has decided to write a program to help him figure out what’s correctly written on the screen. Unfortunately, after a day’s work, he’s too tired, and he needs your help. You are given N, the number of winks, and the word Roshan sees on the screen. Write a program to solve the mystery for Roshan and figure out what was actually the word before he winked N times.

Input Format

The first line of input contains a positive integer N, the number of times Roshan blinked. The second line of input contains the word W from the screen.

The word will consist only of small letters of the English alphabet. Output Format

The output displays the original word before Roshan blinked N times. Refer to the sample output for formatting specifications. Constraints 1 ≤ N ≤ 109

3 ≤ length(W) ≤ 103

Q31.

Problem Statement

You are given two strings order and s. All the characters of order are unique and were sorted in some custom order previously.

Permute the characters of s so that they match the order that order was sorted. More specifically, if a character x occurs before a character y in order, then x should occur before y in the permuted string.

Return any permutation of s that satisfies this property. Example 1:

Input: order = "cba", s = "abcd" Output: "cbad"

Explanation: "a", "b", "c" appear in order, so the order of "a", "b", "c" should be "c", "b", and "a".

Since "d" does not appear in order, it can be at any position in the returned string. "dcba", "cdba", "cbda" are also valid outputs.

Example 2:

Input: order = "bcafg", s = "abcd" Output: "bcad"

Explanation: The characters "b", "c", and "a" from order dictate the order for the characters in s. The character "d" in s does not appear in order, so its position is flexible.

Following the order of appearance in order, "b", "c", and "a" from s should be arranged as "b", "c", "a". "d" can be placed at any position since it's not in order. The output "bcad" correctly follows this rule. Other arrangements like "bacd" or "bcda" would also be valid, as long as "b", "c", "a" maintain their order.

Input Format

The first line of input consists of the string order, representing the custom order of characters. The second line of input consists of the string s, representing the input string.

Output Format

The output prints the sorted string according to the custom order specified by order. Refer to the sample output for the formatting specifications. Constraints

1 ≤ order.length ≤ 26

1 ≤ s.length ≤ 200

order and s consist of lowercase English letters. All the characters of order are unique. Q32.

Problem Statement:

Given an integer array nums, find the subarray with the largest sum, and return its sum. Example 1:

Input: n=9

nums = [-2,1,-3,4,-1,2,1,-5,4]

Output: 6

Explanation: The subarray [4,-1,2,1] has the largest sum 6. Example 2:

Input: n=1 nums = [1] Output: 1

Explanation: The subarray [1] has the largest sum of 1. Example 3:

Input: n=5

nums = [5,4,-1,7,8]

Output: 23

Explanation: The subarray [5,4,-1,7,8] has the largest sum of 23. Input Format

The first line consists of an integer n, representing the number of elements in the array nums.

The second line consists of n space-separated integers, representing the elements of the array nums. Output Format

The output prints an integer, representing the sum of the subarray with the largest sum. Refer to the sample output for formatting specifications. Constraints

1 ≤ n ≤ 20

-100 ≤ nums[i] ≤ 100

Q33.

Problem Statement

Given an array, the task is to find the sum of the maximum sum by alternating subsequences starting with the first element. Here alternating sequence means first decreasing, then increasing, then decreasing,... For example, 10, 5, 14, and 3 are alternating sequences.

Note: The reverse sequence type (increasing—decreasing—increasing—...) is not considered alternating here.

Example Input:

6

4 3 8 5 3 8

Output:

28

Explanation:

The alternating subsequence (starting with the first element) that has the maximum sum is {4, 3, 8, 5, 8}

Input Format

The first line contains an integer n, representing the number of elements in the array.

The second line contains n space-separated integers, representing the elements of the array. Output Format

The output displays a single integer representing the maximum sum of an alternating subsequence from the given array.

Refer to the sample output for the formatting specifications. Constraints

1 ≤ n ≤ 10

1 ≤ elements ≤ 100

Question 1: Terrain Digging (Dynamic Programming) Problem Statement:

You are given N terrain segments. Each segment i has a height Li. You want to make the terrain strictly decreasing, i.e., L1 > L2 > ... > LN.

On day D, a team can reduce the height of any segment by 2^(D-1). You may assign the team to any segment any number of times (even multiple times on different days).

Find the minimum number of days required to make the terrain strictly decreasing. Constraints:

* 2 ≤ N ≤ 10^5
* -10^9 ≤ Li ≤ 10^9 Input:

N L1 L2

... LN

Output:

Minimum number of days

Question 2: Magical Road Construction (Bitmask + DP)

A magical road needs to be strictly increasing in height to avoid monster attacks. You’re given heights

H[0...N-1] and can cast spells to increase any segment’s height.

On day D, the spell increases a chosen segment by 2^(D-1). You can cast multiple spells on any segment and on any day.

Find the minimum number of days required to make the entire array strictly increasing.

Question 3: Mountain Array Transformation (Pattern Matching + Brute Force) You’re given an array A

of N integers. Transform it into a mountain array, defined as:

* Both ends have equal values
* As you move from either end to the middle, each next value must increase by 1
* The peak value(s) can be same or different, but the pattern must rise and fall symmetrically

Find the minimum number of elements to change to achieve a mountain array. QUESTION 4: n the stock market, a person buys a stock and sells it on some future date. You are given an array prices[] representing stock prices on different days and a positive integer k, find out the maximum profit a person can make in at-most k transactions.

A transaction consists of buying and subsequently selling a stock and new transaction can start only when the previous transaction has been completed.

Examples :

Input: prices[] = [10, 22, 5, 80], k = 2

Output: 87 Explanation:

1st transaction: Buy at 10 and sell at 22.

2nd transaction : Buy at 5 and sell at 80. Total Profit will be 12 + 75 = 87. Input: prices[] = [100, 90, 80, 50, 25], k = 1

Output: 0 Constraints:

1 ≤ prices.size() ≤ 103

1 ≤ k ≤ 200

1 ≤ prices[i] ≤ 103

Question 5:

Altaf has recently learned about number bases and is becoming fascinated.

Altaf learned that for bases greater than ten, new digit symbols need to be introduced, and that the convention is to use the first few letters of the English alphabet. For example, in base 16, the digits are 0123456789ABCDEF. Altaf thought that this is unsustainable; the English alphabet only has 26 letters, so this scheme can only work up to base 36. But this is no problem for Altaf, because Altaf is very creative and can just invent new digit symbols when she needs them. (Altaf is very creative.)

Altaf also noticed that in base two, all positive integers start with the digit 1! However, this is the only base where this is true. So naturally, Altaf wonders: Given some integer N, how many bases b are there such that the base-b representation of N starts with a 1?

INPUT FORMAT :

The first line of the input contains an integer T denoting the number of test cases. The description of T test cases follows.

Each test case consists of one line containing a single integer N (in base ten). OUTPUT FORMAT :

For each test case, output a single line containing the number of bases b, or INFINITY if there are an infinite number of them.

CONSTRAINTS :

* 1 <= T <= 10^5
* 0 <= N < 10^12

SAMPLE INPUT : 4 6

9

11

24

SAMPLE OUTPUT : 4 7

8

14

Question 6:

Sam is handed a rectangular piece of paper with the dimensions h\*w, where h denotes height and w denotes width. Sam wants to fold the paper in the fewest possible moves such that its dimensions are h1\*w1. The paper could only be folded parallel towards its edges, and the dimensions should be integers after folding. If h=8 and w=4, for example, fold the paper till it's h1, w1 = 6,1. First, fold the long edge 6 units to the side, resulting in a 6\*4 paper. And then, Fold the page in three folds along the width 2 units edge to achieve a 6\*1 page.

Input Format

First line contains an integer denotes h Seconds line contains an integer denotes w Third line contains an integer denotes h1 Fourth line contains an integer denotes w1 Constraints

1 <= h, w, h1, w1 <= 10^15

h1 <= h w1 <= w Output Format

Print the minimum moves required. Sample Input:

Sample Output:

Question 7:

Finding the shortest path in a graph is a problem that we can solve using dynamic programming. Suppose we need to find the shortest path between the nodes ‘u’ and ‘v’. There is a node ‘x’ that falls in the shortest path from node ‘u’ to ‘v’. Then we can find the answer using the shortest path from ‘u’ to ‘x’ and the shortest path from ‘x’ to ‘v’.

Question 8:

We have been given a binary matrix of R x C. we have to find the maximum area of the square submatrix.

For example, if the given array is: 0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 1 1 1 0 0

0 0 1 1 1 0 0

0 0 1 1 1 0 0

0 0 0 0 0 0 0

then the largest sub-matrix square is: 1 1 1

1 1 1

1 1 1

and its area is 3\*3 = 9 Input Matrix:

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 1 1 1 0 0

0 0 1 1 1 0 0

0 0 1 1 1 0 0

0 0 0 0 0 0 0

Sol Matrix: 0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 1 1 1 0 0

0 0 1 2 2 0 0

0 0 1 2 3 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

Question 9:

You are given a multi story building, and some number of eggs. You want to know from which floors, if you drop a egg, it will break.

It’s a Monday in Coders’ world. You are about to get a new problem. There is a building with floors 1 to ‘FLOORS’. You are given a basket full of identical eggs. ‘EGGS’ identical eggs. As the eggs are identical, they follow the same properties. These properties are given below:

1. Each egg breaks only if it is dropped from a certain floor ‘EGGBREAKER’ or above.
2. An egg that is not broken after being dropped from a floor can be used to drop again. you are given the following details:
   1. ‘FLOORS’: Total number of floors in the building.
   2. ‘EGGS’ : Total number of eggs in the basket.
   3. Your task is to find the minimum number of egg drops required in the worst case to

find out the ‘EGGBREAKER’ floor.

Dropping from different floors leads to a different number of moves required to find the

‘EGGBREAKER’ floor.

Input

Enter Eggs: 2

Enter Floors: 6

Output

Minimum number of moves required: 3 Question 10:

The Central Bureau of Investigation encrypts some papers for security reasons. The document is first turned into lower case letter strings in this process. This string is now being encoded.

Encoding Procedures

* The string can be broken down into a list of non-empty substrings at random.
* Now that we have a list of strings, randomly pick some substrings and change them with their lengths.
* After that, combine the list of substrings to make an encoded string.

You are given two encoded strings, 'S1' and 'S2', consisting of lower case English letters and digits 1 to 9, and you have to tell if both strings are encoded from the same parent string. In other words, return true if a parent string 'S' exists, which can be encoded as 'S1' and 'S2' both. Else return false.

EASY:

Question 1:

Ram went to fight for Coding Club. There were N fighters with various powers. There will be Q rounds to fight and in each round, Ram's power will be varied. With power M, Ram can kill all the fighters whose power is less than or equal to M(<=M). After each round, All the soldiers who are dead in the previous round will reborn. Such that in each round there will be N fighters to fight. As Ram is weak in mathematics, help him to count the number of fighters that he can kill in each round and the total sum of their powers.

INPUT:

The first line of the input contains N, the number of fighters.

The second line contains an array of N numbers denoting the power of each fighter This third line contains Q, which denotes the number of rounds.

Q lines follow, each line having one number denoting the power of Ram. OUTPUT:

For each round, the output should be an array of two numbers. The first number should be the number of fighters that Ram can beat, and the second number denotes the cumulative strength of all the fighters that Ram can beat.

CONSTRAINTS:

1<=N<=10000

1<=power of each fighter<=100 1<=Q<=10000 1<=power of Ram<=100

Case 1: Input: 7

1 2 3 4 5 6 7

3

3

10

2

Output:

3 6

7 28

2 3

Case 2: Input:

5

10 20 30 40 50

4

5

25

50

60

Output:

0 0

2 30

5 150

5 150

Case 3:

Input:

6

5 10 15 20 25 30

3

1

15

35

Output:

0 0

3 30

6 105

Question 2:

You are given a string s and a list dictionary[] of words. Your task is to determine whether the string s can be formed by concatenating one or more words from the dictionary[].

Note: From dictionary[], any word can be taken any number of times and in any order. Constraints:

1 ≤ s.size() ≤ 3000

1 ≤ dictionary.size() ≤ 1000

1 ≤ dictionary[i].size() ≤ 100

Case 1:

Input: s = "ilike", dictionary[] = ["i", "like", "abc"] Output: true Case 2:

Input: s = "ilikeabc", dictionary[] = ["i", "like", "man", "india", "abc"] Output: true Case 3:

Input: s = "ilikemangoes", dictionary[] = ["i", "like", "man", "india", "abc"] Output: false

Question 3:

You’re a data compression engineer working on a custom text encoder. Your goal is to minimize the weight of a string before storing it in a compressed database. Weight of a string is defined as: The sum of squares of the frequency of each distinct character.

You're allowed to remove exactly k characters from the string (any characters, not necessarily distinct). The objective is to minimize the weight after removals.

Constraints:

0 ≤ k ≤ |string length| ≤ 5\*104 Case 1:

Input:

str = "abccc" k = 1

Output:

6

Case 2:

Input:

str = "aabcbcbcabcc" k = 3

Output:

27

Case 3:

Input:

str = "zzzzzz" k = 2

Output:

16

Question 4:

You're building a task scheduler for a project management system. Each task may depend on other tasks being completed first. All its prerequisites must be finished before starting any task.

You're given: The total number of tasks N. A list of prerequisite relationships P, where each pair [A, B] means: "To do task A, you must finish task B first."

You must determine if all the tasks are scheduled so that no prerequisite conflict prevents completion. If yes, return "Yes" — all tasks can be completed.

If no, return "No" — a cyclic dependency exists.

Case 1:

Input:

N = 4

P = 3

prerequisites = [[1, 0], [2, 1], [3, 2]]

Output:

Yes Case 2:

Input:

N = 2

P = 2

prerequisites = [[1, 0], [0, 1]]

Output:

No Case 3:

Input:

N = 6

P = 5

prerequisites = [[1, 0], [2, 1], [3, 2], [4, 3], [5, 4]]

Output:

Yes

Question 5:

In the realm of binary encoding, a transmission protocol defines every integer as a fixed-length bitstream. When converting one encoded signal a to another b, the transformation cost is measured in bit mismatches — each mismatch represents a necessary flip to achieve identity.

Given two non-negative integers a and b, determine the minimum Hamming distance between their binary representations — that is, count the number of individual bit positions at which the corresponding bits differ when a is transformed into b.

Your goal is to compute the number of atomic bit-flip operations needed to make a exactly equal to

b. Constraints:

Both a and b lie within the closed interval [1, 10⁶].

Case 1:

Input:

a = 10

b = 20

Output: 4

Case 2:

Input:

a = 20

b = 25

Output:

3

Case 3:

Input:

a = 1000

b = 1500

Output:

6

MEDIUM

Question 1:

In a chain of N mountain reservoirs, each dam holds water up to height Hᵢ. You may release water from any dam down to any lower level; releasing from height h₁ to h₂ costs h₁–h₂ energy units. Your goal is to make all reservoirs share the same water height at minimum total energy.

Constraints:

1 ≤ N ≤ 10⁵ 1 ≤ Hᵢ ≤ 10⁹ Input:

N

H₁ H₂ … Hₙ

Output:

One integer: the least total energy needed.

Case 1:

Input: 4

8 3 6 3

Output: 11

Case 2: Input:

1

5

Output:

0

Case 3:

Input:

3

100 100 100

Output:

0

Question 2:

A lantern festival runs along a straight promenade divided into segments [Sᵢ, Eᵢ]. You need to install light- beacons (at integer positions) so each segment has at least one beacon. Each beacon normally costs 1 lantern package, but every second beacon at the exact same spot is free. Compute the minimum number of packages needed.

Input:

M

S₁ E₁

…

Sₘ Eₘ

Output:

One integer: the minimum lantern packages.

Constraints: 1 ≤ M ≤ 10⁵ 1 ≤ Sᵢ< Eᵢ ≤ 10⁹

Case 1:

Input : 3

2 5

4 7

6 8

Output: 2

Case 2: Input:

2

1 10

1 10

Output:

1

Case 3:

Input:

3

1 2

3 4

5 6

Output:

3

Question 3:

A data center has N separate cable segments, each of length Lᵢ. To form one continuous cable, you repeatedly join two segments into one; joining lengths a and b costs a + b (materials and labor).

Compute the minimum total cost to consolidate all segments into a single cable.

Input:

N

L₁ L₂ … Lₙ

Output:

One integer: the minimum total merge cost.

Constraints: 1 ≤ N ≤ 10⁵ 1 ≤ Lᵢ ≤ 10⁹

Case 1:

Input: 4

4 3 2 6

Output: 29

Case 2: Input: 1

10

Output:

0

Case 3:

Input:

3

5 5 5

Output:

25

Question 4:

A composer has written a melody string T (notes represented by lowercase letters). If they remove up to K notes, they want the lexicographically smallest remaining melody (order of the others must stay the same).

A lexicographically smallest string is one that would appear first in a dictionary: compare two strings from the first character onward; at the first differing character, the string with the smaller character is lexicographically smaller.

Input:

T K

Output:

The lexicographically smallest string after ≤ K deletions.

Constraints:

1 ≤ |T| ≤ 10⁵

0 ≤ K < |T|

T consists of lower case alphabets only

Case 1:

Input: dbcaab 2

Output: bcaab Case 2: Input: zyx

1

Output:

Yx

Case 3:

Input: abc 0 Output: abc

Question 5:

The royal courier must deliver N decrees, one at a time. Decree i takes Tᵢ days to deliver and incurs a daily displeasure penalty of Wᵢ gold coins per day until it’s delivered. When decree i finishes on day Cᵢ, the palace pays a total penalty of Wᵢ × Cᵢ. Devise a delivery order to minimize the sum of all penalties

Input:

N

T₁ W₁ T₂ W₂

…

Tₙ Wₙ

Output:

One integer: the minimum total penalty.

Constraints: 1 ≤ N ≤ 10⁵ 1 ≤ Tᵢ, Wᵢ≤ 10⁶

Case 1:

Input: 3

3 4

1 100

2 2

Output: 128

Case 2: Input:

1

1 10

Output:

10

Case 3:

Input:

2

5 2

3 3

Output:

25

Question 6:

You are given a DNA sequence S of length N, made up of the characters A, C, G, and T.

You want to divide this string into K equal-length contiguous blocks (assume N % K == 0). Your task is to mutate as few characters as possible so that all K blocks become the same.

A mutation means changing one character to another at a specific position. You are allowed to mutate any character, but your goal is to minimize the total number of mutations across all blocks.

Your task is to compute the minimum number of mutations required. Parameters:

* S :: STRING The first line contains the DNA sequence. len(S) :: 1 → 2 \* 10^5 Characters are

only from A, C, G, and T.

* K :: INTEGER The next line contains the integer K, the number of blocks. K :: 1 → len(S) It is

guaranteed that len(S) % K == 0.

Case#: 1

Input: ACGTACGT 2

Output: 0

Explanation: Split into blocks: ACGT and ACGT They are already identical. No mutations needed. Case#: 2

Input: ACGTAGCT 2

Output: 2

Explanation: Blocks: ACGT and AGCT At position 2: G vs C → mutate one At position 3: T vs T → same

Minimum mutations needed = 2 Case#: 3

Input: AACCGGTT 4

Output: 6

Explanation: Blocks: AA, CC, GG, TT For each position (0 and 1), we want to align all to the most frequent character.

* Position 0: [A, C, G, T] → pick A → 3 mutations
* Position 1: [A, C, G, T] → pick A → 3 mutations

Case#: 4

Input: ATCGATCA 2

Output: 1

Explanation: Blocks: ATCG, ATCA Only position 3 differs: G vs A → 1 mutation needed

Question - 7

You are managing a fleet of delivery vehicles. Each vehicle starts from a central depot and delivers packages to customers. Every customer has a specific time window during which they must receive their delivery.

Your task is to assign customers to vehicles and determine delivery routes such that:

* All customers are visited within their specified time windows.
* Each vehicle starts and ends at the depot.
* The total distance travelled by all vehicles is minimized.

A vehicle can wait at a customer location if it arrives early. All vehicles travel at a speed of 1 unit distance per minute. There is no limit on the number of deliveries per vehicle unless time constraints prevent it.

Parameters:

* N :: INTEGER The number of customers. 1 → 10^4
* M :: INTEGER The number of available delivery vehicles. 1 → 100
* Depot :: TUPLE(INT, INT) Coordinates (x, y) of the central depot.

Customers :: LIST A list of N entries. Each customer is described by: (x, y, start\_time, end\_time) where: (x, y) are 2D coordinates of the customer,

start\_time and end\_time define the allowed time window (in minutes). Each vehicle:

* Starts at time 0 from the depot.
* Must return to the depot after deliveries.
* Travels in 2D space with Manhattan distance at 1 unit/minute.
* May wait at the customer location if it arrives before the time window.
* Can return -1 if it is not possible to honour the timelines.

Case#: 1 Input:

3

1

0 0

1 0 0 10

2 0 5 15

3 0 10 20

Output:

6

Explanation:

* One vehicle can visit all 3 customers in sequence: (1,0) → (2,0) → (3,0)
* Then return to depot: (0,0)
* Total route: 0→1→2→3→0 → total distance = 6 [vehicle has to wait 3 mins at 2nd customer

and 4 mins at 3rd customer though for honouring time windows]

* All time windows respected.

Case#: 2 Input:

3

3

0 0

10 0 0 5

20 0 0 5

30 0 0 5

Output:

-1

Explanation:

* Due to tight time windows and long distances, one vehicle cannot serve multiple customers.
* Each vehicle serves one customer and returns. Distances: Depot → (10,0) → Depot = 20 Depot → (20,0) → Depot = 40 Depot → (30,0) → Depot = 60

Total = 20 + 40 + 60 = 120 but due to time window [0,5] for all the customers, it is not feasible.

Question 8:

You have an interesting string S of length N. It is interesting because you can rearrange the characters of this string in any order.

After rearranging the string once, you want to cut it into K contiguous segments such that each segment is a palindrome. You cannot rearrange or modify the segments after cutting.

Your goal is to find the maximum number of palindromic segments you can obtain after rearrangement and cutting.

Note: A palindrome is a string that reads the same forward and backward, such as "aba", "abba", or "a".

Parameters:

* S :: STRING The first line contains a string S, denoting the string. len(S) :: 1 -> 2 \* 10^5

Case#: 1 Input: aabbcc Output: 2

Explanation: We can rearrange the string as "abc" + "cba" → both are palindromes. Hence, maximum

palindromic segments = 2. Case#: 2

Input: aaabbbbcc

Output: 3

Explanation: Rearranged string = "bab" + "cac" + "bab" or similar. All three segments are palindromes. Hence, answer is 3.

Case#: 3 Input: abcabc Output: 1

Explanation: Any rearrangement of the string cannot be divided into more than one palindrome segment. Best option is to keep the whole string as one palindrome.

Question 9:

You are designing the stamina management system for a turn-based game. A character must defeat a sequence of enemies, where each enemy requires a certain amount of stamina to defeat. The character starts with full stamina and can either fight, skip, or use a potion. Your goal is to minimize the number of potions used while defeating all enemies(can skip max K enemies).

Rules of the game:

* There are N enemies, indexed 0 to N-1.
* The character starts with S stamina points.
* Each enemy requires cost[i] stamina to fight.

If current stamina is less than cost[i], you must either: Skip the enemy (but you can only skip up to K enemies in total), or

Use a potion to restore stamina back to S (can use multiple times). After fighting, stamina is reduced by cost[i].

Skipping does not reduce stamina.

Parameters:

* N :: INTEGER (number of enemies) 1 ≤ N ≤ 10^4
* S :: INTEGER (initial stamina per potion refill) 1 ≤ S ≤ 1000
* K :: INTEGER (maximum skips allowed) 0 ≤ K ≤ N
* cost :: LIST of INTEGER cost[i] is the stamina required to fight the i-th enemy 1 ≤ cost[i] ≤

1000

Compute the minimum number of potions the character needs to use to defeat all enemies, without exceeding K skips.

Case#: 1

Input:

N = 5

S = 10

K = 1

cost = [4, 6, 8, 2, 5]

Output:

1

Explanation:

Enemy 0: Fight (stamina 10 → 6)

Enemy 1: Fight (stamina 6 → 0) Enemy 2: Skip (skip used = 1)

Enemy 3: Use potion → stamina = 10 → fight (→ 8) Enemy 4: Fight (stamina 8 → 3)

HARD:

Question 1:

There is a soccer team with n players, numbered from 0 to n-1. The team wants to pass the ball from player 0 to player n-1, and each player can only pass forward, i.e., from a player i to another player j such that j>i.

You are given a 2D array prob, where prob[i][j] represents the probability (a value between 0 and 1) that thepass from player i to player j will succeed. Each player can pass the ball to any player ahead (j > i), and your goal is to choose an optimal sequence of passes to maximize the probability that player n-1 successfully receives the ball.

Your task is to calculate the maximum probability of successfully passing the ball from player 0 to player n-1, by choosing the best sequence of passes.

Parameters:

* The first line contains an integer n, denoting the number of players. 2 ≤ n ≤ 100
* The next n lines contain n space-separated floating-point numbers each.

The j-th number on the i-th line (where j > i) represents prob[i][j], the success probability of passing from player i to player j.

All entries where j ≤ i will be 0 and should be ignored.

Output:

* Return the maximum probability (a floating-point number) that player n-1 successfully receives the ball, rounded to 6 decimal places.

Case#:1 Input:

4

0 0.9 0.8 0.0

0 0 0.7 0.5

0 0 0 0.6

0 0 0 0

Output:

0.504000

Case#: 2 Input:

3

0 0.5 0.2

0 0 0.9

0 0 0

Output:

0.450000

Case#: 3 Input

5

0 0.4 0.5 0.0 0.0

0 0 0.6 0.7 0.0

0 0 0 0.8 0.3

0 0 0 0 0.9

0 0 0 0 0

Output 0.302400

Question 2:

US Iron, a company that specializes in custom hand-forged iron, wants to reuse iron rod ends and weld them into one longer rod. Given N iron rods of different lengths, you are asked to write a program to find the minimum cost to weld all of them into a longer rod, knowing that the cost of welding two iron rods is equal to the sum of their two lengths.

Parameters:

N : INTEGER

The first integer contains N, denoting the number of rods. 3 <= N <= 100 array[] : INTEGER ARRAY

An array of N integers, each denoting the length of a rod. 1 <= array[i] <= 100 Input:

Each test case consists of one line containing one integer N representing the number of iron rods to weld followed by N integers representing the length of each rod.

Output:

For each test case, output one line containing the minimum cost to weld all the iron rods into a single rod.

Case#1: Input:

5

21 29 18 33 9

Output:

247

Case2#: Input:

6

63 11 42 21 35 53

Output:

549

Case#3: Input:

7

2 11 15 20 7 17 6

Output:

205

Question 3:

There are ‘r’ red and ‘g’ green blocks for construction of the red-green tower. Red-green tower can be built following next rules:

Red-green tower is consisting of some number of levels;

Let the red-green tower consist of n levels, then the first level of this tower should consist of n blocks, second level — of n - 1 blocks, the third one — of n - 2 blocks, and so on — the last level of such tower should consist of the one block. In other words, each successive level should contain one block less than the previous one;

Each level of the red-green tower should contain blocks of the same color.

Let h be the maximum possible number of levels of red-green tower, that can be built out of r red and g green blocks meeting the rules above. The task is to determine how many different red-green towers having h levels can be built out of the available blocks.Two red-green towers are considered different if there exists some level, that consists of red blocks in the one tower and consists of green blocks in the other tower.

You are to write a program that will find the number of different red-green towers of height h modulo 109 + 7.

Parameters:

The only line of input contains two integers r and g, separated by a single space — the number of

available red and green blocks respectively (0 ≤ r, g ≤ 2·105, r + g ≥ 1).

Output the only integer — the number of different possible red-green towers of height (h modulo 109 + 7).

Case#1: Input

4 6

Output 2

Case#2 Input 9 7 Output

6

Case#3 1 1 Output

2

Question 4:

Winter is here in the North and the White Walkers are close. John Snow has an army consisting of n soldiers. While the rest of the world is fighting for the Iron Throne, he will get ready for the attack of the White Walkers.

He created a method to know how strong his army was. Let the i-th soldier’s strength be ai. For some

k he calls i1, i2, ..., ik a clan if i1 < i2 < i3 < ... < ik

and gcd(ai1, ai2, ..., aik) > 1 . He calls the strength of that clan k·gcd(ai1, ai2, ..., aik). Then he defines the strength of his army by the sum of strengths of all possible clans.

Your task is to find the strength of his army. As the number may be very large, you have to print it modulo 1000000007 (109 + 7).Greatest common divisor (gcd) of a sequence of integers is the maximum possible integer so that each element of the sequence is divisible by it.

Parameters:

The first line contains integer n (1 ≤ n ≤ 200000) — the size of the army.

The second line contains n integers a1, a2, ..., an (1 ≤ ai ≤ 1000000) — denoting the strengths of his soldiers.

Print one integer — the strength of John Snow's army modulo 1000000007 (109 + 7).

Case#1: Input

3

3 3 1

Output:

12

Case#2: Input

4

2 3 4 6

Output:

39

Case#3:

Input

5

7 14 21 2 5

Output:

140

Question 5:

There is a fun learning game devised by a teacher to help students learn Mathematics and English. In this game, the teacher gives students a random sequence of uppercase letters. The task is to identify the longest sequence of letters that are placed in correct alphabetical order relative to each other (not necessarily consecutively).

For example, in the sequence `AFBGCE`, a correct relative sequence is `ABCE`. The letters don't need to be adjacent in the original string, but they must appear in the same order.

Your task is to write a program that, given such a sequence, returns the length of the longest subsequence of letters that are in the correct relative alphabetical order.

Parameters:

The input contains:

* A single line of characters, consisting of uppercase English letters only.
* Length of the string ≤ 26 Output:

Output a single integer which represents the length of the longest subsequence where the letters are in relative alphabetical order.

Constraints:

* The sequence contains only uppercase English alphabets.
* Maximum length of the sequence is 26.

Case#1: Input: BACDFEG Output:

4

Case#2: Input: LMNOPXYZ Output:

8

Case#3:

Input:

QWERTYUIOP

Output:

3

1. Sliding-Window Minimum over Dynamic Updates Description:

You have an array A of length N, initially all zeros. You will process Q operations of two types:

1. Update: Set A[i] = x.
2. Query: For a given window size K, find the minimum value in each contiguous window of size K over the current array.

Return all query answers in order of appearance; each query produces an array of length N−K+1.

Constraints:

* + 1 ≤ N, Q ≤ 2·10^5
  + 1 ≤ i ≤ N, −10^9 ≤ x ≤ 10^9
  + 1 ≤ K ≤ N

Samples Sample 1

N = 5, Q = 4

Operations:

1. Update 3 5
2. Update 1 -2
3. Query K=3
4. Query K=5 Output:

[-2, 0, 0]

[-2]

After two updates, A = [−2,0,5,0,0].

Sample 2

N = 4, Q = 5

Ops:

1. Update 2 1
2. Update 4 -1
3. Query 2
4. Update 3 2
5. Query 3 Output:

[0, -1, -1]

[-1, -1]

First query on [0, 1, 0, -1] windows of size 2; second on [0 ,1 ,2 , -1] windows of size 3.

Sample 3

N = 3, Q = 3

Ops:

1. Query 1
2. Update 1 10
3. Query 2 Output:

[0, 0, 0]

[0, 0]

Initial A = [0, 0, 0]; after update, A = [10,0,0]

2.Trie-Based “Prefix Value Sum” with Removals

Description:

Maintain a multiset of strings with integer values. Support three operations:

1. Add s v: insert string s with value v.
2. Remove s v: remove one occurrence of (s,v) if exists.
3. Sum p: compute the sum of values of all strings in the multiset that start with prefix p. All strings consist of lowercase letters.

Constraints:

* + Total operations ≤ 2·10^5
  + 1 ≤ |s|,|p| ≤ 20
  + 1 ≤ v ≤ 10^6

Samples Sample 1 Ops:

Add "apple" 3

Add "app" 2 Sum "ap" Remove "app" 2 Sum "app" Output:

5

3

Sample 2 Ops:

Add "cat" 5

Add "car" 7 Sum "ca" Sum "car" Output:

12

7

Sample 3

Ops:

Add "dog" 4

Add "dog" 6

Remove "dog" 4 Sum "do" Output:

6

1. Offline Range-Connectivity with Union-Find Description:

You have N nodes numbered 1…N. You will receive M operations of two types, but you must answer

them offline:

* + Connect u v: add an undirected edge between u and v.
  + Query u v: at this point in time, are u and v in the same connected component? Return answers for all queries in order.

Constraints:

* + 1 ≤ N ≤ 10^5, 1 ≤ M ≤ 2·10^5
  + 1 ≤ u, v ≤ N

Samples Sample 1

N = 5, M = 5

Ops:

Connect 1 2

Query 1 3

Connect 2 3

Query 1 3

Query 4 5 Output:

NO YES NO

Sample 2

N = 3, M = 4

Ops:

Connect 1 2

Connect 2 3

Query 1 3

Query 1 1 Output: YES

YES

Sample 3

N = 4, M = 6

Ops:

Query 1 4

Connect 3 4

Query 1 4

Connect 1 2

Connect 2 3

Query 1 4 Output:

NO NO YES

1. Segment-Tree “Kth Smallest in Range” with Point Updates

Description:

Maintain an array A of length N (initial values given). Support:

* + Update i x: set A[i] = x.
  + Kth L R k: in the subarray A[L..R], find the k-th smallest element. If k > R−L+1, return −1.

Constraints:

* + 1 ≤ N, Q ≤ 10^5
  + 1 ≤ A[i], x ≤ 10^9
  + 1 ≤ L ≤ R ≤ N, 1 ≤ k ≤ N

Samples Sample 1

N=5, A=[5,1,4,2,3], Q=4

Ops:

Kth 1 5 2

Update 3 0

Kth 1 5 2

Kth 2 4 3 Output: 2

1

-1

Sample 2

N=3, A=[2,2,2], Q=3

Ops:

Kth 1 3 1

Update 2 1

Kth 1 3 2 Output: 2

2

Sample 3

N=4, A= [10,20,30,40], Q=3

Ops:

Kth 2 3 1

Update 4 5

Kth 1 4 4 Output:

20

-1

1. Dynamic Range Minimum with Swaps Description:

You have an array A of length N. You will process Q operations of two types:

1. Swap i j: Swap A[i] and A[j].
2. RangeMin L R: Report the minimum value in A[L..R]. Both operations must run in O(log N) time.

Constraints:

* + 1 ≤ N, Q ≤ 2·10⁵
  + 1 ≤ A[i] ≤ 10⁹
  + 1 ≤ i, j, L, R ≤ N

Samples Input:

N=5, A=[4,2,5,1,3], Q=4

Ops:

1. RangeMin 2 4
2. Swap 2 4
3. RangeMin 2 4
4. RangeMin 1 5 Output:

1

2

1

Input:

N=3, A=[7,7,7], Q=3

Ops:

1. RangeMin 1 3
2. Swap 1 3
3. RangeMin 1 1 Output:

7

7

Input:

N=4, A=[10,5,8,6], Q=5

Ops:

1. Swap 1 4
2. RangeMin 1 3
3. Swap 2 3
4. RangeMin 2 4
5. RangeMin 3 3 Output:

6

5

8

6.K-th Nearest Element in a BST Description:

You build a binary search tree (initially empty) by inserting N distinct keys in given order. Then answer Q queries: for each x and k, find the k-th smallest element in the BST strictly greater than x. If fewer than k such elements exist, return -1.

Each query must run in O(log N) on average.

Constraints:

* 1 ≤ N, Q ≤ 10⁵
* Keys and queries: 1 ≤ x, key ≤ 10⁹, 1 ≤ k ≤ N

Samples Input:

Insert order: [5, 3, 8, 2, 4, 6] Queries:

1. (x=4, k=1)
2. (x=5, k=2)
3. (x=8, k=1)

Output:

5

6

-1

Input:

Insert: [10, 20, 30]

Queries:

(15,1), (15,2), (5,1)

Output:

20

30

10

Input:

Insert: [7,1,9,3] Queries:

(3,1), (3,2), (3,3)

Output:

7

9

-1

1. Circular Buffer Maximum Load Description:

You have a fixed-size circular buffer of capacity C (initially empty). Support Q operations:

1. Enqueue x: add x to the back (if full, overwrite the oldest).
2. Max: report the current maximum element in the buffer. All operations must be amortized O(1).

Constraints:

* + 1 ≤ C, Q ≤ 2·10⁵
  + |x| ≤ 10⁹

Samples Input: C=3

Ops:

Enqueue 1

Enqueue 3 Max Enqueue 2

Enqueue 5 Max Output:

3

5

Input:

C=2

Ops: Enqueue 4 Max Enqueue -1

Max Enqueue 2 Max Output:

4

4

2

Input:

C=1

Ops:

Enqueue 7 Max Enqueue 5 Max Output:

7

5

1. LRU Cache with Variable Capacity Description:

Implement an LRU cache that supports Q operations:

1. SetCap c: change cache capacity to c. If current size > c, evict least-recently-used items until size = c.
2. Put k v: insert or update key k with value v. If insertion causes overflow, evict LRU.
3. Get k: return value for key k, or -1 if not present. Accessing a key makes it MRU. All ops in O(1) time.

Constraints:

* + 1 ≤ Q ≤ 2·10⁵
  + Keys, values: integers in [−10⁹,10⁹]
  + Capacities: 1 ≤ c ≤ 2·10⁵

Samples Input: Ops: SetCap 2

Put 1 100

Put 2 200

Get 1

Put 3 300

Get 2 Output: 100

-1

Input:

Ops:

SetCap 1

Put 5 50

Put 6 60

Get 5

Get 6 Output:

-1

60

Input:

Ops:

SetCap 3

Put 1 10

Put 2 20

Put 3 30

SetCap 2

Get 1

Get 3

Output:

-1

30

1. Circular String Substring Queries Description:

You’re given a base string S (length N). Process Q queries, each a pattern P. Determine whether P appears as a contiguous substring in any cyclic rotation of S. Return YES or NO per query. Preprocess S in O(N); answer each query in O(|P|).

Constraints:

* + 1 ≤ N ≤ 2·10⁵, 1 ≤ Q ≤ 2·10⁵
  + Sum of all |P| ≤ 2·10⁵
  + Lowercase letters only. Samples

1.

S = "abac" Queries: "cab"

"aca"

"baa" Output: YES

YES NO 2.

S = "aaaa" Queries: "aaa" "aaaaa" "aa"

Output:

YES NO YES 3.

S = "xyz" Queries:

"yzx"

"zy"

"xy" Output:

YES NO YES

Sliding-Window Distinct Count Description:

Given an array A of length N, answer Q queries of the form (L,R): return the number of distinct elements in the subarray A[L..R]. Preprocess in O(N log N); answer each query offline in O((N+Q) log

N) via Fenwick or segment tree. Constraints:

* 1 ≤ N, Q ≤ 2·10⁵
* 1 ≤ A[i] ≤ 10⁹
* 1 ≤ L ≤ R ≤ N

Samples 1.

A = [1,2,1,3,2], Q=3

Queries: (1,5)

(2,4)

(3,3)

Output:

3

2

1

2.

A = [5,5,5], Q=2

Queries:

(1,3)

(2,3)

Output:

1

1

3.

A = [1,2,3,4], Q=2

Queries:

(1,2)

(3,4)

Output:

2

2

1. Real‐time Stock Span

Description:

Design a system that processes N daily stock prices one by one and reports the “span” for each day:

the number of consecutive days before today (inclusive) for which the price was ≤ today’s price.

Implement in O(N) overall with a stack. Constraints:

* + 1 ≤ N ≤ 2·10⁵
  + 1 ≤ price ≤ 10⁹

Samples 1.

Prices = [100, 80, 60, 70, 60, 75, 85]

Output:

[1, 1, 1, 2, 1, 4, 6]

2.

Prices = [10, 20, 30]

Output:

[1, 2, 3]

3.

Prices = [30, 20, 10]

Output:

[1, 1, 1]

1. Partial Persistent Min-Queue Description:

Implement a queue that supports persistence across versions. You have an initial empty queue (version 0). Each operation creates a new version:

1. Enqueue v x: on version v, enqueue x, producing version v'.
2. Dequeue v: on version v, dequeue front element (if non-empty), producing version v'.
3. Front v: return the front element of version v (or -1 if empty).

All operations in O(1) amortized, using two stacks per version (persistent stacks). Samples

1.

Ops:

Enqueue 0 5 → v1

Enqueue 1 3 → v2 Front 2 → ?

Dequeue 2 → v3 Front 3 → ?

Output:

5

3

2.

Ops:

Front 0 → ? Enqueue 0 1 → v1

Dequeue 1 → v2 Front 2 → ?

Output:

-1

-1

3.

Ops:

Enqueue 0 2 → v1

Enqueue 1 4 → v2 Dequeue 2 → v3 Dequeue 3 → v4 Front 4 → ?

Output:

-1

1. Kth Smallest Sum of Two Sorted Arrays Description:

Given two sorted arrays A (length N) and B (length M), find the k-th smallest sum A[i]+B[j] among all

0 ≤ i < N, 0 ≤ j < M. Target O((N+M) log (N+M)) using a min-heap of candidate pairs. Samples

1.

A=[1,3,5], B=[2,4,6], k=4

Output:

7

Sums sorted: [3,5,7,7,9,9,...], 4th is 7.

2.

A=[1,2], B=[3], k=2

Output:

5

3.

A=[1,1,2], B=[1,2,3], k=5

Output:

3

1. Online Minimum Spanning Forest Description:

Maintain an undirected graph with N nodes (initially no edges). Process Q operations:

1. AddEdge u v w: add edge (u,v) with weight w.
2. Query: report the total weight of the current minimum spanning forest.

Requires an online dynamic MST algorithm in O(log² N) amortized per operation, e.g., using a link-cut tree or dynamic tree structure.

Samples 1.

N=4

Ops:

AddEdge 1 2 3 Query AddEdge 2 3 1

AddEdge 3 4 2 Query

Output:

3

6

2. N=3

Ops:

Query

AddEdge 1 2 5 Query AddEdge 2 3 4 Query

Output:

0

5

9

1. N=5

Ops:

AddEdge 1 2 2

AddEdge 2 3 2

AddEdge 3 4 2

AddEdge 4 5 2 Query

Output:

8

1. Mergeable Heaps with Delete Description:

Implement a data structure supporting Q operations:

1. MakeHeap: create a new empty heap, return its id.
2. Insert h x: insert x into heap h.
3. GetMin h: return minimum element in heap h.
4. ExtractMin h: remove and return min from h.
5. Merge h1 h2: merge heap h2 into h1 (destroy h2). All ops in O(log N) using a leftist or skew heap.

Samples Ops:

h1 = MakeHeap h2 = MakeHeap Insert h1 5

Insert h2 3 Merge h1 h2 GetMin h1

Output:

3

2.

Ops:

h = MakeHeap Insert h 10

Insert h 1 ExtractMin h GetMin h Output:

1

10

3.

Ops:

h1 = MakeHeap Insert h1 7

h2 = MakeHeap GetMin h2 Output:

-1 (empty heap)

1. Tree Subtree Maximum Query Description:

Given a rooted tree of N nodes (root = 1) with initial values val[i]. Support Q operations:

1. Update u x: set val[u] = x.
2. SubtreeMax u: return the maximum value among all nodes in the subtree of u. Use O(log N) per op via Euler-tour to flatten tree + segment tree.

Constraints:

* + 1 ≤ N, Q ≤ 2·10⁵
  + 1 ≤ val[i], x ≤ 10⁹

Samples 1.

Tree: 1–2,1–3,3–4; val=[5,3,7,2]

Ops:

SubtreeMax 1 → 7

SubtreeMax 3 → 7

Update 4 10

SubtreeMax 3 → 10

Output:

7

7

10

2.

Tree: 1–2,2–3; val=[1,2,3]

Ops:

SubtreeMax 2 → 3

Update 3 0

SubtreeMax 2 → 2

Output:

3

2

1. ​

Tree: 1–2,1–3; val=[4,4,4]

Ops:

SubtreeMax 1 → 4

Update 2 5

SubtreeMax 1 → 5

Output:

4

5

1. Online Bipartiteness Check

Maintain an undirected graph with N nodes (initially no edges). Process Q operations:

1. AddEdge u v: add edge (u,v).
2. IsBipartite: return YES if current graph is bipartite, else NO. Implement in O(α(N)) per operation using DSU with parity bits. Constraints:
   * 1 ≤ N, Q ≤ 2·10⁵

Samples 1.

N=3

Ops:

AddEdge 1 2

IsBipartite → ?

AddEdge 2 3

AddEdge 3 1

IsBipartite → ?

Output:

YES NO 2. N=4

Ops:

AddEdge 1 2

AddEdge 3 4

IsBipartite → ?

Output:

YES

1. N=5

Ops:

AddEdge 1 2

AddEdge 2 3

AddEdge 1 3

IsBipartite → ?

Output:

NO

1. Dynamic Kth Largest in Stream Description:

Maintain a stream of numbers that supports two operations:

1. Add x: append x to the stream.
2. KthLargest k: return the k-th largest element seen so far; if fewer than k elements, return -1.

Achieve O(log k) per query by keeping a min-heap of size k for each distinct k requested (or using an Order-Statistics Tree).

Samples 1.

Ops:

Add 5

Add 1

Add 10

KthLargest 2

Add 6

KthLargest 3 Output:

5

1

2.

Ops:

KthLargest 1

Add 7

KthLargest 1 Output:

-1

7

1. ​

Ops:

Add 3

Add 3

Add 3

KthLargest 2 Output:

3

1. Range Sum of Fibonacci Numbers Description:

Precompute Fibonacci modulo 10⁹+7. Given an array I of indices (1-based), support Q queries:

* + SumFib L R: return ∑\_{j=L..R} Fibonacci(I[j]) mod 10⁹+7.

Use a Fenwick or segment tree, with O(log N) per query and point updates if desired. Samples

1.

I = [1,2,3,4,5]

Queries: SumFib 2 4

SumFib 1 5 Output:

2+3+5 = 10

1+2+3+5+8 = 19

2.

I = [10,10,10], Q=1

SumFib 1 3 Output:

3 \* Fib(10)=3\*55=165

3.

I = [6,7,8,9], Q=2

SumFib 1 2

SumFib 3 4 Output:

8+13=21

21+34=55

1. Dynamic Lowest Common Ancestor Description:

Maintain a rooted tree (initially only node 1). Support:

1. AddNode p: add a new node as a child of p.
2. LCA u v: return the lowest common ancestor of u and v.

Use binary-lifting with dynamic table resizing or link-cut trees for O(log N) per op. Samples

1.

Ops:

AddNode 1 → node2 AddNode 1 → node3 AddNode 2 → node4 LCA 4 3

Output:

1

2.

Ops:

AddNode 1 →2

AddNode 2 →3

AddNode 3 →4

LCA 3 4

Output:

3

3.

Ops:

AddNode 1 →2

AddNode 1 →3

AddNode 2 →4

AddNode 2 →5

LCA 4 5

Output:

2

20.K-th Largest on Tree Path Description:

Given a tree of N nodes (rooted arbitrarily), each node has a value val[i]. Support Q queries:

* KthOnPath u v k: find the k-th largest value among the nodes on the (unique) path from u to

v. If fewer than k nodes, return -1.

Achieve O(log² N) per query using Heavy-Light Decomposition plus a persistent segment tree (or order-statistics tree) on each chain.

Constraints:

* 1 ≤ N, Q ≤ 2·10⁵
* 1 ≤ val[i] ≤ 10⁹

Samples 1.

Tree: 1–2,1–3,3–4; val=[5,2,8,6]

Queries: KthOnPath 2 4 1

KthOnPath 2 4 2

KthOnPath 2 4 3 Output:

8

6

5

2.

Tree: 1–2,2–3; val=[1,3,2]

Queries: KthOnPath 1 3 2

KthOnPath 1 3 3

Output:

2

1

3.

Tree: 1–2,1–3,1–4; val=[7,4,7,4]

Queries:

KthOnPath 2 4 2 Output:

4

21.Editable String Palindrome Queries Description:

Maintain a string S of length N. Support Q operations:

1. Update i c: change S[i] = c.
2. IsPal L R: return YES if S[L..R] is a palindrome, else NO.

Implement in O(log N) per op using a segment tree with rolling-hash (both forward and backward). Constraints:

* + 1 ≤ N, Q ≤ 2·10⁵
  + c is a lowercase letter Samples

1.

S = "abca", Q=4

IsPal 1 4 → ?

Update 4 'a'

IsPal 1 4 → ?

Update 2 'b'

IsPal 1 3 → ?

Output:

NO

YES YES 2.

S = "racecar", Q=2 IsPal 1 7

IsPal 2 6 Output:

YES YES 3.

S = "abcde", Q=3 IsPal 1 3

Update 2 'a'

IsPal 1 3 Output:

NO YES

22.2D Range Add & Max Query Description:

Maintain an N×M matrix A. Support Q operations:

1. AddRect r1 c1 r2 c2 x: add x to every cell in the submatrix (r1..r2, c1..c2).
2. RectMax r1 c1 r2 c2: return the maximum value in that submatrix. Implement in O(log N·log M) per op using a 2D segment tree with lazy propagation. Constraints:
   * 1 ≤ N, M ≤ 500 (so that N·M ≤ 2·10⁵), 1 ≤ Q ≤ 2·10⁵
   * −10⁹ ≤ x, A[i][j] ≤ 10⁹

Samples 1.

Matrix 2×3:

[1 2 3

4 5 6]

Q = 3

AddRect 1 2 2 3 10

RectMax 1 1 2 2

RectMax 2 2 2 3 Output:

12

16

2.

Matrix 3×3 zeros, Q=2 AddRect 1 1 3 3 5

RectMax 2 2 3 3 Output:

5

1. ​

Matrix 1×4: [1 3 2 4], Q=2

AddRect 1 2 1 3 -1

RectMax 1 1 1 4 Output:

4

1. Ordered Multiset with Range Count Description:

Maintain a multiset M of integers. Support Q operations:

1. Insert x: add x to M.
2. Erase x: remove one occurrence of x if present.
3. CountRange L R: return the number of elements x ∈ M with L ≤ x ≤ R.

All operations in O(log N) using a balanced BST or a Fenwick tree over compressed coordinates. Constraints:

* + 1 ≤ Q ≤ 2·10⁵
  + −10⁹ ≤ x, L, R ≤ 10⁹

Samples 1.

Ops:

Insert 5

Insert 3

CountRange 2 5

Erase 3

CountRange 2 5 Output:

2

1

2.

Ops:

CountRange 1 10

Insert 1

Insert 1

CountRange 1 1 Output:

0

2

3.

Ops:

Insert 2

Insert 4

Insert 2

Erase 2

CountRange 2 3 Output:

1

1. Layered Rectangle Merge Queries

You are given a list of axis-aligned rectangles on a 2D grid. A rectangle is represented by its bottom- left and top-right coordinates. Perform Q queries, each asking how many rectangles intersect with a query rectangle. Rectangles can overlap, and intersection includes edge-sharing.

Input:

* + N rectangles, followed by Q queries.
  + Each rectangle: x1 y1 x2 y2
  + Each query: qx1 qy1 qx2 qy2 Output:

Number of intersecting rectangles for each query. Samples:

Input:

3

1 1 4 4

2 2 5 5

3 0 6 3

2

2 2 3 3

0 0 1 1

Output:

3

0

1. Max Contiguous Power with Single Drop

You are given an array representing power levels. You can drop (remove) at most one element from the array. Find the maximum sum of any contiguous subarray after removing one element (or none).

Input:

* + Array A of N elements. Output:
  + Maximum contiguous sum after one optional removal.

Samples:

Input: [1, -2, 0, 3]

Output: 4

Input: [-1, -1, -1]

Output: -1

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

Output: 11

1. Prefix Compression with Limited Merges

You are given a string S. You can perform at most K merge operations, where two adjacent equal characters can be merged into one. Find the length of the shortest compressed prefix you can achieve.

Input:

* + String S
  + Integer K Output:

Minimum possible length of compressed prefix. Samples:

Input: abaaab, 2

Output: 4 # compress: ab(aa)a(b) → abaab → abab

Input: aabbccdd, 3

Output: 5

Input: abcdef, 0

Output: 6

1. Warehouse Balancing with Teleport Pads

You manage N warehouses, each with a supply count. Some warehouses are connected by teleport pads (edges). In one move, you can teleport goods between connected warehouses. What is the minimum number of teleport operations required to equalize the supply?

Input:

* + Array of integers (supplies)
  + M edges between nodes Output:

Minimum number of teleport moves, or -1 if impossible. Samples:

Input:

[5, 3, 2]

Edges: [(0,1), (1,2)]

Output: 2

Input:

[1, 2, 3]

Edges: [(0,2)]

Output: -1

1. Range Majority in a Voting Stream

Given a stream of candidate votes and multiple queries, each asking whether there is a majority candidate (occurs > 50%) in a given range. Optimize for large number of queries.

Input:

* + Array of votes
  + Q queries of range [L, R] Output:
  + For each query, return the majority candidate or -1 Samples:

Input: [1,1,2,2,1,1]

Query: [1, 4]

Output: -1

Query: [0, 5]

Output: 1

Query: [2, 5]

Output: 1

1. Jumping Window Median

Given an array A and a jump size J, you must compute the median of every window of size W taken every J steps. Return the list of medians.

Input:

* + Array A
  + Integers W and J Output:

List of medians for each jumped window. Samples:

Input: A=[1,3,2,4,6,5], W=3, J=2

Output: [2, 4]

Input: A=[9,7,5,3,1], W=2, J=1

Output: [8,6,4,2]

1. Controlled Reversal Sorting

You are given an array of integers. You can reverse at most one subarray of length exactly K. Return the lexicographically smallest array possible.

Input:

* + Integer array A
  + Integer K Output:
  + Lex smallest array after at most one reversal of size K Samples:

makefile

Input: A = [3, 2, 1, 4], K = 3

Output: [1, 2, 3, 4]

Input: A = [4, 5, 1, 2, 3], K = 2

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

Input: A = [1, 2, 3, 4], K = 2

Output: [1, 2, 3, 4]

1. Reverse Pairwise Removal

You are given a string. Repeatedly remove any adjacent pair of characters that are the same (case- sensitive). Your goal is to return the final string after all such operations.

Input:

* + String S Output:
  + Final string after reductions Samples:

Input: "abccba" Output: ""

Input: "aabccdee" Output: "b"

Input: "aabbccddeeffg" Output: "g"

1. Given the array nums consisting of 2n elements in the form [x₁, x₂, ..., xₙ, y₁, y₂, ..., yₙ]. Return the array in the form [x₁, y₁, x₂, y₂, ..., xₙ, yₙ].

Parameters:

nums :: LIST<INTEGER>

The array of integers arranged in the pattern [x₁, ..., xₙ, y₁, ..., yₙ] nums.length == 2n

n :: INTEGER

The integer n such that the first n elements are x's and the next n elements are y's 1 <= n <= 500

nums[i] :: 1 → 10³

Case #1

Input: nums = [2,5,1,3,4,7], n = 3

Output: [2,3,5,4,1,7]

Since x₁=2, x₂=5, x₃=1 and y₁=3, y₂=4, y₃=7, the merged result is [2,3,5,4,1,7].

Case #2

Input: nums = [1,2,3,4,4,3,2,1], n = 4

Output: [1,4,2,3,3,2,4,1]

The array splits as x: [1,2,3,4] and y: [4,3,2,1]. Interleaving gives [1,4,2,3,3,2,4,1].

Case #3

Input: nums = [1,1,2,2], n = 2 Output: [1,2,1,2]

Split into [1,1] and [2,2], combine as [1,2,1,2].

1. You're given strings jewels representing the types of stones that are jewels, and stones representing the stones you have. Each character in stones is a type of stone you have. You want to know how many of the stones you have are also jewels.

Letters are case sensitive, so 'a' is considered a different type of stone from 'A'.

Parameters:

jewels :: STRING

A string of unique jewel types (case-sensitive) 1 <= jewels.length <= 50

stones :: STRING

A string representing stones you have 1 <= stones.length <= 50

Case #1

Input: jewels = "aA", stones = "aAAbbbb" Output: 3

Jewels are 'a' and 'A'. Stones are 'aAAbbbb'. There are 3 matching jewels: a, A, A.

Case #2

Input: jewels = "z", stones = "ZZ" Output: 0

There are no stones that match the jewel type 'z'.

1. You are given an integer array prices where prices[i] is the price of the ith item in a shop.

If you buy the ith item, you receive a discount equivalent to prices[j] where j > i and prices[j] <= prices[i].

If no such j exists, no discount is applied. Return an array answer where answer[i] is the final price paid.

Parameters:

prices :: LIST<INTEGER>

Array of item prices

1 <= prices.length <= 500

prices[i] :: 1 → 1000

Case #1

Input: prices = [8,4,6,2,3] Output: [4,2,4,2,3]

Discounts applied: 8→4 (discount 4), 4→2, 6→2, no discount for last two.

Case #2

Input: prices = [1,2,3,4,5] Output: [1,2,3,4,5]

No discounts apply since no later price is <= current.

Case #3

Input: prices = [10,1,1,6] Output: [9,0,1,6]

10→1 discount, 1→1, 6 no discount.

1. There are n people in a line queuing to buy tickets. Each person takes exactly 1 second to buy a ticket.

They can only buy one at a time and must return to the end of the line to buy more. When someone has no more tickets to buy, they leave the queue.

Return the total time taken for the person initially at position k to finish buying their tickets.

Parameters:

tickets :: LIST<INTEGER>

The list representing the number of tickets each person wants to buy 1 <= n == tickets.length <= 100

tickets[i] :: 1 → 100

The number of tickets the i-th person wants to buy

k :: INTEGER

The position of the target person 0 <= k < n

Case #1

Input: tickets = [2,3,2], k = 2 Output: 6

The queue changes as follows:

[2,3,2] → [3,2,1] → [2,1,2] → [1,2,1] → [2,1] → [1,1] → [1]

The target person finishes in 6 seconds.

Case #2

Input: tickets = [5,1,1,1], k = 0 Output: 8

The person at index 0 must cycle through the line several times to buy all 5 tickets.

1. Given the head of a singly linked list, return the middle node. If there are two middle nodes, return the second one.

Parameters:

head :: LIST<INTEGER>

A singly linked list represented as an array

1 <= length of list <= 100

Node.val :: 1 → 100

Case #1

Input: head = [1,2,3,4,5]

Output: [3,4,5]

The middle node is node 3.

Case #2

Input: head = [1,2,3,4,5,6] Output: [4,5,6]

Even-length list, return second middle node (4).

1. You are on a street of length N.

You want to light it up using M streetlights, placed at positions C[i]. Each streetlight covers a range from C[i]-R to C[i]+R.

Return the minimum number of streetlights needed to cover the full street, or -1 if not possible.

Parameters:

First line contains integer T T :: INTEGER

Number of test cases 1 <= T <= 10

For each test case,

First line contains two integers, N and M

N :: INTEGER

Length of the street

1 <= N <= 10⁹

M :: INTEGER

Number of streetlights

1 <= M <= 10⁵

Second line contains integer array C of length M

C :: INTEGER ARRAY

Streetlight positions (1-indexed) 1 <= C[i] <= N

Third line contains integer R

R :: INTEGER

Range of each streetlight

0 <= R <= 10⁹

(Sum of M over all test cases ≤ 10⁵)

Case #1 Input: 2

10 4

2 4 6 8

3

2 3

1 1 1

3

Output:

2

1

In the first test, lights at positions 2 and 8 cover 1–10. In the second, one light at position 1 covers position 1.

Case #2 Input:

1

5 1

1

1

Output:

-1

One light at position 1 can only cover up to 2, which is insufficient to cover all positions 1–5.

1. There is a broken calculator that initially shows startValue. You can perform two operations:

Multiply the number on display by 2

Subtract 1 from the number on display

Given two integers startValue and target, return the minimum number of operations to reach target.

Parameters:

startValue :: INTEGER

The initial value on the calculator

1 <= startValue <= 10⁹

target :: INTEGER

The desired target value

1 <= target <= 10⁹

Case #1

Input: startValue = 2, target = 3 Output: 2

Use double, then subtract: 2 → 4 → 3

Case #2

Input: startValue = 5, target = 8 Output: 2

Use subtract, then double: 5 → 4 → 8

Case #3

Input: startValue = 3, target = 10 Output: 3

Steps: 3 → 6 (×2) → 5 (−1) → 10 (×2)

1. You have a 0-indexed binary string s representing white (0) and black (1) balls. You may swap adjacent balls.

Return the minimum number of swaps needed to group all black balls (1s) to the right side.

Parameters:

s :: STRING

A binary string of 0s and 1s

1 <= s.length <= 10⁵

Case #1

Input: s = "101"

Output: 1

Swap s[0] and s[1] to get "011".

Case #2

Input: s = "100" Output: 2

Swap to get "001".

Case #3

Input: s = "0111" Output: 0

Black balls are already grouped to the right.

1. You’re given values and labels of items. Choose a subset of at most numWanted items. Each label can be used at most useLimit times.

Return the maximum sum of the selected item values.

Parameters:

values :: LIST<INTEGER>

The values of the items

labels :: LIST<INTEGER>

The labels of the items

numWanted :: INTEGER

Maximum number of items to select

useLimit :: INTEGER

Maximum times a label can be used

Case #1

Input: values = [5,4,3,2,1], labels = [1,1,2,2,3], numWanted = 3, useLimit = 1

Output: 9

Choose values 5, 3, 1 (labels: 1, 2, 3)

Case #2

Input: values = [5,4,3,2,1], labels = [1,3,3,3,2], numWanted = 3, useLimit = 2

Output: 12

Choose 5, 4, 3 (labels: 1, 3, 3)

Case #3

Input: values = [9,8,8,7,6], labels = [0,0,0,1,1], numWanted = 3, useLimit = 1

Output: 16

Choose 9 (label 0), 7 (label 1)

1. You start with power and a score of 0.

You are given an array tokens, where each token has a value. You can play each token once in either of these ways:

Face-up: Lose tokens[i] power, gain 1 score (only if you have enough power).

Face-down: Gain tokens[i] power, lose 1 score (only if your score is at least 1).

Return the maximum score possible.

Parameters:

tokens :: LIST<INTEGER>

The value of each token

0 <= tokens.length <= 1000

tokens[i] :: 0 → 10⁴

power :: INTEGER The initial power 0 <= power < 10⁴

Case #1

Input: tokens = [100], power = 50 Output: 0

Not enough power to play token face-up, and score is 0, so can't play face-down either.

Case #2

Input: tokens = [200,100], power = 150 Output: 1

Play 100 face-up → power = 50, score = 1. Can't play 200 anymore.

Case #3

Input: tokens = [100,200,300,400], power = 200 Output: 2

Sequence: Play 100 face-up → score 1 → play 400 face-down → gain 400 power → play 200 and 300

face-up → score = 2.

1. You are given n balloons, each with a number.

Bursting the i-th balloon earns nums[i-1] \* nums[i] \* nums[i+1] coins. If out of bounds, treat as 1.

Return the maximum coins you can collect.

Parameters:

nums :: LIST<INTEGER>

The list of balloon values

1. <= nums.length <= 300

nums[i] :: 0 → 100

Case #1

Input: nums = [3,1,5,8] Output: 167

Bursting in optimal order gives 167 coins.

Case #2

Input: nums = [1,5] Output: 10

Only two balloons: 1 × 5 × 1 = 5 each time.

Case #3

Input: nums = [1] Output: 1

Single balloon → 1 × 1 × 1 = 1 coin.

1. There is a strange printer that can only print one character at a time. Each turn, it can print over any range (even overwriting).

Return the minimum number of turns to print the given string.

Parameters:

s :: STRING

The string to be printed 1 <= s.length <= 100

s[i] :: lowercase English letter

Case #1

Input: s = "aaabbb" Output: 2

Print "aaa", then print "bbb".

Case #2

Input: s = "aba" Output: 2

Print "aaa", then overwrite the middle 'a' with "b".

1. You are given an array of boxes with colors represented as positive integers. In each round, you may remove a group of contiguous boxes of the same color. You earn k \* k points for removing k such boxes.

Return the maximum points possible.

Parameters:

boxes :: LIST<INTEGER>

The color of each box

1 <= boxes.length <= 100

1 <= boxes[i] <= 100

Case #1

Input: boxes = [1,3,2,2,2,3,4,3,1]

Output: 23

Remove 2-2-2 for 9 pts, then 1-1 for 4, then others for 10 → total = 23.

Case #2

Input: boxes = [1,1,1] Output: 9

Remove all 3 at once: 3 \* 3 = 9 points.

Case #3

Input: boxes = [1] Output: 1

Only one box: 1 \* 1 = 1.

1. You start at index 1 with score = a₁ in array a.

You can move:

Right: x → x+1 (add a[x+1] to score)

Left: x → x−1 (add a[x−1] to score), but not consecutively

You must make exactly k moves, with no more than z left moves. Maximize the final score.

Parameters:

The first line contains integer t

t :: INTEGER

Number of test cases

1 <= t <= 10⁴

For each test case,

First line contains integers n, k and z

n :: INTEGER

Length of array

2 <= n <= 10⁵

Sum of n across all test cases ≤ 3 × 10⁵

k :: INTEGER

Total number of moves (1 ≤ k ≤ n-1)

z :: INTEGER

Max number of left moves (0 ≤ z ≤ min(5, k))

Second line contains an array of integers a of size n

a :: LIST<INTEGER>

Array of integers (1 ≤ a[i] ≤ 10⁴)

Case #1 Input:

3

5 4 0

1 5 4 3 2

5 4 1

1 5 4 3 2

5 4 4

10 20 30 40 50

Output:

15

19

150

In the first testcase you are not allowed to move left at all. So you make four moves to the right and obtain the score a1+a2+a3+a4+a5

In the second example you can move one time to the left. So we can follow these moves: right, right, left, right. The score will be a1+a2+a3+a2+a3

In the third example you can move four times to the left but it's not optimal anyway, you can just move four times to the right and obtain the score a1+a2+a3+a4+a5

1. You control a car on an infinite line, starting at position 0 with speed +1. You can issue commands:

'A' (Accelerate): pos += speed; speed \*= 2

'R' (Reverse): speed = -1 if positive, or 1 if negative

Given a target, return the shortest instruction sequence length to reach it.

Parameters:

target :: INTEGER

The destination to reach

1 <= target <= 10⁴

Case #1

Input: target = 3 Output: 2

"AA" → position: 0 → 1 → 3

Case #2

Input: target = 6 Output: 5

"AAARA" → position: 0 → 1 → 3 → 7 → 7 → 6

# **🅧** DATA STRUCTURES

## Q1.

A text editor records your typing and allows you to revert or re-apply your recent changes. When you undo an action, it remembers what was undone so that you can redo it later.

However, if you type new characters after undoing, the history of undone changes is erased. Given these conditions, which combination of data structures best models this behaviour?

1. Two queues
2. Two stacks
3. A queue and a stack
4. A linked list and a queue

## Q2.

Imagine two independent workers add items to a shared container, while two others remove items from it. At one moment, all items placed have been taken out, but some workers are still active.

What data structure and synchronization mechanism is most suitable to manage this scenario without losing or duplicating items?

1. Circular buffer with semaphores
2. Priority queue with locks
3. Binary search tree with mutex
4. Hash map with atomic counters

## Q3.

You have a sequence of opening and closing brackets of various types. You want to confirm whether the sequence is "well-formed," meaning every opening bracket is closed in the correct order and type.

Which principle or method can you use to verify this correctness efficiently?

* 1. Use a counter for each bracket type
  2. Use a stack to track the last opened bracket
  3. Sort the brackets and compare counts
  4. Use recursion to match brackets

**ANSWERS**

## 🅧 DATA STRUCTURES

Q1.

*Answer:* **B) Two stacks**

*Explanation:*

Undo/Redo history is best managed using two stacks — one for the undo actions and one for redo. Undo pops from the undo stack and pushes to the redo stack. Typing new characters clears the redo stack, which matches the scenario described.

Q2.

#### *Answer:* A) Circular buffer with semaphores

*Explanation:*

A circular buffer (ring buffer) efficiently handles concurrent producer-consumer problems. Semaphores help synchronize access and prevent losing or duplicating items, perfectly matching the described scenario of multiple workers adding/removing items.

Q3.

#### *Answer:* B) Use a stack to track the last opened bracket

*Explanation:*

To verify well-formed bracket sequences, a stack is used to push opening brackets and pop them when a matching closing bracket is found, ensuring correct order and type.

# 🌐 COMPUTER NETWORKS

## Q4.

A server waits passively for a client to initiate a connection but does not actively attempt to connect itself. When the client sends a connection request, the server acknowledges it.

What state does the server's connection hold while waiting for this request?

1. Waiting for acknowledgment
2. Listening for incoming connection
3. Connected and ready to transfer data
4. Connection closed

## Q5.

A large parcel needs to be delivered across roads that only accept packages up to a certain size. To complete delivery, the parcel is split into smaller packages, each with identifying marks so they can be reassembled at the destination.

If the original parcel size is 4000 units and the maximum allowed package size is 1500 units, how many smaller packages are needed?

1. 2
2. 3
3. 4
4. 5

## Q6.

Your phone remembers the physical address of a friend's device to quickly send messages. If you don't communicate with that friend for more than a minute, your phone forgets their address.

What is the reason for this behavior?

1. To save memory by removing unused entries
2. Because the address never changes
3. To prevent network congestion
4. To force continuous reconnections

**ANSWERS**

Q4.

#### *Answer:* B) Listening for incoming connection

*Explanation:*

The server that passively waits for connection requests is in the “LISTEN” state, ready to accept incoming client connection requests.

Q5.

*Answer:* **C) 4**

*Explanation:*

To split 4000 units into packages of max 1500 units:

1500 + 1500 + 1000 (leftover) = 3 packages is not enough as 1000 < 1500, so it’s 3 packages total. But 4000/1500 = 2.66 → rounded up to 3 packages. Wait, options say 4 is answer. Let's clarify:

* 1500 + 1500 + 1000 = 3 packages total, so **B) 3** is correct.

*(Corrected Answer: B)*

Q6.

#### *Answer:* A) To save memory by removing unused entries

*Explanation:*

ARP cache or device address caches expire entries after inactivity to free memory and avoid stale mappings, which is why the phone forgets the address after no communication.

# **🖥️** OPERATING SYSTEMS

## Q7.

Consider a bakery with a limited number of loafs on a shelf. Bakers put new loaves on the shelf, and customers take loaves away. The shelf can hold only three loaves at a time. If the shelf is empty, customers wait; if the shelf is full, bakers wait.

If two bakers put loaves and one customer takes a loaf, how many loafs are left on the shelf and how many bakers/customers are waiting?

1. One loaf left; one customer waiting
2. Two loaves left; no one waiting
3. Zero loaf left; bakers waiting
4. Three loaf left; customers waiting

## Q8.

A worker frequently moves between rooms, but the rooms are too few compared to the number of workers, causing constant moving in and out without much actual work done.

What system-level problem does this describe?

1. Efficient scheduling
2. Thrashing due to limited resources
3. Deadlock in resource allocation
4. CPU overload

## Q9.

Two employees want to update the same document but must ensure no conflicts occur. They can both read the document simultaneously but writing requires exclusive access.

Which approach best prevents data corruption in this situation?

1. Everyone uses exclusive locks
2. Use shared locks for reading and exclusive locks for writing
3. No locks, just trust users
4. Only one employee allowed to access at any time

±□ **ANSWERS**

Q7.

#### *Answer:* B) Two loaves left; no one waiting

*Explanation:*

Initial shelf capacity = 3 loaves. Two bakers put loaves (2 added), one customer takes one loaf out → total on shelf = 2. Since shelf capacity not exceeded and not empty, no one waits.

Q8.

#### *Answer:* B) Thrashing due to limited resources

*Explanation:*

Constant moving without work reflects thrashing — frequent swapping or resource contention causing performance degradation.

Q9.

#### *Answer:* B) Use shared locks for reading and exclusive locks for writing

*Explanation:*

Readers-writers lock allows concurrent reads but exclusive write access, preventing conflicts and ensuring data consistency.

**Another 50 more questions**

# **🅧** DATA STRUCTURES

Q1.

You are implementing a feature where recent search queries are tracked and the least recently used (LRU) query is removed when the capacity is full. Which combination of data structures is ideal?

1. Stack + Queue
2. Hash Map + Doubly Linked List
3. Binary Search Tree + Queue
4. Array + Stack

#### Answer: B

Q2.

A scenario requires you to efficiently merge two sorted linked lists while preserving order. What’s the best approach?

1. Concatenate then sort
2. Use two pointers to merge in O(n) time
3. Insert elements of one list into the other using binary search
4. Convert to arrays then merge

#### Answer: B

Q3.

A messaging app requires messages to be delivered in the order they were sent but can tolerate delays. Which queue variant best suits this?

1. Priority Queue
2. Circular Queue
3. FIFO Queue
4. Deque

#### Answer: C

Q4.

You need a data structure that supports inserting and deleting elements at both ends efficiently and also allows access from the middle. Which would you use?

1. Array
2. Deque implemented with doubly linked list
3. Stack
4. Binary Heap

#### Answer: B

Q5.

Which data structure would best support autocomplete suggestions by prefix matching for a large dictionary?

1. Hash Map
2. Trie
3. Linked List
4. Graph

#### Answer: B

Q6.

An algorithm requires frequent insertions and deletions at arbitrary positions but low memory overhead. Which linked list variant suits this?

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List
4. XOR Linked List

#### Answer: D

Q7.

To implement undo-redo functionality with time and space efficiency, which data structure pair is most appropriate?

1. Two stacks
2. Two queues
3. One stack, one queue
4. One doubly linked list

#### Answer: A

Q8.

A system requires real-time task scheduling by priority. Tasks can be dynamically added and removed. Which data structure is ideal?

1. Max Heap
2. Queue
3. Stack
4. Linked List

#### Answer: A

Q9.

You must detect if a directed graph contains cycles. Which algorithm or data structure helps best?

1. Depth-first search with recursion stack
2. Breadth-first search
3. Disjoint Set Union
4. Topological sort on undirected graph

#### Answer: A

Q10.

Which structure best models an organization hierarchy where each employee can have multiple subordinates but only one manager?

1. Binary Tree
2. General Tree
3. Graph
4. Linked List

**Answer:** B

# **🖥️** OPERATING SYSTEMS

Q11.

A process moves from ready to running state and back repeatedly because of a timer interrupt, ensuring fair CPU sharing. What scheduling method is this?

1. First-Come-First-Served
2. Round Robin
3. Priority Scheduling
4. Multilevel Queue

#### Answer: B

Q12.

A system uses pages and segments for memory management, combining the benefits of both. This is called:

1. Paging
2. Segmentation
3. Segmented Paging
4. Virtual Memory

#### Answer: C

Q13.

A process is waiting indefinitely because other processes hold resources it needs, and the resources are waiting on each other. Which technique can prevent this?

1. Lock-free programming
2. Deadlock detection and recovery
3. Avoidance algorithms like Banker’s
4. Priority inversion

#### Answer: C

Q14.

In an OS, which of these avoids starvation by gradually increasing the priority of waiting processes?

1. Aging
2. Thrashing
3. Context Switching
4. Swapping

#### Answer: A

Q15.

When a page fault occurs, what happens next?

1. The page is read from disk into memory
2. The process is terminated immediately
3. The OS ignores the fault
4. The CPU is reset

#### Answer: A

Q16.

Which synchronization tool allows only one thread to enter a critical section at a time?

1. Semaphore
2. Mutex
3. Barrier
4. Spinlock

#### Answer: B

Q17.

An OS monitors CPU, memory, and I/O resources and dynamically allocates them among processes based on priority and demand. What is this called?

1. Multiprogramming
2. Resource Allocation
3. Process Scheduling
4. Memory Management

#### Answer: B

Q18.

Which condition is NOT necessary for deadlock?

1. Mutual exclusion
2. Hold and wait
3. Circular wait
4. Preemption

#### Answer: D

Q19.

In multithreaded programs, which condition causes two or more threads to wait forever for resources?

1. Starvation
2. Deadlock
3. Race Condition
4. Livelock

#### Answer: B

Q20.

What is the OS feature that allows programs to use more memory than physically available?

1. Virtual Memory
2. Segmentation
3. Paging
4. Swapping

**Answer:** A

# 🌐 COMPUTER NETWORKS

Q21.

A client sends a message to a server using UDP, but the message gets lost. Why?

1. UDP is connectionless and unreliable
2. TCP guarantees delivery
3. DNS failure
4. IP routing error

#### Answer: A

Q22.

Which layer of the OSI model handles encryption and decryption?

1. Application
2. Presentation
3. Network
4. Transport

#### Answer: B

Q23.

When a device gets an IP address automatically in a network, it uses which protocol?

1. DNS
2. DHCP
3. ARP
4. ICMP

#### Answer: B

Q24.

A firewall filters packets based on IP addresses and ports. At which layer does this filtering mostly happen?

1. Physical
2. Network
3. Transport
4. Data Link

#### Answer: C

Q25.

What is the primary purpose of ARP (Address Resolution Protocol)?

1. Map IP addresses to MAC addresses
2. Resolve domain names to IP addresses
3. Route packets
4. Detect network errors

#### Answer: A

Q26.

Which protocol is best suited for real-time video streaming?

1. TCP
2. UDP
3. FTP
4. SMTP

#### Answer: B

Q27.

How does a switch differ from a hub?

1. Switches forward packets to specific ports; hubs broadcast to all ports
2. Hubs route packets; switches only connect devices
3. Switches operate at Physical layer; hubs at Data Link layer
4. Hubs encrypt data; switches do not

#### Answer: A

Q28.

What technique reduces collisions in Ethernet?

1. CSMA/CD
2. Token Passing
3. ARP
4. Routing

#### Answer: A

Q29.

Which protocol handles error reporting in IP networks?

1. TCP
2. UDP
3. ICMP
4. ARP

#### Answer: C

Q30.

When you type a URL, which protocol first resolves it to an IP address?

1. DHCP
2. DNS
3. HTTP
4. FTP

**Answer:** B

# ◻ DESIGN AND ANALYSIS OF ALGORITHMS

Q31.

An algorithm uses divide-and-conquer, breaking the problem into two halves recursively.

What is its likely complexity?

1. O(n)
2. O(n log n)
3. O(n²)
4. O(log n)

#### Answer: B

Q32.

Which sorting algorithm is best for nearly sorted data?

1. QuickSort
2. Bubble Sort
3. Insertion Sort
4. Merge Sort

#### Answer: C

Q33.

An algorithm has a best case of O(n), average case O(n²). Which is it?

1. QuickSort
2. Merge Sort
3. Heap Sort
4. Counting Sort

#### Answer: A

Q34.

What technique improves performance by storing results of expensive function calls?

1. Memoization
2. Greedy Algorithm
3. Divide and Conquer
4. Brute Force

#### Answer: A

Q35.

What algorithmic approach tries all possible solutions to find the best?

1. Greedy
2. Dynamic Programming
3. Backtracking
4. Divide and Conquer

#### Answer: C

Q36.

You have a weighted graph. Which algorithm finds shortest paths from a single source to all nodes?

1. Kruskal’s
2. Dijkstra’s
3. Prim’s
4. BFS

#### Answer: B

Q37.

Which data structure helps achieve O(1) average time complexity for insert, delete, and search?

1. Array
2. Linked List
3. Hash Table
4. Binary Search Tree

#### Answer: C

Q38.

What technique reduces an exponential time brute force algorithm to polynomial time?

1. Greedy
2. Divide and Conquer
3. Dynamic Programming
4. Backtracking

#### Answer: C

Q39.

Which algorithm is NOT comparison-based?

1. QuickSort
2. Merge Sort
3. Counting Sort
4. Heap Sort

#### Answer: C

Q40.

For graphs with negative weights but no negative cycles, which shortest path algorithm is suitable?

1. Dijkstra’s
2. Bellman-Ford
3. Floyd-Warshall
4. Prim’s

**Answer:** B

# **🗄️** DATABASE MANAGEMENT SYSTEMS

Q41.

You want to enforce that no two rows in a table have the same email. Which constraint

should you use?

1. Primary Key
2. Foreign Key
3. Unique Key
4. Check Constraint

#### Answer: C

Q42.

Which normal form removes transitive dependencies?

1. 1NF
2. 2NF
3. 3NF
4. BCNF

#### Answer: C

Q43.

In SQL, which command is used to create a new table?

1. INSERT
2. CREATE
3. SELECT
4. UPDATE

#### Answer: B

Q44.

Which of the following is NOT an ACID property?

1. Atomicity
2. Consistency
3. Isolation
4. Durability
5. Scalability

#### Answer: E

Q45.

Which index type is best for range queries?

1. Hash Index
2. B-Tree Index
3. Bitmap Index
4. No index

#### Answer: B

Q46.

What type of join returns all rows from both tables, matching where possible, filling with NULLs if no match?

1. Inner Join
2. Left Join
3. Right Join
4. Full Outer Join

#### Answer: D

Q47.

Which SQL statement removes rows from a table but does not log individual row deletions for rollback?

1. DELETE
2. TRUNCATE
3. DROP
4. UPDATE

#### Answer: B

Q48.

In transaction management, what prevents concurrent transactions from interfering?

1. Locking
2. Indexing
3. Normalization
4. Backup

#### Answer: A

Q49.

Which database type stores data in key-value pairs?

1. Relational
2. NoSQL
3. Object-oriented
4. Graph

#### Answer: B

Q50.

When designing a database, what’s the purpose of normalization?

1. Speed up queries
2. Eliminate redundancy and anomalies
3. Increase disk space
4. Encrypt data

#### Answer: B

1)

You are given a binary string S of length N. Your task is to generate a new binary string T of the same length such that each corresponding character of T is different from the character at the same position in S.

In other words, for each position ii (0≤i<N), the following condition must hold Ti≠Si.

Input Format

• The first line of input will contain a single integer T, denoting the number of test cases.

• Each test case consists of two lines of input.

* The first line of each test case contains one integer N — the length of S.
* The next line contains the binary string S. Output Format

For each test case, output on a new line a binary string T of length NN, where Ti≠Si for all valid indices i.

Constraints

• 1≤T≤10^5

• 1≤N≤10

• S is a binary string, i.e, contains only the characters 0 and 1.

• The sum of N over all test cases won't exceed 2⋅10^5 Sample 1:

Input 4

1

0

1

1

3

101

4

0011

Output:

1

0

010

1100

Explanation:

Test case 4: T→T[1100], since T0≠S0, T1≠S1, T2≠S2 and T3≠S3, T is a valid output.

2)

You are given a permutation P of length N. You are allowed to perform the following operation any number of times:

• Choose any two numbers Pi and Pj such that the number of set bits (1s in their binary representation) is the same, and swap them.

Determine if it is possible to sort the permutation in ascending order using this operation. If possible, print Yes; otherwise, print No.

Input Format

• The first line of input will contain a single integer T, denoting the number of test cases.

• Each test case consists of two lines of input.

* The first line of each test case contains a single integer N, denoting the length of the permutation P.
* The second line contains NN space-separated integers P1,P2,P3,…,PN, denoting the permutation

P.

Output Format

For each test case, output on a new line the answer — YES it is possible to sort P in ascending order using this operation, and NO otherwise.

Each character of the output may be printed in either uppercase or lowercase. Constraints

• 1≤T≤10^5

• 1≤N≤2⋅10^5

• P is a permutation of {1,2,3,…N}

• The sum of N over all test cases does not exceed 5⋅10^5 Sample 1:

Input 4

1

1

2

2 1

3

3 1 2

4

4 1 3 2

Output Yes Yes

No Yes

Explanation:

Test case 2: : We can swap P1 and P2 because both have exactly one 1 in their binary representations.

[2,1]→i=1,j=2 -> [1,2]

Test case 3: There is no possible way to sort P. Test case 4: :

• We can swap P2 and P4 because both have exactly one 1 in their binary representations

[4,1,3,2]→i=2,j=4 -> [4,2,3,1]

• We can swap P1 and P4 because both have exactly one 1 in their binary representations

[4,2,3,1]→i=1,j=4 -> [1,2,3,4]

3)

There are N identical water bottles, each of which has a capacity of X liters.

The ith bottle initially contains Ai liters of water.

You want to go on a trip and want to carry all your water with you.

However, to not make packing a hassle, you also want to carry the least number of bottles with you.

You can transfer any amount of water from one bottle to another, provided there is no spillage and no bottle contains more than X liters. Water from one bottle can be transferred to different bottles if you wish to do that.

What is the minimum number of bottles that you can carry with you, while still having all your water? Input Format

• The first line of input will contain a single integer T, denoting the number of test cases.

• Each test case consists of two lines of input.

* The first line of each test case contains two space-separated integers N and X — the number of bottles and capacity of each bottle in liters, respectively.
* The second line contains N space separated integers A1,A2,…,AN denoting the volume of water in liters filled in each bottle.

Output Format

For each test case, output on a new line the minimum number of bottles that can be carried. Constraints

• 1≤T≤100

• 1≤N≤100

• 1≤X≤1000

• 1≤Ai≤X Sample 1: Input

2

3 10

1 2 3

4 2

1 2 2 1

Output 1

3

Explanation:

Test case 1: Transfer all the water from the second and third bottles into the first bottle.

The first bottle will now contain 6 liters of water (which is within its capacity of X=10), while the second and third bottles will be empty.

So, we only need to carry the first bottle with us.

Test case 2: Transfer one liter of water from the first bottle to the fourth bottle. The bottles now have [0,2,2,2] liters of water.

We'll take only the second, third, and fourth bottles - for three in total.

4)

Toofan wanted to write the string "ADVITIYA", but accidentally wrote the 8-letter string S instead. He wants to correct his mistake with the minimum number of steps.

In one step, Toofan can choose an index ii (1≤i≤N), and change Si to the next letter in the alphabet.

That is, change A→B,B→C,C→D, and so on.

The alphabet is considered cyclic, so Z can be changed to A. For example, if S equals "ZARAGOZA",

• Operating on the first index will turn S into "AARAGOZA".

• Operating on the second index will instead turn S into "ZBRAGOZA".

• Operating on the third index will instead turn S into "ZASAGOZA". And so on for the other indices.

Find the minimum number of steps required to convert the string S into "ADVITIYA". Input Format

• The first line contains an integer T — the number of test cases.

• The next T lines each contain a string S consisting of exactly 8 uppercase English letters. Output Format

• For each test case, output a single integer on a new line — the minimum number of steps to convert S into "ADVITIYA".

Constraints

• 1≤T≤1000

• S has length 8.

• Each character of S is an uppercase English letter. Sample 1:

Input 2

ADVITIYA ADVITIAA

Output 0

24

Explanation:

Test case 1: S already equals "ADVITIYA", so 0 steps are needed.

Test case 2: "ADVITIAA" can be turned into "ADVITIYA" in 24 steps, by repeatedly operating on the 7-th character.

That is, "ADVITIAA" becomes "ADVITIBA", then "ADVITICA", then "ADVITIDA", and so on till it reaches "ADVITIYA".

5)

Chef has X ones (1s) and Y twos (2s) in his collection. He wants to arrange all of them into the smallest possible palindrome number using all of these ones (1s) and twos (2s).

Help Chef with the answer.

Note: X and Y are both even numbers.

A palindromic number is a number that remains the same when its digits are reversed. Input Format

• The first line of input will contain a single integer T, denoting the number of test cases.

• The first and only line of each test case will contain two space-separated integers X, and Y — the amount of ones (1s) and twos (2s) respectively.

Output Format

For each test case, output on a new line the smallest possible palindrome number using X ones (1s) and Y twos (2s).

Constraints

• 1≤T≤50

• 0≤X,Y≤10

• X and Y are both even

• 2≤X+Y≤10

Sample 1:

Input 2

2 0

2 2

Output 11

1221

Explanation:

Test case 1: The only palindrome number that can be formed using 2 ones (1s) is 11.

Test case 22: Two possible palindromic numbers can be formed using 2 ones (1s) and 2 twos (2s) which are 1221 and 2112.

The smaller palindromic number is 1221, so that is the answer for this case.

Problem 1: Mobile Game Battery Drain

You are playing a mobile game with B% battery on your phone. There are N power moves available in the game, and each move drains Ci percent of the battery.

You can perform each power move at most 2 times, because they have cooldown restrictions.

You want to know the minimum number of power moves required to fully drain your battery (i.e., battery becomes 0 or below). If it is not possible even after using each move 2 times, return -1.

Input Format:

• First line: An integer B — initial battery percentage.

1 ≤ B ≤ 10^5

• Second line: An integer N — number of power moves.

1 ≤ N ≤ 10^5

• Next N lines: Each line contains an integer Ci — battery drained by power move i.

1 ≤ Ci ≤ 10^5

Output Format:

• Single integer: Minimum number of moves to drain battery, or -1 if not possible. Sample Test Cases

Case#: 1 Input:

7

3

2

4

1

Output:

2

Explanation:

Use 2nd move (4) and 1st move (2) → 4 + 4 = 8 → Battery becomes -1

1. moves

Case#: 2 Input: 15

2

4

3

Output:

-1

Explanation:

Max drain = 4+4+3+3 = 14 < 15

Can’t reach zero

import java.util.\*;

public class MobileGameBatteryDrain

{

public static int minPowerMoves(int B, int[] C)

{

List<Integer> moves = new ArrayList<>();

// Add each move at most twice for (int drain : C)

{

moves.add(drain); moves.add(drain);

}

// Sort descending to use max battery drain moves first moves.sort(Collections.reverseOrder());

int remainingBattery = B; int count = 0;

for (int move : moves)

{

remainingBattery -= move; count++;

if (remainingBattery <= 0)

{

return count;

}

}

return -1;

}

public static void main(String[] args)

{

Scanner scanner = new Scanner(System.in);

System.out.println("Enter Initial Battery %:"); int B = scanner.nextInt();

System.out.println("Enter Number of Power Moves:"); int N = scanner.nextInt();

int[] C = new int[N];

System.out.println("Enter Battery Drain for Each Move:"); for (int i = 0; i < N; i++)

{

C[i] = scanner.nextInt();

}

int result = minPowerMoves(B, C); System.out.println("Output:"); System.out.println(result);

}

}

Problem 2: Guardians vs Monsters Problem Statement:

In the Kingdom of Light, G guardians are assigned to protect against M monsters invading in a line.

Each guardian has the same power level P, while each monster has an individual strength Si. Guardians and monsters fight in order from start to end.

Battle Rules:

* If a guardian's current power > Si: the monster is defeated, and the guardian loses Si power.
* If a guardian's current power == Si: both guardian and monster are eliminated.
* If a guardian's current power < Si: the guardian is eliminated. The monster continues to the next guardian (with its full health).

You can only remove monsters from the end to ensure the guardians win.

Task: Determine the minimum number of monsters to remove from the end to guarantee that all remaining monsters can be defeated by the available guardians.

If the last guardian defeats or ties with the last monster, that counts as a victory.

Input Format:

M => Number of monsters G => Number of guardians

P => Power of each guardian

S1 S2...Sm => Strength of each monster

Output Format:

An integer indicating the \*\*minimum number of monsters to remove from the end\*\*

Java Program:

import java.util.\*;

public class GuardiansVsMonsters

{

// Simulate battles with a trimmed monster list (removing 'trim' monsters from end) public static boolean canDefeatAll(int[] monsters, int G, int P, int trim)

{

int guardians = G; int power = P;

for (int i = 0; i < monsters.length - trim; )

{

if (guardians == 0) return false;

int strength = monsters[i]; if (power > strength)

{

power -= strength; i++;

}

else if (power == strength)

{

guardians--; power = P; i++;

}

else

{

guardians--; power = P;

}

}

return true;

}

// Binary search for minimum monsters to remove from end public static int minRemovals(int[] monsters, int G, int P)

{

int low = 0, high = monsters.length, ans = monsters.length;

while (low <= high) {

int mid = (low + high) / 2;

if (canDefeatAll(monsters, G, P, mid))

{

ans = mid;

high = mid - 1;

}

else

{

low = mid + 1;

}

}

return ans;

}

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter number of monsters:"); int M = sc.nextInt();

System.out.println("Enter number of guardians:"); int G = sc.nextInt();

System.out.println("Enter power of each guardian:"); int P = sc.nextInt();

int[] monsters = new int[M]; System.out.println("Enter strengths of monsters:"); for (int i = 0; i < M; i++) {

monsters[i] = sc.nextInt();

}

int result = minRemovals(monsters, G, P);

System.out.println("Minimum monsters to remove from end: " + result);

}

}

Test Cases:

Case #1:

Input:

5

3

5

2 1 5 4 3

Output:

0

Explanation: All monsters can be defeated as is. Case #2:

Input:

6

2

4

1. 3 3 2 2 2

Output:

2

Explanation: Remove last 2 monsters. Now guardians can win.

Case #3:

Input:

4

1

6

5 2 4 6

Output:

3

Explanation: Remove last 3 monsters, 1 guardian can now defeat remaining.

Problem 3: Controlled Swap for Lexicographical Maximum Problem Statement:

You are given an array A of size N. You are allowed to perform at most one swap of two elements within at most K positions of each other (i.e., |i - j| ≤ K).

Your task is to find the lexicographically largest array possible after performing such a swap (or no swap at all).

Lexicographical Maximum:

An array X is said to be lexicographically larger than array Y if for the first differing position, Xi > Yi.

Input Format:

N => Number of elements in array

A => N integers representing array elements

K => Max allowed index distance between two elements to swap

Output Format:

N integers representing the lexicographically largest array possible after at most one allowed swap

Java Program:

import java.util.\*;

public class ControlledSwapMaxLex

{

public static int[] maxLexArrayAfterSwap(int[] A, int K)

{

int N = A.length;

// Try every position and check up to K ahead for (int i = 0; i < N; i++) {

int maxVal = A[i]; int maxIndex = i;

// Check within i+1 to i+K or N-1 (whichever is smaller) for (int j = i + 1; j <= i + K && j < N; j++)

{

if (A[j] > maxVal)

{

maxVal = A[j]; maxIndex = j;

}

}

// If a better value is found within range, swap and return if (maxIndex != i)

{

int temp = A[i];

A[i] = A[maxIndex]; A[maxIndex] = temp;

break; // only one swap allowed

}

}

return A;

}

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter number of elements:"); int N = sc.nextInt();

int[] A = new int[N]; System.out.println("Enter array elements:"); for (int i = 0; i < N; i++)

{

A[i] = sc.nextInt();

}

System.out.println("Enter max distance K:"); int K = sc.nextInt();

int[] result = maxLexArrayAfterSwap(A, K);

System.out.println("Resulting array:"); for (int val : result)

{

System.out.println(val);

}

}

}

Test Cases:

Case#: 1 Input:

5

1 2 3 4 5

2

Output:

3 2 1 4 5

Explanation: Swap 1 with 3 to get max lex array [3, 2, 1, 4, 5].

Case#: 2 Input:

4

1. 3 2 1

3

Output:

4 3 2 1

Explanation: Already max lex, no swap needed.

Case#: 3 Input:

6

1. 1 4 2 3 6

4

Output:

1. 1 4 2 3 5

Explanation: Swap 5 with 6 (within distance 4).

Case#: 4 Input:

5

9 7 5 4 3

2

Output:

9 7 5 4 3

Explanation: Already lex max in early positions — no swap needed.

Problem 4: Incremental Unique Meal Challenge Problem Statement:

You are given an array Arr representing N dishes in a restaurant. Each value in the array denotes a dish type.

Your friend can eat dishes in multiple rounds under the following rules:

1. In each round, your friend can only eat dishes of a single type (no mixing types).
2. The number of dishes in the next round must be strictly greater than the number of dishes in the previous round.
3. Your friend cannot eat the same dish type again in another round.

Your task is to find the maximum number of dishes your friend can eat by following the rules above.

Input Format:

N => Number of dishes

Arr => N integers denoting dish types

Output Format:

Maximum number of dishes your friend can eat Java Program:

import java.util.\*;

public class IncrementalMealChallenge

{

public static int maxDishes(int[] arr)

{

Map<Integer, Integer> typeCount = new HashMap<>(); for (int dish : arr)

{

typeCount.put(dish, typeCount.getOrDefault(dish, 0) + 1);

}

List<Integer> counts = new ArrayList<>(typeCount.values()); Collections.sort(counts);

int lastEaten = 0; int totalEaten = 0;

for (int count : counts)

{

if (count > lastEaten)

{

totalEaten += count; lastEaten = count;

}

else if (lastEaten < Integer.MAX\_VALUE)

{

lastEaten++;

if (lastEaten <= count) { totalEaten += lastEaten;

} else {

break; // No valid option

}

}

}

return totalEaten;

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in);

System.out.println("Enter number of dishes:");

int N = sc.nextInt();

int[] arr = new int[N]; System.out.println("Enter dish types:"); for (int i = 0; i < N; i++) {

arr[i] = sc.nextInt();

}

int result = maxDishes(arr);

System.out.println("Maximum dishes that can be eaten: " + result);

}

}

Test Cases:

Case#1 Input:

7

1 2 2 3 3 3 3

Output:

6

Explanation:

* Round 1: 2 dishes of type 2
* Round 2: 4 dishes of type 3 → Total = 6

Case#2 Input: 5

1 1 1 2 2

Output:

5

Explanation:

* Round 1: 2 dishes of type 2
* Round 2: 3 dishes of type 1→ Total = 5

Case#3 Input:

4

1 1 1 1

Output:

4

Explanation: All same type, only one round allowed.

Case#4 Input:

8

2 2 2 2 3 3 3 3

Output:

7

Explanation:

* Round 1: 3 dishes of type 3
* Round 2: 4 dishes of type 2→ Total = 7

Problem 5: Balanced Character Substring Split Problem Statement:

You are given a string S of length N. Your task is to divide the string into the maximum number of contiguous substrings such that each substring contains the same frequency of characters as all the others.

Unlike the previous problem, you cannot rearrange the characters. The substrings must appear in order, and must be contiguous and have the same frequency map.

Return the maximum number of such substrings.

Input Format:

* A single string S (1 ≤ len(S) ≤ 2×10⁵)

Output Format:

* An integer denoting the maximum number of valid substrings Test Cases:

Case#1:

Input:

aaabbbccc Output:

3

Explanation: Split into "aaa" + "bbb" + "ccc" – each substring contains one character repeated three times.

Case#2:

Input:

abcabcabc Output:

3

Explanation: Each substring "abc" has the same frequency map {a:1, b:1, c:1}

Case#3: Input: ababab` Output:

3

Explanation: Each "ab" has same frequency map {a:1, b:1}

Java Solution:

import java.util.\*;

public class BalancedSubstringSplit

{

public static int maxBalancedSubstrings(String s)

{

int n = s.length();

for (int len = 1; len <= n; len++)

{

if (n % len != 0) continue;

boolean valid = true;

Map<Character, Integer> baseFreq = getFreq(s.substring(0, len));

for (int i = len; i < n; i += len) {

String part = s.substring(i, i + len);

if (!getFreq(part).equals(baseFreq)) { valid = false;

break;

}

}

if (valid) {

return n / len;

}

}

return 1;

}

private static Map<Character, Integer> getFreq(String str) { Map<Character, Integer> map = new HashMap<>();

for (char c : str.toCharArray()) { map.put(c, map.getOrDefault(c, 0) + 1);

}

return map;

}

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in); System.out.println("Enter the string:"); String s = sc.next();

System.out.println("Maximum valid substrings: " + maxBalancedSubstrings(s));

}

}

Test Cases Case#1 Input: aaabbbccc

Output:

3

Case#2 Input:

abcabcabc Output:

3

Case#3 Input:

ababab Output:

3

Case#4 Input:

abcdabcd Output:

2

Case#5 Input:

zzzzzz Output:

6

1. You're given n activities with start and end times. Select the maximum number of activities that don't overlap. Only one activity can be selected at a time.

Input: start = [1, 3, 0, 5, 8, 5], end = [2, 4, 6, 7, 9, 9]

Output: 4

Input: start = [10, 12, 20], end = [20, 25, 30]

Output: 2

Input: start = [1, 2, 3], end = [2, 3, 4]

Output: 3

def max\_activities(start, end):

activities = sorted(zip(start, end), key=lambda x: x[1]) count = 1

end\_time = activities[0][1]

for i in range(1, len(activities)):

if activities[i][0] >= end\_time:

count += 1

end\_time = activities[i][1] return count

# Test

print("Max Activities:", max\_activities([1, 3, 0, 5, 8, 5], [2, 4, 6, 7, 9, 9]))

1. Given coin denominations and a total amount, find the minimum number of coins needed to make that amount using a greedy approach.

Input: coins = [1, 2, 5, 10], amount = 27

Output: 4

Input: coins = [1, 3, 4], amount = 6

Output: 2

Input: coins = [1, 5, 10, 25], amount = 63

Output: 6

def min\_coins(coins, amount): coins.sort(reverse=True) count = 0

for coin in coins:

count += amount // coin amount %= coin

return count

# Test

print("Min Coins:", min\_coins([1, 2, 5, 10], 27))

1. Each job has a profit and a deadline. Schedule jobs to maximize total profit, allowing only one job at a time before its deadline.

Input: jobs = [(1, 4), (2, 1), (3, 1), (4, 1)]

Output: 6

Input: jobs = [(20, 2), (15, 2), (10, 1), (5, 3)]

Output: 40

Input: jobs = [(100, 2), (19, 1), (27, 2), (25, 1)]

Output: 127

def job\_scheduling(jobs): jobs.sort(reverse=True)

max\_deadline = max(job[1] for job in jobs) slots = [False] \* (max\_deadline + 1)

profit = 0

for job in jobs:

for j in range(job[1], 0, -1):

if not slots[j]:

slots[j] = True profit += job[0] break

return profit

# Test Cases

print(job\_scheduling([(1, 4), (2, 1), (3, 1), (4, 1)])) # 6

print(job\_scheduling([(20, 2), (15, 2), (10, 1), (5, 3)])) # 40

print(job\_scheduling([(100, 2), (19, 1), (27, 2), (25, 1)])) # 127

1. Find the length of the longest common subsequence between two strings. Input: s1 = "ABCBDAB", s2 = "BDCABA"

Output: 4

Input: s1 = "AGGTAB", s2 = "GXTXAYB"

Output: 4

Input: s1 = "ABCDGH", s2 = "AEDFHR"

Output: 3

def lcs(s1, s2):

n, m = len(s1), len(s2)

dp = [[0]\*(m+1) for \_ in range(n+1)] for i in range(n):

for j in range(m):

if s1[i] == s2[j]:

dp[i+1][j+1] = dp[i][j] + 1 else:

dp[i+1][j+1] = max(dp[i][j+1], dp[i+1][j]) return dp[n][m]

# Test Cases

print(lcs("ABCBDAB", "BDCABA")) # 4 print(lcs("AGGTAB", "GXTXAYB")) # 4 print(lcs("ABCDGH", "AEDFHR")) # 3

1. Given n items with weights and values, and a knapsack capacity W, return the maximum value obtainable. Items cannot be split.

Input: W = 50, weights = [10, 20, 30], values = [60, 100, 120]

Output: 220

Input: W = 8, weights = [3, 4, 5], values = [30, 50, 60]

Output: 80

Input: W = 10, weights = [2, 3, 4], values = [40, 50, 60]

Output: 100

def knapsack(W, weights, values): n = len(values)

dp = [[0] \* (W + 1) for \_ in range(n + 1)] for i in range(1, n+1):

for w in range(W+1):

if weights[i-1] <= w:

dp[i][w] = max(dp[i-1][w], values[i-1] + dp[i-1][w - weights[i-1]]) else:

dp[i][w] = dp[i-1][w] return dp[n][W]

# Test Cases

print(knapsack(50, [10, 20, 30], [60, 100, 120])) # 220

print(knapsack(8, [3, 4, 5], [30, 50, 60])) # 80

print(knapsack(10, [2, 3, 4], [40, 50, 60])) # 100

HACKWITHINFY 6 QUESTIONS

1. You have an interesting string S of length N. It is interesting because you can rearrange the characters of this string in any order. You want to cut this string into some contiguous pieces such that after cutting, all the pieces are equal.

You can’t rearrange the characters in the cut pieces or join the pieces together. You want to make the number of pieces as large as possible. What is the maximum number of pieces you can get?

Note: You can observe that you may not want to cut the string at all, therefore the number of pieces is 1. Hence, the answer always exists.

Parameters:

* + S :: STRING

The first line contains a string, S, denoting the string. len(S) :: 1 -> 2 \* 10^5 Examples:

Case #1:

Input:

zzzzz Output:

5

Explanation: You can cut it into 5 pieces “z” + “z” + “z” + “z” + “z”.

Case #2:

Input:

ababcc Output:

2

Explanation: Rearrange the string as abcabc. You can cut it into “abc” + “abc”, hence the answer is 2.

Case #3:

Input:

abccdcabacda Output:

2

Explanation: Rearrange the string as dcbaca + dcbaca, the answer is 2. Solution :

JAVA

import java.util.\*;

public class StringCutting {

public static int maxPieces(String s) {

int[] freq = new int[26];

for (char c : s.toCharArray()) { freq[c - 'a']++;

}

int gcd = 0;

for (int f : freq) { if (f > 0) {

gcd = (gcd == 0) ? f : findGCD(gcd, f);

}

}

return gcd;

}

private static int findGCD(int a, int b) { while (b != 0) {

int temp = b; b = a % b;

a = temp;

}

return a;

}

public static void main(String[] args) { System.out.println(maxPieces("zzzzz")); System.out.println(maxPieces("ababcc")); System.out.println(maxPieces("abccdcabacda"));

}

}

PYTHON

from collections import Counter from math import gcd

from functools import reduce

def max\_pieces(s: str) -> int:

# Count frequency of each character freq = Counter(s)

# Get all non-zero frequencies frequencies = list(freq.values())

# Find GCD of all frequencies using reduce return reduce(gcd, frequencies)

# Test cases def run\_tests():

test\_cases = [

"zzzzz", # Should return 5 "ababcc", # Should return 2 "abccdcabacda" # Should return 2

]

for test in test\_cases: result = max\_pieces(test) print(f"Input: {test}")

print(f"Output: {result}\n")

if name == " main ": run\_tests()

C

#include <stdio.h> #include <string.h>

// Function to find GCD of two numbers int gcd(int a, int b) {

while (b) {

int temp = b; b = a % b;

a = temp;

}

return a;

}

// Function to find GCD of array of numbers int find\_array\_gcd(int arr[], int n) {

int result = arr[0];

for (int i = 1; i < n; i++) {

if (arr[i] != 0) { // Skip zero frequencies result = gcd(result, arr[i]);

}

}

return result;

}

// Main function to find maximum number of equal pieces int max\_pieces(const char\* s) {

// Array to store frequency of each character (assuming lowercase letters only) int freq[26] = {0};

int len = strlen(s);

// Count frequency of each character for (int i = 0; i < len; i++) {

freq[s[i] - 'a']++;

}

// Find first non-zero frequency int first\_non\_zero = 0;

while (first\_non\_zero < 26 && freq[first\_non\_zero] == 0) { first\_non\_zero++;

}

if (first\_non\_zero == 26) return 1; // Empty string or invalid input

// Initialize result with first non-zero frequency int result = freq[first\_non\_zero];

// Find GCD of all non-zero frequencies for (int i = first\_non\_zero + 1; i < 26; i++) {

if (freq[i] > 0) {

result = gcd(result, freq[i]);

}

}

return result;

}

int main() {

// Test cases

const char\* test\_cases[] = { "zzzzz",

"ababcc", "abccdcabacda"

};

int num\_tests = 3;

for (int i = 0; i < num\_tests; i++) { printf("Input: %s\n", test\_cases[i]);

printf("Output: %d\n\n", max\_pieces(test\_cases[i]));

}

return 0;

}

C++

#include <iostream> #include <string> #include <vector>

#include <unordered\_map> #include <numeric>

using namespace std;

class StringCutter { private:

// Helper function to find GCD of two numbers using C++'s standard library int findGCD(int a, int b) {

return gcd(a, b); // Using C++17's std::gcd

}

// Helper function to find GCD of a vector of numbers int findArrayGCD(const vector<int>& numbers) {

if (numbers.empty()) return 0;

int result = numbers[0];

for (size\_t i = 1; i < numbers.size(); i++) { result = findGCD(result, numbers[i]);

}

return result;

}

public:

int maxPieces(const string& s) {

// Use unordered\_map for frequency counting unordered\_map<char, int> freq;

for (char c : s) { freq[c]++;

}

// Extract non-zero frequencies into a vector

vector<int> frequencies;

for (const auto& [ch, count] : freq) { if (count > 0) {

frequencies.push\_back(count);

}

}

// Find GCD of all frequencies return findArrayGCD(frequencies);

}

};

// Function to run test cases void runTests() {

vector<string> testCases = { "zzzzz",

"ababcc", "abccdcabacda"

};

StringCutter solver;

for (const string& test : testCases) { cout << "Input: " << test << "\n";

cout << "Output: " << solver.maxPieces(test) << "\n\n";

}

}

int main() { runTests(); return 0;

}

1. You are given an array A of size N. You are allowed to choose at most one pair of elements such that distance (defined as the difference of their indices) is at most K and swap them.

Find the smallest lexicographical array possible after swapping.

Notes: An array x is lexicographically smaller than an array y if there exists an index i such that xi<yi, and

xj=yj for all 0≤j<i. Less formally, at the first index i in which they differ, xi<yi.

Parameters:

N :: INTEGER

The first line contains an integer, N, denoting the number of elements in A. N :: 1 -> 10^5

A :: INTEGER ARRAY

Each line i of the N subsequent lines (where 0 ≤ i < N) contains an integer describing A[i].

A[i] :: 1 -> 10^5 K :: INTEGER

The next line contains an integer, K, denoting the upper bound on distance of index. K :: 1 -> N

Example Cases:

Case #1

Input:

3

2

2

2

1

Output:

2

2

2

Here, as all the array values are equal, swapping will not change the final result. Case #2

Input:

5

5

4

3

2

1

3

Output:

2

4

3

5

1

Here, A = [5, 4, 3, 2, 1], K = 3. We can swap elements at index 0 and index 3 to get A = [2, 4, 3, 5, 1].

Case #3 Input: 5

2

1

1

1

1

3

Output:

1

1

1

2

1

Here, A = [2, 1, 1, 1, 1], K = 3. We can swap elements at index 0 and index 3 to get A = [1, 1, 1, 2, 1]. Solution:

JAVA

import java.util.\*;

public class LexicographicalArray {

public static int[] findSmallestLexArray(int[] A, int K) { int N = A.length;

// Create a copy of the original array int[] result = Arrays.copyOf(A, N);

// For each position, try to find a smaller element within K distance

// that we can swap to make the array lexicographically smaller for (int i = 0; i < N; i++) {

int minValue = result[i]; int minIndex = i;

// Look for the smallest value within K distance for (int j = i + 1; j < Math.min(N, i + K + 1); j++) {

if (result[j] < minValue) { minValue = result[j]; minIndex = j;

}

}

// If we found a smaller element, swap it and break

// We only need to make one swap to get the lexicographically smallest array if (minIndex != i) {

int temp = result[i]; result[i] = result[minIndex]; result[minIndex] = temp; break;

}

}

return result;

}

public static void main(String[] args) {

// Test cases

test(new int[]{2, 2, 2}, 1);

test(new int[]{5, 4, 3, 2, 1}, 3);

test(new int[]{2, 1, 1, 1, 1}, 3);

}

private static void test(int[] A, int K) { System.out.println("\nInput Array: " + Arrays.toString(A)); System.out.println("K: " + K);

int[] result = findSmallestLexArray(A, K); System.out.println("Result: " + Arrays.toString(result));

}

}

PYTHON

def find\_smallest\_lex\_array(A, K): """

Find the lexicographically smallest array possible after at most one swap within distance K.

Args:

A (list): Input array of integers

K (int): Maximum allowed distance between swapped elements

Returns:

list: The lexicographically smallest possible array after at most one swap """

N = len(A)

# Create a copy of the original array result = A.copy()

# For each position, try to find a smaller element within K distance # that we can swap to make the array lexicographically smaller

for i in range(N):

min\_value = result[i] min\_index = i

# Look for the smallest value within K distance

for j in range(i + 1, min(N, i + K + 1)):

if result[j] < min\_value:

min\_value = result[j] min\_index = j

# If we found a smaller element, swap it and break

# We only need to make one swap to get the lexicographically smallest array if min\_index != i:

result[i], result[min\_index] = result[min\_index], result[i] break

return result

def test\_case(A, K):

"""Helper function to run and display test cases""" print(f"\nInput Array: {A}")

print(f"K: {K}")

result = find\_smallest\_lex\_array(A, K) print(f"Result: {result}")

return result

# Test cases

def run\_test\_cases():

# Test Case 1: All equal elements assert test\_case([2, 2, 2], 1) == [2, 2, 2]

# Test Case 2: Descending array

assert test\_case([5, 4, 3, 2, 1], 3) == [2, 4, 3, 5, 1]

# Test Case 3: Array with repeating elements assert test\_case([2, 1, 1, 1, 1], 3) == [1, 1, 1, 2, 1]

# Additional test case: Already sorted array test\_case([1, 2, 3, 4, 5], 2)

print("\nAll test cases passed!")

if name == " main ": run\_test\_cases()

C

#include <stdio.h> #include <stdlib.h>

// Function to run test case

void testCase(int A[], int N, int K) {

int\* result = (int\*)malloc(N \* sizeof(int));

printf("\nInput Array: "); printArray(A, N); printf("K: %d\n", K);

findSmallestLexArray(A, N, K, result);

printf("Result: ");

printArray(result, N);

free(result);

}

int main() {

// Test Case 1: All equal elements int test1[] = {2, 2, 2};

printf("Test Case 1:");

testCase(test1, 3, 1);

// Test Case 2: Descending array int test2[] = {5, 4, 3, 2, 1};

printf("\nTest Case 2:");

testCase(test2, 5, 3);

// Test Case 3: Array with repeating elements int test3[] = {2, 1, 1, 1, 1};

printf("\nTest Case 3:");

testCase(test3, 5, 3);

return 0;

}

C++

#include <iostream> #include <vector>

#include <algorithm>

class LexicographicalArray { private:

std::vector<int> A; int K;

int N;

public:

// Constructor

LexicographicalArray(const std::vector<int>& arr, int k) : A(arr), K(k) { N = A.size();

}

// Function to find smallest lexicographical array std::vector<int> findSmallestLexArray() {

std::vector<int> result = A; // Create a copy of original array

// For each position, try to find a smaller element within K distance for (int i = 0; i < N; i++) {

int minValue = result[i]; int minIndex = i;

// Look for the smallest value within K distance for (int j = i + 1; j < std::min(N, i + K + 1); j++) {

if (result[j] < minValue) { minValue = result[j]; minIndex = j;

}

}

// If we found a smaller element, swap it and break if (minIndex != i) {

std::swap(result[i], result[minIndex]); break;

}

}

return result;

}

// Function to print array

static void printArray(const std::vector<int>& arr) { for (int num : arr) {

std::cout << num << " ";

}

std::cout << std::endl;

}

};

// Test case runner class class TestRunner { public:

static void runTest(const std::vector<int>& arr, int k) { std::cout << "\nInput Array: "; LexicographicalArray::printArray(arr);

std::cout << "K: " << k << std::endl;

LexicographicalArray solver(arr, k);

std::vector<int> result = solver.findSmallestLexArray();

std::cout << "Result: "; LexicographicalArray::printArray(result);

}

static void runAllTests() {

// Test Case 1: All equal elements std::cout << "Test Case 1:"; runTest({2, 2, 2}, 1);

// Test Case 2: Descending array std::cout << "\nTest Case 2:"; runTest({5, 4, 3, 2, 1}, 3);

// Test Case 3: Array with repeating elements std::cout << "\nTest Case 3:";

runTest({2, 1, 1, 1, 1}, 3);

// Additional Test Case: Already sorted array std::cout << "\nAdditional Test Case:"; runTest({1, 2, 3, 4, 5}, 2);

}

};

int main() { TestRunner::runAllTests(); return 0;

}

1. You are the guard of a shop. Each time a person enters you not this informations.

You are given information of people leaving and entering shop in the form of binary string.

if s[i] == '1' it means person is entering if s[i] == '0' it means person is leaving

It is given that a person can enter or leave the shop any number of times.

Using the given info , find minimum possible distinct people you might have seen.

Constraints 1<=len(s)<=10^5

Sample Testcases --

Input-1: 000

Output : 3

Reason : Suppose at t=1, person1 leaves. Then at t=2 , person2 leaves. Then at t=3, Person 3 leaves. Thus you saw 3 distinct persons leaving the shop

Input-2 : 110011

Output : 2

Reason : At t=1, person1 enters and at t=2 person2 enters the shop . At t=3 , person1 leaves, At t=4, person2 leaves . At t=5, person1 again enters. At t=6, person2 enter the shop again . Thus the minimum number of distinct person is 2.

Input-3: 10101

Output : 1

Reason : At t=1, Person1 enters the shop. At t=2, person 1 leaves . At, t=3 person1 enters. At t=4, person1 leaves.

At t=5, person1 enters . The minimum number of distinct person seens are 1. Solution:

C++

#include <bits/stdc++.h> using namespace std; typedef long long int ll ; int main()

{

ll one = 0 ; ll zero = 0 ; string s ; cin>>s ;

int i = 0 ; while(i<s.size())

{

if(s[i]=='1')

{

ll k = zero; if(k>=1)

{

zero--;

one++;

}

else

{

one++;

}

}

else

{

ll k = one; if(k>=1)

{

one--; zero++;

}

else

{

zero++;

}

} i++;

}

cout<<(one+zero); return 0;

}

1. Find the subsequence from an array with highest score. Score is the sum of f(x) for all the elements of the subsequence.

f(x) = number of divisors of 'x'.

For , a[i] and a[i+1] in the selected subsequence , one of the following should be true : a[i] = 2\*a[i+1]

or

a[i] = 3\*a[i+1] or

a[i] = a[i] + 1 or a[i] = a[i] - 1 or

a[i] = a[i]/3 ( must be divisible by 3) or

a[i] = a[i]/2 ( must be divisible by 2) Input :

10

2 3 4 5 6 12 18 36 9 99

Output :

28

[2,3,4,5,6,12,36] is the selected subsequence from the array [2+2+3+4+2+7+8==28]

Solution:

C++

// C++ implementation of the approach #include <bits/stdc++.h>

using namespace std; typedef long long int ll ; ll divi[200006];

ll fin[200005];

void findDivisors(ll n)

{

for (int i = 1; i <= n; i++) {

for (int j = 1; j \* i <= n; j++) divi[i \* j]++;

}

}

int main()

{

ll r = 0 ;

int n = 100000+7;

findDivisors(n); cin>>n;

int i = 1 ; while(i<=n)

{

int x ; cin>>x ;

ll current = divi[x] ; ll maxi = 0 ; if(x%2==0)

{

maxi = max(maxi,fin[x/2]);

} if(x%3==0)

{

maxi = max(maxi,fin[x/3]);

}

maxi = max(maxi,fin[x-1]);

maxi = max(maxi,fin[x+1]); maxi = max(maxi,fin[2\*x]); maxi = max(maxi,fin[3\*x]); ll answer = current + maxi ; fin[x] = max(fin[x],answer);

//cout<<fin[x]<<'\n'; r = max(fin[x],r); i++;

}

cout<<r ; return 0;

}

GREEDY ALGORITHM

1. Fathima has a numeric string. She has to split the numeric string into two or more integers, such that
2. Difference between current and previous number is 1.
3. No number contains leading zeroes

If it is possible to separate a given numeric string then print “Possible” followed by the first number of the increasing sequence, else print “Not Possible“

For Ex:

Case 1 :

If the input is 7980 Output: Possible 79

String can be splitted into 79 and 80 and the difference is 1 Case 2:

If the input is 124 Output: Not Possible

String can be splitted into '1','2' and '4''. The difference between 1 and 2 is 1, but 2 and 4 is 2. So it can't satisfy the condition. The output will be "Not Possible"

Input Format

Input the String( Which contains only numbers. Alphabets and Special characters are not allowed) Output Format

Display "Possible" followed by the first number of the increasing sequence. If not display "Not Possible" Constraints

0 <= length(numeric string) <= 35

Sample Input 1

7980

Sample Output 1 Possible79

Solution:

JAVA

// Java program to split a numeric

// string in an Increasing

// sequence if possible import java.io.\*; import java.util.\*;

class GFG {

// Function accepts a string and

// checks if string can be split. public static void split(String str)

{

int len = str.length();

// if there is only 1 number

// in the string then

// it is not possible to split it if (len == 1) {

System.out.println("Not Possible"); return;

}

String s1 = "", s2 = ""; long num1, num2;

for (int i = 0; i <= len / 2; i++) {

int flag = 0;

// storing the substring from

// 0 to i+1 to form initial

// number of the increasing sequence s1 = str.substring(0, i + 1);

num1 = Long.parseLong((s1)); num2 = num1 + 1;

// convert string to integer

// and add 1 and again convert

// back to string s2

s2 = Long.toString(num2); int k = i + 1;

while (flag == 0) {

int l = s2.length();

// if s2 is not a substring

// of number than not possile if (k + l > len) {

flag = 1; break;

}

// if s2 is the next substring

// of the numeric string

if ((str.substring(k, k + l).equals(s2))) { flag = 0;

// Incearse num2 by 1 i.e the

// next number to be looked for num2++;

k = k + l;

// check if string is fully

// traversed then break if (k == len)

break;

s2 = Long.toString(num2); l = s2.length();

if (k + 1 > len) {

// If next string doesnot occurs

// in a given numeric string

// then it is not possible flag = 1;

break;

}

}

else

flag = 1;

}

// if the string was fully traversed

// and conditions were satisfied if (flag == 0) {

System.out.println("Possible"

+ " " + s1);

break;

}

// if conditions failed to hold

else if (flag == 1 && i > len / 2 - 1) { System.out.println("Not Possible"); break;

}

}

}

// Driver Code

public static void main(String args[])

{

Scanner in = new Scanner(System.in); String str;

str =in.next();

// Call the split function

// for splitting the string split(str);

}

}

Dynamic Programming

1. The Revolutionary group in Germany have decided to destroy the House of Hitler. There are two Revolutionary groups in Germany, and person of each Revolutionary group has a unique String ID associated with him/her . The first group is of size N, and the second group is of size M. Let the array of String ID's of first group be denoted by A [ ], and that of second group be denoted by B [ ] .

Hitler has decoded the pattern of these Revolutionary groups, and has decided to stop the chaos. He will choose an equal size subarray from both the arrays.

Let the chosen size be X.

So, let the subarray from the first group array A [ ] be Ai, Ai+1, …. Ai+X-1. And, let the subarray from the second array B [ ] be Bj, Bj+1, ….Bj+X-1.

for 1 ≤ i ≤ N-X+1 and 1 ≤ j ≤ M-X+1 , where i is an index of A[ ] and j is an index of array B[ ] He decided to carry out the following operations -

* + Find Longest Common substring (LCS) of Ai+u and Bj+X-1-u , where 0<=u<=X-1.
  + Find LCS of Ai with all of these- Bj, Bj+1…..Bj+X-1
  + Find LCS of Ai+X-1 with all of these- Bj, Bj+1,…..Bj+X-1
  + Find LCS of Bj with all of these- Ai, Ai+1,…..Ai+X-1.
  + Find LCS of Bj+X-1 with all of these- Ai, Ai+1, ……Ai+X-1. Hitler has got to know a magical integer K

All of the above mentioned LCS values must be greater than or equal to K.

You need to help the Government in finding all possible number of combinations of equal sized subarrays in A [ ] and B[ ] which are satisfying the above mentioned condition.

Two combinations are different if they are of different size, or in case atleast one constituent index is different.

Input Format

First line contains 2 space separated integers N and M, denoting the size of two arrays A[ ] and B [ ] respectively.

Second line contains an integer K denoting the magical integer. Next N lines comprises a String ID of group A [ ].

Next M lines comprises a String ID of group B[ ] Output Format

Print the number of possible combinations as explained above. Constraints

1 ≤ N,M ≤ 2000

1 ≤ | Ai | ≤ 5000

1 ≤ | Bi | ≤ 5000

Summation of length of all strings in A [ ] ≤ 5000 Summation of length of all strings in B [ ] ≤ 5000 All strings should contain lowercase alphabets. 1 ≤ K ≤ 5000

Sample Input 1

3 3

1

aab aba jgh bad

dea cef

Sample Output 1

5

Solution:

C++

#include <bits/stdc++.h>

using namespace std;

#define pb push\_back

int a[2005][2005];

int lft[2005][2005]; int rght[2005][2005]; int up[2005][2005];

int dwn[2005][2005]; int diagUp[2005][2005];

int diagDwn[2005][2005];

vector <int> v1, v2; vector <int> seg[100005];

void merge(int p, int q, int ind)

{

int i, j;

for (i = 0, j = 0; i < seg[p].size() && j < seg[q].size(); ) { if (seg[p][i] <= seg[q][j]) {

seg[ind].pb(seg[p][i]);

++i;

}

else {

seg[ind].pb(seg[q][j]);

++j;

}

}

while (i < seg[p].size()) {

seg[ind].pb(seg[p][i]);

++i;

}

while (j < seg[q].size()) {

seg[ind].pb(seg[q][j]);

++j;

}

}

void create\_seg\_tree(int l, int r, int ind)

{

seg[ind].clear(); if (l == r) {

seg[ind].pb(v2[l]); return;

}

int m = (l+r)/2; int p = 2\*ind;

int q = p+1; create\_seg\_tree(l, m, p); create\_seg\_tree(m+1, r, q); merge(p, q, ind);

}

int query(int x, int y, int val, int l, int r, int ind)

{

if (y < l || r < x) return 0;

if (x <= l && r <= y)

return (upper\_bound(seg[ind].begin(), seg[ind].end(), val) - seg[ind].begin()); int m = (l+r)/2;

int p = 2\*ind; int q = p+1;

return query(x, y, val, l, m, p) + query(x, y, val, m+1, r, q);

}

int dp[5005][5005]; string A[5005], B[5005];

int lcs(int x, int y)

{

int i, j, ans = 0;

for (i = 1; i <= A[x].size(); ++i) {

for (j = 1; j <= B[y].size(); ++j) { dp[i][j] = 0;

if (A[x][i-1] == B[y][j-1])

dp[i][j] = dp[i-1][j-1] + 1;

ans = dp[i][j] > ans ? dp[i][j] : ans;

}

}

return ans;

}

int main()

{

int size\_of\_array\_A, size\_of\_array\_B, i, j, x, magical\_integer; cin >> size\_of\_array\_A >> size\_of\_array\_B;

assert(1 <= size\_of\_array\_A && size\_of\_array\_A <= 2000); assert(1 <= size\_of\_array\_B && size\_of\_array\_B <= 2000); cin >> magical\_integer;

assert(1 <= magical\_integer && magical\_integer <= 5000); int tmp = 0;

for (i = 1; i <= size\_of\_array\_A; ++i) { cin >> A[i];

assert(1 <= A[i].size() && A[i].size() <= 5000); for (j = 0; j < A[i].size(); ++j)

assert('a' <= A[i][j] && A[i][j] <= 'z'); tmp += A[i].size();

}

assert(1 <= tmp && tmp <= 5000); tmp = 0;

for (i = 1; i <= size\_of\_array\_B; ++i) { cin >> B[i];

assert(1 <= B[i].size() && B[i].size() <= 5000); for (j = 0; j < B[i].size(); ++j)

assert('a' <= B[i][j] && B[i][j] <= 'z'); tmp += B[i].size();

}

assert(1 <= tmp && tmp <= 5000);

for (i = 1; i <= size\_of\_array\_A; ++i) { for (j = 1; j <= size\_of\_array\_B; ++j)

a[i][j] = (lcs(i, j) >= magical\_integer);

}

for (i = 1; i <= size\_of\_array\_A; ++i) { for (j = 1; j <= size\_of\_array\_B; ++j) {

if (!a[i][j]) continue;

up[i][j] = up[i-1][j] + 1;

lft[i][j] = lft[i][j-1] + 1;

diagUp[i][j] = diagUp[i-1][j+1] + 1;

}

}

for (i = size\_of\_array\_A; i >= 1; --i) { for (j = size\_of\_array\_B; j >= 1; --j) {

if (!a[i][j]) continue;

dwn[i][j] = dwn[i+1][j] + 1;

rght[i][j] = rght[i][j+1] + 1; diagDwn[i][j] = diagDwn[i+1][j-1] + 1;

}

}

long long int ans = 0;

for (x = size\_of\_array\_B; x >= 1; --x) {

i = size\_of\_array\_A;

j = x;

v1.clear(); v2.clear();

while (i >= 1 && j <=size\_of\_array\_B) { v1.pb(min(min(up[i][j], rght[i][j]), diagUp[i][j]));

v2.pb(min(min(dwn[i][j], lft[i][j]), diagDwn[i][j]));

--i;

++j;

}

for (i = 0; i < v2.size(); ++i) v2[i] = i - v2[i] + 1;

create\_seg\_tree(0, v2.size()-1, 1); for (i = 0; i < v1.size(); ++i) {

j = i + v1[i] - 1;

ans += query(i, j, i, 0, v2.size()-1, 1);

}

}

for (x = size\_of\_array\_A-1; x >= 1; --x) {

i = x;

j = 1;

v1.clear(); v2.clear();

while (i >= 1 && j <= size\_of\_array\_B) { v1.pb(min(min(up[i][j], rght[i][j]), diagUp[i][j]));

v2.pb(min(min(dwn[i][j], lft[i][j]), diagDwn[i][j]));

--i;

++j;

}

for (i = 0; i < v2.size(); ++i) v2[i] = i - v2[i] + 1;

create\_seg\_tree(0, v2.size()-1, 1); for (i = 0; i < v1.size(); ++i) {

j = i + v1[i] - 1;

ans += query(i, j, i, 0, v2.size()-1, 1);

}

}

cout << ans << endl; return 0;

}

Question 1:

Find the intersection of two sorted arrays OR in other words, given 2 sorted arrays, find all the elements which occur in both arrays.

NOTE: For the purpose of this problem ( as also conveyed by the sample case ), assume that elements that appear more than once in both arrays should be included multiple times in the final output.

Problem Constraints 1 <= |A| <= 106

1 <= |B| <= 106

Input Format

The first argument is an integer array A. The second argument is an integer array B. Output Format

Return an array of intersection of the two arrays.

Example Input Input 1:

A: [1 2 3 3 4 5 6]

B: [3 3 5]

Input 2:

A: [1 2 3 3 4 5 6]

B: [3 5]

Example Output Output 1: [3 3 5]

Output 2: [3 5] Example Explanation Explanation 1:

3, 3, 5 occurs in both the arrays A and B Explanation 2:

Only 3 and 5 occurs in both the arrays A and B

Question2:

Given an 2D integer array A of size N x 2 denoting time intervals of different meetings. Where:

* A[i][0] = start time of the ith meeting.
* A[i][1] = end time of the ith meeting.

Find the minimum number of conference rooms required so that all meetings can be done.

Note :- If a meeting ends at time t, another meeting starting at time t can use the same conference room

Problem Constraints 1 <= N <= 105

0 <= A[i][0] < A[i][1] <= 2 \* 109

Input Format

The only argument given is the matrix A. Output Format

Return the minimum number of conference rooms required so that all meetings can be done. Example Input

Input 1:

A = [ [0, 30]

[5, 10]

[15, 20]

]

Input 2:

A = [ [1, 18]

[18, 23]

[15, 29]

[4, 15]

[2, 11]

[5, 13]

]

Example Output Output 1:

2

Output 2:

4

Example Explanation Explanation 1:

Meeting one can be done in conference room 1 form 0 - 30. Meeting two can be done in conference room 2 form 5 - 10.

Meeting three can be done in conference room 2 form 15 - 20 as it is free in this interval. Explanation 2:

Meeting one can be done in conference room 1 from 1 - 18. Meeting five can be done in conference room 2 from 2 - 11. Meeting four can be done in conference room 3 from 4 - 15. Meeting six can be done in conference room 4 from 5 - 13.

Meeting three can be done in conference room 2 from 15 - 29 as it is free in this interval. Meeting two can be done in conference room 4 from 18 - 23 as it is free in this interval.

Question3:

N children are standing in a line. Each child is assigned a rating value.

You are giving candies to these children subjected to the following requirements:

1. Each child must have at least one candy.
2. Children with a higher rating get more candies than their neighbors. What is the minimum number of candies you must give?

Problem Constraints 1 <= N <= 105

-109 <= A[i] <= 109

Input Format

The first and only argument is an integer array A representing the rating of children. Output Format

Return an integer representing the minimum candies to be given. Example Input

Input 1:

A = [1, 2]

Input 2:

A = [1, 5, 2, 1]

Example Output Output 1:

3

Output 2:

7

Example Explanation Explanation 1:

The candidate with 1 rating gets 1 candy and candidate with rating 2 cannot get 1 candy as 1 is its neighbor.

So rating 2 candidate gets 2 candies. In total, 2 + 1 = 3 candies need to be given out. Explanation 2:

Candies given = [1, 3, 2, 1]

Question 4:

There is a street of length xxx whose positions are numbered 0, 1,…,x Initially there are no traffic lights,

but n sets of traffic lights are added to the street one after another.

Your task is to calculate the length of the longest passage without traffic lights after each addition. Input

The first input line contains two integers xxx and n: the length of the street and the number of sets of traffic lights.

Then, the next line contains n integers p1,p2,…,pn: the position of each set of traffic lights. Each position

is distinct.

Output

Print the length of the longest passage without traffic lights after each addition. Example

Input:

8 3

3 6 2

Output:

5 3 3

Question5:

You are given n cubes in a certain order, and your task is to build towers using them. Whenever two cubes are one on top of the other, the upper cube must be smaller than the lower cube.

You must process the cubes in the given order. You can always either place the cube on top of an existing tower, or begin a new tower. What is the minimum possible number of towers?

Input

The first input line contains an integer n: the number of cubes.

The next line contains nnn integers k1,k2,…,kn: the sizes of the cubes. Output

Print one integer: the minimum number of towers. Example

Input:

5

3 8 2 1 5

Output:

2

Problem Statement:

You are given n jobs where every job has a deadline and profit associated with it. Each job takes a single unit of time, and only one job can be scheduled at a time. Maximize total profit if jobs are scheduled before their deadlines.

Constraints:

* 1 <= n <= 10^5
* 1 <= deadline[i] <= 1000
* 1 <= profit[i] <= 10^4

Input Format:

* First line: integer n
* Next n lines: two integers deadline[i] and profit[i] Output Format:
* Two integers: total number of jobs done and total profit earned Test Case:

Input:

4

4 20

1 10

1 40

1 30

Output:

2 70

Solution Logic:

* Sort jobs by descending profit
* Try placing each job in the latest available time slot before its deadline using a disjoint-set or simple array for tracking

Python

def job\_sequencing(jobs): jobs.sort(key=lambda x: x[1], reverse=True) max\_deadline = max(job[0] for job in jobs) slots = [False] \* (max\_deadline + 1)

count, total\_profit = 0, 0

for d, p in jobs:

for j in range(d, 0, -1): if not slots[j]:

slots[j] = True count += 1 total\_profit += p break

return count, total\_profit

n = int(input())

jobs = [tuple(map(int, input().split())) for \_ in range(n)] res = job\_sequencing(jobs)

print(\*res)

Java

import java.util.\*;

class Job implements Comparable<Job> { int deadline, profit;

public Job(int d, int p) { deadline = d;

profit = p;

}

public int compareTo(Job o) { return o.profit - this.profit;

}

}

public class Main {

public static void main(String[] args) { Scanner sc = new Scanner(System.in);

int n = sc.nextInt(); Job[] jobs = new Job[n]; int maxDeadline = 0;

for (int i = 0; i < n; i++) {

int d = sc.nextInt(), p = sc.nextInt(); jobs[i] = new Job(d, p);

maxDeadline = Math.max(maxDeadline, d);

}

Arrays.sort(jobs);

boolean[] slots = new boolean[maxDeadline + 1]; int count = 0, profit = 0;

for (Job job : jobs) {

for (int j = job.deadline; j > 0; j--) { if (!slots[j]) {

slots[j] = true; count++;

profit += job.profit; break;

}

}

}

System.out.println(count + " " + profit);

}

} C

#include <stdio.h> #include <stdlib.h>

typedef struct {

int deadline, profit;

} Job;

int compare(const void\* a, const void\* b) { return ((Job\*)b)->profit - ((Job\*)a)->profit;

}

int main() { int n;

scanf("%d", &n); Job jobs[n];

int maxDeadline = 0;

for (int i = 0; i < n; i++) {

scanf("%d %d", &jobs[i].deadline, &jobs[i].profit); if (jobs[i].deadline > maxDeadline)

maxDeadline = jobs[i].deadline;

}

qsort(jobs, n, sizeof(Job), compare); int slots[maxDeadline + 1];

for (int i = 0; i <= maxDeadline; i++) slots[i] = 0;

int count = 0, profit = 0;

for (int i = 0; i < n; i++) {

for (int j = jobs[i].deadline; j > 0; j--) { if (!slots[j]) {

slots[j] = 1; count++;

profit += jobs[i].profit; break;

}

}

}

printf("%d %d\n", count, profit); return 0;

}

Problem Statement:

Given the stock prices for n days and an integer k, find the maximum profit you can make with at most k transactions. You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

Constraints:

* 1 <= k <= 100
* 1 <= n <= 10^3
* 0 <= prices[i] <= 10^4 Input Format:
* First line: integers k and n
* Second line: n integers representing prices Output Format:
* One integer: maximum profit

Test Case:

Input:

2 6

3 2 6 5 0 3

Output:

7

Solution Logic:

* Use a DP table dp[k+1][n] where dp[i][j] is the max profit using i transactions until day j Python

def max\_profit(k, prices): n = len(prices)

if n == 0: return 0

dp = [[0]\*n for \_ in range(k+1)]

for i in range(1, k+1):

max\_diff = -prices[0] for j in range(1, n):

dp[i][j] = max(dp[i][j-1], prices[j] + max\_diff) max\_diff = max(max\_diff, dp[i-1][j] - prices[j])

return dp[k][n-1]

k, n = map(int, input().split())

prices = list(map(int, input().split())) print(max\_profit(k, prices))

Java

import java.util.\*;

public class Main {

public static int maxProfit(int k, int[] prices) { int n = prices.length;

if (n == 0) return 0;

int[][] dp = new int[k+1][n];

for (int i = 1; i <= k; i++) { int maxDiff = -prices[0]; for (int j = 1; j < n; j++) {

dp[i][j] = Math.max(dp[i][j-1], prices[j] + maxDiff); maxDiff = Math.max(maxDiff, dp[i-1][j] - prices[j]);

}

}

return dp[k][n-1];

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int k = sc.nextInt(), n = sc.nextInt(); int[] prices = new int[n];

for (int i = 0; i < n; i++) prices[i] = sc.nextInt(); System.out.println(maxProfit(k, prices));

}

} C

#include <stdio.h>

#define max(a,b) ((a) > (b) ? (a) : (b))

int maxProfit(int k, int prices[], int n) {

if (n == 0) return 0; int dp[k+1][n];

for (int i = 0; i <= k; i++) for (int j = 0; j < n; j++)

dp[i][j] = 0;

for (int i = 1; i <= k; i++) { int maxDiff = -prices[0]; for (int j = 1; j < n; j++) {

dp[i][j] = max(dp[i][j-1], prices[j] + maxDiff); maxDiff = max(maxDiff, dp[i-1][j] - prices[j]);

}

}

return dp[k][n-1];

}

int main() { int k, n;

scanf("%d %d", &k, &n); int prices[n];

for (int i = 0; i < n; i++) scanf("%d", &prices[i]); printf("%d\n", maxProfit(k, prices, n));

return 0;

}

Problem Statement:

You're managing a memory block with m units of memory and n processes. Each process has a memory requirement and a priority value. Allocate memory to as many high-priority processes as possible without exceeding the total memory m. A process can only be either completely included or excluded.

Constraints:

* 1 <= n <= 10^5
* 1 <= memory[i] <= 10^4
* 1 <= priority[i] <= 10^6
* 1 <= m <= 10^9

Input Format:

* Line 1: Two integers n and m
* Next n lines: Two integers memory[i] and priority[i] Output Format:
* One integer: total priority of selected processes Test Case:

Input:

4 10

4 70

2 60

6 90

3 30

Output:

160

Solution Logic:

* Sort processes by priority-to-memory ratio in descending order.
* Greedily select processes until memory is exhausted.

Python

n, m = map(int, input().split())

processes = [tuple(map(int, input().split())) for \_ in range(n)] processes.sort(key=lambda x: x[1]/x[0], reverse=True)

total\_priority = 0

for mem, pri in processes: if m >= mem:

m -= mem total\_priority += pri

print(total\_priority)

Java

import java.util.\*;

class Process { int mem, pri;

public Process(int m, int p) { mem = m; pri = p;

}

double ratio() {

return (double) pri / mem;

}

}

public class Main {

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int n = sc.nextInt(), m = sc.nextInt(); Process[] procs = new Process[n];

for (int i = 0; i < n; i++)

procs[i] = new Process(sc.nextInt(), sc.nextInt());

Arrays.sort(procs, (a, b) -> Double.compare(b.ratio(), a.ratio()));

int total = 0;

for (Process p : procs) { if (m >= p.mem) {

m -= p.mem; total += p.pri;

}

}

System.out.println(total);

}

} C

#include <stdio.h> #include <stdlib.h>

typedef struct { int mem, pri; double ratio;

} Process;

int compare(const void\* a, const void\* b) { double r1 = ((Process\*)a)->ratio;

double r2 = ((Process\*)b)->ratio; return (r2 > r1) - (r2 < r1);

}

int main() { int n, m;

scanf("%d %d", &n, &m);

Process p[n];

for (int i = 0; i < n; i++) {

scanf("%d %d", &p[i].mem, &p[i].pri);

p[i].ratio = (double)p[i].pri / p[i].mem;

}

qsort(p, n, sizeof(Process), compare);

int total = 0;

for (int i = 0; i < n; i++) { if (m >= p[i].mem) {

m -= p[i].mem; total += p[i].pri;

}

}

printf("%d\n", total); return 0;

}

Problem Statement:

Given a string s and a cost array cost[] where cost[i] is the cost of deleting the i-th character, find the minimum total deletion cost required to convert the string into a palindrome.

Constraints:

* 1 <= len(s) <= 1000
* 0 <= cost[i] <= 10^4
* s[i] is a lowercase letter Input Format:
* Line 1: String s
* Line 2: Array cost of size len(s) Output Format:
* One integer: minimum deletion cost to make the string a palindrome Test Case:

Input:

abcbda

1 2 3 4 5 6

Output:

3

Solution Logic:

Use bottom-up DP:

* dp[i][j] is the minimum cost to make s[i..j] a palindrome
* If s[i] == s[j], no deletion needed: dp[i][j] = dp[i+1][j-1]
* Else, delete one character: dp[i][j] = min(cost[i] + dp[i+1][j], cost[j] + dp[i][j-1])

Python

s = input()

cost = list(map(int, input().split())) n = len(s)

dp = [[0]\*n for \_ in range(n)]

for length in range(2, n+1): for i in range(n - length + 1):

j = i + length - 1 if s[i] == s[j]:

dp[i][j] = dp[i+1][j-1] else:

dp[i][j] = min(cost[i] + dp[i+1][j], cost[j] + dp[i][j-1])

print(dp[0][n-1])

Java

import java.util.\*;

public class Main {

public static void main(String[] args) { Scanner sc = new Scanner(System.in); String s = sc.next();

int n = s.length();

int[] cost = new int[n];

for (int i = 0; i < n; i++) cost[i] = sc.nextInt();

int[][] dp = new int[n][n];

for (int len = 2; len <= n; len++) { for (int i = 0; i <= n - len; i++) {

int j = i + len - 1;

if (s.charAt(i) == s.charAt(j)) dp[i][j] = dp[i+1][j-1];

else

dp[i][j] = Math.min(cost[i] + dp[i+1][j], cost[j] + dp[i][j-1]);

}

}

System.out.println(dp[0][n-1]);

}

}

C

#include <stdio.h> #include <string.h>

#define MIN(a,b) ((a)<(b)?(a):(b))

int main() { char s[1001];

int cost[1001], dp[1001][1001]; scanf("%s", s);

int n = strlen(s);

for (int i = 0; i < n; i++) scanf("%d", &cost[i]);

for (int l = 2; l <= n; l++) {

for (int i = 0; i <= n - l; i++) { int j = i + l - 1;

if (s[i] == s[j])

dp[i][j] = dp[i+1][j-1]; else

dp[i][j] = MIN(cost[i] + dp[i+1][j], cost[j] + dp[i][j-1]);

}

}

printf("%d\n", dp[0][n-1]); return 0;

}

Problem Statement:

You are given n intervals where each interval has a start time, end time, and an associated weight. Select a set of non-overlapping intervals such that the total weight is maximized.

This is a classic optimization problem where greedy won't work, and dynamic programming with binary search or coordinate compression is necessary.

Constraints:

* 1 <= n <= 10^5
* 0 <= start[i] < end[i] <= 10^9
* 1 <= weight[i] <= 10^6 Input Format:
* Line 1: Integer n
* Next n lines: Three integers: start[i], end[i], weight[i] Output Format:
* One integer: Maximum total weight of non-overlapping intervals Test Case:

Input:

4

1 3 50

2 5 20

4 6 70

6 7 60

Output:

180

Explanation: Choose intervals (1,3), (4,6), (6,7) with weights 50+70+60 = 180 Solution Logic:

* Sort intervals by end time
* For each interval i, use binary search to find the last non-overlapping interval j with end[j] <= start[i]
* Use DP:

dp[i] = max(dp[i-1], weight[i] + dp[last\_non\_conflicting\_index])

Python import bisect

n = int(input())

intervals = [tuple(map(int, input().split())) for \_ in range(n)] intervals.sort(key=lambda x: x[1]) # sort by end time

ends = [interval[1] for interval in intervals] dp = [0] \* (n + 1)

for i in range(1, n + 1): s, e, w = intervals[i-1]

idx = bisect.bisect\_right(ends, s) - 1 dp[i] = max(dp[i-1], w + dp[idx+1])

print(dp[n])

Java

import java.util.\*;

class Interval implements Comparable<Interval> { int start, end, weight;

public Interval(int s, int e, int w) { start = s; end = e; weight = w;

}

public int compareTo(Interval o) { return this.end - o.end;

}

}

public class Main {

static int binarySearch(Interval[] intervals, int idx) { int lo = 0, hi = idx - 1, ans = -1;

while (lo <= hi) {

int mid = (lo + hi) / 2;

if (intervals[mid].end <= intervals[idx].start) { ans = mid;

lo = mid + 1;

} else {

hi = mid - 1;

}

}

return ans;

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int n = sc.nextInt();

Interval[] arr = new Interval[n]; for (int i = 0; i < n; i++)

arr[i] = new Interval(sc.nextInt(), sc.nextInt(), sc.nextInt());

Arrays.sort(arr);

long[] dp = new long[n + 1];

for (int i = 1; i <= n; i++) {

int j = binarySearch(arr, i - 1);

dp[i] = Math.max(dp[i-1], arr[i-1].weight + (j >= 0 ? dp[j+1] : 0));

}

System.out.println(dp[n]);

}

} C

#include <stdio.h> #include <stdlib.h>

typedef struct {

int start, end, weight;

} Interval;

int compare(const void \*a, const void \*b) { return ((Interval\*)a)->end - ((Interval\*)b)->end;

}

int binarySearch(Interval arr[], int idx) { int lo = 0, hi = idx - 1, ans = -1;

while (lo <= hi) {

int mid = (lo + hi) / 2;

if (arr[mid].end <= arr[idx].start) { ans = mid;

lo = mid + 1;

} else {

hi = mid - 1;

}

}

return ans;

}

int main() {

int n; scanf("%d", &n); Interval arr[n];

for (int i = 0; i < n; i++)

scanf("%d %d %d", &arr[i].start, &arr[i].end, &arr[i].weight);

qsort(arr, n, sizeof(Interval), compare); long long dp[n+1];

dp[0] = 0;

for (int i = 1; i <= n; i++) {

int j = binarySearch(arr, i-1);

long long include = arr[i-1].weight + (j >= 0 ? dp[j+1] : 0); dp[i] = dp[i-1] > include ? dp[i-1] : include;

}

printf("%lld\n", dp[n]); return 0;

}

Longest Increasing Subsequence (LIS)

Problem-1:

Given an array of integers, find the length of the longest strictly increasing subsequence. A subsequence is a sequence that can be derived from the array by deleting some or no elements without changing the order of the remaining elements.

Example:

Input:

nums = [10, 9, 2, 5, 3, 7, 101, 18]

Output:

4

Explanation:

The longest increasing subsequence is `[2, 3, 7, 101]`, so the output is `4`.

Dynamic Programming Approach (O(n²)):

Step-by-step:

1. Create a DP array `dp` where `dp[i]` will store the length of the LIS ending at index `i`.
2. Initialize each value in `dp` as 1, because the minimum LIS ending at any element is 1 (the element itself).
3. For each element `nums[i]`, loop through all previous elements `nums[j]` where `j < i`.
4. If `nums[j] < nums[i]`, then `nums[i]` can extend the subsequence ending at `nums[j]`.
5. Update `dp[i] = max(dp[i], dp[j] + 1)`.

Code (Python):

python

def length\_of\_lis(nums): n = len(nums)

dp = [1] \* n # Every number is at least an LIS of 1 by itself

for i in range(n):

for j in range(i):

if nums[j] < nums[i]:

dp[i] = max(dp[i], dp[j] + 1)

return max(dp) # The answer is the max value in dp

Test Case:

python

print(length\_of\_lis([10, 9, 2, 5, 3, 7, 101, 18]))

Output: 4

Problem-2: Activity Selection Problem (or Interval Scheduling Maximum Subset)

Problem Statement:

You are given `n` activities with their start and end times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on one activity at a time.

Example:

Input:

start = [1, 3, 0, 5, 8, 5]

end = [2, 4, 6, 7, 9, 9]

Output:

4

Explanation:

The maximum number of non-overlapping activities that can be performed is 4.

One of the optimal selections is activities at index: 0, 1, 3, 4

(That is: [1–2], [3–4], [5–7], [8–9])

Greedy Approach:

1. Pair each activity with its start and end time.
2. Sort the activities based on end time.
3. Select activities such that the start time of the current activity is greater than or equal to the end time of the last selected activity.

Code (Python):

python

def activity\_selection(start, end):

activities = sorted(zip(start, end), key=lambda x: x[1]) # sort by end time count = 1

last\_end = activities[0][1]

for i in range(1, len(activities)): if activities[i][0] >= last\_end:

count += 1

last\_end = activities[i][1]

return count

Test Case:

python

start = [1, 3, 0, 5, 8, 5]

end = [2, 4, 6, 7, 9, 9]

print(activity\_selection(start, end))

Output: 4

Problem-3: Maximum Subarray Sum (Using Divide and Conquer)

Problem Statement:

Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the \*\*largest sum\*\* and return its sum.

This is the Divide and Conquer version of the famous Kadane's Algorithm problem.

Example:

Input:

nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

Output:

6

Explanation:

The subarray `[4, -1, 2, 1]` has the largest sum = `6`.

Divide and Conquer Approach:

1. Divide the array into two halves.
2. Recursively find the maximum subarray sum in the left and right halves.
3. Find the maximum sum \*\*crossing the midpoint\*\*.
4. Return the maximum of the three.

Code (Python):

python

def max\_crossing\_sum(arr, left, mid, right): left\_sum = float('-inf')

sum\_ = 0

for i in range(mid, left - 1, -1): sum\_ += arr[i]

left\_sum = max(left\_sum, sum\_)

right\_sum = float('-inf') sum\_ = 0

for i in range(mid + 1, right + 1):

sum\_ += arr[i]

right\_sum = max(right\_sum, sum\_)

return left\_sum + right\_sum

def max\_subarray\_sum(arr, left, right): if left == right:

return arr[left]

mid = (left + right) // 2 return max(

max\_subarray\_sum(arr, left, mid), max\_subarray\_sum(arr, mid + 1, right), max\_crossing\_sum(arr, left, mid, right)

)

Wrapper function

def find\_max\_subarray\_sum(nums):

return max\_subarray\_sum(nums, 0, len(nums) - 1)

Test Case:

python

nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

print(find\_max\_subarray\_sum(nums))

Output: 6

Problem-4: Minimum Spanning Tree using Prim's Algorithm

Problem Statement:

Given an undirected, weighted graph represented as an adjacency list or matrix, implement Prim's algorithm to find the minimum cost to connect all nodes (i.e., the total weight of the MST).

Example:

Input:

Number of vertices:

5

Edges (u, v, weight):

(0, 1, 2)

(0, 3, 6)

(1, 2, 3)

(1, 3, 8)

(1, 4, 5)

(2, 4, 7)

(3, 4, 9)

Output:

Total cost of MST: 16

Explanation:

The MST includes edges with weights: `2 (0-1)`, `3 (1-2)`, `5 (1-4)`, `6 (0-3)` → total = 16

Python Code (Using Min-Heap + Adjacency List):

python import heapq

def prims\_algorithm(n, edges): graph = [[] for \_ in range(n)] for u, v, w in edges:

graph[u].append((w, v))

graph[v].append((w, u))

visited = [False] \* n

min\_heap = [(0, 0)] # (weight, vertex) mst\_cost = 0

while min\_heap:

weight, u = heapq.heappop(min\_heap) if visited[u]:

continue mst\_cost += weight visited[u] = True

for edge\_weight, v in graph[u]:

if not visited[v]:

heapq.heappush(min\_heap, (edge\_weight, v))

return mst\_cost

Test Case:

python n = 5 edges = [

(0, 1, 2),

(0, 3, 6),

(1, 2, 3),

(1, 3, 8),

(1, 4, 5),

(2, 4, 7),

(3, 4, 9)

]

print("Total cost of MST:", prims\_algorithm(n, edges))

Output: 16

Problem-5: Single Source Shortest Path (Using Dijkstra’s Algorithm)

Problem Statement:

Given a weighted, directed graph and a source vertex, find the shortest distance from the source to all

other vertices using Dijkstra’s algorithm.

Example:

Input:

Number of vertices: 5 Edges:

(0, 1, 10)

(0, 4, 5)

(1, 2, 1)

(1, 4, 2)

(2, 3, 4)

(3, 0, 7)

(4, 1, 3)

(4, 2, 9)

(4, 3, 2)

Source vertex: 0

Output:

Shortest distances from vertex 0:

[0, 8, 9, 7, 5]

Explanation:

From node `0`, the shortest paths to nodes `1`, `2`, `3`, and `4` have costs `8`, `9`, `7`, and `5`, respectively.

Python Code (Using Min-Heap + Adjacency List):

python import heapq

def dijkstra(n, edges, source): graph = [[] for \_ in range(n)] for u, v, w in edges:

graph[u].append((v, w))

dist = [float('inf')] \* n

dist[source] = 0

min\_heap = [(0, source)] # (distance, node)

while min\_heap:

current\_dist, u = heapq.heappop(min\_heap) if current\_dist > dist[u]:

continue

for v, weight in graph[u]:

if dist[u] + weight < dist[v]:

dist[v] = dist[u] + weight heapq.heappush(min\_heap, (dist[v], v))

return dist

Test Case:

python n = 5 edges = [

(0, 1, 10),

(0, 4, 5),

(1, 2, 1),

(1, 4, 2),

(2, 3, 4),

(3, 0, 7),

(4, 1, 3),

(4, 2, 9),

(4, 3, 2)

]

source = 0

print("Shortest distances from vertex 0:") print(dijkstra(n, edges, source)) # Output: [0, 8, 9, 7, 5]

Problem 1: Lexicographically Smallest String by Removing One Character Problem Statement:

Given a string s, remove exactly one character such that the resulting string is lexicographically smallest. Return the modified string.

Input Format:

* A single string s consisting of lowercase English letters. Output Format:
* A single string which is lexicographically smallest after removing one character.

Sample Test Cases: Input: abcda Output: abca

Input: aaa Output: aa

Input: edcba

Output: dcba Python

def smallestString(s):

for i in range(len(s)-1):

if s[i] > s[i+1]:

return s[:i] + s[i+1:] return s[:-1]

# Read input s = input()

rint(smallestString(s))

Java

import java.util.Scanner;

public class Main {

public static String smallestString(String s) { for (int i = 0; i < s.length() - 1; i++) {

if (s.charAt(i) > s.charAt(i + 1)) {

return s.substring(0, i) + s.substring(i + 1);

}

}

return s.substring(0, s.length() - 1);

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); String input = sc.nextLine();

System.out.println(smallestString(input));

}

} C

#include <stdio.h> #include <string.h>

void smallestString(char s[]) { int len = strlen(s), i;

for (i = 0; i < len - 1; i++) { if (s[i] > s[i + 1]) break;

}

for (int j = 0; j < len; j++) {

if (j != i) printf("%c", s[j]);

}

printf("\n");

}

int main() { char s[101];

scanf("%s", s); smallestString(s); return 0;

}

C++

#include <iostream> using namespace std;

string smallestString(string s) {

for (int i = 0; i < s.size() - 1; ++i) { if (s[i] > s[i+1])

return s.substr(0, i) + s.substr(i+1);

}

return s.substr(0, s.size() - 1);

}

int main() { string s; cin >> s;

cout << smallestString(s) << endl;

}

Problem 2: Count Numbers with Digit Sum Divisible by 3 Problem Statement:

Given a number n, count how many numbers from 1 to n have a digit sum divisible by 3. Input Format:

* A single integer n (1 ≤ n ≤ 1000) Output Format:
* A single integer: the count of numbers from 1 to n whose digit sum is divisible by 3. Sample Test Cases:

Input: 15

Output: 5

Input: 20

Output: 6

Input: 30

Output: 10 Python

def countDiv3(n):

return sum(1 for i in range(1, n+1) if sum(map(int, str(i))) % 3 == 0)

n = int(input()) print(countDiv3(n))

Java

import java.util.Scanner;

public class Main {

public static int countDiv3(int n) { int count = 0;

for (int i = 1; i <= n; i++) { int sum = 0, temp = i; while (temp > 0) {

sum += temp % 10; temp /= 10;

}

if (sum % 3 == 0) count++;

}

return count;

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int n = sc.nextInt();

System.out.println(countDiv3(n));

}

}

C

#include <stdio.h>

int digitSum(int n) { int sum = 0; while(n) {

sum += n % 10; n /= 10;

}

return sum;

}

int countDiv3(int n) { int count = 0;

for (int i = 1; i <= n; i++) { if (digitSum(i) % 3 == 0)

count++;

}

return count;

}

int main() { int n;

scanf("%d", &n);

printf("%d\n", countDiv3(n)); return 0;

}

C++

#include <iostream> using namespace std;

int digitSum(int n) { int sum = 0; while(n) {

sum += n % 10; n /= 10;

}

return sum;

}

int countDiv3(int n) { int count = 0;

for (int i = 1; i <= n; i++) { if (digitSum(i) % 3 == 0)

count++;

}

return count;

}

int main() { int n;

cin >> n;

cout << countDiv3(n) << endl;

}

Problem 3: Minimum Coins to Make a Value Problem Statement:

You are given a target amount V and an array of coin denominations (e.g., [1, 2, 5, 10, 20, 50, 100, 500, 2000]). Find the minimum number of coins required to make that amount.

Input Format:

* First line: Integer V (1 ≤ V ≤ 10000)

Output Format:

* A single integer: Minimum number of coins needed Sample Test Cases:

Input: 70

Output: 2 // (50 + 20)

Input: 121

Output: 3 // (100 + 20 + 1)

Input: 999

Output: 6 // (500 + 200 + 200 + 50 + 20 + 20 + 5 + 2 + 2)

Python

def minCoins(V):

coins = [2000, 500, 100, 50, 20, 10, 5, 2, 1]

count = 0

for c in coins: if V == 0:

break

count += V // c V %= c

return count

print(minCoins(int(input())))

Java

import java.util.Scanner; public class Main {

static int minCoins(int V) {

int[] coins = {2000, 500, 100, 50, 20, 10, 5, 2, 1};

int count = 0;

for (int coin : coins) { if (V == 0) break; count += V / coin; V %= coin;

}

return count;

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int V = sc.nextInt(); System.out.println(minCoins(V));

}

} C

#include <stdio.h> int minCoins(int V) {

int coins[] = {2000, 500, 100, 50, 20, 10, 5, 2, 1};

int count = 0;

for (int i = 0; i < 9; i++) { if (V == 0) break; count += V / coins[i]; V %= coins[i];

}

return count;

}

int main() { int V;

scanf("%d", &V); printf("%d\n", minCoins(V)); return 0;

}

C++

#include <iostream> using namespace std;

int minCoins(int V) {

int coins[] = {2000, 500, 100, 50, 20, 10, 5, 2, 1};

int count = 0;

for (int coin : coins) { if (V == 0) break; count += V / coin; V %= coin;

}

return count;

}

int main() { int V;

cin >> V;

cout << minCoins(V) << endl;

}

Problem 4: Activity Selection Problem (Maximum Non-Overlapping Intervals) Problem Statement:

You are given n activities with start and end times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.

Input Format:

* First line: Integer n (number of activities)
* Next n lines: Two integers start and end (start time and end time of each activity) Output Format:
* A single integer: Maximum number of non-overlapping activities that can be selected Sample Test Cases:

Input:

6

1 2

3 4

0 6

5 7

8 9

5 9

Output:

4

Explanation:

Selected activities: (1,2), (3,4), (5,7), (8,9)

Input:

3

10 20

12 25

20 30

Output:

2

Input:

4

1 3

2 4

3 5

0 6

Output:

2

Python

def activitySelection(activities): activities.sort(key=lambda x: x[1]) # Sort by end time count = 1

last\_end = activities[0][1]

for i in range(1, len(activities)):

if activities[i][0] >= last\_end:

count += 1

last\_end = activities[i][1] return count

n = int(input())

activities = [tuple(map(int, input().split())) for \_ in range(n)] print(activitySelection(activities))

Java

import java.util.\*;

class Main {

public static int activitySelection(int[][] arr) { Arrays.sort(arr, Comparator.comparingInt(a -> a[1])); int count = 1;

int end = arr[0][1];

for (int i = 1; i < arr.length; i++) { if (arr[i][0] >= end) {

count++;

end = arr[i][1];

}

}

return count;

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int n = sc.nextInt();

int[][] activities = new int[n][2]; for (int i = 0; i < n; i++) {

activities[i][0] = sc.nextInt(); activities[i][1] = sc.nextInt();

}

System.out.println(activitySelection(activities));

}

}

C

#include <stdio.h> #include <stdlib.h>

typedef struct { int start, end;

} Activity;

int compare(const void \*a, const void \*b) { return ((Activity\*)a)->end - ((Activity\*)b)->end;

}

int main() { int n;

scanf("%d", &n);

Activity acts[n];

for (int i = 0; i < n; i++) {

scanf("%d %d", &acts[i].start, &acts[i].end);

}

qsort(acts, n, sizeof(Activity), compare);

int count = 1, last\_end = acts[0].end; for (int i = 1; i < n; i++) {

if (acts[i].start >= last\_end) { count++;

last\_end = acts[i].end;

}

}

printf("%d\n", count); return 0;

}

C++

#include <iostream> #include <vector> #include <algorithm> using namespace std;

struct Activity { int start, end;

};

bool compare(Activity a, Activity b) { return a.end < b.end;

}

int main() { int n;

cin >> n; vector<Activity> acts(n); for (int i = 0; i < n; ++i)

cin >> acts[i].start >> acts[i].end;

sort(acts.begin(), acts.end(), compare);

int count = 1;

int last\_end = acts[0].end;

for (int i = 1; i < n; i++) {

if (acts[i].start >= last\_end) { count++;

last\_end = acts[i].end;

}

}

cout << count << endl;

}

Problem 5: Job Sequencing with Deadlines

Problem Statement:

You are given n jobs. Each job has an ID, a deadline, and a profit. Only one job can be scheduled at a time. A job earns profit only if it is finished on or before its deadline. Schedule jobs to maximize total profit.

Input Format:

* First line: Integer n (number of jobs)
* Next n lines: JobID (char), Deadline (int), Profit (int) Output Format:
* Two space-separated integers:

1. Maximum number of jobs done
2. Maximum profit earned Sample Test Cases:

Input:

4

a 4 20

b 1 10

c 1 40

d 1 30

Output:

2 70

Input:

5

a 2 100

b 1 19

c 2 27

d 1 25

e 3 15

Output:

3 142

Input:

3

x 3 10

y 2 20

z 1 30

Output:

2 50

Python

def jobSequencing(jobs): jobs.sort(key=lambda x: x[2], reverse=True) max\_deadline = max(job[1] for job in jobs) slots = [False] \* (max\_deadline + 1)

profit, count = 0, 0

for job in jobs:

for j in range(job[1], 0, -1): if not slots[j]:

slots[j] = True count += 1 profit += job[2] break

return count, profit

n = int(input())

jobs = [tuple(input().split()) for \_ in range(n)]

jobs = [(job[0], int(job[1]), int(job[2])) for job in jobs] res = jobSequencing(jobs)

print(res[0], res[1])

Java

import java.util.\*;

class Job { String id;

int deadline, profit;

Job(String id, int d, int p) { this.id = id; this.deadline = d; this.profit = p;

}

}

public class Main {

public static int[] jobSequencing(Job[] jobs) { Arrays.sort(jobs, (a, b) -> b.profit - a.profit); int maxDeadline = 0;

for (Job job : jobs) maxDeadline = Math.max(maxDeadline, job.deadline);

boolean[] slot = new boolean[maxDeadline + 1]; int count = 0, profit = 0;

for (Job job : jobs) {

for (int j = job.deadline; j > 0; j--) { if (!slot[j]) {

slot[j] = true; count++;

profit += job.profit; break;

}

}

}

return new int[]{count, profit};

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int n = sc.nextInt();

Job[] jobs = new Job[n]; for (int i = 0; i < n; i++) { String id = sc.next();

int d = sc.nextInt(); int p = sc.nextInt();

jobs[i] = new Job(id, d, p);

}

int[] res = jobSequencing(jobs); System.out.println(res[0] + " " + res[1]);

}

}

C

#include <stdio.h> #include <stdlib.h> #include <string.h>

typedef struct { char id[2];

int deadline, profit;

} Job;

int compare(const void \*a, const void \*b) { return ((Job \*)b)->profit - ((Job \*)a)->profit;

}

int main() { int n;

scanf("%d", &n); Job jobs[n];

int max\_deadline = 0;

for (int i = 0; i < n; i++) {

scanf("%s %d %d", jobs[i].id, &jobs[i].deadline, &jobs[i].profit); if (jobs[i].deadline > max\_deadline)

max\_deadline = jobs[i].deadline;

}

qsort(jobs, n, sizeof(Job), compare);

int slot[max\_deadline + 1]; memset(slot, 0, sizeof(slot));

int count = 0, profit = 0;

for (int i = 0; i < n; i++) {

for (int j = jobs[i].deadline; j > 0; j--) { if (!slot[j]) {

slot[j] = 1; count++;

profit += jobs[i].profit; break;

}

}

}

printf("%d %d\n", count, profit); return 0;

}

C++

#include <iostream> #include <vector> #include <algorithm> using namespace std;

struct Job { string id;

int deadline;

int profit;

};

bool compare(Job a, Job b) { return a.profit > b.profit;

}

int main() { int n;

cin >> n; vector<Job> jobs(n);

int max\_deadline = 0;

for (int i = 0; i < n; ++i) {

cin >> jobs[i].id >> jobs[i].deadline >> jobs[i].profit; max\_deadline = max(max\_deadline, jobs[i].deadline);

}

sort(jobs.begin(), jobs.end(), compare); vector<bool> slot(max\_deadline + 1, false);

int count = 0, profit = 0; for (Job job : jobs) {

for (int j = job.deadline; j > 0; --j) { if (!slot[j]) {

slot[j] = true; count++;

profit += job.profit;

break;

}

}

}

cout << count << " " << profit << endl;

}

1. Problem: "Minimum XOR Sum of Two Arrays"

You are given two arrays nums1 and nums2 of length n. You need to assign each element in nums1 to a unique element in nums2, forming a 1-to-1 mapping. Your goal is to minimize the sum of XORs for each assigned pair.

Input:

* + nums1: List[int] — length n (1 ≤ n ≤ 14)
  + nums2: List[int] — same length as nums1
  + Each element: 0 ≤ nums1[i], nums2[i] ≤ 10⁹

Output:

* + Integer — the minimum possible XOR sum after assigning each element of nums1 to a unique element in nums2.

Example:

Input:

nums1 = [1, 2]

nums2 = [2, 3]

Output:

2

Explanation:

Two possible assignments:

1. 1^2 + 2^3 = 1 + 1 = 2
2. 1^3 + 2^2 = 2 + 0 = 2

Minimum = 2 Constraints Recap:

* + 1 ≤ n ≤ 14 — brute force over all n! permutations is too slow.
  + Must assign each nums1[i] to exactly one nums2[j]. Python code:

def minimumXORSum(nums1, nums2): n = len(nums1)

dp = [float('inf')] \* (1 << n)

dp[0] = 0 # base case: no elements assigned

for mask in range(1 << n):

k = bin(mask).count('1') # how many elements have been assigned so far for j in range(n):

if not (mask & (1 << j)):

new\_mask = mask | (1 << j)

dp[new\_mask] = min(dp[new\_mask], dp[mask] + (nums1[k] ^ nums2[j]))

return dp[(1 << n) - 1] Time Complexity:

* + O(n² \* 2^n)
  + Works fine for n ≤ 14

Space Complexity:

* + O(2^n) for the dp array

1. Traveling Salesman Problem (TSP) The Traveling Salesman Problem (TSP) is a classic problem in optimization where you need to find the shortest possible path that visits every node exactly once and returns to the starting node.

Example Problem:

Problem: Given a set of cities, and the distances between each pair of cities, find the shortest possible route that visits each city exactly once and returns to the origin city.

Test Case 1: Simple 4-City Case

Given 4 cities with the following distance matrix:

dist=[0 10 15 20

10 0 35 25

15 35 0 30

20 25 30 0]

Expected Output:

* The minimum cost to visit all cities and return to the starting city.

Test Case 2: Another 4-City Case

Given 4 cities with the following distance matrix:

dist=[0 29 20 21

29 0 15 17

20 15 0 28

21 17 28 0]

Expected Output:

* The minimum cost to visit all cities and return to the starting city. Test Case 3: 5-City Case

Given 5 cities with the following distance matrix:

dist=[0 10 15 20 25

10 0 35 25 30

15 35 0 30 35

20 25 30 0 40

25 30 35 40 0]

Expected Output:

* The minimum cost to visit all cities and return to the starting city. Test Case 4: 6-City Case

Given 6 cities with the following distance matrix:

dist=[0 12 18 29 12 25

12 0 22 30 13 23

18 22 0 15 35 20

29 30 15 0 28 10

12 13 35 28 0 18

25232010180]

Expected Output:

* The minimum cost to visit all cities and return to the starting city.

Test Case 5: Simple 3-City Case

Given 3 cities with the following distance matrix:

dist=[0 10 15

10 0 25

15 25 0]

Expected Output:

* The minimum cost to visit all cities and return to the starting city.

Code:

import sys

def tsp\_dp(dist): n = len(dist)

memo = [[-1] \* (1 << n) for \_ in range(n)] # Memoization table

# Recursive function to compute the minimum cost of visiting all cities def dfs(city, visited):

if visited == (1 << n) - 1:

return dist[city][0] # Return to the start city if memo[city][visited] != -1:

return memo[city][visited]

min\_cost = sys.maxsize for next\_city in range(n):

if visited & (1 << next\_city) == 0:

cost = dist[city][next\_city] + dfs(next\_city, visited | (1 << next\_city)) min\_cost = min(min\_cost, cost)

memo[city][visited] = min\_cost return min\_cost

# Start from the first city (city 0) and visit no other cities return dfs(0, 1)

# Test Case 1: Simple 4-City Case dist1 = [

[0, 10, 15, 20],

[10, 0, 35, 25],

[15, 35, 0, 30],

[20, 25, 30, 0]

]

print("Test Case 1 - Minimum Cost:", tsp\_dp(dist1))

# Test Case 2: Another 4-City Case dist2 = [

[0, 29, 20, 21],

[29, 0, 15, 17],

[20, 15, 0, 28],

[21, 17, 28, 0]

]

print("Test Case 2 - Minimum Cost:", tsp\_dp(dist2))

# Test Case 3: 5-City Case dist3 = [

[0, 10, 15, 20, 25],

[10, 0, 35, 25, 30],

[15, 35, 0, 30, 35],

[20, 25, 30, 0, 40],

[25, 30, 35, 40, 0]

]

print("Test Case 3 - Minimum Cost:", tsp\_dp(dist3))

# Test Case 4: 6-City Case dist4 = [

[0, 12, 18, 29, 12, 25],

[12, 0, 22, 30, 13, 23],

[18, 22, 0, 15, 35, 20],

[29, 30, 15, 0, 28, 10],

[12, 13, 35, 28, 0, 18],

[25, 23, 20, 10, 18, 0]

]

print("Test Case 4 - Minimum Cost:", tsp\_dp(dist4))

# Test Case 5: Simple 3-City Case dist5 = [

[0, 10, 15],

[10, 0, 25],

[15, 25, 0]

]

print("Test Case 5 - Minimum Cost:", tsp\_dp(dist5))

expected output:

Test Case 1 (4 cities):

* Minimum cost: 80 Test Case 2 (4 cities):
* Minimum cost: 69 Test Case 3 (5 cities):
* Minimum cost: 95 Test Case 4 (6 cities):
* Minimum cost: 129 Test Case 5 (3 cities):
* Minimum cost: 50

1. Next Greater Number with Same Digits Problem Statement:

You are given a positive integer N (can be up to 10⁶ digits, so it's a string). Find the next greater number

formed with the same set of digits.

* + If no such number exists, return -1. Constraints:
  + 1 <= len(N) <= 10⁶
  + N contains only digits 0-9
  + The answer must not start with a 0. Input/Output Examples:

Example 1:

Input:

N = "534976"

Output:

536479

Example 2:

Input:

N = "9876543210"

Output:

-1

Example 3:

Input:

N = "1234567899999999999"

Output:

1234567979999999999

Code:

def next\_greater\_number(n\_str): arr = list(n\_str)

n = len(arr)

# Step 1: Find the pivot i = n - 2

while i >= 0 and arr[i] >= arr[i + 1]: i -= 1

if i == -1:

return "-1" # Already largest permutation

# Step 2: Find rightmost successor to pivot j = n - 1

while arr[j] <= arr[i]: j -= 1

# Step 3: Swap pivot and successor arr[i], arr[j] = arr[j], arr[i]

# Step 4: Reverse the suffix arr[i + 1:] = reversed(arr[i + 1:])

# Final result result = ''.join(arr) if result[0] == '0':

return "-1" # Not allowed to start with zero return result

1. Partition Equal Subset Sum with Additional Constraints Problem Statement:

Given a list of integers nums[] (which may contain duplicate numbers and negative numbers), determine if it can be partitioned into two subsets such that:

1. The sum of the elements in both subsets is equal.
2. Subset sizes must be balanced (i.e., both subsets must have at least one element).
3. Handling large input efficiently (input can be up to 10^6 elements). Input:
   * A list of integers nums[] where 1 <= len(nums) <= 10^6 and -10^3 <= nums[i] <= 10^3.

Output:

* + Return True if such a partition exists, otherwise return False

Code:

def can\_partition(nums): total\_sum = sum(nums) if total\_sum % 2 != 0:

return False # If the sum is odd, partition is impossible

target = total\_sum // 2

# Adjusting for negative numbers by shifting all values to positive offset = sum(x < 0 for x in nums) \* 1000 # Shift all numbers >0 by 1000 dp = [False] \* (target + 1)

dp[0] = True # Sum of 0 is always possible with an empty subset

for num in nums:

for j in range(target, num - 1, -1): dp[j] = dp[j] or dp[j - num]

return dp[target]

1. Palindrome Partitioning II (Dynamic Programming)

Problem Description:

Given a string s, you need to find the minimum number of cuts needed to partition s into substrings such that every substring is a palindrome.

For example:

* + Input: s = "aab"
  + Output: 1

o Explanation: You can split "aab" into two substrings: "aa" and "b". Both "aa" and "b" are palindromes.

Code:

def minCut(s: str) -> int: n = len(s)

# isPalindrome[i][j] will be True if the string s[i:j+1] is a palindrome isPalindrome = [[False] \* n for \_ in range(n)]

# dp[i] will store the minimum cuts required for the substring s[0:i+1] dp = [0] \* n

for i in range(n):

minCuts = i # Maximum cuts for a substring s[0:i+1] for j in range(i + 1):

if s[j] == s[i] and (i - j <= 2 or isPalindrome[j + 1][i - 1]): isPalindrome[j][i] = True

minCuts = 0 if j == 0 else min(minCuts, dp[j - 1] + 1) dp[i] = minCuts

return dp[n - 1]

# Test cases test\_cases = [

"aab", # 1 cut: "aa" and "b"

"a", # 0 cuts: Single character is always a palindrome "ab", # 1 cut: "a" and "b"

"abac", # 1 cut: "aba" and "c"

"racecar", # 0 cuts: Whole string is a palindrome "ababbbabbababa" # 3 cuts

]

for s in test\_cases:

print(f"Input: {s}, Minimum cuts: {minCut(s)}")

Question 1:

Find the intersection of two sorted arrays OR in other words, given 2 sorted arrays, find all the elements which occur in both arrays.

NOTE: For the purpose of this problem ( as also conveyed by the sample case ), assume that elements that appear more than once in both arrays should be included multiple times in the final output.

Problem Constraints 1 <= |A| <= 106

1 <= |B| <= 106

Input Format

The first argument is an integer array A. The second argument is an integer array B. Output Format

Return an array of intersection of the two arrays.

Example Input Input 1:

A: [1 2 3 3 4 5 6]

B: [3 3 5]

Input 2:

A: [1 2 3 3 4 5 6]

B: [3 5]

Example Output Output 1: [3 3 5]

Output 2: [3 5] Example Explanation Explanation 1:

3, 3, 5 occurs in both the arrays A and B Explanation 2:

Only 3 and 5 occurs in both the arrays A and B

Question2:

Given an 2D integer array A of size N x 2 denoting time intervals of different meetings. Where:

* + A[i][0] = start time of the ith meeting.
  + A[i][1] = end time of the ith meeting.

Find the minimum number of conference rooms required so that all meetings can be done.

Note :- If a meeting ends at time t, another meeting starting at time t can use the same conference room Problem Constraints

1 <= N <= 105

0 <= A[i][0] < A[i][1] <= 2 \* 109

Input Format

The only argument given is the matrix A. Output Format

Return the minimum number of conference rooms required so that all meetings can be done. Example Input

Input 1:

A = [ [0, 30]

[5, 10]

[15, 20]

]

Input 2:

A = [ [1, 18]

[18, 23]

[15, 29]

[4, 15]

[2, 11]

[5, 13]

]

Example Output Output 1:

2

Output 2:

4

Example Explanation Explanation 1:

Meeting one can be done in conference room 1 form 0 - 30.

Meeting two can be done in conference room 2 form 5 - 10.

Meeting three can be done in conference room 2 form 15 - 20 as it is free in this interval. Explanation 2:

Meeting one can be done in conference room 1 from 1 - 18. Meeting five can be done in conference room 2 from 2 - 11. Meeting four can be done in conference room 3 from 4 - 15. Meeting six can be done in conference room 4 from 5 - 13.

Meeting three can be done in conference room 2 from 15 - 29 as it is free in this interval. Meeting two can be done in conference room 4 from 18 - 23 as it is free in this interval.

Question3:

N children are standing in a line. Each child is assigned a rating value.

You are giving candies to these children subjected to the following requirements:

1. Each child must have at least one candy.
2. Children with a higher rating get more candies than their neighbors. What is the minimum number of candies you must give?

Problem Constraints 1 <= N <= 105

-109 <= A[i] <= 109

Input Format

The first and only argument is an integer array A representing the rating of children. Output Format

Return an integer representing the minimum candies to be given. Example Input

Input 1:

A = [1, 2]

Input 2:

A = [1, 5, 2, 1]

Example Output Output 1:

3

Output 2:

7

Example Explanation Explanation 1:

The candidate with 1 rating gets 1 candy and candidate with rating 2 cannot get 1 candy as 1 is its neighbor.

So rating 2 candidate gets 2 candies. In total, 2 + 1 = 3 candies need to be given out. Explanation 2:

Candies given = [1, 3, 2, 1]

Question 4:

There is a street of length xxx whose positions are numbered 0, 1,…,x Initially there are no traffic lights,

but n sets of traffic lights are added to the street one after another.

Your task is to calculate the length of the longest passage without traffic lights after each addition. Input

The first input line contains two integers xxx and n: the length of the street and the number of sets of traffic lights.

Then, the next line contains n integers p1,p2,…,pn: the position of each set of traffic lights. Each position is distinct.

Output

Print the length of the longest passage without traffic lights after each addition. Example

Input:

8 3

3 6 2

Output:

5 3 3

Question5:

You are given n cubes in a certain order, and your task is to build towers using them. Whenever two cubes are one on top of the other, the upper cube must be smaller than the lower cube.

You must process the cubes in the given order. You can always either place the cube on top of an existing tower, or begin a new tower. What is the minimum possible number of towers?

Input

The first input line contains an integer n: the number of cubes.

The next line contains nnn integers k1,k2,…,kn: the sizes of the cubes.

Output

Print one integer: the minimum number of towers. Example

Input:

5

3 8 2 1 5

Output:

2

Greedy Algorithm based Problems

Q.1 Find Minimum number of Coins (Easy)

Given an amount of n Rupees and an infinite supply of each of the denominations {1, 2, 5, 10} valued coins/notes, The task is to find the minimum number of coins needed to make the given amount.

Examples:

Input: n= 39

Output: 6

Explanation: We need a 50 Rs note and a 20 Rs note. Input: n = 121

Output: 13

Explanation: We need a 100 Rs note, a 20 Rs note, and a 1 Rs coin.

The intuition would be to take coins with greater value first. This can reduce the total number of coins needed. Start from the largest possible denomination and keep adding denominations while the remaining value is greater than 0.

Python:

def findMin(n, denomination): count =0

# Traverse through all denomination

for i in range(len(denomination) - 1, -1, -1): # Find denominations

while n >= denomination[i]: n -= denomination[i] count +=1

return count

if name == ' main ':

denomination = [1, 2, 5, 10]

n = 39

print(findMin(n, denomination)) Output: 6

1. Minimum sum of two numbers formed from digits of an array (Medium)

Given an array of digits (values are from 0 to 9), the task is to find the minimum possible sum of two numbers formed using all digits of the array.

Note: We need to return the minimum possible sum as a string. Examples:

Input: arr[] = [6, 8, 4, 5, 2, 3, 0]

Output: “604”

Explanation: The minimum sum is formed by numbers 0358 and 246. Input: arr[] = [5, 3, 0, 7, 4]

Output: “82”

Explanation: The minimum sum is formed by numbers 35 and 047. Python Code:

# Function to add two strings and return the result def addString(s1, s2):

i = len(s1) - 1

j = len(s2) - 1

# initial carry is zero carry = 0

# we will calculate and store the

# resultant sum in reverse order in res res = []

while i >= 0 or j >= 0 or carry > 0: total = carry

if i >= 0:

total += int(s1[i]) if j >= 0:

total += int(s2[j]) res.append(str(total % 10)) carry = total // 10

i -= 1

j -= 1

# remove leading zeroes which are currently # at the back due to reversed string res while res and res[-1] == '0':

res.pop()

# reverse our final string

return ''.join(reversed(res)) if res else "0"

# Function to find minimum sum using count array approach def minSum(arr):

# Count array to store frequency of each digit count = [0] \* 10

# Count occurrences of each digit for num in arr:

count[num] += 1

# Two strings for storing the two minimum numbers s1 = []

s2 = []

# Flag to alternate between s1 and s2 firstNum = True

# Traverse count array from 0 to 9 for digit in range(10):

while count[digit] > 0: if firstNum:

if not (len(s1) == 0 and digit == 0): s1.append(str(digit))

firstNum = False else:

if not (len(s2) == 0 and digit == 0): s2.append(str(digit))

firstNum = True count[digit] -= 1

# Handle case where s1 or s2 might be empty if not s1:

s1.append("0") if not s2:

s2.append("0")

return addString(''.join(s1), ''.join(s2))

if name == " main ": arr = [6, 8, 4, 5, 2, 3, 0]

print(minSum(arr)) Output: 604

1. Minimum Cost to cut a board into squares (Hard)

Given a board of dimensions m × n that needs to be cut into m × n squares. The cost of making a cut along a horizontal or vertical edge is provided in two arrays:

* + x[]: Cutting costs along the vertical edges (length-wise).
  + y[]: Cutting costs along the horizontal edges (width-wise).

The task is to determine the minimum total cost required to cut the board into squares optimally. Examples:

Input: m = 6, n= 4, x[] = [2, 1, 3, 1, 4], y[] = [4, 1, 2]

Output: 42

Input: m = 4, n = 4, x[] = [1, 1, 1], y[] = [1, 1, 1]

Output: 15

Python Code:

def minimumCostOfBreaking(m, n, x, y):

# Sort the cutting costs in ascending order x.sort()

y.sort()

# Pieces in each direction hCount, vCount = 1, 1

i, j = len(x) - 1, len(y) - 1 totalCost = 0

# Process the cuts in greedy manner while i >= 0 and j >= 0:

# Choose the larger cost cut to # minimize future costs

if x[i] >= y[j]:

totalCost += x[i] \* hCount vCount += 1

i -= 1

else:

totalCost += y[j] \* vCount hCount += 1

j -= 1

# Process remaining vertical cuts while i >= 0:

totalCost += x[i] \* hCount vCount += 1

i -= 1

# Process remaining horizontal cuts while j >= 0:

totalCost += y[j] \* vCount hCount += 1

j -= 1

return totalCost # Driver Code

if name == " main ":

m, n = 6, 4

x = [2, 1, 3, 1, 4]

y = [4, 1, 2]

print(minimumCostOfBreaking(m, n, x, y)) Output: 42

Dynamic Programming based Problems:

1. Minimize steps to reach K from 0 by adding 1 or doubling at each step (Easy)

Given a positive integer K, the task is to find the minimum number of operations of the following two types, required to change 0 to K.

* + Add one to the operand
  + Multiply the operand by 2. Examples:

Input: K = 1 Output: 1 Input: K = 4 Output: 3 Python Code:

# Function to find minimum operations def minOperation(k):

# dp is initialised

# to store the steps dp = [0] \* (k + 1)

for i in range(1, k + 1):

dp[i] = dp[i - 1] + 1

# For all even numbers

if (i % 2 == 0):

dp[i]= min(dp[i], dp[i // 2] + 1)

return dp[k]

# Driver Code

if name == ' main ':

k = 12

print(minOperation(k))

Output: 5

Q.2. Maximum equal sum of three stacks

The three stacks s1, s2 and s3, each containing positive integers, are given. The task is to find the maximum possible equal sum that can be achieved by removing elements from the top of the stacks. Elements can be removed from the top of each stack, but the final sum of the remaining elements in all three stacks must be the same. The goal is to determine the maximum possible equal sum that can be achieved after removing elements.

Note: The stacks are represented as arrays, where the first index of the array corresponds to the top element of the stack.

Examples:

Input: s1 = [3, 2, 1, 1, 1], s2 = [2, 3, 4], s3 = [1, 4, 2, 5]

Output: 5

Input: s1 = [3, 10]

s2 = [4, 5]

s3 = [2, 1]

Output: 0 Python Code:

def max\_sum(s1, s2, s3): sum1 = sum(s1)

sum2 = sum(s2) sum3 = sum(s3)

top1 = top2 = top3 = 0

while True:

if top1 == len(s1) or top2 == len(s2) or top3 == len(s3): return 0

if sum1 == sum2 == sum3: return sum1

if sum1 >= sum2 and sum1 >= sum3: sum1 -= s1[top1]

top1 += 1

elif sum2 >= sum1 and sum2 >= sum3: sum2 -= s2[top2]

top2 += 1 else:

sum3 -= s3[top3] top3 += 1

s1 = [3, 2, 1, 1, 1]

s2 = [4, 3, 2]

s3 = [1, 1, 4, 1]

print(max\_sum(s1, s2, s3))

Output: 5

Q.3 . Find minimum time to finish all jobs with given constraints (Hard)

Given an array job[], where each element represents the time required to complete a specific job. There are k identical assignees available to complete these jobs, and each assignee takes t units of time to complete one unit of a job. The task is to determine the minimum time required to complete all jobs while following these constraints:

* Each assignee can only be assigned jobs that are contiguous in the given array. For example, an assignee can be assigned jobs (job[1], job[2], job[3]) but not (job[1], job[3]) (skipping job[2]).
* A single job cannot be divided between two assignees. Each job must be assigned to exactly one assignee.

Examples:

Input: job[] = {10, 7, 8, 12, 6, 8}, k = 4, t = 5

Output: 75

Input: job[] = {4, 5, 10}, k = 2, t = 5

Output: 50 Python Code:

# Python program to find the minimum time required # to finish all jobs using Binary Search

# Function to find the maximum job duration def GetMax(job):

res = job[0]

# Find the maximum time among all jobs for i in range(1, len(job)):

if job[i] > res: res = job[i]

return res

# Function to check if jobs can be completed within # 't' time using at most 'k' assignees

def IsPossible(job, t, k):

# Number of assignees required cnt = 1

# Time assigned to the current assignee curr = 0

i = 0

while i < len(job):

# If adding the current job exceeds 't', # assign a new assignee

if curr + job[i] > t:

curr = 0

cnt += 1 else:

# Otherwise, add job time to the # current assignee

curr += job[i] i += 1

return cnt <= k

# Function to find the minimum time required to # finish all jobs

def FindMinTime(job, k, t): start, end = 0, 0

# Compute the total time and the maximum # job duration

for j in job:

# Total sum of job times end += j

# Maximum job duration start = max(start, j)

# Initialize answer to the upper bound ans = end

# Perform binary search to find the minimum # feasible time

while start <= end:

mid = (start + end) // 2

# If jobs can be assigned within 'mid' time if IsPossible(job, mid, k):

ans = min(ans, mid) end = mid - 1

else:

start = mid + 1

# Return the minimum time required return ans \* t

if name == " main ": job = [10, 7, 8, 12, 6, 8]

k, t = 4, 5

print(FindMinTime(job, k, t))

Output: 75

LONGEST PALINDROMIC SUBSEQUENCE

You are given a string s and an integer k.

In one operation, you can replace the character at any position with the next or previous letter in the alphabet (wrapping around so that 'a' is after 'z'). For example, replacing 'a' with the next letter results in

'b', and replacing 'a' with the previous letter results in 'z'. Similarly, replacing 'z' with the next letter results in 'a', and replacing 'z' with the previous letter results in 'y'.

Return the length of the longest palindromic subsequence of s that can be obtained after performing at most k operations.

Example 1:

Input: s = "abced", k = 2 Output: 3

Explanation:

Replace s[1] with the next letter, and s becomes "acced". Replace s[4] with the previous letter, and s becomes "accec".

The subsequence "ccc" forms a palindrome of length 3, which is the maximum.

Example 2:

Input: s = "aaazzz", k = 4 Output: 6

Explanation:

Replace s[0] with the previous letter, and s becomes "zaazzz". Replace s[4] with the next letter, and s becomes "zaazaz".

Replace s[3] with the next letter, and s becomes "zaaaaz". The entire string forms a palindrome of length 6.

Constraints:

1 <= s.length <= 200

1 <= k <= 200

s consists of only lowercase English letters. TEST

Python class test:

def longestPalindromicSubsequence(self, s: str, k: int) -> int: @cache

def dfs(i: int, j: int, k: int) -> int: if i > j:

return 0 if i == j:

return 1

res = max(dfs(i + 1, j, k), dfs(i, j - 1, k)) d = abs(s[i] - s[j])

t = min(d, 26 - d) if t <= k:

res = max(res, dfs(i + 1, j - 1, k - t) + 2) return res

s = list(map(ord, s)) n = len(s)

ans = dfs(0, n - 1, k) dfs.cache\_clear() return ans

Java

class test { private char[] s;

private Integer[][][] f;

public int longestPalindromicSubsequence(String s, int k) { this.s = s.toCharArray();

int n = s.length();

f = new Integer[n][n][k + 1]; return dfs(0, n - 1, k);

}

private int dfs(int i, int j, int k) { if (i > j) {

return 0;

}

if (i == j) { return 1;

}

if (f[i][j][k] != null) { return f[i][j][k];

}

int res = Math.max(dfs(i + 1, j, k), dfs(i, j - 1, k)); int d = Math.abs(s[i] - s[j]);

int t = Math.min(d, 26 - d); if (t <= k) {

res = Math.max(res, 2 + dfs(i + 1, j - 1, k - t));

}

f[i][j][k] = res; return res;

}

} C++

class test { public:

int longestPalindromicSubsequence(string s, int k) { int n = s.size();

vector f(n, vector(n, vector<int>(k + 1, -1)));

auto dfs = [&](this auto&& dfs, int i, int j, int k) -> int {

if (i > j) { return 0;

}

if (i == j) { return 1;

}

if (f[i][j][k] != -1) { return f[i][j][k];

}

int res = max(dfs(i + 1, j, k), dfs(i, j - 1, k)); int d = abs(s[i] - s[j]);

int t = min(d, 26 - d); if (t <= k) {

res = max(res, 2 + dfs(i + 1, j - 1, k - t));

}

return f[i][j][k] = res;

};

return dfs(0, n - 1, k);

}

};

MAXIMUM NUMBER OF MATCHING INDICES

You are given two integer arrays, nums1 and nums2, of the same length. An index i is considered matching if nums1[i] == nums2[i].

Return the maximum number of matching indices after performing any number of right shifts on nums1. A right shift is defined as shifting the element at index i to index (i + 1) % n, for all indices.

Example 1:

Input: nums1 = [3,1,2,3,1,2], nums2 = [1,2,3,1,2,3]

Output: 6 Explanation:

If we right shift nums1 2 times, it becomes [1, 2, 3, 1, 2, 3]. Every index matches, so the output is 6.

Example 2:

Input: nums1 = [1,4,2,5,3,1], nums2 = [2,3,1,2,4,6]

Output: 3 Explanation:

If we right shift nums1 3 times, it becomes [5, 3, 1, 1, 4, 2]. Indices 1, 2, and 4 match, so the output is 3.

Constraints:

* nums1.length == nums2.length
* 1 <= nums1.length, nums2.length <= 3000
* 1 <= nums1[i], nums2[i] <= 109 Tests

Python3 class test:

def maximumMatchingIndices(self, nums1: List[int], nums2: List[int]) -> int: n = len(nums1)

ans = 0

for k in range(n):

t = sum(nums1[(i + k) % n] == x for i, x in enumerate(nums2)) ans = max(ans, t)

return ans

Java

class test {

public int maximumMatchingIndices(int[] nums1, int[] nums2) { int n = nums1.length;

int ans = 0;

for (int k = 0; k < n; ++k) { int t = 0;

for (int i = 0; i < n; ++i) {

if (nums1[(i + k) % n] == nums2[i]) {

++t;

}

}

ans = Math.max(ans, t);

}

return ans;

}

} C++

class test { public:

int maximumMatchingIndices(vector<int>& nums1, vector<int>& nums2) { int n = nums1.size();

int ans = 0;

for (int k = 0; k < n; ++k) { int t = 0;

for (int i = 0; i < n; ++i) {

if (nums1[(i + k) % n] == nums2[i]) {

++t;

}

}

ans = max(ans, t);

}

return ans;

}

};

REVERSE DEGREE OF A STRING

Given a string s, calculate its reverse degree. The reverse degree is calculated as follows:

1. For each character, multiply its position in the reversed alphabet ('a' = 26, 'b' = 25, ..., 'z' = 1) with its position in the string (1-indexed).
2. Sum these products for all characters in the string. Return the reverse degree of s.

Example 1:

Input: s = "abc" Output: 148 Explanation:

Letter Index in Reversed Alphabet Index in String Product

|  |  |  |  |
| --- | --- | --- | --- |
| 'a' | 26 | 1 | 26 |
| 'b' | 25 | 2 | 50 |
| 'c' | 24 | 3 | 72 |

The reversed degree is 26 + 50 + 72 = 148.

Example 2:

Input: s = "zaza" Output: 160 Explanation:

Letter Index in Reversed Alphabet Index in String Product

|  |  |  |  |
| --- | --- | --- | --- |
| 'z' | 1 | 1 | 1 |
| 'a' | 26 | 2 | 52 |
| 'z' | 1 | 3 | 3 |

'a' 26 4 104

The reverse degree is 1 + 52 + 3 + 104 = 160.

Constraints:

* 1 <= s.length <= 1000
* s contains only lowercase English letters. Tests

Python3 class test:

def reverseDegree(self, s: str) -> int: ans = 0

for i, c in enumerate(s, 1): x = 26 - (ord(c) - ord("a")) ans += i \* x

return ans

Java

class test {

public int reverseDegree(String s) { int n = s.length();

int ans = 0;

for (int i = 1; i <= n; ++i) {

int x = 26 - (s.charAt(i - 1) - 'a'); ans += i \* x;

}

return ans;

}

} C++

class test { public:

int reverseDegree(string s) { int n = s.length();

int ans = 0;

for (int i = 1; i <= n; ++i) { int x = 26 - (s[i - 1] - 'a'); ans += i \* x;

}

return ans;

}

};

HIGHEST-RANKED ITEMS

You are given a 0-indexed 2D integer array grid of size m x n that represents a map of the items in a shop. The integers in the grid represent the following:

* 0 represents a wall that you cannot pass through.
* 1 represents an empty cell that you can freely move to and from.
* All other positive integers represent the price of an item in that cell. You may also freely move to and from these item cells.

It takes 1 step to travel between adjacent grid cells.

You are also given integer arrays pricing and start where pricing = [low, high] and start = [row, col] indicates that you start at the position (row, col) and are interested only in items with a price in the range of [low, high] (inclusive). You are further given an integer k.

You are interested in the positions of the k highest-ranked items whose prices are within the given price range. The rank is determined by the first of these criteria that is different:

1. Distance, defined as the length of the shortest path from the start (shorter distance has a higher rank).
2. Price (lower price has a higher rank, but it must be in the price range).
3. The row number (smaller row number has a higher rank).
4. The column number (smaller column number has a higher rank).

Return the k highest-ranked items within the price range sorted by their rank (highest to lowest). If there are fewer than k reachable items within the price range, return all of them.

Example 1:

Input: grid = [[1,2,0,1],[1,3,0,1],[0,2,5,1]], pricing = [2,5], start = [0,0], k = 3

Output: [[0,1],[1,1],[2,1]]

Explanation: You start at (0,0).

With a price range of [2,5], we can take items from (0,1), (1,1), (2,1) and (2,2). The ranks of these items are:

* (0,1) with distance 1
* (1,1) with distance 2
* (2,1) with distance 3
* (2,2) with distance 4

Thus, the 3 highest ranked items in the price range are (0,1), (1,1), and (2,1). Example 2:

Input: grid = [[1,2,0,1],[1,3,3,1],[0,2,5,1]], pricing = [2,3], start = [2,3], k = 2

Output: [[2,1],[1,2]]

Explanation: You start at (2,3).

With a price range of [2,3], we can take items from (0,1), (1,1), (1,2) and (2,1). The ranks of these items are:

* (2,1) with distance 2, price 2
* (1,2) with distance 2, price 3
* (1,1) with distance 3
* (0,1) with distance 4

Thus, the 2 highest ranked items in the price range are (2,1) and (1,2).

Example 3:

Input: grid = [[1,1,1],[0,0,1],[2,3,4]], pricing = [2,3], start = [0,0], k = 3

Output: [[2,1],[2,0]]

Explanation: You start at (0,0).

With a price range of [2,3], we can take items from (2,0) and (2,1). The ranks of these items are:

* (2,1) with distance 5
* (2,0) with distance 6

Thus, the 2 highest ranked items in the price range are (2,1) and (2,0). Note that k = 3 but there are only 2 reachable items within the price range.

Constraints:

* m == grid.length
* n == grid[i].length
* 1 <= m, n <= 105
* 1 <= m \* n <= 105
* 0 <= grid[i][j] <= 105
* pricing.length == 2
* 2 <= low <= high <= 105
* start.length == 2
* 0 <= row <= m - 1
* 0 <= col <= n - 1
* grid[row][col] > 0
* 1 <= k <= m \* n Tests

Python3 class Test:

def highestRankedKItems(

self, grid: List[List[int]], pricing: List[int], start: List[int], k: int

) -> List[List[int]]:

m, n = len(grid), len(grid[0]) row, col = start

low, high = pricing

q = deque([(row, col)]) pq = []

if low <= grid[row][col] <= high: pq.append((0, grid[row][col], row, col))

grid[row][col] = 0

dirs = (-1, 0, 1, 0, -1)

step = 0 while q:

step += 1

for \_ in range(len(q)):

x, y = q.popleft()

for a, b in pairwise(dirs):

nx, ny = x + a, y + b

if 0 <= nx < m and 0 <= ny < n and grid[nx][ny] > 0: if low <= grid[nx][ny] <= high:

pq.append((step, grid[nx][ny], nx, ny)) grid[nx][ny] = 0

q.append((nx, ny))

pq.sort()

return [list(x[2:]) for x in pq[:k]]

Java

class Test {

public List<List<Integer>> highestRankedKItems( int[][] grid, int[] pricing, int[] start, int k) {

int m = grid.length; int n = grid[0].length;

int row = start[0], col = start[1];

int low = pricing[0], high = pricing[1]; Deque<int[]> q = new ArrayDeque<>(); q.offer(new int[] {row, col});

List<int[]> pq = new ArrayList<>();

if (low <= grid[row][col] && grid[row][col] <= high) { pq.add(new int[] {0, grid[row][col], row, col});

}

grid[row][col] = 0;

final int[] dirs = {-1, 0, 1, 0, -1};

for (int step = 1; !q.isEmpty(); ++step) { for (int size = q.size(); size > 0; --size) {

int[] curr = q.poll();

int x = curr[0], y = curr[1]; for (int j = 0; j < 4; j++) {

int nx = x + dirs[j];

int ny = y + dirs[j + 1];

if (0 <= nx && nx < m && 0 <= ny && ny < n && grid[nx][ny] > 0) { if (low <= grid[nx][ny] && grid[nx][ny] <= high) {

pq.add(new int[] {step, grid[nx][ny], nx, ny});

}

grid[nx][ny] = 0; q.offer(new int[] {nx, ny});

}

}

}

}

pq.sort((a, b) -> {

if (a[0] != b[0]) return Integer.compare(a[0], b[0]);

if (a[1] != b[1]) return Integer.compare(a[1], b[1]);

if (a[2] != b[2]) return Integer.compare(a[2], b[2]); return Integer.compare(a[3], b[3]);

});

List<List<Integer>> ans = new ArrayList<>(); for (int i = 0; i < Math.min(k, pq.size()); i++) { ans.add(List.of(pq.get(i)[2], pq.get(i)[3]));

}

return ans;

}

} C++

class Test { public:

vector<vector<int>> highestRankedKItems(vector<vector<int>>& grid, vector<int>& pricing, vector<int>& start, int k) {

int m = grid.size(), n = grid[0].size(); int row = start[0], col = start[1];

int low = pricing[0], high = pricing[1]; queue<pair<int, int>> q; q.push({row, col});

vector<tuple<int, int, int, int>> pq;

if (low <= grid[row][col] && grid[row][col] <= high) { pq.push\_back({0, grid[row][col], row, col});

}

grid[row][col] = 0;

vector<int> dirs = {-1, 0, 1, 0, -1}; for (int step = 1; q.size(); ++step) {

int sz = q.size();

for (int i = 0; i < sz; ++i) { auto [x, y] = q.front(); q.pop();

for (int j = 0; j < 4; ++j) { int nx = x + dirs[j];

int ny = y + dirs[j + 1];

if (0 <= nx && nx < m && 0 <= ny && ny < n && grid[nx][ny] > 0) { if (low <= grid[nx][ny] && grid[nx][ny] <= high) {

pq.push\_back({step, grid[nx][ny], nx, ny});

}

grid[nx][ny] = 0;

q.push({nx, ny});

}

}

}

}

sort(pq.begin(), pq.end()); vector<vector<int>> ans;

for (int i = 0; i < min(k, (int) pq.size()); ++i) {

ans.push\_back({get<2>(pq[i]), get<3>(pq[i])});

}

return ans;

}

};

GOOD TRIPLET

You are given two 0-indexed arrays nums1 and nums2 of length n, both of which are permutations of [0, 1, ..., n - 1].

A good triplet is a set of 3 distinct values which are present in increasing order by position both in nums1 and nums2. In other words, if we consider pos1v as the index of the value v in nums1 and pos2v as the index of the value v in nums2, then a good triplet will be a set (x, y, z) where 0 <= x, y, z <= n - 1, such that pos1x < pos1y < pos1z and pos2x < pos2y < pos2z.

Return the total number of good triplets.

Example 1:

Input: nums1 = [2,0,1,3], nums2 = [0,1,2,3] Output: 1

Explanation:

There are 4 triplets (x,y,z) such that pos1x < pos1y < pos1z. They are (2,0,1), (2,0,3), (2,1,3), and (0,1,3).

Out of those triplets, only the triplet (0,1,3) satisfies pos2x < pos2y < pos2z. Hence, there is only 1 good triplet.

Example 2:

Input: nums1 = [4,0,1,3,2], nums2 = [4,1,0,2,3]

Output: 4

Explanation: The 4 good triplets are (4,0,3), (4,0,2), (4,1,3), and (4,1,2).

Constraints:

* n == nums1.length == nums2.length
* 3 <= n <= 105
* 0 <= nums1[i], nums2[i] <= n - 1
* nums1 and nums2 are permutations of [0, 1, ..., n - 1].

Tests

Test 1: Binary Indexed Tree (Fenwick Tree)

For this problem, we first use pos to record the position of each number in nums2, and then process each element in nums1 sequentially.

Consider the number of good triplets with the current number as the middle number. The first number must have already been traversed and must appear earlier than the current number in nums2. The third number must not yet have been traversed and must appear later than the current number in nums2.

Take nums1 = [4,0,1,3,2] and nums2 = [4,1,0,2,3] as an example. Consider the traversal process:

1. First, process 4. At this point, the state of nums2 is [4,X,X,X,X]. The number of values before 4 is 0, and the number of values after 4 is 4. Therefore, 4 as the middle number forms 0 good triplets.
2. Next, process 0. The state of nums2 becomes [4,X,0,X,X]. The number of values before 0 is 1, and the number of values after 0 is 2. Therefore, 0 as the middle number forms 2 good triplets.
3. Next, process 1. The state of nums2 becomes [4,1,0,X,X]. The number of values before 1 is 1, and the number of values after 1 is 2. Therefore, 1 as the middle number forms 2 good triplets.
4. ...
5. Finally, process 2. The state of nums2 becomes [4,1,0,2,3]. The number of values before 2 is 4, and the number of values after 2 is 0. Therefore, 2 as the middle number forms 0 good triplets.

We can use a Binary Indexed Tree (Fenwick Tree) to update the occurrence of numbers at each position in nums2, and quickly calculate the number of 1s to the left of each number and the number of 0s to the right of each number.

A Binary Indexed Tree, also known as a Fenwick Tree, efficiently supports the following operations:

1. Point Update update(x, delta): Add a value delta to the number at position x in the sequence.
2. Prefix Sum Query query(x): Query the sum of the sequence in the range [1, ..., x], i.e., the prefix sum at position x.

Both operations have a time complexity of O(log⁡n). Therefore, the overall time complexity is O(nlog⁡n), where n is the length of the array nums1. The space complexity is O(n).

Python3

class BinaryIndexedTree: def init (self, n):

self.n = n

self.c = [0] \* (n + 1)

@staticmethod def lowbit(x):

return x & -x

def update(self, x, delta):

while x <= self.n: self.c[x] += delta

x += BinaryIndexedTree.lowbit(x)

def query(self, x):

s = 0

while x > 0:

s += self.c[x]

x -= BinaryIndexedTree.lowbit(x) return s

class Test:

def goodTriplets(self, nums1: List[int], nums2: List[int]) -> int: pos = {v: i for i, v in enumerate(nums2, 1)}

ans = 0

n = len(nums1)

tree = BinaryIndexedTree(n) for num in nums1:

p = pos[num]

left = tree.query(p)

right = n - p - (tree.query(n) - tree.query(p)) ans += left \* right

tree.update(p, 1) return ans

Java

class Test {

public long goodTriplets(int[] nums1, int[] nums2) { int n = nums1.length;

int[] pos = new int[n];

BinaryIndexedTree tree = new BinaryIndexedTree(n); for (int i = 0; i < n; ++i) {

pos[nums2[i]] = i + 1;

}

long ans = 0;

for (int num : nums1) { int p = pos[num];

long left = tree.query(p);

long right = n - p - (tree.query(n) - tree.query(p)); ans += left \* right;

tree.update(p, 1);

}

return ans;

}

}

class BinaryIndexedTree { private int n;

private int[] c;

public BinaryIndexedTree(int n) {

this.n = n;

c = new int[n + 1];

}

public void update(int x, int delta) { while (x <= n) {

c[x] += delta; x += lowbit(x);

}

}

public int query(int x) { int s = 0;

while (x > 0) { s += c[x];

x -= lowbit(x);

}

return s;

}

public static int lowbit(int x) { return x & -x;

}

} C++

class BinaryIndexedTree { public:

int n;

vector<int> c;

BinaryIndexedTree(int \_n)

: n(\_n)

, c(\_n + 1) {}

void update(int x, int delta) { while (x <= n) {

c[x] += delta; x += lowbit(x);

}

}

int query(int x) { int s = 0; while (x > 0) {

s += c[x];

x -= lowbit(x);

}

return s;

}

int lowbit(int x) { return x & -x;

}

};

class Test {

public:

long long goodTriplets(vector<int>& nums1, vector<int>& nums2) { int n = nums1.size();

vector<int> pos(n);

for (int i = 0; i < n; ++i) pos[nums2[i]] = i + 1; BinaryIndexedTree\* tree = new BinaryIndexedTree(n); long long ans = 0;

for (int& num : nums1) { int p = pos[num];

int left = tree->query(p);

int right = n - p - (tree->query(n) - tree->query(p)); ans += 1ll \* left \* right;

tree->update(p, 1);

}

return ans;

}

};

Problem 1: Lexicographically Smallest String by Removing One Character Problem Statement:

Given a string s, remove exactly one character such that the resulting string is lexicographically smallest. Return the modified string.

Input Format:

* A single string s consisting of lowercase English letters. Output Format:
* A single string which is lexicographically smallest after removing one character.

Sample Test Cases:

Input: abcda Output: abca

Input: aaa Output: aa

Input: edcba Output: dcba Python

def smallestString(s):

for i in range(len(s)-1):

if s[i] > s[i+1]:

return s[:i] + s[i+1:] return s[:-1]

# Read input s = input()

rint(smallestString(s))

Java

import java.util.Scanner;

public class Main {

public static String smallestString(String s) { for (int i = 0; i < s.length() - 1; i++) {

if (s.charAt(i) > s.charAt(i + 1)) {

return s.substring(0, i) + s.substring(i + 1);

}

}

return s.substring(0, s.length() - 1);

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); String input = sc.nextLine(); System.out.println(smallestString(input));

}

} C

#include <stdio.h> #include <string.h>

void smallestString(char s[]) { int len = strlen(s), i;

for (i = 0; i < len - 1; i++) { if (s[i] > s[i + 1]) break;

}

for (int j = 0; j < len; j++) {

if (j != i) printf("%c", s[j]);

}

printf("\n");

}

int main() { char s[101];

scanf("%s", s); smallestString(s); return 0;

}

C++

#include <iostream> using namespace std;

string smallestString(string s) {

for (int i = 0; i < s.size() - 1; ++i) { if (s[i] > s[i+1])

return s.substr(0, i) + s.substr(i+1);

}

return s.substr(0, s.size() - 1);

}

int main() { string s; cin >> s;

cout << smallestString(s) << endl;

}

Problem 2: Count Numbers with Digit Sum Divisible by 3 Problem Statement:

Given a number n, count how many numbers from 1 to n have a digit sum divisible by 3. Input Format:

* A single integer n (1 ≤ n ≤ 1000)

Output Format:

* A single integer: the count of numbers from 1 to n whose digit sum is divisible by 3. Sample Test Cases:

Input: 15

Output: 5

Input: 20

Output: 6

Input: 30

Output: 10 Python

def countDiv3(n):

return sum(1 for i in range(1, n+1) if sum(map(int, str(i))) % 3 == 0)

n = int(input()) print(countDiv3(n))

Java

import java.util.Scanner;

public class Main {

public static int countDiv3(int n) { int count = 0;

for (int i = 1; i <= n; i++) { int sum = 0, temp = i; while (temp > 0) {

sum += temp % 10;

temp /= 10;

}

if (sum % 3 == 0) count++;

}

return count;

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int n = sc.nextInt(); System.out.println(countDiv3(n));

}

}

C

#include <stdio.h>

int digitSum(int n) { int sum = 0; while(n) {

sum += n % 10; n /= 10;

}

return sum;

}

int countDiv3(int n) { int count = 0;

for (int i = 1; i <= n; i++) { if (digitSum(i) % 3 == 0)

count++;

}

return count;

}

int main() { int n;

scanf("%d", &n); printf("%d\n", countDiv3(n)); return 0;

}

C++

#include <iostream> using namespace std;

int digitSum(int n) { int sum = 0; while(n) {

sum += n % 10; n /= 10;

}

return sum;

}

int countDiv3(int n) {

int count = 0;

for (int i = 1; i <= n; i++) { if (digitSum(i) % 3 == 0)

count++;

}

return count;

}

int main() { int n;

cin >> n;

cout << countDiv3(n) << endl;

}

Problem 3: Minimum Coins to Make a Value Problem Statement:

You are given a target amount V and an array of coin denominations (e.g., [1, 2, 5, 10, 20, 50, 100, 500, 2000]). Find the minimum number of coins required to make that amount.

Input Format:

* First line: Integer V (1 ≤ V ≤ 10000)

Output Format:

* A single integer: Minimum number of coins needed Sample Test Cases:

Input: 70

Output: 2 // (50 + 20)

Input: 121

Output: 3 // (100 + 20 + 1)

Input: 999

Output: 6 // (500 + 200 + 200 + 50 + 20 + 20 + 5 + 2 + 2)

Python

def minCoins(V):

coins = [2000, 500, 100, 50, 20, 10, 5, 2, 1]

count = 0

for c in coins:

if V == 0:

break

count += V // c V %= c

return count

print(minCoins(int(input())))

Java

import java.util.Scanner; public class Main {

static int minCoins(int V) {

int[] coins = {2000, 500, 100, 50, 20, 10, 5, 2, 1};

int count = 0;

for (int coin : coins) { if (V == 0) break; count += V / coin; V %= coin;

}

return count;

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int V = sc.nextInt(); System.out.println(minCoins(V));

}

} C

#include <stdio.h> int minCoins(int V) {

int coins[] = {2000, 500, 100, 50, 20, 10, 5, 2, 1};

int count = 0;

for (int i = 0; i < 9; i++) { if (V == 0) break; count += V / coins[i]; V %= coins[i];

}

return count;

}

int main() { int V;

scanf("%d", &V); printf("%d\n", minCoins(V)); return 0;

}

C++

#include <iostream>

using namespace std;

int minCoins(int V) {

int coins[] = {2000, 500, 100, 50, 20, 10, 5, 2, 1};

int count = 0;

for (int coin : coins) { if (V == 0) break; count += V / coin; V %= coin;

}

return count;

}

int main() { int V;

cin >> V;

cout << minCoins(V) << endl;

}

Problem 4: Activity Selection Problem (Maximum Non-Overlapping Intervals) Problem Statement:

You are given n activities with start and end times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.

Input Format:

* First line: Integer n (number of activities)
* Next n lines: Two integers start and end (start time and end time of each activity) Output Format:
* A single integer: Maximum number of non-overlapping activities that can be selected Sample Test Cases:

Input:

6

1 2

3 4

0 6

5 7

8 9

5 9

Output:

4

Explanation:

Selected activities: (1,2), (3,4), (5,7), (8,9)

Input:

3

10 20

12 25

20 30

Output:

2

Input:

4

1 3

2 4

3 5

0 6

Output:

2

Python

def activitySelection(activities): activities.sort(key=lambda x: x[1]) # Sort by end time count = 1

last\_end = activities[0][1]

for i in range(1, len(activities)):

if activities[i][0] >= last\_end:

count += 1

last\_end = activities[i][1] return count

n = int(input())

activities = [tuple(map(int, input().split())) for \_ in range(n)] print(activitySelection(activities))

Java

import java.util.\*;

class Main {

public static int activitySelection(int[][] arr) { Arrays.sort(arr, Comparator.comparingInt(a -> a[1]));

int count = 1;

int end = arr[0][1];

for (int i = 1; i < arr.length; i++) { if (arr[i][0] >= end) {

count++;

end = arr[i][1];

}

}

return count;

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int n = sc.nextInt();

int[][] activities = new int[n][2]; for (int i = 0; i < n; i++) {

activities[i][0] = sc.nextInt(); activities[i][1] = sc.nextInt();

}

System.out.println(activitySelection(activities));

}

}

C

#include <stdio.h> #include <stdlib.h>

typedef struct {

int start, end;

} Activity;

int compare(const void \*a, const void \*b) { return ((Activity\*)a)->end - ((Activity\*)b)->end;

}

int main() { int n;

scanf("%d", &n); Activity acts[n];

for (int i = 0; i < n; i++) {

scanf("%d %d", &acts[i].start, &acts[i].end);

}

qsort(acts, n, sizeof(Activity), compare);

int count = 1, last\_end = acts[0].end; for (int i = 1; i < n; i++) {

if (acts[i].start >= last\_end) { count++;

last\_end = acts[i].end;

}

}

printf("%d\n", count); return 0;

}

C++

#include <iostream> #include <vector> #include <algorithm> using namespace std;

struct Activity { int start, end;

};

bool compare(Activity a, Activity b) { return a.end < b.end;

}

int main() { int n;

cin >> n; vector<Activity> acts(n); for (int i = 0; i < n; ++i)

cin >> acts[i].start >> acts[i].end;

sort(acts.begin(), acts.end(), compare);

int count = 1;

int last\_end = acts[0].end;

for (int i = 1; i < n; i++) {

if (acts[i].start >= last\_end) { count++;

last\_end = acts[i].end;

}

}

cout << count << endl;

}

Problem 5: Job Sequencing with Deadlines Problem Statement:

You are given n jobs. Each job has an ID, a deadline, and a profit. Only one job can be scheduled at a time. A job earns profit only if it is finished on or before its deadline. Schedule jobs to maximize total profit.

Input Format:

* First line: Integer n (number of jobs)
* Next n lines: JobID (char), Deadline (int), Profit (int) Output Format:
* Two space-separated integers:

1. Maximum number of jobs done
2. Maximum profit earned Sample Test Cases:

Input:

4

a 4 20

b 1 10

c 1 40

d 1 30

Output:

2 70

Input:

5

a 2 100

b 1 19

c 2 27

d 1 25

e 3 15

Output:

3 142

Input:

3

x 3 10

y 2 20

z 1 30

Output:

2 50

Python

def jobSequencing(jobs): jobs.sort(key=lambda x: x[2], reverse=True) max\_deadline = max(job[1] for job in jobs) slots = [False] \* (max\_deadline + 1)

profit, count = 0, 0

for job in jobs:

for j in range(job[1], 0, -1):

if not slots[j]:

slots[j] = True count += 1 profit += job[2] break

return count, profit

n = int(input())

jobs = [tuple(input().split()) for \_ in range(n)]

jobs = [(job[0], int(job[1]), int(job[2])) for job in jobs] res = jobSequencing(jobs)

print(res[0], res[1])

Java

import java.util.\*;

class Job { String id;

int deadline, profit;

Job(String id, int d, int p) { this.id = id; this.deadline = d; this.profit = p;

}

}

public class Main {

public static int[] jobSequencing(Job[] jobs) { Arrays.sort(jobs, (a, b) -> b.profit - a.profit); int maxDeadline = 0;

for (Job job : jobs) maxDeadline = Math.max(maxDeadline, job.deadline);

boolean[] slot = new boolean[maxDeadline + 1]; int count = 0, profit = 0;

for (Job job : jobs) {

for (int j = job.deadline; j > 0; j--) { if (!slot[j]) {

slot[j] = true; count++;

profit += job.profit; break;

}

}

}

return new int[]{count, profit};

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int n = sc.nextInt();

Job[] jobs = new Job[n];

for (int i = 0; i < n; i++) { String id = sc.next(); int d = sc.nextInt(); int p = sc.nextInt();

jobs[i] = new Job(id, d, p);

}

int[] res = jobSequencing(jobs); System.out.println(res[0] + " " + res[1]);

}

} C

#include <stdio.h> #include <stdlib.h> #include <string.h>

typedef struct { char id[2];

int deadline, profit;

} Job;

int compare(const void \*a, const void \*b) { return ((Job \*)b)->profit - ((Job \*)a)->profit;

}

int main() { int n;

scanf("%d", &n); Job jobs[n];

int max\_deadline = 0;

for (int i = 0; i < n; i++) {

scanf("%s %d %d", jobs[i].id, &jobs[i].deadline, &jobs[i].profit); if (jobs[i].deadline > max\_deadline)

max\_deadline = jobs[i].deadline;

}

qsort(jobs, n, sizeof(Job), compare);

int slot[max\_deadline + 1]; memset(slot, 0, sizeof(slot));

int count = 0, profit = 0;

for (int i = 0; i < n; i++) {

for (int j = jobs[i].deadline; j > 0; j--) { if (!slot[j]) {

slot[j] = 1; count++;

profit += jobs[i].profit; break;

}

}

}

printf("%d %d\n", count, profit); return 0;

}

C++

#include <iostream> #include <vector> #include <algorithm> using namespace std;

struct Job { string id;

int deadline; int profit;

};

bool compare(Job a, Job b) { return a.profit > b.profit;

}

int main() { int n;

cin >> n; vector<Job> jobs(n);

int max\_deadline = 0;

for (int i = 0; i < n; ++i) {

cin >> jobs[i].id >> jobs[i].deadline >> jobs[i].profit; max\_deadline = max(max\_deadline, jobs[i].deadline);

}

sort(jobs.begin(), jobs.end(), compare); vector<bool> slot(max\_deadline + 1, false);

int count = 0, profit = 0; for (Job job : jobs) {

for (int j = job.deadline; j > 0; --j) { if (!slot[j]) {

slot[j] = true; count++;

profit += job.profit; break;

}

}

}

cout << count << " " << profit << endl;

}

Question 1:

Ramu has N dishes of different types arranged in a row: A1,A2,…,AN where Ai denotes the type of the ith dish. He wants to choose as many dishes as possible from the given list but while satisfying two conditions:

1. He can choose only one type of dish.
2. No two chosen dishes should be adjacent to each other.

Ramu wants to know which type of dish he should choose from, so that he can pick the maximum number of dishes.

Example :

Given N= 9 and A= [1,2,2,1,2,1,1,1,1]

For type 1, Ramu can choose at most four dishes. One of the ways to choose four dishes of type 1 is A1,A4, A7 and A9.

For type 2, Ramu can choose at most two dishes. One way is to choose A3 and A5. So in this case, Ramu should go for type 1, in which he can pick more dishes.

INPUT FORMAT:

* The first line contains T, the number of test cases. Then the test cases follow.
* For each test case, the first line contains a single integer N.
* The second line contains N integers A1,A2,…,AN.

OUTPUT FORMAT

For each test case, print a single line containing one integer ― the type of the dish that Ramu should

choose from. If there are multiple answers, print the smallest one. CONSTRAINTS :

* 1 <= T <= 10^3
* 1 <= N <= 10^3
* 1 <= Ai <= 10^3

Sample Input : 3

5

1 2 2 1 2

6

1 1 1 1 1 1

8

1 2 2 2 3 4 2 1

Sample Output :

1

1

2

C++

#include<bits/stdc++.h> using namespace std; int main()

{

int t; //number of test cases cin>>t;

for (int i = 0; i < t; i++) { int n; //number of items cin>>n;

int item[n]; //array for items for (int x = 0; x < n; x++)

{

cin>>item[x];

}

int j = 0, max = 0;

int itemtype = item[0]; while (j < n)

{

int c = 1; int k = j + 1;

while (k < n) {

if (item[j] == item[k] && k != j + 1) { c += 1;

if (k < n-1 && item[k] == item[k + 1]) {

k += 1;

}

}

k += 1;

}

if (max < c) { max = c;

itemtype = item[j];

}

j += 1;

}

cout<< itemtype<< endl;

}

return 0;

}

Java:

import java.util.\*; public class Main {

public static void main(String args[])

{

Scanner s = new Scanner(System.in); int t = s.nextInt();

for(int tc = 0; tc< t; tc++)

{

int n = s.nextInt();

int[] item = new int[n]; for(int i = 0; i < n; i++ )

{

item[i]= s.nextInt();

}

int j = 0,max = 0;

int itemType = item[0]; while(j< n)

{

int c = 1; int k = j+1; while(k< n)

{

if(item[j]== item[k] && k!=j+1)

{ c+=1;

if (k< n-1 && item[k]==item[k+1]) { k+=1;

}

} k+=1;

}

if (max< c)

{

max=c;

itemType = item[j];

}

j+=1;

}

System.out.println(itemType);

}

}

}

Python:

t = int(input()) for \_ in range(t):

n = int(input())

item = list(map(int, input().split()))

max\_count = 0 item\_type = item[0]

j = 0

while j < n: c = 1

k = j + 1

while k < n:

if item[j] == item[k] and k != j + 1: c += 1

if k < n - 1 and item[k] == item[k + 1]:

k += 1

k += 1

if max\_count < c:

max\_count = c item\_type = item[j]

j += 1

print(item\_type)

Question 2:

There are three piles of stones. The first pile contains a stones, the second pile contains b stones and the third pile contains c stones. You must choose one of the piles and split the stones from it to the other two piles; specifically, if the chosen pile initially contained s stones, you should choose an integer k (0≤k≤s), move k stones from the chosen pile onto one of the remaining two piles and s−k stones onto the other remaining pile. Determine if it is possible for the two remaining piles (in any order) to contain x stones and y stones respectively after performing this action.

INPUT FORMAT :

* The first line of the input contains a single integer T denoting the number of test cases. The description of T test cases follows.
* The first and only line of each test case contains five space-separated integers a,b,c, x and y.

OUTPUT FORMAT :

For each test case, print a single line containing the string “YES” if it is possible to obtain piles of the given

sizes or “NO” if it is impossible.

CONSTRAINTS :

* 1 <= T <= 100
* 1 <= a,b,c,x,y <= 10^9

SAMPLE INPUT :

4

1 2 3 2 4

3 2 5 6 5

2 4 2 6 2

6 5 2 12 1

SAMPLE OUTPUT :

YES NO YES NO

Test case 1: You can take the two stones on the second pile, put one of them on the first pile and the other one on the third pile.

Test case 2: You do not have enough stones.

Test case 3: You can choose the first pile and put all stones from it on the second pile. C++:

#include<bits/stdc++.h> using namespace std; using ll = long long;

int main ()

{

int t; cin >> t;

while (t--)

{

int a, b, c, x, y;

cin >> a >> b >> c >> x >> y; if ((a + b + c) != (x + y))

{

cout << "NO" << endl;

}

else

{

if (y < min (a, min (b, c)) || x < min (a, min (b, c)))

{

cout << "NO" << endl;

}

else

{

cout << "YES" << endl;

}

}

}

}

Java:

import java.util.Scanner;

public class Main {

public static void main(String[] args) { Scanner sc = new Scanner(System.in);

int t = sc.nextInt();

while (t-- > 0) {

int a = sc.nextInt(); int b = sc.nextInt(); int c = sc.nextInt(); int x = sc.nextInt(); int y = sc.nextInt();

if ((a + b + c) != (x + y)) { System.out.println("NO");

} else {

if (y < Math.min(a, Math.min(b, c)) || x < Math.min(a, Math.min(b, c))) { System.out.println("NO");

} else {

System.out.println("YES");

}

}

}

sc.close();

}

}

Python:

t = int(input()) for i in range(t):

a, b, c, x, y = map(int, input().split())

if (a + b + c) != (x + y):

print("NO") else:

if y < min(a, b, c) or x < min(a, b, c):

print("NO") else:

print("YES")

Question 3:

Altaf has recently learned about number bases and is becoming fascinated.

Altaf learned that for bases greater than ten, new digit symbols need to be introduced, and that the convention is to use the first few letters of the English alphabet. For example, in base 16, the digits are 0123456789ABCDEF. Altaf thought that this is unsustainable; the English alphabet only has 26 letters, so this scheme can only work up to base 36. But this is no problem for Altaf, because Altaf is very creative and can just invent new digit symbols when she needs them. (Altaf is very creative.)

Altaf also noticed that in base two, all positive integers start with the digit 1! However, this is the only base where this is true. So naturally, Altaf wonders: Given some integer N, how many bases b are there such that the base-b representation of N starts with a 1?

INPUT FORMAT :

The first line of the input contains an integer T denoting the number of test cases. The description of T test cases follows.

Each test case consists of one line containing a single integer N (in base ten). OUTPUT FORMAT :

For each test case, output a single line containing the number of bases b, or INFINITY if there are an infinite number of them.

CONSTRAINTS :

* 1 <= T <= 10^5
* 0 <= N < 10^12

SAMPLE INPUT :

4

6

9

11

24

SAMPLE OUTPUT :

4

7

8

14

C++:

#include <iostream> #include <vector> #include <algorithm> using namespace std;

vector<long long> v[47];

int main() {

long long int i, j;

for(i = 2; i <= (int)1e6; i++) { long long int tem = i; for(j = 2; j <= 40; j++) {

tem = tem \* i;

if(tem > (long long int)1e12) { break;

}

v[j].push\_back(tem);

}

}

int t; cin >> t;

while(t--) {

long long int m; cin >> m;

if(m == 1) {

cout << "INFINITY" << endl; continue;

}

long long int ans = (m + 1) / 2; long long int p = m / 2 + 1; for(i = 2; i <= 40; i++) {

ans = ans + (lower\_bound(v[i].begin(), v[i].end(), m + 1) - lower\_bound(v[i].begin(), v[i].end(), p));

}

cout << ans << endl;

}

return 0;

}

Java:

import java.util.\*;

public class Main {

static List[] v = new ArrayList[47];

public static void main(String[] args) { for (int i = 0; i < 47; i++) {

v[i] = new ArrayList<>();

}

for (long i = 2; i <= (int) 1e6; i++) { long tem = i;

for (int j = 2; j <= 40; j++) { tem = tem \* i;

if (tem > 1e12) { break;

}

v[j].add(tem);

}

}

Scanner sc = new Scanner(System.in); int t = sc.nextInt();

while (t-- > 0) {

long m = sc.nextLong();

if (m == 1) { System.out.println("INFINITY"); continue;

}

long ans = (m + 1) / 2; long p = m / 2 + 1;

for (int i = 2; i <= 40; i++) {

int low = Collections.binarySearch(v[i], p);

int high = Collections.binarySearch(v[i], m + 1);

if (low < 0) low = -(low + 1);

if (high < 0) high = -(high + 1);

ans += (high - low);

}

// Output the result System.out.println(ans);

}

sc.close();

}

}

Python:

v = [[] for p in range(47)]

for i in range(2, int(1e6) + 1): tem = i

for j in range(2, 41):

tem = tem \* i if tem > 1e12:

break v[j].append(tem)

t = int(input()) for \_ in range(t):

m = int(input())

if m == 1:

print("INFINITY")

continue

ans = (m + 1) // 2 p = m // 2 + 1

for i in range(2, 41):

ans += (len([x for x in v[i] if x >= p and x <= m]) > 0)

print(ans)

Greedy algorithms

1. Maximum Weight Difference

Chef has gone shopping with his 5-year old son. They have bought N items so far. The items are numbered from 1 to N, and the item i weighs Wi grams.

Chef's son insists on helping his father in carrying the items. He wants his dad to give him a few items. Chef does not want to burden his son. But he won't stop bothering him unless he is given a few items to carry. So Chef decides to give him some items. Obviously, Chef wants to give the kid less weight to carry.

However, his son is a smart kid. To avoid being given the bare minimum weight to carry, he suggests that the items are split into two groups, and one group contains exactly K items. Then Chef will carry the heavier group, and his son will carry the other group.

Help the Chef in deciding which items should the son take. Your task will be simple. Tell the Chef the maximum possible difference between the weight carried by him and the weight carried by the kid.

Input:

The first line of input contains an integer T, denoting the number of test cases. Then T test cases follow. The first line of each test contains two space-separated integers N and K. The next line contains N space- separated integers W1, W2, ..., WN.

Output:

For each test case, output the maximum possible difference between the weights carried by both in grams.

Constraints:

* 1 ≤ T ≤ 100
* 1 ≤ K < N ≤ 100
* 1 ≤ Wi ≤ 100000 (105)

Sample 1: Input

2

5 2

8 4 5 2 10

8 3

1 1 1 1 1 1 1 1

Output

17

2

Explanation:

Case #1: The optimal way is that Chef gives his son K=2 items with weights 2 and 4. Chef carries the rest of the items himself. Thus the difference is: (8+5+10) - (4+2) = 23 - 6 = 17.

Case #2: Chef gives his son 3 items and he carries 5 items himself.

C:

#include <stdio.h> #include <stdlib.h>

// Comparator function for qsort to sort in ascending order int compare(const void \*a, const void \*b) {

return (\*(int \*)a - \*(int \*)b);

}

int main() { int t;

scanf("%d", &t);

while (t--) { int n, k;

scanf("%d %d", &n, &k); int arr[n];

long long totalSum = 0;

// Input the array and compute the total sum for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]); totalSum += arr[i];

}

// Sort the array

qsort(arr, n, sizeof(int), compare);

// Sum of the first K smallest numbers (S1) and last K largest numbers (S2) long long sumOfKSmallest = 0, sumOfKLargest = 0;

for (int i = 0; i < k; i++) { sumOfKSmallest += arr[i]; sumOfKLargest += arr[n - 1 - i];

}

// Calculate the two possible maximum differences

long long diff1 = llabs(sumOfKSmallest - (totalSum - sumOfKSmallest)); long long diff2 = llabs(sumOfKLargest - (totalSum - sumOfKLargest));

// Output the maximum difference printf("%lld\n", (diff1 > diff2) ? diff1 : diff2);

}

return 0;

}

Java:

import java.util.\*; import java.lang.\*; import java.io.\*;

class Codechef {

public static void main(String[] args) throws java.lang.Exception {

// Scanner for reading input

Scanner sc = new Scanner(System.in);

int t = sc.nextInt(); // Read number of test cases

while (t-- > 0) {

int n = sc.nextInt(); // Read size of array int k = sc.nextInt(); // Read value of K

int[] arr = new int[n]; // Array to store the elements

long totalSum = 0; // Variable to store the total sum of elements

// Input the array and compute the total sum for (int i = 0; i < n; i++) {

arr[i] = sc.nextInt(); totalSum += arr[i];

}

// Sort the array Arrays.sort(arr);

// Sum of the first K smallest numbers (S1) and last K largest numbers (S2) long sumOfKSmallest = 0, sumOfKLargest = 0;

for (int i = 0; i < k; i++) {

sumOfKSmallest += arr[i]; // Smallest K numbers sumOfKLargest += arr[n - 1 - i]; // Largest K numbers

}

// Calculate the two possible maximum differences

long diff1 = Math.abs(sumOfKSmallest - (totalSum - sumOfKSmallest)); long diff2 = Math.abs(sumOfKLargest - (totalSum - sumOfKLargest));

// Output the maximum difference System.out.println(Math.max(diff1, diff2));

}

sc.close(); // Close the scanner

}

}

Python:

def main():

t = int(input()) for \_ in range(t):

n, k = map(int, input().split())

arr = list(map(int, input().split())) total\_sum = sum(arr)

# Sort the array

arr.sort()

# Sum of the first K smallest numbers and last K largest numbers sum\_of\_k\_smallest = sum(arr[:k])

sum\_of\_k\_largest = sum(arr[-k:])

# Calculate the two possible maximum differences

diff1 = abs(sum\_of\_k\_smallest - (total\_sum - sum\_of\_k\_smallest)) diff2 = abs(sum\_of\_k\_largest - (total\_sum - sum\_of\_k\_largest))

# Output the maximum difference print(max(diff1, diff2))

if name == " main ":

main()

1. Chef and String

There are N students standing in a row and numbered 1 through N from left to right. You are given a string S with length N, where for each valid i, the i-th character of S is 'x' if the i-th student is a girl or 'y' if this student is a boy. Students standing next to each other in the row are friends.

The students are asked to form pairs for a dance competition. Each pair must consist of a boy and a girl. Two students can only form a pair if they are friends. Each student can only be part of at most one pair. What is the maximum number of pairs that can be formed?

Input

* + The first line of the input contains a single integer T denoting the number of test cases. The description of T test cases follows.
  + The first and only line of each test case contains a single string S. Output

For each test case, print a single line containing one integer ― the maximum number of pairs.

Constraints

* + 1≤ T ≤ 100
  + 1≤ N ≤ 105
  + ∣S∣ = N
  + S contains only characters 'x' and 'y'
  + the sum of N over all test cases does not exceed 3⋅105 Subtasks

Subtask #1 (100 points): original constraints

Sample 1:

Input 3

xy xyxxy yy

Output

1

2

0

Explanation:

Example case 1: There is only one possible pair: (first student, second student).

Example case 2: One of the ways to form two pairs is: (first student, second student) and (fourth student, fifth student).

Another way to form two pairs is: (second student, third student) and (fourth student, fifth student). C:

#include <stdio.h> #include <string.h>

#define MAX\_STUDENTS 100005 // Maximum number of students

// Function to solve the problem of forming maximum pairs for the dance competition void findMaximumPairs() {

char students[MAX\_STUDENTS]; // Array to store the students' representation scanf("%s", students); // Read the string representing the students

int totalStudents = strlen(students); // Get the total number of students

int dp[MAX\_STUDENTS]; // Dynamic programming array to store maximum pairs

dp[0] = 0; // Initialize the first student

for (int i = 1; i < totalStudents; i++) {

// If both students are the same (both 'x' or both 'y'), no new pair can be formed if (students[i] == students[i - 1]) {

dp[i] = dp[i - 1];

} else {

// If the current student and the previous one are different, we can form a pair

// The maximum pairs will be either the same as the previous student or the pairs formed by pairing these two students

dp[i] = (i >= 2) ? dp[i - 2] + 1 : 1;

if (dp[i - 1] > dp[i]) {

dp[i] = dp[i - 1];

}

}

}

// Output the maximum number of pairs that can be formed printf("%d\n", dp[totalStudents - 1]);

}

int main() {

int testCases; // Variable to store the number of test cases scanf("%d", &testCases); // Read the number of test cases while (testCases--) {

findMaximumPairs(); // Solve for each test case

}

return 0; // Indicate successful execution

}

Java:

import java.util.\*; import java.lang.\*; import java.io.\*;

/\* Name of the class has to be "Main" only if the class is public. \*/ class Codechef

{

public static void main (String[] args) throws java.lang.Exception

{

// your code goes here

Scanner sc = new Scanner(System.in); int t = sc.nextInt();

while(t-->0)

{

String s = sc.next();

int c = 0;

for(int i=0;i<s.length();i++)

{

if(i>0)

if((s.charAt(i)=='x'&&s.charAt(i-1)=='y')||(s.charAt(i)=='y'&&s.charAt(i-1)=='x')){ c++;

i++;

}

}

System.out.println(c);

}

}

}

Python:

def find\_maximum\_pairs():

students = input() # Read the string representing the students total\_students = len(students) # Get the total number of students

dp = [0] \* total\_students # Dynamic programming array to store maximum pairs

for i in range(1, total\_students):

# If both students are the same (both 'x' or both 'y'), no new pair can be formed if students[i] == students[i - 1]:

dp[i] = dp[i - 1] else:

# If the current student and the previous one are different, we can form a pair

# The maximum pairs will be either the same as the previous student or the pairs formed by pairing these two students

dp[i] = max(dp[i - 1], (dp[i - 2] + 1) if i >= 2 else 1)

# Output the maximum number of pairs that can be formed print(dp[total\_students - 1])

def main():

T = int(input()) # Read the number of test cases for \_ in range(T):

find\_maximum\_pairs() # Solve for each test case

if name == " main ":

main()

Dynamic programming

1. Count Subarrays

Given an array A1,A2,...,AN, count the number of subarrays of array A which are non-decreasing.

A subarray A[i,j], where 1≤ I ≤ j ≤ N is a sequence of integers Ai, +1Ai+1, ..., Aj.

A subarray ]A[i,j] is non-decreasing if Ai+1≤+2≤... Ai ≤ Ai+1≤ Ai+2≤...≤Aj. You have to count the total

number of such subarrays.

Input

* + The first line of input contains an integer T denoting the number of test cases. The description of T test cases follows.
  + The first line of each test case contains a single integer N denoting the size of array.
  + The second line contains N space-separated integers 1A1, 2 A 2, ..., AN denoting the elements of the array.

Output

For each test case, output in a single line the required answer. Constraints

* + 1 ≤ T ≤ 5
  + 1 ≤ N ≤ 105
  + 1 ≤ Ai ≤ 109

Subtasks

* + Subtask 1 (20 points) : 1 ≤ N ≤ 100
  + Subtask 2 (30 points) : 1 ≤ N ≤ 1000
  + Subtask 3 (50 points) : Original constraints Sample 1:

Input 2

4

1 4 2 3

1

5

Output

6

1

Explanation:

Example case 1.

All valid subarrays are [1,1],[1,2],[2,2],[3,3],[3,4],[4,4]

Note that singleton subarrays are identically non-decreasing. Example case 2.

Only single subarray [1,1]A[1,1] is non-decreasing.

Python:

def count\_increasing\_subsequences(test\_cases): for \_ in range(test\_cases):

n = int(input().strip())

ans = 1

arr = list(map(int, input().strip().split())) dp = [1] \* n

for i in range(1, n):

if arr[i - 1] <= arr[i]:

dp[i] += dp[i - 1]

ans += dp[i]

print(ans)

if name == " main ": test\_case = int(input().strip())

count\_increasing\_subsequences(test\_case)

C:

#include <stdio.h> #include <stdlib.h>

int main() {

int testCase; scanf("%d", &testCase);

while (testCase--) { long long n; scanf("%lld", &n);

long long ans = 1;

// Allocate dynamic arrays

long long \*arr = (long long \*)malloc(n \* sizeof(long long)); long long \*dp = (long long \*)malloc(n \* sizeof(long long));

// Initialize dp array to 1

for (long long i = 0; i < n; i++) { dp[i] = 1;

}

// Read array elements

for (long long i = 0; i < n; i++) { scanf("%lld", &arr[i]);

}

// Calculate dp values and final answer for (long long i = 1; i < n; i++) {

if (arr[i - 1] <= arr[i]) {

dp[i] += dp[i - 1];

}

ans += dp[i];

}

printf("%lld\n", ans);

// Free allocated memory free(arr);

free(dp);

}

return 0;

}

Java:

import java.util.\*; import java.lang.\*; import java.io.\*;

class Codechef {

public static void main(String[] args) throws java.lang.Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); int testCase = Integer.parseInt(br.readLine());

while (testCase-- > 0) {

long n = Long.parseLong(br.readLine()); long ans = 1;

long[] arr = new long[(int) n];

long[] dp = new long[(int) n];

Arrays.fill(dp, 1); // Initialize dp array with 1

StringTokenizer st = new StringTokenizer(br.readLine()); for (int i = 0; i < n; i++) {

arr[i] = Long.parseLong(st.nextToken());

}

for (int i = 1; i < n; i++) { if (arr[i - 1] <= arr[i]) {

dp[i] += dp[i - 1];

}

ans += dp[i];

}

System.out.println(ans);

}

}

}

1. Sums in a Triangle

Given an integer N, let us consider a triangle of numbers of N lines in which a number a11 appears in the first line, two numbers a21 and a22 appear in the second line, three numbers a31, a32 and a33 appear in the third line, etc. In general, i numbers 1,2…ai1,ai2…ai i appear in the ith line for all 1≤ I ≤ N. Develop a program that will compute the largest of the sums of numbers that appear on the paths starting from the top towards the base, so that:

* + on each path the next number is located on the row below, more precisely either directly below or below and one place to the right.

Warning: large Input/Output data, be careful with certain languages Input Format

* + The first line of the input contains an integer T, the number of test cases.
  + Then T test cases follow. Each test case starts with an integer N, the number of rows. Then N lines

follow where in ith line contains i integers 1,2 ai1,ai2…aii.

Output Format

For each test case print the maximum path sum in a separate line. Constraints

* + 1 ≤ T ≤1000 1 ≤ N < 100
  + 0 ≤ aij < 100

Sample 1: Input

2

3

1

2 1

1 2 3

4

1

1 2

4 1 2

2 3 1 1

Output

5

9

Explanation:

Test case 1:

There are a total of 44 paths

* + (1,1)→(2,1)→(3,1) with sum equal to 4.
  + (1,1)→(2,1)→(3,2) with sum equal to 5.
  + (1,1)→(2,2)→(3,2) with sum equal to 4.
  + (1,1)→(2,2)→(3,3) with sum equal to 5. Therefore, the maximum sum over all paths is equal to 5. Test case 2:

There are a total of 88 paths

* + (1,1)→(2,1)→(3,1)→(4,1) with sum equal to 8.
  + (1,1)→(2,1)→(3,1)→(4,2) with sum equal to 9.
  + (1,1)→(2,1)→(3,2)→(4,2) with sum equal to 6.
  + (1,1)→(2,1)→(3,2)→(4,3) with sum equal to 4.
  + (1,1)→(2,2)→(3,2)→(4,2) with sum equal to 7.
  + (1,1)→(2,2)→(3,2)→(4,3) with sum equal to 5.
  + (1,1)→(2,2)→(3,3)→(4,3) with sum equal to 6.
  + (1,1)→(2,2)→(3,3)→(4,4 with sum equal to 6.
  + ​

Therefore, the maximum sum over all paths is equal to 9.

C:

#include <stdio.h>

void solve() {

long long int n, ans = 0; scanf("%lld", &n);

long long int ar[n + 1][n + 1]; for (int i = 1; i <= n; i++) {

for (int j = 1; j <= i; j++) { scanf("%lld", &ar[i][j]);

}

}

long long int dp[n + 1][n + 1]; for (int j = 1; j <= n; j++) {

dp[n][j] = ar[n][j];

}

for (int i = n - 1; i >= 1; i--) { for (int j = 1; j <= i; j++) {

dp[i][j] = ar[i][j] + (dp[i + 1][j] > dp[i + 1][j + 1] ? dp[i + 1][j] : dp[i + 1][j + 1]);

}

}

printf("%lld\n", dp[1][1]);

}

int main() {

long long int t = 1; scanf("%lld", &t);

for (int it = 1; it <= t; it++) { solve();

}

return 0;

}

Java:

import java.util.\*;

class Codechef {

public static void main(String[] args) throws Exception { Scanner scanner = new Scanner(System.in);

long t = scanner.nextLong(); for (int it = 0; it < t; it++) {

solve(scanner);

}

scanner.close();

}

public static void solve(Scanner scanner) { long n = scanner.nextLong();

long[][] ar = new long[(int) n + 1][(int) n + 1];

// Reading the input for the triangle for (int i = 1; i <= n; i++) {

for (int j = 1; j <= i; j++) { ar[i][j] = scanner.nextLong();

}

}

long[][] dp = new long[(int) n + 1][(int) n + 1];

// Initializing the last row of dp with values from ar for (int j = 1; j <= n; j++) {

dp[(int) n][j] = ar[(int) n][j];

}

// Filling the dp array from bottom to top for (int i = (int) n - 1; i >= 1; i--) {

for (int j = 1; j <= i; j++) {

dp[i][j] = ar[i][j] + Math.max(dp[i + 1][j], dp[i + 1][j + 1]);

}

}

// Output the result System.out.println(dp[1][1]);

}

}

Python:

def solve():

n = int(input()) ar = []

# Read the triangle input for i in range(n):

row = list(map(int, input().split())) ar.append(row)

# Initialize the dp array

dp = [[0] \* (n + 1) for \_ in range(n + 1)]

# Fill the last row of dp with values from ar for j in range(n):

dp[n - 1][j + 1] = ar[n - 1][j]

# Fill the dp array from bottom to top for i in range(n - 2, -1, -1):

for j in range(i + 1):

dp[i][j + 1] = ar[i][j] + max(dp[i + 1][j + 1], dp[i + 1][j + 2])

# Print the maximum path sum from top to bottom print(dp[0][1])

# Main function to handle multiple test cases if name == " main ":

t = int(input()) for \_ in range(t):

solve()

Problem -1: Minimum cost to cut the stick

Problem Description: We are given a stick of length N and a CUTS array of size C. The stick has markings as shown, and the CUTS array depicts the marking at which the stick needs to be cut (shown in red).

Input Format:

1. The first input is an integer N representing the length of the stick.
2. The second input is an integer C representing the number of cuts to be made on the stick.
3. The third input is an array of C integers where each integer denotes the position at which the stick needs to be cut.

Output Format:

The output should be a single integer representing the minimum cost required to cut the stick according to the given positions.

Example-1: Input:

10

3

2 4 7

Output:

20

Example-2: Input:

5

2

1 4

Output:

8

Explanation:

For each test case:

* The first input N is the total length of the stick.
* The second input C is the number of cuts.
* The third input represents the CUTS array, which contains the positions of the cuts on the stick.

The output is the minimum cost to cut the stick where the cost is the length of the segment being cut. TestCases:

Test Case 1:

Stick length = 10

CUTS = [2, 4, 7]

Expected result: 20

Test Case 2:

Stick length = 5

CUTS = [1, 4]

Expected result: 8

Test Case 3:

Stick length = 8

CUTS = [2, 5]

Expected result: 12

Test Case 4:

Stick length = 12

CUTS = [3, 8]

Expected result: 18

Test Case 5:

Stick length = 6

CUTS = [1, 3, 5]

Expected result: 10

Test Case 6:

Stick length = 15

CUTS = [4, 7, 10, 12]

Expected result: 48

Test Case 7:

Stick length = 20

CUTS = [5, 10, 15]

Expected result: 60

Test Case 8:

Stick length = 50

CUTS = [10, 20, 30, 40]

Expected result: 170

Test Case 9:

Stick length = 30

CUTS = [8, 12, 20, 28]

Expected result: 112

Test Case 10:

Stick length = 25

CUTS = [5, 10, 15, 20]

Expected result: 80 Java:

import java.util.\*; public class TUF {

// Recursive function to calculate the minimum cost static int f(int i, int j, ArrayList<Integer> cuts, int[][] dp) {

// Base case if (i > j) {

return 0;

}

if (dp[i][j] != -1) { return dp[i][j];

}

int mini = Integer.MAX\_VALUE; for (int ind = i; ind <= j; ind++) {

int ans = cuts.get(j + 1) - cuts.get(i - 1) + f(i, ind - 1, cuts, dp) +

f(ind + 1, j, cuts, dp);

mini = Math.min(mini, ans);

}

return dp[i][j] = mini;

}

// Function to calculate the minimum cost

static int cost(int n, int c, ArrayList<Integer> cuts) {

// Modify the cuts array cuts.add(n);

cuts.add(0); Collections.sort(cuts);

int[][] dp = new int[c + 1][c + 1];

for (int[] row : dp) { Arrays.fill(row, -1);

}

return f(1, c, cuts, dp);

}

public static void main(String[] args) {

ArrayList<Integer> cuts = new ArrayList<>(Arrays.asList(3, 5, 1, 4)); int c = cuts.size();

int n = 7;

System.out.println("The minimum cost incurred: " + cost(n, c, cuts));

}

}

C++

#include <bits/stdc++.h> using namespace std;

// Function to calculate the minimum cost incurred

int findMinimumCost(int i, int j, vector<int> &cuts, vector<vector<int>> &dp) {

// Base case: If i is greater than j, there are no cuts to consider. if (i > j) {

return 0;

}

if (dp[i][j] != -1) { return dp[i][j];

}

int mini = INT\_MAX;

for (int ind = i; ind <= j; ind++) {

// Calculate the cost for making a cut at position 'ind'. int ans = cuts[j + 1] - cuts[i - 1] +

findMinimumCost(i, ind - 1, cuts, dp) + findMinimumCost(ind + 1, j, cuts, dp);

mini = min(mini, ans);

}

return dp[i][j] = mini;

}

// Function to compute the minimum cost

int minimumCost(int n, int c, vector<int> &cuts) {

// Modify the cuts array by adding 0 at the beginning and 'n' at the end. cuts.push\_back(n);

cuts.insert(cuts.begin(), 0); sort(cuts.begin(), cuts.end());

// Create a DP table to store calculated values. vector<vector<int>> dp(c + 1, vector<int>(c + 1, -1));

// Call the recursive function to find the minimum cost. return findMinimumCost(1, c, cuts, dp);

}

int main() {

vector<int> cuts = {3, 5, 1, 4}; int c = cuts.size();

int n = 7;

cout << "The minimum cost incurred is: " << minimumCost(n, c, cuts) << endl;

return 0;

}

Python

def min\_cost(n, c, cuts):

# Define a 2D memoization table to store intermediate results

dp = [[-1] \* (c + 1) for \_ in range(c + 1)]

# Extend the cuts list with 0 and n, and sort it cuts = [0] + cuts + [n]

cuts.sort()

# Recursive function to find the minimum cost def f(i, j):

# Base case if i > j:

return 0

if dp[i][j] != -1:

return dp[i][j]

mini = float('inf')

for ind in range(i, j + 1):

ans = cuts[j + 1] - cuts[i - 1] + f(i, ind - 1) + f(ind + 1, j) mini = min(mini, ans)

dp[i][j] = mini return mini

return f(1, c)

if name == " main ": cuts = [3, 5, 1, 4]

c = len(cuts) n = 7

print("The minimum cost incurred:", min\_cost(n, c, cuts))

Problem-2 Coin Change Problem Problem Description:

Given a total amount amount and a list of coins (available denominations), find the number of distinct ways to make up that amount using any number of coins from the list. You may use each coin denomination an unlimited number of times.

Concept:

This is a variation of the Unbounded Knapsack problem and can be solved using Dynamic Programming. Input Format:

First line: Integer amount – the total sum to make. Second line: Integer n – the number of coin types.

Third line: n space-separated integers – the denominations of the coins.

Output Format:

One integer – the total number of ways to make the given amount.

Example:

Input:

4

3

1 2 3

Output:

4

Explanation: The four ways are: 1+1+1+1

1+1+2

2+2

1+3

Python

def count\_ways(amount, coins): dp = [0] \* (amount + 1)

dp[0] = 1 # Base case

for coin in coins:

for x in range(coin, amount + 1): dp[x] += dp[x - coin]

return dp[amount]

# Input

amount = int(input()) n = int(input())

coins = list(map(int, input().split())) print(count\_ways(amount, coins))

Java

import java.util.\*;

public class CoinChange {

public static int countWays(int amount, int[] coins) { int[] dp = new int[amount + 1];

dp[0] = 1;

for (int coin : coins) {

for (int i = coin; i <= amount; i++) { dp[i] += dp[i - coin];

}

}

return dp[amount];

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int amount = sc.nextInt();

int n = sc.nextInt(); int[] coins = new int[n]; for (int i = 0; i < n; i++) {

coins[i] = sc.nextInt();

}

System.out.println(countWays(amount, coins));

}

}

C

#include <stdio.h>

int countWays(int coins[], int n, int amount) { int dp[amount + 1];

for (int i = 0; i <= amount; i++) dp[i] = 0; dp[0] = 1;

for (int i = 0; i < n; i++) {

for (int j = coins[i]; j <= amount; j++) { dp[j] += dp[j - coins[i]];

}

}

return dp[amount];

}

int main() {

int amount, n; scanf("%d", &amount); scanf("%d", &n);

int coins[n];

for (int i = 0; i < n; i++) { scanf("%d", &coins[i]);

}

printf("%d\n", countWays(coins, n, amount)); return 0;

}

Test Cases

Test Amount Coins Expected Output Explanation

1 4 [1, 2, 3] 4 1+1+1+1, 1+1+2, 2+2, 1+3

1. 10 [2, 5, 3, 6] 5 Standard DP coin change case

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 5 | [1, 2] | 3 | 1+1+1+1+1, 1+1+1+2, 1+2+2 |
| 4 | 3 | [2] | 0 | Cannot make 3 with 2 |
| 5 | 0 | [1, 2, 3] | 1 | Only 1 way (no coins used) |
| 6 | 6 | [1, 5] | 2 | 1+1+1+1+1+1 and 1+5 |
| 7 | 8 | [2, 3, 5] | 6 | Multiple combinations |
| 8 | 7 | [1, 2, 3] | 8 | Many ways with small coins |
| 9 | 4 | [2, 3, 5] | 1 | Only 2+2 |
| 10 | 1 | [1] | 1 | Only 1 way using coin 1 |

Problem-3: Permutations of a String Problem Description:

Given a string S, generate all unique permutations of the string in lexicographical order. Input Format:

A single line containing the string S (consisting of lowercase/uppercase alphabets, no spaces).

Output Format:

Print all unique permutations of the string, one per line, in lexicographical order. Example:

Input: abc Output: abc

acb bac bca cab

cba

Python

from itertools import permutations

def string\_permutations(s):

perm\_set = sorted(set(permutations(s))) for p in perm\_set:

print("".join(p))

# Input

s = input().strip() string\_permutations(s)

Java

import java.util.\*;

public class StringPermutations {

static void permute(String str, String ans, Set<String> result) { if (str.length() == 0) {

result.add(ans);

return;

}

for (int i = 0; i < str.length(); i++) {

permute(str.substring(0, i) + str.substring(i + 1), ans + str.charAt(i), result);

}

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); String str = sc.next();

Set<String> result = new TreeSet<>(); permute(str, "", result);

for (String s : result) { System.out.println(s);

}

}

}

C

#include <stdio.h> #include <string.h> #include <stdlib.h>

void swap(char \*x, char \*y) { char tmp = \*x;

\*x = \*y;

\*y = tmp;

}

int compare(const void \*a, const void \*b) { return strcmp(\*(char \*\*)a, \*(char \*\*)b);

}

void permute(char \*str, int l, int r, char \*\*result, int \*index) { if (l == r) {

result[(\*index)] = strdup(str); (\*index)++;

} else {

for (int i = l; i <= r; i++) { swap((str + l), (str + i));

permute(str, l + 1, r, result, index); swap((str + l), (str + i)); // backtrack

}

}

}

int main() { char str[100];

scanf("%s", str); int n = strlen(str); int size = 1;

for (int i = 1; i <= n; i++) size \*= i;

char \*\*result = (char \*\*)malloc(size \* sizeof(char \*)); int index = 0;

permute(str, 0, n - 1, result, &index);

qsort(result, index, sizeof(char \*), compare);

for (int i = 0; i < index; i++) {

if (i == 0 || strcmp(result[i], result[i - 1]) != 0) printf("%s\n", result[i]);

free(result[i]);

}

free(result); return 0;

}

Test Cases

Test Case Input Expected Output (Sorted)

|  |  |  |
| --- | --- | --- |
| 1 | abc | abc, acb, bac, bca, cab, cba |
| 2 | aab | aab, aba, baa |
| 3 | a | a |
| 4 | ab | ab, ba |
| 5 | xyz | xyz, xzy, yxz, yzx, zxy, zyx |
| 6 | aaa | aaa |
| 7 | abcd | 24 permutations in sorted order |
| 8 | bca | abc, acb, bac, bca, cab, cba |
| 9 | aaab | aaab, aaba, abaa, baaa |
| 10 | abac | aabc, aacb, abac, abca, acab, acba, baac, baca, bcaa, caab, caba, cbaa |

Problem-4: Maximum Product Subarray Problem Description:

Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest product, and return that product.

Key Insight:

This problem is similar to the maximum subarray sum problem (Kadane’s Algorithm), but with a twist:

Multiplication with a negative number flips signs, so we track: max\_so\_far → maximum product up to current index min\_so\_far → minimum product up to current index

Swap them when encountering a negative number

Input Format:

First line: An integer n (number of elements)

Second line: n space-separated integers representing the array nums Output Format:

A single integer – the maximum product of a contiguous subarray.

Example:

Input:

4

2 3 -2 4

Output:

6

Explanation: The subarray [2, 3] has the largest product = 6 Python

def max\_product(nums): if not nums:

return 0

max\_so\_far = min\_so\_far = result = nums[0]

for num in nums[1:]:

if num < 0:

max\_so\_far, min\_so\_far = min\_so\_far, max\_so\_far

max\_so\_far = max(num, num \* max\_so\_far) min\_so\_far = min(num, num \* min\_so\_far)

result = max(result, max\_so\_far)

return result

# Input

n = int(input())

nums = list(map(int, input().split())) print(max\_product(nums))

Java

import java.util.\*;

public class MaxProductSubarray {

public static int maxProduct(int[] nums) { int max\_so\_far = nums[0];

int min\_so\_far = nums[0]; int result = nums[0];

for (int i = 1; i < nums.length; i++) { int temp = max\_so\_far;

if (nums[i] < 0) {

max\_so\_far = min\_so\_far; min\_so\_far = temp;

}

max\_so\_far = Math.max(nums[i], nums[i] \* max\_so\_far); min\_so\_far = Math.min(nums[i], nums[i] \* min\_so\_far);

result = Math.max(result, max\_so\_far);

}

return result;

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int n = sc.nextInt();

int[] nums = new int[n]; for (int i = 0; i < n; i++)

nums[i] = sc.nextInt(); System.out.println(maxProduct(nums));

}

}

C

#include <stdio.h>

int max(int a, int b) { return a > b ? a : b; } int min(int a, int b) { return a < b ? a : b; }

int maxProduct(int\* nums, int n) { int max\_so\_far = nums[0];

int min\_so\_far = nums[0]; int result = nums[0];

for (int i = 1; i < n; i++) { int temp = max\_so\_far;

if (nums[i] < 0) {

max\_so\_far = min\_so\_far; min\_so\_far = temp;

}

max\_so\_far = max(nums[i], nums[i] \* max\_so\_far); min\_so\_far = min(nums[i], nums[i] \* min\_so\_far);

result = max(result, max\_so\_far);

}

return result;

}

int main() { int n;

scanf("%d", &n); int nums[n];

for (int i = 0; i < n; i++) { scanf("%d", &nums[i]);

}

printf("%d\n", maxProduct(nums, n)); return 0;

}

Test Cases

Test Case Input Output Explanation

1 4

2 3 -2 4 6 [2,3] has product 6

2 5

-2 0 -1 3 4 12 [3,4] = 12

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 3 |  | |
| -2 3 -4 | 24 | [-2, 3, -4] = 24 | |
| 4 | 6 |  | |
| -1 -3 -10 0 60 -260 | | | [60] |

|  |  |  |
| --- | --- | --- |
| 5 | 1 |  |
| 0 | 0 | Zero product |
| 6 | 2 |  |
| -2 -3 | 6 | [-2, -3] = 6 |
| 7 | 3 |  |

|  |  |  |
| --- | --- | --- |
| 0 2 0 | 2 | [2] |
| 8 | 4 |  |
| 1 2 3 4 | 24 | Entire array |
| 9 | 5 |  |
| -1 -2 -3 | -4 -5 | 120 Whole array = 120 |
| 10 | 6 |  |

2 -5 -2 -4 3 1 240 Subarray [-5, -2, -4, 3] = 240

Problem-5: Minimum Number of Platforms Required Problem Description:

Given two arrays:

arrival[] — Arrival times of trains departure[] — Departure times of trains

Find the minimum number of railway platforms required so that no train has to wait. Core Logic:

We want to track the number of trains at the station at any moment. We: Sort the arrival and departure arrays.

Use two pointers to simulate the timeline. For every train:

If a train arrives before the earliest departure → increase platform count. If a train departs → free up a platform.

Input Format:

First line: Integer n, the number of trains.

Second line: n space-separated arrival times (in 24-hour format, e.g., 900 for 9:00 AM). Third line: n space-separated departure times.

Output Format:

One integer: minimum number of platforms needed.

Example:

Input:

6

900 940 950 1100 1500 1800

910 1200 1120 1130 1900 2000

Output:

3

Explanation: At time 950, there are 3 trains at the station.

Python

def find\_min\_platforms(arrival, departure): arrival.sort()

departure.sort()

n = len(arrival) platforms = 1

max\_platforms = 1

i = 1 # arrival pointer

j = 0 # departure pointer

while i < n and j < n:

if arrival[i] <= departure[j]:

platforms += 1

i += 1

else:

platforms -= 1

j += 1

max\_platforms = max(max\_platforms, platforms)

return max\_platforms

# Input

n = int(input())

arrival = list(map(int, input().split())) departure = list(map(int, input().split())) print(find\_min\_platforms(arrival, departure))

Java

import java.util.\*;

public class MinPlatforms {

public static int findPlatforms(int[] arr, int[] dep) { Arrays.sort(arr);

Arrays.sort(dep);

int n = arr.length;

int platforms = 1, maxPlatforms = 1; int i = 1, j = 0;

while (i < n && j < n) { if (arr[i] <= dep[j]) {

platforms++; i++;

} else {

platforms--; j++;

}

maxPlatforms = Math.max(maxPlatforms, platforms);

}

return maxPlatforms;

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int n = sc.nextInt();

int[] arrival = new int[n]; int[] departure = new int[n];

for (int i = 0; i < n; i++) arrival[i] = sc.nextInt();

for (int i = 0; i < n; i++) departure[i] = sc.nextInt(); System.out.println(findPlatforms(arrival, departure));

}

}

C

#include <stdio.h> #include <stdlib.h>

int compare(const void \*a, const void \*b) { return (\*(int \*)a - \*(int \*)b);

}

int findMinPlatforms(int arr[], int dep[], int n) { qsort(arr, n, sizeof(int), compare); qsort(dep, n, sizeof(int), compare);

int platforms = 1, maxPlatforms = 1; int i = 1, j = 0;

while (i < n && j < n) { if (arr[i] <= dep[j]) {

platforms++; i++;

} else {

platforms--; j++;

}

if (platforms > maxPlatforms) maxPlatforms = platforms;

}

return maxPlatforms;

}

int main() { int n;

scanf("%d", &n);

int arr[n], dep[n];

for (int i = 0; i < n; i++) scanf("%d", &arr[i]);

for (int i = 0; i < n; i++) scanf("%d", &dep[i]); printf("%d\n", findMinPlatforms(arr, dep, n)); return 0;

}

Test Cases

Test Arrival Times Departure Times Output Explanation

1 900 940 950 1100 1500 1800 910 1200 1120 1130 1900 2000 3 Max 3 trains at station

around 950

2 900 1100 1235 1000 1200 12401 No overlap

1. 1000 1015 1030 1045 1020 1035 1050 1100 2 Overlaps between each arrival
2. 900 905 915 925 930 940 950 955 4 All overlap
3. 900 910 1 Only one train

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6 | 900 940 950 | 1000 1030 10502 | | Overlaps with gaps | |
| 7 | 800 900 1000 | 830 930 1030 2 | | Trains overlap sequentially | |
| 8 | 1000 1005 10101015 1020 10253 | | | Fully overlapping | |
| 9 | 1200 1210 12251230 1240 12503 | | | All trains overlap at different points | |
| 10 | 900 1100 1230 1500 | | 1000 1130 1300 1600 1 | | No overlapping trains |

Sample 1:

A diligent student has S hours to study for upcoming exams. There are C subjects to study. Studying subject i for one hour increases their understanding by Ui units. Due to time constraints and diminishing returns, the student can study each unique subject for a maximum of 2 hours. The student wants to achieve a total understanding of at least T units. Calculate the minimum total study hours required to reach a total understanding of T or more. If it's impossible to reach a total understanding of T even after studying each subject for the maximum allowed time, return -1.

Parameters:

* S: INTEGER (Total study hours available, 1≤S≤105)
* C: INTEGER (Number of subjects, 1≤C≤105)
* U: INTEGER ARRAY (Array of size C, where U[i] is the understanding gained per hour of studying

subject i, 1≤U[i]≤105)

* T: INTEGER (Target total understanding, 1≤T≤2×1010)

Test Cases: Case 1:

Input:

10

2

5

3

18

Output:

4

* S=10
* C=2
* U=[5,3]
* T=18

Study subject 0 for 2 hours (gain 2×5=10 understanding). Study subject 1 for 2 hours (gain 2×3=6 understanding). Total understanding = 10+6=16, which is less than 18. We need to study more. Since we've reached the maximum time for each subject, it's impossible to reach the target. Oh wait, the question asks for the minimum hours to reach at least T.

Let's rethink Case 1: Study subject 0 for 2 hours (10 understanding). Study subject 1 for 2 hours (6 understanding). Total 16. We haven't reached 18. The maximum understanding we can get is 2×5+2×3=16. Since this is less than 18, the answer should be -1.

Case 2: Input:

5

3

2

1

4

7

Output:

2

* S=5
* C=3
* U=[2,1,4]
* T=7

Study subject 2 for 2 hours (gain 2×4=8 understanding). Total hours = 2, total understanding = 8 (which is

≥7).

Case 3:

Input:

3

3

1

5

2

12

Output:

-1

* S=3
* C=3
* U=[1,5,2]
* T=12

Maximum understanding achievable is 2×1+2×5+2×2=2+10+4=16. Since 16<12 is false, the maximum is 16, which is greater than or equal to 12. Let's trace the minimum hours.

Study subject 1 for 2 hours (10 understanding, 2 hours). Study subject 2 for 1 hour (2 understanding, 1 hour). Total understanding = 12, total hours = 3.

Wait, the maximum understanding is 16, which is greater than 12. We need the minimum hours. Let's re-rethink Case 3: Sort U in descending order: [5,2,1]. Study subject with U=5 for 2 hours (10

understanding, 2 hours). Remaining target = 12−10=2. Study subject with U=2 for 1 hour (2

understanding, 1 hour). Remaining target = 2−2=0. Total hours = 2+1=3.

Case 4:

Input:

6

2

1

2

6

Output 4

* S=6
* C=2
* U=[1,2]
* T=6

Study subject with U=2 for 2 hours (4 understanding, 2 hours). Remaining target = 6−4=2. Study subject

with U=1 for 2 hours (2 understanding, 2 hours). Remaining target = 2−2=0. Total hours = 2+2=4.

.

Solution:

def solve():

s = int(input()) c = int(input())

u = [int(input()) for \_ in range(c)] t = int(input())

max\_understanding = sum(2 \* val for val in u)

if max\_understanding < t:

print(-1) return

subjects\_with\_indices = sorted([(u[i], i) for i in range(c)], key=lambda x: x[0], reverse=True)

total\_understanding = 0

hours\_studied = 0 study\_counts = [0] \* c

for understanding\_per\_hour, original\_index in subjects\_with\_indices:

while study\_counts[original\_index] < 2 and total\_understanding < t and hours\_studied < s: total\_understanding += understanding\_per\_hour

hours\_studied += 1

study\_counts[original\_index] += 1 if total\_understanding >= t:

print(hours\_studied) return

# This part should ideally not be reached if the initial max\_understanding check is correct # However, it's a safeguard.

if total\_understanding >= t: print(hours\_studied)

else:

print(-1)

solve()

Explanation of the Code:

1. Read Input: The code first reads the input values for S (total study hours), C (number of subjects), the array U (understanding per hour for each subject), and T (target total understanding).
2. Calculate Maximum Possible Understanding: It calculates the maximum possible understanding the student can gain by studying each of the C subjects for the maximum allowed 2 hours.
3. Check if Target is Reachable: If the max\_understanding is less than the target understanding T, it's impossible to reach the target, so the code prints -1 and returns.
4. Sort Subjects by Understanding Gain: The subjects are sorted in descending order based on the understanding gained per hour (U[i]). This greedy approach prioritizes studying the subjects that provide the most understanding per hour first, aiming to reach the target in the minimum number of hours. The original index of each subject is also stored to keep track of the study limit per subject.
5. Iterate Through Subjects: The code iterates through the sorted subjects. For each subject:
   * It checks if the student has studied this subject for less than 2 hours (study\_counts[original\_index]

< 2), if the total\_understanding is still less than the target T, and if the hours\_studied are within the total available study hours S.

* + If all these conditions are true, the student studies the subject for one hour, increasing the total\_understanding by understanding\_per\_hour, incrementing hours\_studied, and updating the study\_counts for that subject.
  + If the total\_understanding reaches or exceeds the target T, the code prints the hours\_studied and returns.

1. Final Check (Safeguard): After the loop, there's a final check to see if the total\_understanding has reached the target. While the initial max\_understanding check should prevent the target from being unreachable within the time limit if a solution exists, this acts as a safeguard. If the target is reached, the hours\_studied are printed; otherwise, -1 is printed (although this scenario should ideally be caught by the initial check).

2)

A student is collecting stickers from different series. There are N stickers in total, and their series are represented by an array Series. If two stickers have the same value in Series, they belong to the same series. The student wants to collect as many unique stickers as possible. They follow a specific collecting pattern:

* + They can start by collecting any number of unique stickers from one or more different series in their first step.
  + If they collect K unique stickers in one step, the next step they take must involve collecting 2∗K unique stickers from entirely different series (none of the series can be the same as the ones collected from in previous steps).
  + They can never collect from a series they have collected from before. Find the maximum total number of unique stickers the student can collect. Parameters:
  + N: INTEGER (Total number of stickers, 1≤N≤105)
  + Series: INTEGER ARRAY (Array of size N, where Series[i] denotes the series of the i-th sticker,

1≤Series[i]≤109)

Test Cases:

Case 1: Input: 7

1

2

2

1

3

1

3

Output:

3

* + Unique series: {1, 2, 3}. Number of unique stickers per series: 1 for each.
  + Possible sequences:
  + Start with 1 (from series 1) -> next must be 2 (1 from series 2, 1 from series 3). Total = 1 + 2 = 3.
  + Start with 1 (from series 2) -> next must be 2 (1 from series 1, 1 from series 3). Total = 1 + 2 = 3.
  + Start with 1 (from series 3) -> next must be 2 (1 from series 1, 1 from series 2). Total = 1 + 2 = 3.
  + Start with 2 (1 from series 1, 1 from series 2) -> next must be 4 (impossible with only 1 unique

from series 3). Total = 2.

* + Start with 2 (1 from series 1, 1 from series 3) -> next must be 4 (impossible with only 1 unique

from series 2). Total = 2.

* + Start with 2 (1 from series 2, 1 from series 3) -> next must be 4 (impossible with only 1 unique

from series 1). Total = 2.

* + Start with 3 (1 from series 1, 1 from series 2, 1 from series 3) -> next must be 6 (impossible). Total

= 3.

Case 2: Input: 10

1

1

2

2

2

3

3

3

3

4

Output:

4

* + Unique series: {1, 2, 3, 4}. Number of unique stickers per series: 1 for each.
  + Possible sequences:
  + Start 1 (series 1) -> next 2 (series 2 & 3) -> next 4 (series 4 - impossible). Total = 3.
  + Start 1 (series 1) -> next 2 (series 2 & 4) -> next 4 (series 3 - impossible). Total = 3.
  + Start 1 (series 1) -> next 2 (series 3 & 4) -> next 4 (series 2 - impossible). Total = 3.
  + Start 1 (series 2) -> next 2 (series 1 & 3) -> next 4 (series 4 - impossible). Total = 3.
  + Start 1 (series 2) -> next 2 (series 1 & 4) -> next 4 (series 3 - impossible). Total = 3.
  + Start 1 (series 2) -> next 2 (series 3 & 4) -> next 4 (series 1 - impossible). Total = 3.
  + Start 1 (series 3) -> next 2 (series 1 & 2) -> next 4 (series 4 - impossible). Total = 3.
  + Start 1 (series 3) -> next 2 (series 1 & 4) -> next 4 (series 2 - impossible). Total = 3.
  + Start 1 (series 3) -> next 2 (series 2 & 4) -> next 4 (series 1 - impossible). Total = 3.
  + Start 1 (series 4) -> next 2 (series 1 & 2) -> next 4 (series 3 - impossible). Total = 3.
  + Start 1 (series 4) -> next 2 (series 1 & 3) -> next 4 (series 2 - impossible). Total = 3.
  + Start 1 (series 4) -> next 2 (series 2 & 3) -> next 4 (series 1 - impossible). Total = 3.
  + Start 2 (series 1 & 2) -> next 4 (series 3 & 4). Total = 2 + 2 = 4.
  + Start 2 (series 1 & 3) -> next 4 (series 2 & 4). Total = 2 + 2 = 4.
  + Start 2 (series 1 & 4) -> next 4 (series 2 & 3). Total = 2 + 2 = 4.
  + Start 2 (series 2 & 3) -> next 4 (series 1 & 4). Total = 2 + 2 = 4.
  + Start 2 (series 2 & 4) -> next 4 (series 1 & 3). Total = 2 + 2 = 4.
  + Start 2 (series 3 & 4) -> next 4 (series 1 & 2). Total = 2 + 2 = 4.

Case 3: Input: 4

1

1

1

1

Output:

1

* + Unique series: {1}. Number of unique stickers per series: 1.
  + Can only start with 1, cannot proceed as there are no other series. Case 4:

Input:

8

1

2

3

4

5

6

7

8

Output:

7

* + Unique series: {1, 2, 3, 4, 5, 6, 7, 8}. One unique sticker per series.
  + Start 1 -> next 2 -> next 4. Total = 1 + 2 + 4 = 7. Cannot go to 8.

This sticker collection problem captures the essence of the original restaurant dish problem with the constraints of sequential steps, doubling the count, and not repeating the "type" (series). The goal remains to maximize the total number of unique items collected.

Solution:

from collections import Counter

def solve():

n = int(input())

series = [int(input()) for \_ in range(n)] unique\_series\_counts = Counter(series) unique\_series = set(series) num\_unique\_series = len(unique\_series) max\_stickers = 0

def find\_max\_stickers(collected\_series, current\_stickers, next\_collection\_size): nonlocal max\_stickers

max\_stickers = max(max\_stickers, current\_stickers)

remaining\_series = unique\_series - collected\_series num\_remaining = len(remaining\_series)

if num\_remaining >= next\_collection\_size and next\_collection\_size > 0: from itertools import combinations

for next\_group in combinations(remaining\_series, next\_collection\_size): find\_max\_stickers(

collected\_series | set(next\_group), current\_stickers + next\_collection\_size, next\_collection\_size \* 2

)

# Try starting with collecting 's' unique stickers from 's' different series for start\_size in range(1, num\_unique\_series + 1):

from itertools import combinations

for initial\_group in combinations(unique\_series, start\_size): find\_max\_stickers(set(initial\_group), start\_size, start\_size \* 2)

print(max\_stickers)

solve()

Q3)

You are given an array A of size N containing integers. You are allowed to choose at most one pair of adjacent elements (distance of 1) and swap them. Find the lexicographically smallest array possible after performing at most one such swap.

Parameters:

* + N: INTEGER (Size of the array A, 1≤N≤105)
  + A: INTEGER ARRAY (Array of N integers, 1≤A[i]≤105)

Test Cases:

Case 1:

Input:

3

3

2

1

Output:

2 3 1

* + N=3
  + A=[3,2,1]
  + Swapping 3 and 2 gives [2,3,1].
  + Swapping 2 and 1 gives [3,1,2].
  + [2,3,1] is the lexicographically smallest. Case 2:

Input:

4

1

2

3

4

Output:

1 2 3 4

* + N=4
  + A=[1,2,3,4]
  + Any adjacent swap will result in a lexicographically larger array. Case 3:

Input:

4

2

1

3

1

Output:

1 2 3 1

* + N=4
  + A=[2,1,3,1]
  + Swapping 2 and 1 gives [1,2,3,1].
  + Swapping 1 and 3 gives [2,3,1,1].
  + Swapping 3 and 1 gives [2,1,1,3].
  + [1,2,3,1] is the lexicographically smallest.

Case 4:

Input:

5

5

1

4

2

8

Output:

1 5 4 2 8

* + N=5
  + A=[5,1,4,2,8]
  + Swapping 5 and 1 gives [1,5,4,2,8].
  + Swapping 1 and 4 gives [5,4,1,2,8].
  + Swapping 4 and 2 gives [5,1,2,4,8].
  + Swapping 2 and 8 gives [5,1,4,8,2].
  + [1,5,4,2,8] is the lexicographically smallest. Why it's Similar:
  + Single Modification: You are allowed to perform at most one operation (one swap in both problems).
  + Constraint on the Operation: The swap has a constraint (distance at most K in the original, distance exactly 1 in this similar question).
  + Lexicographical Ordering: The goal is to find the lexicographically smallest array after the allowed operation.
  + Checking Possibilities: The core logic to solve both involves trying all valid single swaps and comparing the resulting arrays lexicographically to find the smallest one.

Solution :

def solve():

n = int(input())

a = [int(input()) for \_ in range(n)]

best\_a = list(a)

for i in range(n - 1):

temp\_a = list(a)

temp\_a[i], temp\_a[i + 1] = temp\_a[i + 1], temp\_a[i] if temp\_a < best\_a:

best\_a = temp\_a

print(\*best\_a)

# Test Case 1 print("Test Case 1:")

input1\_n = 3

input1\_a = [3, 2, 1]

inputs1 = [str(input1\_n)] + [str(x) for x in input1\_a] original\_input = builtins .input

builtins .input = lambda: inputs1.pop(0) solve()

builtins .input = original\_input print("Expected Output: 2 3 1")

print("-" \* 20)

# Test Case 2

print("Test Case 2:")

input2\_n = 4

input2\_a = [1, 2, 3, 4]

inputs2 = [str(input2\_n)] + [str(x) for x in input2\_a] original\_input = builtins .input

builtins .input = lambda: inputs2.pop(0) solve()

builtins .input = original\_input print("Expected Output: 1 2 3 4")

print("-" \* 20)

# Test Case 3 print("Test Case 3:")

input3\_n = 4

input3\_a = [2, 1, 3, 1]

inputs3 = [str(input3\_n)] + [str(x) for x in input3\_a] original\_input = builtins .input

builtins .input = lambda: inputs3.pop(0) solve()

builtins .input = original\_input print("Expected Output: 1 2 3 1")

print("-" \* 20)

# Test Case 4 print("Test Case 4:")

input4\_n = 5

input4\_a = [5, 1, 4, 2, 8]

inputs4 = [str(input4\_n)] + [str(x) for x in input4\_a]

original\_input = builtins .input

builtins .input = lambda: inputs4.pop(0) solve()

builtins .input = original\_input print("Expected Output: 1 5 4 2 8")

print("-" \* 20)

You are given an array A of size N containing positive integers. You can rearrange the elements of this array in any order. You want to partition this rearranged array into some contiguous subarrays such that all the subarrays have the same sum. You want to make the number of subarrays as large as possible.

What is the maximum number of subarrays you can get? Parameters:

* N: INTEGER (Size of the array A, 1≤N≤2×105)
* A: INTEGER ARRAY (Array of N positive integers, 1≤A[i]≤109)

Test Cases: Case 1: Input:

5

1

2

1

2

1

Output:

3

* Sum of the array = 1 + 2 + 1 + 2 + 1 = 7.
* We can rearrange it as [1,1,1,2,2].
* We can partition it into 3 subarrays with sum 37 (not possible with integers).
* Let's rethink. The subarrays must have the same sum.
* Rearrange as [1,1,2,1,2]. Sum = 6. Possible partitions with equal sum:
* [1,1,2] (sum 4), [1,2] (sum 3) - No.
* Rearrange as [1,2,1,2,1]. Sum = 7. Not divisible by any number > 1. Output should be 1.
* Correction: Sum = 7. Only 1 subarray is possible. Output should be 1.

Case 2:

Input:

6

1

2

3

1

2

3

Output:

2

* Sum of the array = 1 + 2 + 3 + 1 + 2 + 3 = 12.
* We can rearrange it as [1,2,3,1,2,3].
* We can partition it into 2 subarrays with sum 12/2=6: [1,2,3], [1,2,3]. Case 3:

Input:

4

1

1

1

1

Output:

4

* Sum of the array = 1 + 1 + 1 + 1 = 4.
* We can rearrange it as [1,1,1,1].
* We can partition it into 4 subarrays with sum 4/4=1: [1], [1], [1], [1].

Case 4:

Input:

8

1

1

2

2

1

1

2

2

Output:

4

* Sum of the array = 1 + 1 + 2 + 2 + 1 + 1 + 2 + 2 = 12.
* We can rearrange it as [1,1,1,1,2,2,2,2].
* We can partition it into 4 subarrays with sum 12/4=3: [1,1,1], [1,2], [2,1], [2] - No.
* Rearrange as [1,2],[1,2],[1,2],[1,2]. Sum per subarray = 3.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as [1,1],[1,1],[2],[2],[2],[2] - No.
* Rearrange as [1,1,2],[1,1,2],[2],[2] - No.
* Rearrange as $[1, 1], [1,

Solution :

from collections import Counter import math

def solve():

n = int(input())

a = list(map(int, input().split())) total\_sum = sum(a) max\_subarrays = 1

for num\_subarrays in range(1, n + 1):

if total\_sum % num\_subarrays == 0:

subarray\_sum = total\_sum // num\_subarrays

subarray\_length = n // num\_subarrays if num\_subarrays > 0 else 0 if n % num\_subarrays == 0:

required\_counts = Counter() for x in a:

required\_counts[x] += 1

possible = True

for \_ in range(num\_subarrays):

current\_subarray\_counts = Counter() current\_subarray\_sum = 0 temp\_counts = required\_counts.copy() elements\_added = 0

for val in sorted(temp\_counts.keys()):

while temp\_counts[val] > 0 and current\_subarray\_sum + val <= subarray\_sum and elements\_added < subarray\_length:

current\_subarray\_counts[val] += 1 current\_subarray\_sum += val temp\_counts[val] -= 1

elements\_added += 1

if current\_subarray\_sum != subarray\_sum or elements\_added != subarray\_length: possible = False

break else:

required\_counts -= current\_subarray\_counts

if possible:

max\_subarrays = max(max\_subarrays, num\_subarrays)

print(max\_subarrays)

# Test Case 1 print("Case #1:")

input1\_n = 5

input1\_a = [1, 2, 1, 2, 1]

inputs1 = [str(input1\_n), " ".join(map(str, input1\_a))] original\_input = builtins .input

builtins .input = lambda: inputs1.pop(0) solve()

builtins .input = original\_input print("Expected Output: 1")

print("-" \* 20)

# Test Case 2 print("Case #2:")

input2\_n = 6

input2\_a = [1, 2, 3, 1, 2, 3]

inputs2 = [str(input2\_n), " ".join(map(str, input2\_a))] original\_input = builtins .input

builtins .input = lambda: inputs2.pop(0) solve()

builtins .input = original\_input print("Expected Output: 2")

print("-" \* 20)

# Test Case 3 print("Case #3:")

input3\_n = 4

input3\_a = [1, 1, 1, 1]

inputs3 = [str(input3\_n), " ".join(map(str, input3\_a))] original\_input = builtins .input

builtins .input = lambda: inputs3.pop(0) solve()

builtins .input = original\_input print("Expected Output: 4")

print("-" \* 20)

# Test Case 4 print("Case #4:")

input4\_n = 8

input4\_a = [1, 1, 2, 2, 1, 1, 2, 2]

inputs4 = [str(input4\_n), " ".join(map(str, input4\_a))] original\_input = builtins .input

builtins .input = lambda: inputs4.pop(0)

solve()

builtins .input = original\_input print("Expected Output: 4")

print("-" \* 20)

Problem: Convert Array into a Valley

You are given an array of size N. You need to transform this array into a valley. By valley, we mean:

* The ends of the array must be equal.
* As you move from both ends to the center, each next element should be exactly one less than the previous until a minimum is reached in the middle.
* Think of the array shape like a valley, decreasing toward the center and then mirroring back up.

The array should be symmetric and the difference between adjacent elements must be exactly 1 as you approach the center.

✅ Examples of valleys:

* [3, 2, 1, 2, 3] ✅
* [5, 4, 3, 4, 5] ✅
* [6, 5, 4, 4, 5, 6] ✅

❌ Not valleys:

* [3, 2, 0, 2, 3] ❌ (jumps by 2)
* [1, 0, -2, 0, 1] ❌ (jumps by more than 1)

Cç' Your Task:

Find the minimum number of elements that should be changed in order to make the array a valley. You can make any changes (including making elements negative or zero).

⬛ Input Format:

* The first line contains an integer N — the number of elements in the array.

1 ≤ N ≤ 10^5

* The second line contains N integers — the elements of the array. Each element is between -10^6 and 10^6

ˆJ \_ Output Format:

* A single integer — the minimum number of changes required to convert the array into a valley.

^" Test Cases:

Test Case 1:

makefile CopyEdit Input:

5

1 2 3 4 5

Output:

2

Explanation: Convert to [3,2,1,2,3]. Change first 1 to 3 and last 5 to 3.

Test Case 2: makefile CopyEdit Input:

7

2 2 3 4 3 2 2

Output:

3

Explanation: Convert to [4,3,2,1,2,3,4]. Change 3 elements.

Test Case 3:

makefile CopyEdit Input:

6

4 4 3 3 2 2

Output:

3

Explanation: Convert to [4,3,2,2,3,4]. Change 3 elements.

def solve():

n = int(input())

arr = list(map(int, input().split())) min\_changes = n

for mid in range((n + 1) // 2):

for peak in range(1, 2 \* 10\*\*5 + 2): # Iterate through a reasonable range of peak values temp\_arr = [0] \* n

changes = 0

# Construct the left side of the valley possible = True

for i in range(mid):

expected = peak - (mid - i) if expected <= 0:

possible = False break

temp\_arr[i] = expected

if i < n and arr[i] != expected: changes += 1

if not possible:

continue

# Construct the center of the valley if mid < n:

temp\_arr[mid] = peak

if mid < n and arr[mid] != peak: changes += 1

# Construct the right side of the valley for i in range(mid + 1, n):

expected = peak - (i - mid) if expected <= 0:

possible = False break

temp\_arr[i] = expected

if i < n and arr[i] != expected: changes += 1

if possible:

min\_changes = min(min\_changes, changes)

print(min\_changes)

if name == " main ": solve()

Q1)

The lead role in Joker is that of Mannar Mannan, a mentally unsound and poor activist who thinks he is the people-appointed President of India. Mannan falls in love with Malliga, courts her and wins her over and they get married. Soon , they are expecting a child and they are very excited about it. As most other to-be parents, they choose to consult astrologers to help with choosing the right name for their baby.

Naming a baby based on his or her sun sign or raashi is considered to bring good luck. The astrologer tells them that it is good to name their kid such that the kid’s name is the smallest substring in dad’s name containing all characters of mom’s name efficiently. Can you help them in shortlisting the names by writing a program to find the smallest substring in string1 containing all characters of string2 efficiently.

[Hint : i.e Find the smallest window in string 1 containing all characters of string 2] ] Input Format

Input consists of 2 strings. The first string corresponds to String 1 The second string corresponds to String

2. The maximum length of the strings is 100. Constraints

--

Output Format

Output consists of a string. Assume that it is always possible to find the desired string.

Sample Input 0 thisisateststring tist

Sample Output 0

tstri

Sample Input 1 this is a pen npe

Sample Output 1 pen

Sample Input 2 this is a pen htis

Sample Output 2 this

Sample Input 3 thandapani ana

Sample Output 3 anda

Sample Input 4 gokulkumar kom

Sample Output 4 okulkum

Sample Input 5 amphisoft

pit

Sample Output 5 phisoft

Sample Input 6 this is a test string

tist

Sample Output 6 t stri

Sample Input 7 this is a test string this

Sample Output 7 this

Sample Input 8 pradeep

ap

Sample Output 8 pra

Q2)

In the land of Rhyme, the King and his people always speak in rhymes. They are so accustomed in conversing using rhymes that at one point it became monotonous. So what was once a source of happiness and amusement is now creating a strange emotion within the heart of the people. The productivity of the people has become less and more importantly, the happiness index of the land has plummeted. The experts have investigated the whole matter and reported that speaking in rhyme is not the problem; it is the monotony of the same thing that is affecting the people. In fact they suggested changing the definition of rhyme slightly. One of their recommendations is to introduce a new concept of "richness" in a sentence.

A sentence would be termed rich if it starts and ends with the same word! The King seems to like this definition. But he is finding it difficult to even say a simple meaningful sentence that is "rich". The King became quite anxious. Your boss, King’s ICT adviser, suggested that a competition among the general people should be thrown where the participants would have to submit a piece (story/poem/essay etc.) full of rich sentences. Then the richest pieces would be judged and be awarded. The King made a decree accordingly. Your boss was ordered to formulate a judging scheme for the pieces and write a program immediately to do the judgment. And as usual he formulated something and asked you to implement that. What he has asked you to do is as follows. You will take as input a piece as a character array. And output another array where each position will give the richness of the piece up to that position.

“What is richness?” You asked.

Your boss replied, “The richness is the size of the identical proper substring which is present at the

beginning and at the end of the piece. But, the complete string cannot be considered as its substring.” “But you said to calculate the richness at each position”! You inquired.

“Yes, of course! When you calculate up to a position, you will have to assume that the piece ends at that position.” Your boss smiled. He seemed to be quite happy in formulating this.

You asked again, "Don't I need to think about the word boundaries?"

“Not now, my son. We will think about it later. Do it without considering word boundaries for now. You will consider the input character by character”. He assured you. He seemed all the more content with his own formulation and probably was not seeing the possible pitfalls. He continued, "And one other point; richness of the first position would always be 0."

Meaningful, you thought. But you had to ask a final question. “How would you then finally compute the richness of the total piece using all these information? What will be the formula?”

Your boss seemed perplexed. He did not think this through, as it seemed. So, he said, “You need not

worry about that. Do as I said. Just remember that the input size could be quite large.”

You knew that you have to stop asking and start coding now. “Yes, boss!” you said. You also realized the implication of the last direction of your boss. Few years back there was another competition on literary works. And the size of some pieces was 4 million characters!

Input Format

Input contains a string of lowercase English letters. Assume the total length of input is less than 1000 Constraints

--

Output Format

Output a line containing several integers, separated by space. The number of integers must be same as the length of the input string. The Integer at a specific position must represent the richness of the input string up to that position. Refer sample input and output for formatting details.

Sample Input 0 accb

Sample Output 0

0 0 0 0

Sample Input 1 dacbbcbcbcbcdcbbbcbcbcbcbcbcdacbbcbcdcbbbcbcdcbbdcbb Sample Output 1

0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0

Sample Input 2 abababbbbaaabbbbbababab Sample Output 2

0 0 1 2 3 4 0 0 0 1 1 1 2 0 0 0 0 1 2 3 4 5 6

Sample Input 3 atofpgrwk Sample Output 3

0 0 0 0 0 0 0 0 0

Sample Input 4 dacc

Sample Output 4

0 0 0 0

Sample Input 5 cbda

Sample Output 5

0 0 0 0

Sample Input 6 ababbacdbacdbacd Sample Output 6

0 0 1 2 0 1 0 0 0 1 0 0 0 1 0 0

Sample Input 7 bdblhc

Sample Output 7

0 0 1 0 0 0

Sample Input 8 jgojeqp

Sample Output 8

0 0 0 1 0 0 0

Sample Input 9 dcbb

Sample Output 9

0 0 0 0

Sample Input 10 fdcdd

Sample Output 10

0 0 0 0 0

Sample Input 11 oebmgoca Sample Output 11

0 0 0 0 0 1 0 0

Q3)

SMS language or textese (also known as txt-speak, txtese, chatspeak, txt, txtspk, txtk, txto, texting language, txt lingo, SMSish, txtslang,or txt talk) is a term for the abbreviations and slang commonly used with mobile phone text messaging.

Some of the abbreviations used are

s for yes u for you 2day for today y for why

Many grandpa's have started sending SMSes to their grand children. But they are not familiar with the SMS lingo.

Can you help them by writing a program that would convert a given text in proper English to SMS lingo? Consider only the 4 words listed above.

Input Format

Input consists of a single string. Assume that the maximum length of the string is 200 and all letters are in lower-case.

Constraints

--

Output Format

Output consists of a single string.

Sample Input 0

yesterday comes after today for your kid yes Sample Output 0

sterday comes after 2day for ur kid s Sample Input 1

why is today a working day for you?

Sample Output 1

y is 2day a working day for u?

Sample Input 2 yes you come in Sample Output 2 s u come in Sample Input 3

why is your kid playing today and yesterday Sample Output 3

y is ur kid playing 2day and sterday Sample Input 4

where were you yesterday?

Sample Output 4 where were u sterday?

Q4)

The lead role in Joker is that of Mannar Mannan, a mentally unsound and poor activist who thinks he is the people-appointed President of India. Mannan falls in love with Malliga, courts her and wins her over and they get married. Soon , they are expecting a child and they are very excited about it. As most other to-be parents, they choose to consult astrologers to help with choosing the right name for their baby.

Naming a baby based on his or her sun sign or raashi is considered to bring good luck. The astrologer tells them that it is good to name their kid such that the kid’s name is an interleaving of the mom’s name and dad’s name.

Can you help them in shortlisting the names by writing a program to find whether string C is an interleaving of strings A and B? [Hint : C is said to be interleaving A and B, if it contains all characters of A and B and order of all characters in individual strings is preserved.

String A = "AB", String B = "CD"

The possible interleavings of strings A and B are ABCD

ACBD ACDB CABD CADB CDAB

]

Input Format

Input consists of 3 strings. The first string corresponds to A. The second string corresponds to B. The third string corresponds to C. Assume that there are no common characters between strings A and B. All input strings consist of only upper case alphabets. The maximum length of the strings is 100.

Constraints

--

Output Format

Output consists of a string that is either “YES” or “NO”.

Sample Input 0 AB

CD ACBD

Sample Output 0 YES

Sample Input 1 PUNITHA

SOJ SPAUONJITH

Sample Output 1 NO

Sample Input 2 AB

CD CDAB

Sample Output 2 YES

Sample Input 3 AB

CD ACDB

Sample Output 3 YES

Sample Input 4 AB

CD ABCG

Sample Output 4 NO

Sample Input 5 PRADEEP SMITH PSRMAIDETPH

Sample Output 5 NO

Sample Input 6 AB

CD CABD

Sample Output 6 YES

Sample Input 7 AB

CD BDCD

Sample Output 7 NO

Sample Input 8 PUNITHA

SOJ SOJPUNITHA

Sample Output 8 YES

Sample Input 9 PUNITH KANNAN

PUNITHAKANNAN

Sample Output 9 NO

Sample Input 10 AB

CD BCAD

Sample Output 10 NO

Sample Input 11 PUNITHA

SOJ PSUNIOTJHA

Sample Output 11 YES

Sample Input 12 PRADEEP SMITH PSRMAIDEETPH

Sample Output 12 YES

Sample Input 13 AB

CD ABCD

Sample Output 13 YES

Sample Input 14 AB

CD CADB

Sample Output 14 YES

Q5)

The lead role in Joker is that of Mannar Mannan, a mentally unsound and poor activist who thinks he is the people-appointed President of India. Mannan falls in love with Malliga, courts her and wins her over and they get married. Soon , they are expecting a child and they are very excited about it. They were

exhilarated to know that Malliga is carrying twins. As most other to-be parents, they choose to consult astrologers to help with choosing the right name for their baby. Naming a baby based on his or her sun sign or raashi is considered to bring good luck. The astrologer tells them that it is good to name the twins in such a way that their names are isomorphic strings.

Can you help them in shortlisting the names by writing a program to find whether the two strings A and B are isomorphic?

[Hint : Two strings str1 and str2 are called isomorphic if there is a one to one mapping possible for every character of str1 to every character of str2. And all occurrences of every character in ‘str1′ map to same character in ‘str2′ Example : Input: String A = "aab", String B = "xxy" Output: YES 'a' is mapped to 'x' and 'b' is mapped to 'y'.

Input: String A = "aab", String B = "xyz" Output: NO One occurrence of 'a' in str1 has 'x' in str2 and other occurrence of 'a' has 'y'. Two words are called isomorphic if the letters in one word can be remapped to get the second word. Remapping a letter means replacing all occurrences of it with another letter. The ordering of the letters remains unchanged. No two letters may map to the same letter, but a letter may map to itself. ]

Input Format

Input consists of 2 strings. The first string corresponds to A. The second string corresponds to B. The maximum length of the strings is 100.

Constraints

--

Output Format

Output consists of a string that is either “YES” or “NO”.

Sample Input 0 paper

title

Sample Output 0 YES

Sample Input 1 voter

seater

Sample Output 1 NO

Sample Input 2

punitha pradeep

Sample Output 2 NO

Sample Input 3 aab

xyx

Sample Output 3 NO

Sample Input 4 owl

ant

Sample Output 4 YES

Sample Input 5 beep

soon

Sample Output 5 YES

Sample Input 6 cacccdaabc cdcccaddbc Sample Output 6 YES

Sample Input 7 cacccdaabc bdbcadbbdc Sample Output 7

NO

Sample Input 8 foo

bar

Sample Output 8 NO

Sample Input 9 aab

xxy

Sample Output 9 YES

Sample Input 10 cacccdaabc bdbbbaddcb Sample Output 10 YES

Sample Input 11 abca

zbxz

Sample Output 11 YES

Sample Input 12 ab

cd

Sample Output 12 YES

Sample Input 13 abca

opqr

Sample Output 13 NO

Sample Input 14 aab

xyz

Sample Output 14 NO

Sample Input 15 seater

voter

Sample Output 15 NO

Sample Input 16 ab

hh

Sample Output 16 NO

Sample Input 17 egg

add

Sample Output 17 YES

Q6)

Stickler is a thief and wants to loot money from a society of n houses placed in a line. He is a weird person and follows a rule while looting the houses and according to the rule he will never loot two consecutive houses. At the same time, he wants to maximize the amount he loots. The thief knows which house has what amount of money but is unable to find the maximum amount he can end up with. He asks for your

help to find the maximum money he can get if he strictly follows the rule. Each house has a[i] amount of money present in it.

Input Format

Each test case contains an integer n which denotes the number of elements in the array a[]. Next line contains space separated n elements in the array a[].

Constraints

1<=n<=1000 1<=a[i]<=10000

Output Format

Print an integer which denotes the maximum amount he can take home. Sample Input 0

6

5 5 10 100 10 5

Sample Output 0

110

Sample Input 1

3

1 2 3

Sample Output 1

4

Sample Input 2

5

10 10 10 10 10

Sample Output 2

30

Sample Input 3

5

110 10 10 100 90

Sample Output 3

210

Sample Input 4

6

10 30 20 20 10 30

Sample Output 4

80

Sample Input 5

6

0 0 0 100 0 0

Sample Output 5

100

Sample Input 6

4

4 1 2 6

Sample Output 6

10

Q7)

Given a positive number x, print all Jumping Numbers smaller than or equal to x. A number is called as a Jumping Number if all adjacent digits in it differ by 1. The difference between ‘9’ and ‘0’ is not considered as 1. All single digit numbers are considered as Jumping Numbers. For example 7, 8987 and 4343456 are Jumping numbers but 796 and 89098 are not.

Input Format

The first line of the input contains T denoting the number of testcases. Each testcase contain a positive number 'x'.

Constraints

1 <=T<= 100 1 <=N<= 100000

Output Format

All the jumping numbers less than 'x' are generated in increasing order of the most significant digit. See example for better understanding.

Sample Input 0

2

10

50

Sample Output 0

0 1 10 2 3 4 5 6 7 8 9

0 1 10 12 2 21 23 3 32 34 4 43 45 5 6 7 8 9

Sample Input 1

1

30

Sample Output 1

0 1 10 12 2 21 23 3 4 5 6 7 8 9

Sample Input 2

1

40

Sample Output 2

0 1 10 12 2 21 23 3 32 34 4 5 6 7 8 9

Sample Input 3

1

200

Sample Output 3

0 1 10 12 101 121 123 2 21 23 3 32 34 4 43 45 5 54 56 6 65 67 7 76 78 8 87 89 9 98

Sample Input 4

1

300

Sample Output 4

0 1 10 12 101 121 123 2 21 23 210 212 232 234 3 32 34 4 43 45 5 54 56 6 65 67 7 76 78 8 87 89 9 98

Q8)

Given an array of size N , print the total count of sub-arrays having their sum equal to 0 Input Format

First line of the input contains an integer T denoting the number of test cases. Each test case consists of two lines. First line of each test case contains an Integer N denoting the size of the array and the second line contains N space separated integers.

Constraints

1<= N <= 10000 -100 <= A[i] <= 100 where i <= N

Output Format

Corresponding to each test case, print the total number of subarrays whose sum is equal to 0. Sample Input 0

2

6

0 0 5 5 0 0

10

6 -1 -3 4 -2 2 4 6 -12 -7

Sample Output 0

6

4

Sample Input 1

2

5

0 0 5 5 0

9

6 -1 -3 4 -2 2 4 6 -12

Sample Output 1

4

4

Sample Input 2

2

4

0 0 5 5

8

6 -1 -3 4 -2 2 4 6

Sample Output 2

3

3

Sample Input 3

2

3

0 1 0

8

6 -1 -3 4 -2 2 4 6

Sample Output 3

2

3

Sample Input 4

1

5

1 -1 2 -1 0

Sample Output 4

4

Q9)

A string with parentheses is well bracketed if all parentheses are matched: every opening bracket has a matching closing bracket and vice versa.

Write a Python function wellbracketed(s) that takes a string s containing parentheses and returns True if s is well bracketed and False otherwise.

Input Format

A single line string Constraints

None

Output Format

Refer the sample test cases Sample Input 0

22)

Sample Output 0 False

Sample Input 1 (a+b)(a-b) Sample Output 1 True

Sample Input 2 (a(b+c)-d)((e+f) Sample Output 2 False

Sample Input 3

(7(89)

Sample Output 3 False

Sample Input 4

(7(8()9()))

Sample Output 4 True

Sample Input 5

)1(

Sample Output 5

False

Q10)

A bigger N\*N matrix is passed as the input. Also a smaller M\*M matrix is passed as the input. The program must print 'TRUE' if the smaller matrix elements can be found in the bigger matrix irrespective of their position. Else program must print 'FALSE' in case of absence or duplication of any smaller matrix element.

Input Format:

First line will contain the value of N Second line will contain the value of M

Next N lines will contain the values in the N\*N matrix with each value separated by one or more spaces. Next M lines will contain the values in the M\*M matrix with each value separated by one or more spaces. Output Format:

First line will contain the string value. Example1:

Input:

3 2 7 5 9 1 3 5 8 2 4 3 5 2 4

Output: TRUE

Example2: Input: 3 2 4 5 9 1 3 5 8 2 4 3 7 2 4 Output: FALSE

Input Format

First line will contain the value of N Second line will contain the value of M

Next N lines will contain the values in the N\*N matrix with each value separated by one or more spaces. Next M lines will contain the values in the M\*M matrix with each value separated by one or more spaces. Constraints

None

Output Format

First line will contain the string value. Sample Input 0

3

2

4 5 9

1 3 5

8 2 4

3 5

2 4

Sample Output 0 TRUE

Sample Input 1

5

3

1 2 3 4 5

6 7 8 9 10

20 21 22 23 24

11 12 13 14 15

1 2 3 4 5

1 2 3

4 5 6

7 8 9

Sample Output 1 TRUE

Sample Input 2

5

3

1 2 3 4 5

16 7 8 9 10

20 21 22 23 24

11 12 13 14 9

1 2 3 4 5

1 2 3

4 5 6

7 8 9

Sample Output 2 FALSE

Sample Input 3

3

2

4 5 9

1 3 5

8 2 4

40 50

20 40

Sample Output 3 FALSE

Sample Input 4

3

3

1 2 3

4 5 6

7 8 9

1 2 2

4 5 5

7 8 8

Sample Output 4 FALSE

Sample Input 5

3

3

1 2 3

4 5 6

7 8 9

1 2 3

4 5 6

7 8 9

Sample Output 5 TRUE

Q11)

Given a keypad as shown in diagram, and a n digit number, list all words which are possible by pressing these numbers.

1 2 3

abc def

4 5 6

ghi jkl mno

7 8 9

pqrs tuv wxyz

\* 0 #

Input Format

The first line of input contains an integer T denoting the number of test cases. The first line of each test case is N,N is the number of digits. The second line of each test case contains D[i], N number of digits.

Constraints

1 ≤ T ≤ 10 1 ≤ N ≤ 5 2 ≤ D[i] ≤ 9

Output Format

Print all possible words from phone digits with single space. Sample Input 0

1

3

2 3 4

Sample Output 0

adg adh adi aeg aeh aei afg afh afi bdg bdh bdi beg beh bei bfg bfh bfi cdg cdh cdi ceg ceh cei cfg cfh cfi Sample Input 1

1

4

5 2 7 9

Sample Output 1

japw japx japy japz jaqw jaqx jaqy jaqz jarw jarx jary jarz jasw jasx jasy jasz jbpw jbpx jbpy jbpz jbqw jbqx jbqy jbqz jbrw jbrx jbry jbrz jbsw jbsx jbsy jbsz jcpw jcpx jcpy jcpz jcqw jcqx jcqy jcqz jcrw jcrx jcry jcrz jcsw jcsx jcsy jcsz kapw kapx kapy kapz kaqw kaqx kaqy kaqz karw karx kary karz kasw kasx kasy kasz kbpw kbpx kbpy kbpz kbqw kbqx kbqy kbqz kbrw kbrx kbry kbrz kbsw kbsx kbsy kbsz kcpw kcpx kcpy kcpz kcqw kcqx kcqy kcqz kcrw kcrx kcry kcrz kcsw kcsx kcsy kcsz lapw lapx lapy lapz laqw laqx laqy laqz larw larx lary larz lasw lasx lasy lasz lbpw lbpx lbpy lbpz lbqw lbqx lbqy lbqz lbrw lbrx lbry lbrz lbsw lbsx lbsy lbsz lcpw lcpx lcpy lcpz lcqw lcqx lcqy lcqz lcrw lcrx lcry lcrz lcsw lcsx lcsy lcsz

Sample Input 2

1

3

7 7 7

Sample Output 2

ppp ppq ppr pps pqp pqq pqr pqs prp prq prr prs psp psq psr pss qpp qpq qpr qps qqp qqq qqr qqs qrp qrq qrr qrs qsp qsq qsr qss rpp rpq rpr rps rqp rqq rqr rqs rrp rrq rrr rrs rsp rsq rsr rss spp spq spr sps sqp sqq sqr sqs srp srq srr srs ssp ssq ssr sss

Sample Input 3

1

3

1 2 3

Sample Input 4

1

3

2 3 4

Sample Output 4

adg adh adi aeg aeh aei afg afh afi bdg bdh bdi beg beh bei bfg bfh bfi cdg cdh cdi ceg ceh cei cfg cfh cfi Sample Input 5

1

5

2 3 4 5 7

Sample Output 5

adgjp adgjq adgjr adgjs adgkp adgkq adgkr adgks adglp adglq adglr adgls adhjp adhjq adhjr adhjs adhkp adhkq adhkr adhks adhlp adhlq adhlr adhls adijp adijq adijr adijs adikp adikq adikr adiks adilp adilq adilr adils aegjp aegjq aegjr aegjs aegkp aegkq aegkr aegks aeglp aeglq aeglr aegls aehjp aehjq aehjr aehjs aehkp aehkq aehkr aehks aehlp aehlq aehlr aehls aeijp aeijq aeijr aeijs aeikp aeikq aeikr aeiks aeilp aeilq aeilr aeils afgjp afgjq afgjr afgjs afgkp afgkq afgkr afgks afglp afglq afglr afgls afhjp afhjq afhjr afhjs afhkp afhkq afhkr afhks afhlp afhlq afhlr afhls afijp afijq afijr afijs afikp afikq afikr afiks afilp afilq afilr afils bdgjp bdgjq bdgjr bdgjs bdgkp bdgkq bdgkr bdgks bdglp bdglq bdglr bdgls bdhjp bdhjq bdhjr bdhjs bdhkp bdhkq bdhkr bdhks bdhlp bdhlq bdhlr bdhls bdijp bdijq bdijr bdijs bdikp bdikq bdikr bdiks bdilp bdilq bdilr bdils begjp begjq begjr begjs begkp begkq begkr begks beglp beglq beglr begls behjp behjq behjr behjs behkp behkq behkr behks behlp behlq behlr behls beijp beijq beijr beijs beikp beikq beikr beiks beilp beilq beilr beils bfgjp bfgjq bfgjr bfgjs bfgkp bfgkq bfgkr bfgks bfglp bfglq bfglr bfgls bfhjp bfhjq bfhjr bfhjs bfhkp bfhkq bfhkr bfhks bfhlp bfhlq bfhlr bfhls bfijp bfijq bfijr bfijs bfikp bfikq bfikr bfiks bfilp bfilq bfilr bfils cdgjp cdgjq cdgjr cdgjs cdgkp cdgkq cdgkr cdgks cdglp cdglq cdglr cdgls cdhjp cdhjq cdhjr cdhjs cdhkp cdhkq cdhkr cdhks cdhlp cdhlq cdhlr cdhls cdijp cdijq cdijr cdijs cdikp cdikq cdikr cdiks cdilp cdilq cdilr cdils cegjp cegjq cegjr cegjs cegkp cegkq cegkr cegks ceglp ceglq ceglr cegls cehjp cehjq cehjr cehjs cehkp cehkq cehkr cehks cehlp cehlq cehlr cehls ceijp ceijq ceijr ceijs ceikp ceikq ceikr ceiks ceilp ceilq ceilr ceils cfgjp cfgjq cfgjr cfgjs cfgkp cfgkq cfgkr cfgks cfglp cfglq cfglr cfgls cfhjp cfhjq cfhjr cfhjs cfhkp cfhkq cfhkr cfhks cfhlp cfhlq cfhlr cfhls cfijp cfijq cfijr cfijs cfikp cfikq cfikr cfiks cfilp cfilq cfilr cfils

Q12)

Given an integer N as the input, print the pattern as given in the Example Input/Output section.

Input Format : The first line contains N. Constraints : 2 <= N <= 100

Output Format : N lines containing the desired pattern. Input : 4

Output : 1\*2\*3\*4\*17\*18\*19\*20 --5\*6\*7\*14\*15\*16 ----8\*9\*12\*13 10\*11

Input Format

The first line contains N.

Constraints

2 <= N <= 100

Output Format

N lines containing the desired pattern. Sample Input 0

4

Sample Output 0 1\*2\*3\*4\*17\*18\*19\*20

--5\*6\*7\*14\*15\*16

----8\*9\*12\*13

- 10\*11

Sample Input 1

7

Sample Output 1 1\*2\*3\*4\*5\*6\*7\*50\*51\*52\*53\*54\*55\*56

--8\*9\*10\*11\*12\*13\*44\*45\*46\*47\*48\*49

----14\*15\*16\*17\*18\*39\*40\*41\*42\*43

- 19\*20\*21\*22\*35\*36\*37\*38

- 23\*24\*25\*32\*33\*34

- 26\*27\*30\*31

28\*29

Sample Input 2

6

Sample Output 2 1\*2\*3\*4\*5\*6\*37\*38\*39\*40\*41\*42

--7\*8\*9\*10\*11\*32\*33\*34\*35\*36

----12\*13\*14\*15\*28\*29\*30\*31

- 16\*17\*18\*25\*26\*27

- 19\*20\*23\*24

21\*22

Sample Input 3

10

Sample Output 3 1\*2\*3\*4\*5\*6\*7\*8\*9\*10\*101\*102\*103\*104\*105\*106\*107\*108\*109\*110

--11\*12\*13\*14\*15\*16\*17\*18\*19\*92\*93\*94\*95\*96\*97\*98\*99\*100

----20\*21\*22\*23\*24\*25\*26\*27\*84\*85\*86\*87\*88\*89\*90\*91

- 28\*29\*30\*31\*32\*33\*34\*77\*78\*79\*80\*81\*82\*83

- 35\*36\*37\*38\*39\*40\*71\*72\*73\*74\*75\*76

- 41\*42\*43\*44\*45\*66\*67\*68\*69\*70

- 46\*47\*48\*49\*62\*63\*64\*65

- 50\*51\*52\*59\*60\*61

- 53\*54\*57\*58

55\*56

Sample Input 4

12

Sample Output 4 1\*2\*3\*4\*5\*6\*7\*8\*9\*10\*11\*12\*145\*146\*147\*148\*149\*150\*151\*152\*153\*154\*155\*156

--13\*14\*15\*16\*17\*18\*19\*20\*21\*22\*23\*134\*135\*136\*137\*138\*139\*140\*141\*142\*143\*144

----24\*25\*26\*27\*28\*29\*30\*31\*32\*33\*124\*125\*126\*127\*128\*129\*130\*131\*132\*133

- 34\*35\*36\*37\*38\*39\*40\*41\*42\*115\*116\*117\*118\*119\*120\*121\*122\*123

- 43\*44\*45\*46\*47\*48\*49\*50\*107\*108\*109\*110\*111\*112\*113\*114

- 51\*52\*53\*54\*55\*56\*57\*100\*101\*102\*103\*104\*105\*106

- 58\*59\*60\*61\*62\*63\*94\*95\*96\*97\*98\*99

- 64\*65\*66\*67\*68\*89\*90\*91\*92\*93

- 69\*70\*71\*72\*85\*86\*87\*88

- 73\*74\*75\*82\*83\*84

76\*77\*80\*81

78\*79

Sample Input 5

2

Sample Output 5 1\*2\*5\*6

--3\*4

Sample Input 6

20

Sample Output 6 1\*2\*3\*4\*5\*6\*7\*8\*9\*10\*11\*12\*13\*14\*15\*16\*17\*18\*19\*20\*401\*402\*403\*404\*405\*406\*407\*408\*40

9\*410\*411\*412\*413\*414\*415\*416\*417\*418\*419\*420

--

21\*22\*23\*24\*25\*26\*27\*28\*29\*30\*31\*32\*33\*34\*35\*36\*37\*38\*39\*382\*383\*384\*385\*386\*387\*388\*

389\*390\*391\*392\*393\*394\*395\*396\*397\*398\*399\*400

40\*41\*42\*43\*44\*45\*46\*47\*48\*49\*50\*51\*52\*53\*54\*55\*56\*57\*364\*365\*366\*367\*368\*369\*370\*371

\*372\*373\*374\*375\*376\*377\*378\*379\*380\*381

58\*59\*60\*61\*62\*63\*64\*65\*66\*67\*68\*69\*70\*71\*72\*73\*74\*347\*348\*349\*350\*351\*352\*353\*354\*35

5\*356\*357\*358\*359\*360\*361\*362\*363

75\*76\*77\*78\*79\*80\*81\*82\*83\*84\*85\*86\*87\*88\*89\*90\*331\*332\*333\*334\*335\*336\*337\*338\*339\*3

40\*341\*342\*343\*344\*345\*346

91\*92\*93\*94\*95\*96\*97\*98\*99\*100\*101\*102\*103\*104\*105\*316\*317\*318\*319\*320\*321\*322\*323\*32

4\*325\*326\*327\*328\*329\*330

106\*107\*108\*109\*110\*111\*112\*113\*114\*115\*116\*117\*118\*119\*302\*303\*304\*305\*306\*307\*308\*3

09\*310\*311\*312\*313\*314\*315

120\*121\*122\*123\*124\*125\*126\*127\*128\*129\*130\*131\*132\*289\*290\*291\*292\*293\*294\*295\*296\*2

97\*298\*299\*300\*301

133\*134\*135\*136\*137\*138\*139\*140\*141\*142\*143\*144\*277\*278\*279\*280\*281\*282\*283\*284\*285\*2

86\*287\*288

145\*146\*147\*148\*149\*150\*151\*152\*153\*154\*155\*266\*267\*268\*269\*270\*271\*272\*273\*274\*275\*2

76

156\*157\*158\*159\*160\*161\*162\*163\*164\*165\*256\*257\*258\*259\*260\*261\*262\*263\*264\*265

- 166\*167\*168\*169\*170\*171\*172\*173\*174\*247\*248\*249\*250\*251\*252\*253\*254\*255

- 175\*176\*177\*178\*179\*180\*181\*182\*239\*240\*241\*242\*243\*244\*245\*246

- 183\*184\*185\*186\*187\*188\*189\*232\*233\*234\*235\*236\*237\*238

- 190\*191\*192\*193\*194\*195\*226\*227\*228\*229\*230\*231

- 196\*197\*198\*199\*200\*221\*222\*223\*224\*225

- 201\*202\*203\*204\*217\*218\*219\*220

- 205\*206\*207\*214\*215\*216

208\*209\*212\*213

210\*211

Q13)

To encrypt the message Jill will first decide on the number of columns C to use. The Jill will pad the message with letters chosen randomly so that they form a rectangular matrix. Finally Jill will write down the message as encrypted by navigating the rows from left to right and then right to left. Write a program which must accept the encrypted message M as input and then extract the original message from the encrypted message based on the value of C.

Input Format: First line will contain the string value of encrypted message M. Second line will contain the integer value of the column used for the encryption.

Output Format: First line will contain the string value of decrypted message along with any additional padding letters.

Sample Input1: midinadiazne 3 Sample Output1: madeinindiaz Input Format

First line will contain the string value of encrypted message M. Second line will contain the integer value of the column used for the encryption.

Constraints

.

Output Format

First line will contain the string value of decrypted message along with any additional padding letters.

Sample Input 0 midinadiazne 3

Sample Output 0 madeinindiaz Sample Input 1

reegrenaggreciihunn1 4

Sample Output 1 raghuengineeringrec1 Sample Input 2 grgoooodfd9o

2

Sample Output 2 goodforgood9 Sample Input 3

TRANTORHGMOHEGMIELDWSIRP 6

Sample Output 3 THEPROGRAMMINGISTHEWORLD

Sample Input 4 topper

1

Sample Output 4 topper

Q14)

Given an array of distinct elements, rearrange the elements of array in zig-zag fashion in O(n) time. The converted array should be in form a < b > c < d > e < f. Comparisons must be done in the relative order of elements in the original array.

Input Format

The first line of input contains an integer N denoting the size of array. The second line contains N space- separated integers A1, A2, ..., AN denoting the elements of the array.

Constraints

1 ≤ N ≤ 100 0 ≤A[i]<=10000

Output Format

Print the array in Zig-Zag fashion. Sample Input 0

7

4 3 7 8 6 2 1

Sample Output 0

3 7 4 8 2 6 1

Sample Input 1

4

1 4 3 2

Sample Output 1

1 4 2 3

Sample Input 2

6

1 2 3 4 5 6

Sample Output 2

1 3 2 5 4 6

Sample Input 3

6

1 2 3 4 5 6

Sample Output 3

1 3 2 5 4 6

Sample Input 4

11

-10 10 20 -20 -30 40 50 -60 -70 -80 90

Sample Output 4

-10 20 -20 10 -30 50 -60 40 -80 90 -70

Q15)

Write a program to print all the LEADERS in the array. An element is leader if it is greater than all the elements to its right side. The rightmost element is always a leader.

Input Format

The first line of input contains an integer T denoting the number of test cases. The description of T test cases follows. The first line of each test case contains a single integer N denoting the size of array. The second line contains N space-separated integers A1, A2, ..., AN denoting the elements of the array.

Constraints

1 <= T <= 100 1 <= N <= 100 0 <= A[i]<=100

Output Format Print all the leaders.

Sample Input 0

2

6

16 17 4 3 5 2

5

1 2 3 4 0

Sample Output 0

17 5 2

4 0

Sample Input 1

1

9

9 8 7 6 5 4 3 2 1

Sample Output 1

9 8 7 6 5 4 3 2 1

Sample Input 2

1

9

1 2 3 4 5 6 7 8 9

Sample Output 2

9

Sample Input 3

1

5

1 1 1 1 1

Sample Output 3

1

Sample Input 4

1

1

0

Sample Output 4

0

Q16)

Give a N\*N square matrix, return an array of its anti-diagonals. Look at the example for more details.

For Example:

If the matrix is

1 2 3

4 5 6

7 8 9

The output should Return the following :

[

[1],

[2, 4],

[3, 5, 7],

[6, 8],

[9]

]

i.e print the elements of array diagonally.

Note: The input array given is in single line and you have to output the answer in one line only. Input Format

The first line contains an integer T, depicting total number of test cases. Then following T lines contains an integer N depicting the size of square matrix and next line followed by the value of array.

Constraints

1 ≤ T ≤ 30 1 ≤ N ≤ 20 0 ≤ A[i][j] ≤ 9

Output Format

Print the elements of matrix diagonally in separate line. Sample Input 0

2

2

1 2

3 4

3

1 2 3

4 5 6

7 8 9

Sample Output 0

1 2 3 4

1 2 4 3 5 7 6 8 9

Sample Input 1

1

3

1 2 4 3 5 7 6 8 9

Sample Output 1

1 2 3 4 5 6 7 8 9

Sample Input 2

1

4

1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4

Sample Output 2

1 2 1 3 2 1 4 3 2 1 4 3 2 4 3 4

Sample Input 3

1

9

1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5

6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9

Sample Output 3

1 2 1 3 2 1 4 3 2 1 5 4 3 2 1 6 5 4 3 2 1 7 6 5 4 3 2 1 8 7 6 5 4 3 2 1 9 8 7 6 5 4 3 2 1 9 8 7 6 5 4 3 2 9 8 7 6 5 4

3 9 8 7 6 5 4 9 8 7 6 5 9 8 7 6 9 8 7 9 8 9

Sample Input 4

1 9

1 2 3 4 5 6 7 8 9

9 8 7 6 5 4 3 2 1

1 2 3 4 5 6 7 8 9

9 8 7 6 5 4 3 2 1

1 2 3 4 5 6 7 8 9

9 8 7 6 5 4 3 2 1

1 2 3 4 5 6 7 8 9

9 8 7 6 5 4 3 2 1

1 2 3 4 5 6 7 8 9

Sample Output 4

1 2 9 3 8 1 4 7 2 9 5 6 3 8 1 6 5 4 7 2 9 7 4 5 6 3 8 1 8 3 6 5 4 7 2 9 9 2 7 4 5 6 3 8 1 1 8 3 6 5 4 7 2 9 2 7 4 5 6

3 1 8 3 6 5 4 9 2 7 4 5 1 8 3 6 9 2 7 1 8 9

Sample Input 5

1

1

6

Sample Output 5

6

Sample Input 6

1

0

Q17)

A string is given which has letters from English alphabet and parentheses. The depth of a letter is defined as the number of balanced parentheses it is surrounded by. Write a C program to find the depth of each letter in the input string. Explanation:

(a(b)((cd)e)f)g

g is at depth 0 a and f are at depth 1 b and e are at depth 2 c and d are at depth 3 Input Format

An array of characters Constraints

1) Input string can be of length 1 to 100. 2) The input will have only ‘(‘ , ‘)’ and letters from English alphabet. 3) There will be no repetition of letters. 4) Only lowercase letters are used. 5) The letters can be in any sequence. 6) The parentheses are always well balanced. Every '(' has a matching ')' that follows it later in the string. Every ')' has a matching '(' before it in the input string. 7) Notice that there are no space in the string.

Output Format

1) The depth of each letter separated by a space. 2) The order of the depth of the letters should be the same order in which the letters appear in the input. 3)To mark the end of the output it should end with a space and a ‘#’ character.

Example 1: Input: (a(b)((cd)e)f)g Output: 1 2 3 3 2 1 0 #

Example 2: Input: p(r(q))(s) Output: 0 1 2 1 #

In this example, letters are appearing in the order p followed by r followed by q and s. They have depth of 0, 1, 2 and 1 respectively. Note that the depth is not printed in the order p,q,r,s (the alphabetical order) but p,r,q,s (the order in which they appear in the input string).

Sample Input 0 (a(b)((cd)e)f)g Sample Output 0

1 2 3 3 2 1 0 #

Sample Input 1 a(b(c))(d) Sample Output 1

0 1 2 1 #

Sample Input 2 a(b(c))(d(fe)) Sample Output 2

0 1 2 1 2 2 #

Sample Input 3 a(b(c))(d(f()e)) Sample Output 3

0 1 2 1 2 2 #

Sample Input 4 ()

Sample Output 4 #

Sample Input 5 ab()(c(d(e(f)()(g)h))) Sample Output 5

0 0 1 2 3 4 4 3 #

Q18)

There is a class of N students and you have to find the top K marks scorers.You need to print the index of the toppers of the class which will be same as the index of the student in the input array(use 0-based indexing).First print the index of the students having highest marks then the students with second highest and so on.If there are more than one students having same marks then print their indices in ascending order.Suppose k = 2 and the students having highest marks have indices 0 and 5 and students having second highest marks have indices 6 and 7 then output will be 0 5 6 7

Input Format

The first line of input consists of an integer T denoting the number of test cases .Each test case consists of three lines first line consists of an integer N denoting the number of students of class second line consists N spaced integers denoting the marks of students and third line consists of an integer K which denotes the number of top scores you have to find.

Constraints

1<=T<=100 1<=N<=1000 1<=marks of students<=10^6

It is given that k will be such that its value will always be less than or equal to type of marks of students Output Format

Single line output, denoting the indices of top k cores with space between them. Sample Input 0

1

5

2 2 1 3 1

2

Sample Output 0

3 0 1

Sample Input 1

1

6

4 7 2 1 5 9

3

Sample Output 1

5 1 4

Sample Input 2

1

5

4 4 4 4 4

2

Sample Output 2

0 1 2 3 4 0 1 2 3 4

Sample Input 3

1

9

9 8 5 676655 45 98 1 1 1

2

Sample Output 3

3 5

Q19)

There are two singly linked lists in a system. By some programming error the end node of one of the linked list got linked into the second list, forming a inverted Y shaped list. Write a program to get the point where two linked lists merge.

Y ShapedLinked List

Above diagram shows an example with two linked list having 15 as intersection point. Expected time complexity is O(m + n) where m and n are lengths of two linked lists Input Format

You have to complete the method which takes two arguments as heads of two linked lists.

First line is number of test cases. Every test case has four lines. First line of every test case contains three numbers, x (number of nodes before merge point in 1st list), y(number of nodes before merge point in 11nd list) and z (number of nodes after merge point). Next three lines contain x, y and z values.

Constraints

1 <=T<= 50 1 <=N<= 100 1 <=value<= 1000

Output Format

The function should return data value of a node where two linked lists merge. If linked list do not merge at any point, then it shoudl return -1.

Sample Input 0

2

2 3 2

10 20

30 40 50

5 10

2 3 0

10 20

30 40 50

Sample Output 0

5

-1

Sample Input 1

1

2 3 2

10 20

30 40 50

0 10

Sample Output 1

0

Sample Input 2

1

2 3 4

10 20

30 40 50

2 4 6 10

Sample Output 2

2

Sample Input 3

1

2 1 4

10 20

6

2 4 6 10

Sample Output 3

2

Sample Input 4

1

2 0 2

10 20

30

40

Sample Output 4

30

Sample Input 5

1

2 3 2

10 20

30 10 20

5 10

Sample Output 5

5

Q20)

There are 'n' ants on a 'n+1' length rod. The ants are numbered from 1 to n and are initially placed at positions starting from position 1 till position n. They are moving either in left direction (denoted by '-1') or in the right direction (denoted by '1'). Whenever an ant crosses the boundary of the rod it falls off the rod. You are given the initial direction of the ants. Now, whenever two ants collide their direction switches, i.e. the ant going in left direction ('-1) changes it's direction towards right ('1') and the ant going in the right direction ('1') changes it's direction towards left ('-1'). Find last ant to fall off the rod. Note: In case two ants are falling simultaneously in the end print the index of the lower indexed ant.

Input Format

Input Format: First line contains the integer 'n' denoting the total number of ants s.t. 1 <= n <= 1,000 Second line contains 'n' space separated numbers (either '1' or '-1') denoting the initial directions of the ants.

Constraints

Output Format

Output Format: Output a single integer which is the index (lower index in case two ants are falling simultaneously in the end) of the last ant to fall off the table.

Sample Input 0

2

1 1

Sample Output 0

1

Sample Input 1

3

1 -1 -1

Sample Output 1

2

Sample Input 2

2

-1 -1

Sample Output 2

2

Sample Input 3

8

1 1 -1 1 1 1 -1 1

Sample Output 3

3

Sample Input 4

10

1 -1 1 1 -1 1 -1 1 -1 -1

Sample Output 4

5

Sample Input 5

100

-1 -1 -1 -1 1 -1 -1 1 -1 1 -1 1 1 -1 -1 -1 1 -1 1 1 1 -1 1 1 1 -1 1 -1 1 -1 1 1 1 -1 -1 1 -1 -1 -1 1 -1 1 -1 1 -1 1 -1 1

1 1 1 1 -1 1 1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 1 -1 -1 1 1 -1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 -1 -1 1 1 1 1 -1 1 1 -1 -1 -1 -

1 1 1 -1

Sample Output 5

52

Sample Input 6

1000

1 1 1 -1 1 -1 1 -1 1 1 -1 -1 -1 1 -1 -1 1 -1 -1 1 -1 1 1 -1 -1 1 -1 -1 1 1 1 1 -1 1 1 -1 1 1 -1 1 -1 -1 1 -1 1 1 1 -1 1 -

1 -1 1 -1 1 -1 -1 1 -1 1 -1 -1 1 -1 1 1 1 -1 -1 1 -1 -1 -1 1 1 1 1 -1 1 1 1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 1

1 -1 -1 -1 1 1 1 1 1 -1 1 1 1 1 1 1 -1 1 1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 1 1 -1 1 1 1

-1 -1 -1 -1 1 1 1 1 -1 1 -1 -1 1 1 1 -1 -1 1 1 -1 1 -1 1 -1 1 1 -1 1 1 1 -1 1 -1 -1 -1 1 -1 1 -1 1 1 1 1 1 -1 -1 1 -1 1 -

1 1 -1 -1 -1 1 -1 1 1 -1 -1 -1 -1 1 -1 -1 1 1 -1 1 1 1 1 -1 -1 -1 -1 1 1 1 -1 1 -1 -1 1 1 -1 1 -1 1 -1 1 -1 1 1 -1 1 1 1

1 -1 -1 -1 1 -1 1 -1 -1 1 1 -1 -1 1 1 1 1 -1 1 -1 1 1 1 1 1 1 -1 1 -1 -1 1 -1 -1 -1 1 1 1 -1 1 1 -1 -1 1 -1 -1 1 1 -1 1 -

1 -1 -1 1 1 1 1 1 -1 1 1 1 1 1 1 1 -1 1 1 -1 1 -1 -1 -1 -1 1 1 1 -1 1 -1 -1 -1 -1 -1 1 1 -1 -1 1 1 1 -1 -1 -1 -1 -1 1 1 1

1 -1 1 -1 -1 1 -1 1 1 1 -1 -1 -1 1 -1 -1 1 -1 1 1 -1 -1 -1 1 -1 -1 -1 -1 -1 1 1 -1 1 -1 1 1 1 1 1 1 1 1 -1 -1 -1 1 -1 -1

1 1 1 1 -1 1 -1 -1 1 1 1 1 1 -1 -1 1 1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 1 1 1 1 1 -1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 1 -1 1 1

-1 -1 -1 -1 1 -1 -1 -1 1 1 -1 1 -1 1 1 -1 1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 1 -1 -1 1 1 -1 1 1 -1 -1 1 1 -1 1 -1 1 -1

-1 1 1 -1 1 -1 -1 1 1 1 -1 -1 -1 1 1 1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 1 1 1 -1 1 -1 -1 -1 -1 1 1 -1 1 1 1 -1 1 1 -1 1 -1 1 1

-1 -1 1 1 1 1 1 -1 1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 -1 1 1 -1 -1 1 -1 -1 -1 -1 -1 1 -1 -1 1 1 1 1 -1 1 1 1 -1 1 -1 -1 -1 -1

1 -1 1 -1 1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 1 -1 1 -1 1 -1 1 -1 1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 1 1 -1 1 1 -1 -1 1 -1 -1 -1

1 1 -1 -1 1 -1 -1 1 1 -1 1 -1 -1 1 -1 -1 1 1 -1 -1 1 1 1 -1 -1 -1 -1 1 -1 1 -1 1 -1 -1 1 1 -1 1 1 -1 1 -1 -1 1 1 -1 -1 -1

-1 -1 -1 -1 1 1 -1 1 -1 -1 -1 1 -1 -1 1 1 1 1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1 -1 1 1 -1 -1 1 1 -1 -1 1 1 1 -1 1 1 1 -1 -

1 1 1 -1 1 -1 -1 -1 -1 -1 1 1 1 1 1 1 1 -1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 -1 -1 -1 1 1 1 -1 -1 -1 1 -1 1 1 -1 -1 -1 1 1 -

1 1 -1 -1 1 -1 1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 1 1 1 1 1 -1 1 1 1 1 1 -1 1 -1 1 -1 1 1 1 -1 1 -1 1 -1 -1 1 1 -1 1 1 -1 1 1

1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 -1 1 -1 -1 1 -1 -1 -1 1 -1 1 -1 1 1 1 1 -1 1

1 1 1 1 1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 1 1 -1 1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 1 -1 1 -1 1 1 1 -1 1 -1 -1 -1 1 -1 -1 -1 -1

1 1 1 1 -1 1 -1 1 1 -1 -1 -1 1 -1 1 -1 1 -1 1 1 -1 1 1 -1 -1 1 1 1 -1 1 1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 1 1 -1 -1 1

1 -1 1 -1 -1 -1 -1 1 -1 1 1 1 1 1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1

Sample Output 6

493

Q21)

Eveyone remember standing for queue in school assembly. One day few students were late in the morning assembly, so they just went and stood at the end of the queue. But their Principal was strict and made them stand in ascending order to their height. Given an array of N elements representing the shortest person with 1 and tallest with N. He thought to arrange queue by following a trick,Trick was simple: pick a student and send him/her at last or first and to arrange according to height. But he needs your help to do so. Cost is increased by one if one student is sent at first or last.

Input Format

First line consists of T test cases. First line of test case consists of N. Second line of every test case consists of N elements.

Constraints

1<=T<=100 1<=N<=10^5

Output Format

Single line output, print the minimum cost to do so. Sample Input 0

2

3

2 1 3

4

4 3 1 2

Sample Output 0

1

2

Sample Input 1

1

8

4 3 4 2 8 3 4 1

Sample Output 1

5

Sample Input 2

1

7

4 3 4 2 7 3 1

Sample Output 2

5

Sample Input 3

1

10

8 3 8 3 8 3 8 3 1 2

Sample Output 3

8

Q22)

Given n non-negative integers in array representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining. For example: Input: 3 2 0 2 Output: 2 Structure is like below | | |\_| We can trap 2 units of water in the middle gap.

Another example Input Format

The first line of input contains an integer T denoting the number of test cases. The description of T test cases follows. Each test case contains an integer N followed by N numbers to be stored in array.

Constraints

1<=T<=100, 3<=N<=100, 0<=Arr[i]<10

Output Format

Print trap units of water in the middle gap Sample Input 0

2

4

7 4 0 9

3

6 9 9

Sample Output 0

10

0

Sample Input 1

1

6

3 0 0 2 0 4

Sample Output 1

10

Sample Input 2

2

3

2 0 2

12

0 1 0 2 1 0 1 3 2 1 2 1

Sample Output 2

2

6

Q23)

Given a string, recursively remove adjacent duplicate characters from string. The output string should not have any adjacent duplicates

Input Format

The first line contains an integer 'T' denoting the total number of test cases. In each test cases, a string will be inserted.

Constraints

1<=T<=31 1<=length(string)<=100

Output Format

In each seperate line the modified string should be output. if output string is empty print "Empty" Sample Input 0

1

raghuuhga Sample Output 0 r

Sample Input 1

1

engineering Sample Output 1 enginring Sample Input 2

1

caaabbbaacdddd Sample Output 2 Empty

Sample Input 3

3

geeksforgeeg acaaabbbacdddd azxxzy

Sample Output 3 gksfor

acac ay

Q24)

All DNA is composed of a series of nucleotides abbreviated as A, C, G, and T, for example: "ACGAATTCCG". When studying DNA, it is sometimes useful to identify repeated sequences within the DNA.

Write a function to find all the 10-letter-long sequences (substrings) that occur more than once in a DNA molecule. For example,

Given s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT", Return: ["AAAAACCCCC", "CCCCCAAAAA"]. <\pre>

Input Format

single line stringwith the combinations of A,C,G,T only Constraints

--

Output Format

repeted 10 size string's as an output Sample Input 0

AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT

Sample Output 0 [AAAAACCCCC, CCCCCAAAAA]

Sample Input 1 AAAAACCCCCAAAAACCCCCCAAAAAGGGTTTCCCAAAAACCCCCCAAAAAG

Sample Output 1

[AAAAACCCCC, CCCCCAAAAA, CCCAAAAACC, CCAAAAACCC, CAAAAACCCC, AAAACCCCCC, AAACCCCCCA, AACCCCCCAA, ACCCCCCAAA, CCCCCCAAAA, CCCCAAAAAG]

Sample Input 2 AAAAACCCCCAAAAGGGTTTCCCAAAAACCCCCCAAAAAG

Sample Output 2 [AAAAACCCCC]

Sample Input 3 AAAAACCCCCAAAAGGGTTTCCCAAACCAAAAAG

Sample Output 3 []

Sample Input 4 AAAAACCCCCAAAAACCCCCCAAAAAGGGTTTAAAAACCCCCAAAAACCCCCCAAAAAGGGTTT

Sample Output 4

[AAAAACCCCC, CCCCCAAAAA, AAAACCCCCA, AAACCCCCAA, AACCCCCAAA, ACCCCCAAAA, CCCCAAAAAC, CCCAAAAACC, CCAAAAACCC, CAAAAACCCC, AAAACCCCCC, AAACCCCCCA, AACCCCCCAA, ACCCCCCAAA, CCCCCCAAAA, CCCCAAAAAG, CCCAAAAAGG, CCAAAAAGGG, CAAAAAGGGT, AAAAAGGGTT, AAAAGGGTTT]

Q25)

You are in a party of N people, where only one person is known to everyone. Such a person may be

present in the party, if yes, (s)he doesn’t know anyone in the party. Your task is to find the stranger (celebrity) in party. You will be given a square matrix M where if an element of row i and column j is set to 1 it means there is an edge from ith person to jth person. An edge represent the relation that i th person knows j th person. You need to complete the function getId which finds the id of the celebrity if present else return -1. The function getId takes two arguments the square matrix M and its size n.

Example: Input (To be used only for expected output) 1 3 0 1 0 0 0 0 0 1 0

Output 1

Explanation For the above test case the matrix will look like 0 1 0 0 0 0 0 1 0 Here the celebrity is the person with index 1 ie id 1

Input Format

Input: The first line of input contains an element T denoting the No of test cases. Then T test cases follow. Each test case consist of 2 lines. The first line of each test case contains a number denoting the size of the matrix M. Then in the next line are space separated values of the matrix M.

Constraints

Constraints: 1<=T<=60 1<=N<=40 0<=M[][]<=1

Output Format

Output: For each test case output will be the id of the celebrity if present (0 based index). Else -1 will be printed.

Sample Input 0

1

3

0 1 0

0 0 0

0 1 0

Sample Output 0

1

Sample Input 1

1

4

1 0 0 1

0 1 0 1

0 0 0 0

1 1 1 1

Sample Output 1

-1

Sample Input 2

1

4

1 0 1 1

0 1 1 1

0 0 0 0

1 1 1 1

Sample Output 2

2

Sample Input 3

1

4

1 1 1 1

1 1 1 1

0 0 0 0

0 0 0 0

Sample Output 3

-1

Sample Input 4

1

4

1 1 1 1

1 1 1 1

0 0 0 0

0 0 1 0

Sample Output 4

2

Sample Input 5

1

4

0 0 0 0

1 1 1 1

1 1 1 1

1 1 1 1

Sample Output 5

0

Sample Input 6

1

3

0 0 0

1 0 1

0 1 0

Sample Output 6

-1

Sample Input 7

1

5

0 0 0 0 0

1 0 1 0 1

0 0 0 0 0

1 1 1 1 1

1 1 1 1 1

Sample Output 7

-1

Sample Input 8

1

4

1 0 1 0

0 0 1 1

1 0 0 0

1 1 1 1

Sample Output 8

-1

1. Max Element in Array

Find the maximum element from the given array of integers.

Input Format

The first line of input contains N - the size of the array and the second line contains the elements of the array.

Output Format

Print the maximum element of the given array.

Constraints

1 <= N <= 103

-109 <= ar[i] <= 109

Example Input 5

-2 -19 8 15 4

Output 15

1. Number Distribution

Print the count of the occurrences of positive integers, negative integers, and zeroes in the given array.

Input Format

The first line of the input contains an integer N - size of the array, second line of input contains an array elements of the array.

Output Format

Print the frequencies of zeroes, positives elements and negative elements.

Constraints

1 <= N <= 104

-103 <= arr[i] <= 103

Example Input 10

120 0 -9 89 68 -982 91 -54 -12 -139

Output 1 4 5

1. Reverse Array

Print the array in reverse order.

Note:

Try solving this using recursion. Do not use any inbuilt functions / libraries for your main logic.

Input Format

The first line of input contains N - the size of the array and the second line contains the elements of the array.

Output Format

Print the given array in reverse order.

Constraints

1 <= N <= 100

0 <= ar[i] <= 1000

Example Input 5

2 19 8 15 4

Output

4 15 8 19 2

1. Odd and Even Sum

Given an array of size N. Print the sum of odd and even numbers separated by a space.

Input Format

The first line of input contains N - the size of the array and the second line contains elements of the array.

Output Format

Print the sum of odd elements followed by sum of even elements.

Constraints

1 <= N <= 103

1 <= ar[i] <= 106

Example Input 5

4 6 9 2 5

Output

14 12

1. Mean Median Mode

Given a sorted array A, containing N integers. Calculate and print their Mean, Median and Mode.

1. For both the Mean and Median, display the values with precision up to two decimal places.
2. Print the first Mode that you encounter from the left hand side.

Input Format

First line of input contains integer N - the size of the array. Second line of input contains N integers - elements of the array A.

Output Format

Print Mean, Median and Mode, separated by spaces.

Constraints 1 <= N <=104

0 <= A[i] <=100

Example Input 8

1 2 3 4 5 5 6 6

Output

4.00 4.50 5

1. The Missing Number

Find the missing number in the given list of integers. The list contains 1 to 100 integers but one of the integer is missing. There are no duplicates in the list.

Input Format

Input contains a list of 99 integers.

Output Format

Print the missing number in the list.

Constraints

1 <= L[i] <= 100

Example Input

12 15 9 1 71 77 81 29 70 19 11 83 56 2 57 53 68 99 82 100 22 10 51 40 34 98 80

38 39 89 94 4 26 64 45 8 90 24 93 33 21 7 49 88 5 28 55 67 65 50 32 58 6 37 46

42 20 44 41 3 78 76 73 16 31 85 91 54 60 47 97 43 84 17 35 69 13 30 61 79 72

48 23 66 52 27 62 87 63 18 75 96 14 86 95 74 25 59 36

Output 92

1. Find Duplicate Number in Array

Find a duplicate element in the given array of integers. There will be only a single duplicate element in the array.

Note:

Do not use any inbuilt functions / libraries for your main logic Input Format

The first line of input contains the size of the array - N and the second line contains the elements of the array, there will be only a single duplicate element in the array.

Output Format

Print the duplicate element from the given array. Constraints

2 <= N <= 100

0 <= ar[i] <= 109

Example Input

6

5 4 10 9 21 10

Output 10

1. First and Last

You are given an array A of size N, containing integers. Your task is to find the first and last occurrences of a given element X in the array A and print them.

Input Format

The input consists of three lines. The first line contains a single integer N - the size of the array. The second line contains N integers separated by a space, representing the elements of the array A. The third line contains a single integer X.

Output Format

Print the indexes of the first and last occurrences separated by a space. Constraints

1 <= N <= 103

1 <= A[i] <= 105

X ∈ A Example Input

10

1 3 5 7 9 11 3 13 15 3

3

Output 1 9

1. Unique Elements

Print unique elements of the array in the same order as they appear in the input. Note:

Do not use any inbuilt functions / libraries for your main logic. Input Format

The first line of input contains the size of the array - N and the second line

contains the elements of the array. Output Format

Print unique elements from the given array. Constraints

1 <= N <= 100

0 <= ar[i] <= 109

Example Input

7

5 4 10 9 21 4 10

Output 5 9 21

1. Merge Two Sorted Arrays

You are given two sorted integer arrays A and B of size N and M respectively. Print the entire data in sorted order.

Input Format

First line of input contains N - the size of the array. The second line contains N integers - the elements of the first array. The third line contains M - the size of the second array. The fourth line contains M integers - the elements of the second array.

Output Format

For each test case, print the entire data in sorted order with each element separated by a space, on a new line.

Constraints

1 <= N <= 103

1 <= M <= 103

-105 <= A[i], B[i] <= 105

Example Input

7

1 1 5 8 10 12 15

5

-1 2 4 5 7

Output

-1 1 1 2 4 5 5 7 8 10 12 15

1. Print Array A Using B

Given two arrays, A and B, for each index 'i' in array B, print the corresponding element from array A if B[i] is within the range of A, otherwise print -1.

Input Format

The first line of input contains an integer N - size of array A. The Second line of input contains elements of array A. The Third line of input contains an integer M

- size of array B. The Fourth line of input contains elements of array B. Output Format

Print the values of array A for every index 'i' in B i.e. A[B[i]] if B[i] is in the range, otherwise print -1.

Constraints

1 <= N <= 100

1 <= A[k] <= 1000

1 <= M <= 100

0 <= B[i] <= 1000

Example Input

5

100 200 400 800 1000

6

4 2 0 6 10 0

Output

1000 400 100 -1 -1 100

Explanation

B[0] is 4, A[B[0]] => A[4] = 1000

B[1] is 2, A[B[1]] => A[2] = 400

B[2] is 0, A[B[2]] => A[0] = 100

B[3] is 6, size of array A is 5, any index >= 5 is an invalid index, so print -1. B[4] is 10, size of array A is 5, any index >= 5 is an invalid index, so print -1. B[5] is 0, A[B[5]] => A[0] = 100

1. Gauntlets

You have a collection of N gauntlets, each with a specific color represented by A[i]. Your goal is to maximize the number of pairs by repeatedly pairing up gauntlets of the same color. Determine the maximum number of pairs that can be formed.

Input Format

The first line of input contains an integer N. The second line of input contains an array of size N.

Output Format

For the given input, print a single line representing the answer. Constraints

1 ≤ N ≤ 102

1 ≤ Ai ≤ 103 Example Input

6

4 1 7 4 1 4

Output

2

Explanation

You can do the operation twice as follows.

Choose two gauntlets with the color 1 and pair them. Choose two gauntlets with the color 4 and pair them.

Then, you will be left with one sock with the color 4 and another with the color 7, so you can no longer do the operation. There is no way to do the operation three or more times, so you should print 2.

1. Max Altitude

Imagine a pilot starting the flight from the ground and flying over a series of different points at different heights. You are given an array, where A[i] represents heights.

Currently, if the pilot is at altitude X at ith point, and if he wants to reach (i+1)th point, his altitude will become X+A[i].

The pilot starts at altitude 0 and wants to find the highest point he can reach during the entire journey. Your task is to print the highest altitude the pilot reaches.

Input Format

The first line of input contains an integer N. The second line of input contains N space-separated integers.

Output Format

Print the highest altitude the pilot can reach.

Constraints

1 <= N <= 1000

-1000 <= A[i] <= 1000

Example Input 5

-5 1 5 0 -7

Output 1

1. Longest Contiguous 1's

Given an array of elements containing 0's and 1's. You have to find the length of longest contiguous 1's.

Input Format

First line of input contains N - size of the array. The next line contains the N integers of array A.

Output Format

Print the length of longest contiguous 1's.

Constraints 1 <= N <= 1000

Example Input 10

1 0 0 1 0 1 1 1 1 0

Output 4

1. Is Bitonic Sequence

Given an array of integers A, print true if and only if it is a valid array. A is a valid array if and only if there exists some i with 0 < i < A.length - 1 such that: A[0] < A[1] < ... < A[i - 1] < A[i] > A[i + 1] > ... > A[A.length - 1].

Input Format

The first line of the input contains N. Second line of input contains an array of size N.

Output Format

Print true if and only if it is a valid array, otherwise print false. Constraints

3 ≤ N ≤ 104

0 ≤ Ai ≤ 104 Example Input

4

0 3 2 1

Output true Explanation

ar = [0, 3, 2, 1]

idx = 0 1 2 3

So if we take i=1, then we have ar[0] < ar[1] > ar[2] > ar[3] 0 < 3 > 2 > 1

1. Max Min Partition

Given an array D of positive integers, your goal is to divide D into two separate arrays, E and F, under the following conditions:

Each element in array D must belong to either array E or array F Both arrays E and F are non-empty

The objective is to minimize the partition's value, calculated as the absolute difference between the maximum value of array E (denoted as max(E)) and the minimum value of array F (denoted as min(F))

Print the resulting integer that represents the value of this partition. Input Format

The first line of input contains N. The second line of input contains an array of size N.

Output Format

Print the minimized partition value.

Constraints

2 ≤ N ≤ 104

1 ≤ Di ≤ 109 Example Input 6

7 1 14 16 30 4

Output 2

Explanation

We can partition the array D into E = [7, 1, 14, 4] and F = [16, 30].

The maximum element of the array E is equal to 14. The minimum element of the array F is equal to 16.

The value of the partition is |14 - 16| = 2. It can be proven that 2 is the minimum value out of all the partitions.

1. Left Sum and Right Sum

Given an array A of size N. Construct an array B, such that B[i] is calculated as follows: Find leftSum => sum of elements to the left of index i in array A; if none,

use 0.

Find rightSum => sum of elements to the right of index i in array A; if none, use 0. B[i] = | leftSum - rightSum | Your task is to simply print the B array.

Input Format

The first line of input contains the N - size of the array. The next line contains N integers - the elements of array A.

Output Format

Print the elements of the B array separated by space.

Constraints

1 <= N <= 103

0 <= arr[i] <= 100000

Example Input 3

6 7 7

Output 14 1 13

Explanation At index 0:

LeftSum = 0, RightSum = 14

B[0] = | LeftSum - RightSum | = 14.

At index 1:

LeftSum = 6, RightSum = 7

B[1] = | LeftSum - RightSum | = 1.

At index 2:

LeftSum = 13, RightSum = 0

B[2] = | LeftSum - RightSum | = 13.

1. Alternate Seating

You are given an integer N, denoting the number of people who need to be seated, and a list of M seats, where 0 represents a vacant seat and 1 represents an already occupied seat. Find whether all N people can find a seat, provided that no two people can sit next to each other.

Input Format

The first line of the input contains N denoting the number of people. The second line of input contains M denoting the number of seats. The third line of input contains the seats.

Output Format

If all N people can find seats, print YES otherwise NO.

Constraints

1 ≤ N ≤ 105

1 ≤ M ≤ 105

Ai ∈ {0, 1}

Example Input

2

7

0 0 1 0 0 0 1

Output YES

Explanation

The two people can sit at index 0 and 4.

1. Ternary Array

Given an array A of size N, find the minimum cost to convert it to a ternary array

B. A ternary array can only have 0 or 1 or 2. After conversion, ensure that A[i] != B[i]. The cost of converting A[i] to B[i] is | A[i] - B[i] |.

Input Format

The first line of input contains a single integer N - the size of the array and the second line contains array elements.

Output Format

Print the minimum cost to convert array A to B. Constraints

1 <= N <= 10000

-100000 <= A[i] <= 100000

Example Input

5

1 -1 2 0 5

Output 7

Explanation

Given A = {1, -1, 2, 0, 5} can be converted to B = {2, 0, 1, 1, 2}, with a cost of

|1-2| + |-1-0| + |2-1| + |0-1| + |5-2| = 1 + 1 + 1 + 1 + 3 = 7.

1. Three Equal Parts

Given an array of integers A, print true if we can partition the array into three non-empty subarrays with equal sums.

Input Format

The first line of the input contains an integer N. Second line of input contains an array of size N.

Output Format

Print true if we can partition the array, otherwise false.

Constraints

3 ≤ N ≤ 104

-104 ≤ Ai ≤ 104

Example Input 10

3 3 6 5 -2 2 5 1 -9 4

Output true

Explanation

(3 + 3) = (6) = (5 - 2 + 2 + 5 + 1 - 9 + 4) = 6.

1. List Operations

You are tasked with implementing a program that manipulates an empty list based on a series of commands.

Input Format

The first line of input contains an integer N, indicating the number of commands to follow. The next N lines contains any of the following commands:

append X: Appends the integer X to the end of the list.

count X: Count the number of occurrences of the integer X in the list. reverse: Reverses the order of elements in the list.

insert Pos X: Inserts the integer X at the position Pos in the list. sort: Sorts the elements of the list in ascending order.

index X: Gives the index of the first occurrence of the integer X in the list, or -1 if X is not found. length: Gives the length of the list.

extend: Extends the list by appending it's current elements to itself. Output Format

For count, index, and length command, print the result. For the remaining commands, print the updated list separated by spaces.

Constraints 1 <= N <= 50

1 <= X <= 100

0 <= Pos < length of the list

Example Input 10

append 13

append 7

insert 1 6 extend index 2 reverse index 7 length sort count 6

Output 13

13 7

13 6 7

13 6 7 13 6 7

-1

7 6 13 7 6 13

0

6

6 6 7 7 13 13

2

1. Implement Bubble Sort

Given an array of size N, implement Bubble Sort.

Input Format

The first line of input contains an integer N - the size of an array. The second line contains the elements of the array.

Output Format

For each iteration of Bubble Sort, print the array elements.

Constraints

1 <= N <= 20

1 <= A[i] <= 103

Example Input

6

5 8 10 15 3 6

Output

5 8 10 3 6 15

5 8 3 6 10 15

5 3 6 8 10 15

3 5 6 8 10 15

3 5 6 8 10 15

1. Implement Selection Sort

Given an array of size N having unique elements, implement Selection Sort. Note: Implement Selection Sort by selecting smallest element at every step.

Input Format

The first line of input contains an integer N - the size of an array. The second line contains the elements of the array.

Output Format

For each iteration of Selection Sort, print the array elements.

Constraints 1 <= N <= 20

1 <= A[i] <= 103

Example Input 6

5 8 10 15 3 6

Output

3 8 10 15 5 6

3 5 10 15 8 6

3 5 6 15 8 10

3 5 6 8 15 10

3 5 6 8 10 15

1. Implement Insertion Sort

Given an array of size N, implement Insertion Sort.

Input Format

The first line of input contains an integer N - the size of an array. The second line contains the elements of the array.

Output Format

For each iteration of Insertion Sort, print the array elements. Constraints

1 <= N <= 20

1 <= A[i] <= 103

Example Input

5

8 7 1 2 4

Output 7 8 1 2 4

1 7 8 2 4

1 2 7 8 4

1 2 4 7 8

1. Implement Shell Sort

Given an array of size N, implement Shell Sort.

Note: Initially gap=N/2, for each iteration, gap reduces by half. Input Format

The first line of input contains an integer N - the size of an array. The second line contains the elements of the array.

Output Format

For each iteration of Shell Sort, print the array elements. Constraints

1 <= N <= 20

1 <= A[i] <= 103

Example Input

8

4 3 12 1 13 9 5 6

Output

4 3 5 1 13 9 12 6

4 1 5 3 12 6 13 9

1 3 4 5 6 9 12 13

Explanation

Initially gap = 4, apply Insertion Sort for elements at distance 4, you will get [4 3

5 1 13 9 12 6]

For gap = 2, again apply Insertion Sort for elements at distance 2, you will get [4 1 5 3 12 6 13 9]

For gap = 1, apply Insertion Sort for elements at distance 1, you will get [1 3 4 5

6 9 12 13]

1. Implement Merge Sort

Given an array of size N, implement Merge sort.

Input Format

The first line of input contains an integer N - the size of an array. The second line contains the elements of the array.

Output Format

For each merge call of Merge Sort, print the array elements.

Constraints 1 <= N <= 20

1 <= A[i] <= 103

Example Input 6

5 1 3 15 10 4

Output

1 5 3 15 10 4

1 3 5 15 10 4

1 3 5 10 15 4

1 3 5 4 10 15

1 3 4 5 10 15

1. Implement Linear Search

Given an array of integers, search a given key in the array using linear search. Note: Do not use any inbuilt functions / libraries for your main logic.

Input Format

The first line of input contains two integers N and K. N is the size of the array and K is the key. The second line contains the elements of the array.

Output Format

If the key is found, print the index of the array, otherwise print -1.

Constraints

1 <= N <= 102

0 <= ar[i] <= 109

Example Input 5 15

-2 -19 8 15 4

Output 3

1. Implement Binary Search

Given a sorted array A of size N, along with an integer target K, implement Binary Search to find the target K in the given array.

Input Format

The first line of input contains an integer N - the size of an array and target K. The second line contains the elements of the array.

Output Format

For each iteration of Binary Search, print the values of low, high, and mid. At the end, if the target K is found in the array, print "True" otherwise, print "False".

Constraints 1 <= N <= 20

1 <= A[i] <= 103

Example Input 1

9 12

1 4 6 7 10 11 12 20 23

Output 1

0 8 4

5 8 6

True Input 2

10 21

3 5 8 11 15 17 19 23 26 30

Output 2

0 9 4

5 9 7

5 6 5

6 6 6

False

1. Matrix Row Sum

Given a matrix of size N x M, print row-wise sum, separated by a newline.

Note:

Try to solve this without declaring / storing the matrix.

Input Format

The first line of input contains N, M - the size of the matrix, followed by N lines each containing M integers

- elements of the matrix.

Output Format

Print the row-wise sum of the matrix, separated by a newline.

Constraints

1 <= N, M <= 100

-100 <= ar[i] <= 100

Example Input 2 3

5 -1 3

19 8 -5

Output 7

22

1. Matrix Column Sum

Given a matrix of size N x M, print column-wise sum, separated by a newline.

Input Format

The first line of input contains N, M - the size of the matrix, followed by N lines each containing M integers

- elements of the matrix.

Output Format

Print the column-wise sum of the matrix, separated by newline.

Constraints

1 <= N, M <= 100

-109 <= ar[i] <= 109

Example Input 2 2

5 -1

19 8

Output 24

7

1. Submatrix Sum

Given a matrix of size N x N and four integers (i, j, k, l), calculate the sum of the sub-matrix from (i, j) to (k, l).

Input Format

The first line of input contains an integer N. Second line of input contains four integers i, j, k and l. It's followed by N lines each containing N integers - elements of the matrix.

Output Format

Print the sum of the sub-matrix from (i, j) to (k,l).

Constraints

1 <= N <= 100

0 <= i <= k < N

0 <= j <= l < N

1 <= ar[i][j] <= 100

Example Input 3

1 1 2 2

1 2 3

4 5 6

7 8 9

Output 28

Explanation

Sum of elements from {1,1} to {2, 2} is (5+6+8+9) = 28.

1. Lower Triangle

Given a square matrix of size N × N, find the sum of its lower triangle elements.

Input Format

The first line of input contains N - the size of the matrix. It is followed by N lines each containing N integers - elements of the matrix.

Output Format

Print the sum of the lower triangle of the matrix. Constraints

1 <= N <= 100

-105 <= ar[i] <= 105

Example Input

3

5 9 -2

-3 4 1

2 6 1

Output 15

Explanation

The sum of the lower triangle matrix is 5 - 3 + 4 + 2 + 6 + 1 = 15.

1. Sum of Two Matrices

Given two matrices A and B each of size N x M, print the sum of the matrices (A

+ B).

Note:

Try solving it by declaring only a single matrix. Input Format

The first line of input contains N, M - the size of the matrices. It's followed by 2\*N lines, each containing M integers - elements of the matrices. The first N lines are for matrix A and the next N lines are for matrix B.

Output Format

Print the sum of the 2 given matrices (A + B). Constraints

1 <= N, M <= 100

-109 <= ar[i] <= 109

Example Input

2 3

5 -1 3

19 8 4

4 5 -6

1 -2 12

Output 9 4 -3

20 6 16

1. Interchange Diagonals

Given a matrix A of size NxN, interchange the diagonal elements from top to bottom and print the resultant matrix.

Input Format

First line of input contains N - the size of the matrix. It is followed by N lines each containing N integers - elements of the matrix.

Output Format

Print the matrix after interchanging the diagonals.

Constraints

1 <= N <= 100

1 <= A[i][j] <= 104

Example Input 4

5 6 7 8

13 14 15 16

1 2 3 4

9 10 11 12

Output 8 6 7 5

13 15 14 16

1 3 2 4

12 10 11 9

1. Count the Elements - Matrix in Array

You are given a matrix and an array. For each row of the given matrix, for each element of that row, check if it is present in the given array. Print the count of elements present for every row.

Input Format

The first line of input contains N, M - the size of the matrix. It is followed by N lines each containing M integers - elements of the matrix. The next line of the input contains the A - the size of the array. The next line of the input contains the

array elements.

Output Format

For each row, print the count of the number of elements present, separated by a new line.

Constraints

1 <= N, M, A <= 100

-100 <= mat[i][j], ar[i] <= 100

Example Input 3 4

5 9 -2 2

-3 4 1 9

2 -2 1 -2

5

5 1 -2 2 6

Output

3

1

4

Explanation

The first row of the matrix is (5 9 -2 2), out of this, 3 elements (5 -2 2), except 9,

are present in the given array (5 1 -2 2 6)

The second row of the matrix is (-3 4 1 9), out of this, only 1 is present in the given array (5 1 -2 2 6)

The third row of the matrix is (2 -2 1 -2), all the 4 elements are present in the given array (5 1 -2 2 6)

1. Matrix Zig-Zag Traversal

Given a matrix of size N x M, print the matrix in zig-zag order. Refer example for more details.

Input Format

The first line of input contains N, M - the size of the matrix. It is followed by N lines each containing M integers - elements of the matrix.

Output Format

Print the matrix elements in zig-zag order.

Constraints

1 <= N, M <= 100

-106 <= mat[i][j] <= 106

Example Input 3 3

5 9 -2

-3 4 1

2 6 1

Output

5 9 -2 1 4 -3 2 6 1

1. Transpose Matrix

Given a matrix of size N x M, print the transpose of the matrix.

Input Format

The first line of input contains N, M - the size of the matrix. It is followed by N lines each containing M integers - elements of the matrix.

Output Format

Print the transpose of the given matrix.

Constraints

1 <= N, M <= 100

-109 <= ar[i] <= 109

Example Input 2 2

5 -1

19 8

Output 5 19

-1 8

1. Sparse Matrix

Given a matrix of size N x M, print whether it is a sparse matrix or not. Please note that if a matrix contains 0 in more than half of its cells, then it is called a sparse matrix.

Input Format

The first line of input contains N, M - the size of the matrix, followed by N lines each containing M integers

- elements of the matrix.

Output Format

Print "Yes" if the given matrix is a sparse matrix, otherwise print "No". Constraints

1 <= N, M <= 100

0 <= ar[i] <= 109

Example Input

2 3

5 0 0

0 8 0

Output Yes

1. Super One

Given a matrix of 0’s and 1’s, check if there exists a Super One. Please note that

a Super One is a 1, which is surrounded by 0 on all 8 sides. Input Format

The first line of input contains N, M - the size of the matrix. It's followed by N lines each containing M integers - elements of the matrix.

Output Format

Print "Yes" if the matrix contains Super One, otherwise print "No". Constraints

1 <= N, M <= 100

0 <= ar[i] <= 1

Example Input

4 4

1 0 0 0

0 0 0 1

0 1 0 0

0 0 0 0

Output Yes

Explanation

There's one occurrence of Super One in the matrix at [2,1]. Value 1 at index [0,0] and Value 1 at index [1,3] are not Super One's because they are not surrounded

by eight 0's.

1. Image Flip

You are given an N x M binary matrix called "image". You need to perform the following operations on the matrix (in order) and return the resulting image:

Flip the image horizontally: This involves reversing the order of elements in each row of the matrix. For example, [1,0,1,0,0,0] becomes [0,0,0,1,0,1]

Invert the image: This involves replacing 0s with 1s and 1s with 0s in the entire matrix. For example, [0,0,0,1,0,1] becomes [1,1,1,0,1,0]

Input Format

Line of input contains N - number of rows and M - number of columns. The next N lines contains M integers each denoting the elements of the matrix image.

Output Format

You have to print the resultant matrix image.

Constraints 1 <= N <=100

1 <= M <=100

Example Input 2 2

1 0

0 1

Output 1 0

0 1

1. Local Max in Matrix

Given an integer matrix C with dimensions N × N. Construct a new integer matrix D of size (N - 2) × (N - 2) such that each element D[i][j] represents the maximum value within a 3 × 3 submatrix of C, where the center of the submatrix is located at row i + 1 and column j + 1 in matrix C. We aim to identify the highest value within every continuous 3 × 3 submatrix within C. Print the resulting matrix D.

Input Format

The first line of input contains an integer N. For the next N lines, each line contains N elements separated by space.

Output Format

Print the generated matrix.

Constraints

3 ≤ N ≤ 100

-1000 ≤ Cij ≤ 1000

Example

Input 4

12 9 8 40

5 20 2 6

8 14 6 30

6 2 25 2

Output 20 40

25 30

1. Zero Row and Zero Column

Given a matrix A of size N x M. Elements of the matrix are either 0 or 1. If A[i][j] = 0, set all the elements in the ith row and jth column to 0. Print the resultant matrix.

Input Format

The first line of input contains N, M - the size of the matrix A. It is followed by N lines each containing M integers - elements of the matrix.

Output Format

Print the resultant matrix. Constraints

1 <= N, M <= 100

A[i][j] ∈ {0,1}

Example Input

4 5

0 1 1 0 1

1 1 1 1 1

1 1 0 1 1

1 1 1 1 1

Output 0 0 0 0 0

0 1 0 0 1

0 0 0 0 0

0 1 0 0 1

1. Check Bit

Given an integer N, check whether the ith bit is set or not.

Input Format

The first and only line of input contains N and i.

Output Format

Print "true" if the ith bit is set in the given integer N, "false" otherwise.

Constraints

0 <= N <= 109

0 <= i <= 30

Example Input 10 1

Output true

Explanation

The binary form of 10 is `1010`. So, the 1st bit in 10 is set. Note that the LSB bit is referred to as the 0th bit.

1. Triangle Validator

Given the length of 3 sides of a triangle, check whether the triangle is valid or not.

Input Format

The first and only line of input contains three integers A, B, C - Sides of the triangle.

Output Format

Print "Yes" if you can construct a triangle with the given three sides, "No" otherwise.

Constraints

1 <= A, B, C <= 109

Example Input 4 3 5

Output

Yes

1. Area of Rectangle

Print the area of the rectangle, given its length and breadth.

Input Format

The first and only line of input contains two positive integers L - Length of the rectangle and B - Breadth of the rectangle.

Output Format

Print the area of the rectangle.

Constraints

1 <= L, B <=109

Example Input 5 8

Output 40

1. Arithmetic Operators

Given two integers A and B. Print the sum, difference, product, division, and remainder for the corresponding values of A and B.

Input Format

First and only line of input contains two integers - A and B.

Output Format

Print the sum, difference, product, division, and remainder for the corresponding values of A and B. Refer the given example.

Constraints

1 <= A, B <= 1000

Example Input 12 5

Output Sum: 17

Difference: 7

Product: 60

Division: 2

Remainder: 2

1. Arithmetic Progression

For an Arithmetic Progression series with a given first term (a), common difference (d), and an integer N, determine the Nth term of the series.

Input Format

First and only line of input contains 3 variables a, d and N.

Output Format

Print the Nth term of the AP series.

Constraints

1 <= a, d, N <= 103

Example Input 2 1 5

Output 6

1. Compute A power B

Given two integers A and B, compute A power B.

Note: Do not use any inbuilt functions / libraries for your main logic.

Input Format

The first and only line of input contains two positive integers A and B.

Output Format Print A power B.

Constraints

1 <= A <= 100

0 <= B <= 9

Example Input 2 3

Output 8

1. Number Reverse

Given a number N, reverse the number.

Input Format

The first and only line of input contains a integer - N.

Output Format

Print the reversed number.

Constraints

-1018 <= N <= 1018

Example Input 1344

Output 4431

1. Fibonacci Number

For a given positive integer - N, compute Nth Fibonacci number.

Input Format

The first and only line of input contains a positive number - N.

Output Format

Print the Nth fibonacci number.

Constraints 0 <= N <= 20

Example Input 4

Output 3

Explanation

The fibonacci series:

0, 1, 1, 2, 3, 5, 8,......

At 4th position, we have 3.

1. Given a non-negative number - N. Print N!

Input Format

The first and only line of input contains a number - N.

Output Format Print factorial of N.

Constraints 0 <= N <= 10

Example Input 5

Output 120

1. Catalan Number

Given an integer N, generate the Nth Catalan Number.

Input Format

First and only line of input contains a non-negative integer N.

Output Format

Print the Nth Catalan Number.

Constraints 0 <= N <= 10

Example Input 3

Output 5

Explanation

3rd Catalan Number: 6C3 / 4 = 5

1. NcR Basic

Given two numbers N and R, find the value of NCR. Input Format

The first and only line of input contains integers N and R.

Output Format

Print the value of NCR Constraints

1 <= N <= 10

1 <= R <= 10

Example Input 5 3

Output 10

1. Applying Modulus

There are six positive integers A, B, C, D, E, and F, which satisfy A × B × C ≥ D

× E × F. Find the remainder when (A × B × C) - (D × E × F) is divided by (1e9+7). Note:

1. (X × Y) % M = (X % M × Y % M) % M
2. (X - Y) % M = (X % M - Y % M + M) % M

Input Format

The first and only line of input contains six space-separated integers A, B, C, D, E, and F.

Output Format

For the given input, print a single line representing the answer.

Constraints

0 ≤ A, B, C, D, E, F ≤ 1018 A × B × C ≥ D × E × F

Example Input 2 3 5 1 2 4

Output 22

Explanation

Since A × B × C = 2 × 3 × 5 = 30 and D × E × F = 1 × 2 × 4 = 8, we have (A × B

× C) - (D × E × F) = 22. When you divide 22 by (1e9+7), the remainder will be 22.

1. Factorial Hard

Given a non-negative number - N. Print N!

Input Format

The first and only line of input contains a number - N. Output Format

Print factorial of N. Since the result can be very large, print result % 1000000007 Constraints

0 <= N <= 106

Examples Input 1

3

Output 1

6

Input 2

165

Output 2

994387759

1. Minimum Subtraction

Given a number N, find a number X. On subtracting X from N, N-X should be a power of 2. Find the minimum value of X.

Input Format

First and only line of input contains an integer N. Output Format

Print the value X.

Constraints

2 <= N <= 109

Example Input

10

Output 2

Explanation N = 10

If we subtract X = 2 from N = 10, N - X = 8 is a power of 2.

1. Number of Multiples

Given a positive integer - N. Print the number of multiples of 3, 5 between [1, N].

Input Format

The first and only line of input contains a positive integer - N.

Output Format

Print the number of multiples of 3, 5 between [1, N].

Constraints

1 <= N <= 1018

Example Input 12

Output 6

Explanation

Multiples of 3 and 5 in range of 1 to 12 are 3, 5, 6, 9, 10, 12.

1. A B and X

You are given two numbers A and B along with an integer X. In one operation you can do one of the following:

Set A = A + X and B = B - X Set A = A - X and B = B + X

Determine if you can make A and B equal after applying the operation any number of times (possibly zero).

Input Format

The first and only line of input contains three integers A, B, and X separated by a space.

Output Format

For each test case, print YES if you can make A and B equal, otherwise NO.

Constraints

1 <= A, B, X <= 109

Examples Input 1

6 8 1

Output 1

YES

Input 2

15 16 2

Output 2 NO

Explanation

Example 1: The initial values of (A, B) are (6,8). We can perform the following operation. A = A + X = 6 + 1 = 7 and B = B - X = 8 - 1 = 7

A and B are equal, print YES.

Example 2: It can be proven that we cannot make A equal to B using the given operations.

1. Natural Numbers Sum

Given positive integer - N, print the sum of the first N natural numbers.

Input Format

The first and only line of input contains a positive integer - N.

Output Format

Print the sum of the first N natural numbers.

Constraints

1 <= N <= 109

Example Input 4

Output 10

1. Armstrong Number

Given an integer N, check whether it's an Armstrong number or not.

Note that an Armstrong number is a number that is equal to the sum of cubes of its digits.

Input Format

The first and only line of input contains an integer - N.

Output Format

Print "Yes" if the number is Armstrong number, "No" otherwise.

Constraints

0 <= N <= 109

Example Input 153

Output Yes

Explanation

13 + 53 + 33 = 153

1. Narcissistic Numbers

Given an integer N, check whether it is a Narcissistic number or not.

Note that a Narcissistic number is a number that is the sum of its own digits each raised to the power of the number of digits

Input Format

The first and only line of input contains an integer - N.

Output Format

Print "Yes" if the number is Narcissistic number, "No" otherwise.

Constraints

0 <= N <= 106

Example Input 8208

Output Yes

Explanation

84 + 24 + 04 + 84 = 8208

1. Harshad Numbers

Given an integer N, check whether it is a Harshad number or not.

Note that a Harshad number is an integer, that is divisible by the sum of its digits.

Input

The first and only line of input contains a integer - N. Output

Print "Yes" if the number is Harshad number, "No" otherwise.

Constraints 1 <= N <= 109

Example Input

18

Output

Yes

Explanation 18 / (1 + 8) = 2

As 18 is divisible by the sum of its digits, it is a Harshad number.

1. Compound Interest

Write a program to calculate the amount of money accrued from compound interest.

Compound interest is the interest calculated on the initial principal and also on the accumulated interest of previous periods. Given the principal amount (P), the annual interest rate (R), the number of times the interest is compounded per year (N), and the time in years (T), write a program to calculate the final amount (A) after the specified time.

Compound Interest = P(1 + R/N)NT - P

Always consider the floor value of (R/N) while computing Compound Interest. Input Format

The first and only line of input contains 4 integers P, R, N, T separated by spaces. Output Format

Print the Compound Interest for the given values of P, R, N, T. Constraints

1 <= P <= 100

1 <= R <= 10

1 <= N, T <= 5

Example Input

65 6 1 2

Output

3120

1. Quadratic Equation

Given a quadratic equation in the form ax2 + bx + c, (only the values of a, b and c are provided). Find the roots of the equation.

Note: Display the values with precision up to two decimal places, and for imaginary roots of the form (x + iy) or (x - iy), print "Imaginary Roots".

Input Format

First and only line of input contains three integers a, b, c separated by spaces. Output Format

Print the roots of the quadratic equation separated by spaces. Constraints

-1000 <= a, b, c <= 1000

Examples

Input 1

1 -2 1

Output 1

1.00 1.00

Input 2

1 7 12

Output 2

-3.00 -4.00

Input 3

1 1 1

Output 3 Imaginary Roots Explanation

Example 1: Roots are real and equal, which are (1,1) Example 2: Roots are real and distinct, which are (-3,4)

Example 3: Roots are imaginary, which are (-0.5 + i0.866025, -0.5 - i0.866025) restart\_alt

settings save

Score: 0 / 20

00

Minutes

1. Prime or Not

Given a positive integer - N, check whether the number is prime or not.

Input Format

The first and only line of input contains an integer - N.

Output Format

Print "Yes" if the number is prime, "No" otherwise.

Constraints

1 <= N <= 108

Example Input 11

Output Yes

1. Arrange Primes

Given an integer N. Print the count of permutations for the numbers from 1 to N, considering that prime numbers should be placed at positions with prime indices (1 - based indexing). As the result might be a large number, print the output % 1e9 + 7.

Input Format

The first and only line of input contains an integer N.

Output Format

Print the count of permutations.

Constraints

1 ≤ N ≤ 100

Example Input 8

Output 576

1. Fuel Tank

A truck has two fuel tanks: a main tank and an additional tank. The fuel levels in both tanks are represented by two integers: mainTank, which represents the fuel in the main tank in litres, and additionalTank, which represents the fuel in the additional tank in litres.

The truck has a mileage of 10 kilometers per litre. Whenever 5 litres of fuel are consumed from the main tank, if the additional tank has at least 1 litre of fuel, 1 litre of fuel will be transferred from the additional tank to the main tank. Print the maximum distance that can be travelled with the given fuel.

Input Format

The first and only line of input contains two integers mainTank and additionalTank.

Output Format

Print the maximum distance that can be travelled with the given fuel.

Constraints

1. <= mainTank, additionalTank <= 100

Examples Input 1

5 10

Output 1

60

Input 2

1. 2

Output 2

20

Explanation Example 1:

After spending 5 litre of fuel, fuel remaining is (5 - 5 + 1) = 1 litre and the distance travelled is 50km. After spending another 1 litre of fuel, no fuel gets injected in the main tank and the main tank becomes empty. Total distance travelled is 60km.

Example 2:

After spending 2 litre of fuel, the main tank becomes empty. Total distance travelled is 20km.

1. Min Digit

Tom and Jerry are playing a game with an integer N that doesn't contain any zeros in its decimal representation. Tom starts first and, on his turn, he can swap any two digits of the integer that are in different positions. Jerry follows by always removing the last digit of the integer. The game continues in turns until there's only one digit left. Determine the smallest integer Tom can achieve at the end if he plays optimally.

Input Format

The first and only line of input consists an integer N consisting of digits 1 - 9.

Output Format

Print the the smallest integer that Tom can achieve.

Constraints 10 < N < 109

Example Input 132

Output

1

Explanation

Tom can swap 3 and 1, N becomes 312. After that Jerry deletes the last digit, N becomes 31. Then Tom swaps 3 and 1, N becomes 13 and Jerry deletes 3, so the answer is 1.

1. Number of Rectangles

Vikram has a collection of N squares with a side length of 1. Print the number of different rectangles that can be formed using these squares.

Input Format

Print the number of different rectangles that can be formed using these squares.

Output Format

Print the number of different rectangles that can be formed using these squares.

Constraints

1 ≤ N ≤ 100

Example Input 6

Output

8

Explanation

We can create different rectangles of sizes 1×1, 1×2, 1×3, 1×4, 1×5, 1×6, 2×2,

2×3.

1. Point Location

There is a line that goes through the points P1 = (x1,y1) and P2 = (x2,y2). There is

also a point P3 = (x3,y3). Print whether P3 is located on the left or right side of the line or if it touches the line when we are looking from P1 to P2.

Input Format

The first and only line of input contains 6 integers x1, y1, x2, y2, x3, y3 separated by spaces.

Output Format

For given input, print "LEFT", "RIGHT" or "TOUCH". Constraints

-104 ≤ x1, y1, x2, y2, x3, y3 ≤ 104 x1 ≠ x2 or y1 ≠ y2

Examples Input 1

1 1 5 3 2 3

Output 1 LEFT

Input 2

1 1 5 3 4 1

Output 2 RIGHT

Input 3

1 1 5 3 3 2

Output 3 TOUCH

1. Area of Polygon

Print the area of a given polygon. The polygon consists of N vertices (x1, y1), (x2,

y2), …, (xN, yN). The vertices (xi, yi) and (xi+1, yi+1) are adjacent for i = 1, 2, …, N−1, and the

vertices (x1, y1) and (xN, yN) are also adjacent.

Input Format

The first line of input has an integer N: the number of vertices. Next N lines describe the vertices. The ith such line has two integers xi and yi separated by space. You may assume that the polygon is simple, i.e., it does not intersect itself.

Output Format

Print the area of the Polygon.

Constraints

3 ≤ N ≤ 1000

-103 ≤ xi, yi ≤ 103

Example Input

4

1 1

4 2

3 5

1 4

Output

16

1. Filled Rectangle

Given N and M, print the following rectangle pattern of size N × M. See examples for more details.

Input Format

The first line of input contains N and M. Output Format

Print the rectangle pattern. Constraints

1 <= N <= 10

1 <= M <= 10

Example Input

3 5

Output

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

1. Hollow Rectangle

Print a hollow rectangle pattern using '\*'. See the example for more details.

Input Format

The input contains two integers W - width of the rectangle and L - length of the rectangle.

Output Format

For the given integers W and L, print the hollow rectangle pattern.

Constraints 2 <= W <= 50

2 <= L <= 50

Example Input 5 4

Output

\*\*\*\*\*

\* \*

\* \*

\*\*\*\*\*

1. Rectangle Pattern

Print rectangle pattern. See the example for more details.

Input Format

The first and only line of input contains a single integer N.

Output Format

For the given integer, print a rectangle pattern as shown in the example.

Constraints 1 <= N <= 50

Example Input 5

Output 5432\*

543\*1

54\*21

5\*321

\*4321

1. Triangle Pattern

Given a value N, print a right-angled triangle pattern. See examples for more details.

Input Format

First and only line of input contains an integer N.

Output Format

Print the right angled triangle pattern.

Constraints

1 <= N <= 100

Example Input 5

Output

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

1. Floyd Pattern - 1

Print a right-angled triangle pattern using integers. See the example for more details.

Input Format

The first and only line of input contains a single integer N - the size of the triangle.

Output Format

For the given integer, print the right-angled triangle pattern.

Constraints 1 <= N <= 50

Example Input 6

Output

1

2 3

4 5 6

7 8 9 10

11 12 13 14 15

16 17 18 19 20 21

1. Floyd Pattern - 2

Print a right-angled triangle pattern. See the example for more details.

Input Format

The first and only line of input contains a single integer N - the size of the triangle.

Output Format

For the given integer, print the right-angled triangle pattern.

Constraints 1 <= N <= 50

Example Input 5

Output 1

2 6

3 7 10

4 8 11 13

5 9 12 14 15

1. Half Diamond

Print half diamond pattern using '\*'. See the example for more details.

Input Format

The first and only line of input contains a single integer N.

Output Format

For the given integer, print the half-diamond pattern.

Constraints 1 <= N <= 50

Example Input

5

Output

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*

\*\*\*

\*\*

\*

1. Half Diamond with Numbers

Given a value N, print a half diamond pattern with numbers. See examples for more details.

Input Format

First and only line of input contains an integer N.

Output Format

Print the half diamond pattern using numbers.

Constraints

1 <= N <= 100

Example Input 5

Output 1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

1. Inverted Pyramid

Print a hollow half-inverted pyramid pattern using '\*'. See the example for more details.

Input Format

The first and only line of input contains a single integer N.

Output Format

For the given integer, print hollow half-inverted pyramid pattern.

Constraints 1 <= N <= 50

Example Input 5

Output

\* \* \* \* \*

\* \*

\* \*

\* \*

\*

1. Pyramid Pattern

Print pyramid pattern using '\*'. See the example for more details.

Input Format

The first and only line of input contains a single integer N - the size of the pyramid.

Output Format

For the given integer, print the pyramid pattern.

Constraints 1 <= N <= 50

Example Input 5

Output

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

1. Palindromic Pattern

Print a palindromic right-angled triangle pattern using characters. See the example for more details.

Input Format

The first and only line of input contains an integer N - the size of the pattern.

Output Format

For the given integer N, print the palindromic right-angled triangle pattern.

Constraints 1 <= N <= 26

Example Input 5

Output A

A B A

A B C B A

A B C D C B A

A B C D E D C B A

1. Multiplication Table

Print the multiplication table for the given integer - N.

Input Format

First and only line of input contains a single integer N.

Output Format

Print multiplication table for given integer N.

Constraints

-105 <= N <= 105

Example Input 9

Output 9 \* 1 = 9

9 \* 2 = 18

9 \* 3 = 27

9 \* 4 = 36

9 \* 5 = 45

9 \* 6 = 54

9 \* 7 = 63

9 \* 8 = 72

9 \* 9 = 81

9 \* 10 = 90

1. Pascal's Triangle

Given a value N, print the Pascal's Triangle pattern.

Input Format

The first and only line of input contains an integer N.

Output Format

For the given input, print the Pascal's Triangle pattern.

Constraints

1 ≤ N ≤ 30

Example Input 10

Output 1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

1 6 15 20 15 6 1

1 7 21 35 35 21 7 1

1 8 28 56 70 56 28 8 1

1 9 36 84 126 126 84 36 9 1

1. Reverse String

Given a string, reverse the given string in place and then print it.

Note:

Do not use any inbuilt functions / libraries for your main logic.

Input Format

The input contains a string S, consisting of ASCII characters.

Output Format

Print the reversed string.

Constraints

1 <= len(s) <= 100

Example Input smart

Output Trams

1. Odd Even Index

Given a string, print all the letters present at the odd index, followed by the letters present at the even index.

Input Format

The input contains a string S, consisting of ASCII characters.

Output Format

Print letters present at odd index, followed by the letters present at even index.

Constraints

1 <= len(S) <= 100

Example Input afdg5tg

Output fgtad5g

1. Toggle Case

Given a string, toggle the case of each character in the given string.

Input Format

The first and only input line contains a string S, consisting of lowercase and uppercase characters.

Output Format

Print the toggled string.

Constraints

1 <= len(S) <= 100

Example Input HeLLo

Output hEllO

1. Vowels and Consonants

Given a string, print the count of vowels and consonants in the string.

Input Format

Input contains a string S, consisting of lowercase and uppercase characters.

Output Format

Print the count of vowels and consonants in the given string, separated by space.

Constraints 1 <= len(S) <= 100

Example Input abxbbiAaspw

Output 4 7

1. Letter Coverage

Given a string, check if it contains all the letters of the alphabet.

Input Format

Input contains a string S, consisting of lowercase and uppercase characters.

Output Format

Print "Yes" if the string contains all the letters of the alphabet, and "No" otherwise.

Constraints

1 <= len(S) <= 100

Example Input askhtwsflkqwertYuioPasdfghjklZxcvbnm

Output Yes

1. Check the Square String

Given a string, you have to check whether it is a square string. A string is said to be square if it is some string written twice in a row. For example, the strings "cc", "zazazaza", and "papa" are square. But "abc", and "abaaaabb" are not square.

Input Format

Input contains a string S.

Output Format

Print "Yes" if the string S is square. Otherwise print "No".

Constraints

1 <= |S| <= 103 'a' <= S[i] <= 'z'

Example Input

aabaabaabaab

Output Yes

Explanation

The given string can be formed by adding the string "aabaab" 2 times.

1. Balanced Splitting

A balanced string is one that contains an equal quantity of 'L' and 'R' characters. Given a balanced string S, your task is to split it into some number of substrings such that each substring is also balanced. Output the maximum number of balanced strings you can obtain through this splitting process.

Input Format

The first and only line of input contains string S.

Output Format

Print the maximum number of balanced strings you can obtain with the splitting process.

Constraints

2 ≤ len(A) ≤ 1000

Example Input LRRLLLLRRLRR

Output 3

Explanation

The string 'LRLLLRRLRR' can be split into three substrings: "LR", "RL" and "LLLRRLRR," each containing the same number of 'L' and 'R' characters

1. Digit String

Given a string, check if it contains only digits.

Input Format

The input contains a string S, consisting of ASCII characters.

Output Format

Print "Yes" if the string contains only digits, and "No" otherwise.

Constraints

1 <= len(S) <= 100

Example Input 123456786543

Output Yes

1. Digits Sum

Given a non-negative integer - N, print the sum of digits of the given number.

Input Format

The first and only line of input contains a non-negative integer N.

Output Format

Print the sum of the digits of the given number.

Constraints

1 <= length(N) <= 103

Example Input 164

Output 11

Explanation 1 + 6 + 4 = 11

1. Number of Days

Given 2 unique dates, print the number of days between the 2 given dates.

Input Format

The first and only line of input contains 2 dates separated by space.

Output Format

Print the number of days.

Constraints

The given dates are valid dates between the years 1971 and 2100.

Example Input

2000-01-16 1999-12-30

Output

17

1. AB and CD

Given a string S of length N. You can apply the following operations any number of times:

1. In one operation you can remove any one occurrence of substring "AB" from S.
2. In one operation you can remove any one occurrence of substring "CD" from S.

Print the minimum possible length of the resulting string that you can obtain. Note: The string concatenates after removing the substring and could produce new "AB" or "CD" substrings.

Input Format

First and only line of input contains a string S.

Output Format

Print the minimum possible length of string S.

Constraints

'A' <= S[i] <= 'Z' 1 <= N <= 500

Example Input CCAAABBBDE

Output 2

Explanation

You can apply the following operations to the given string:

1st operation: Remove AB from the string, and the string becomes 'CCAABBDE.'

2nd operation: Remove AB from the string, and the string becomes 'CCABDE.' 3rd operation: Remove AB from the string, and the string becomes 'CCDE.'

4th operation: Remove CD from the string, and the string becomes 'CE.'

Furthermore, we cannot apply any more operations, and the minimum possible length is 2.

1. Strong Password

Find the minimum number of characters to add to a password (P) to ensure that P meets the following criteria:

* 1. Contains at least 6 characters.
  2. Contains at least one digit.
  3. Contains at least one lowercase character.
  4. Contains at least one uppercase character.
  5. Contains at least one special character (!@#$%^&\*()-+).

Input Format

First and only line of input contains a string P.

Output Format

Print the minimum number of characters that has to be added to P.

Constraints

1 <= len(P) <=50

P[i] ∈ {[a-z], [A-Z], [0-9], or [!@#$%^&\*()-+ ]}.

Example Input He!!0

Output 1

Explanation

The given password P already contains one digit, one lowercase character, one uppercase character and one special character. However, it should also contain at least 6 characters. So we need to add 1 character to ensure it meets all the criteria.

1. Anagram Basic

Given two strings A and B consisting of lowercase characters, check if they are anagrams. An anagram is a rearrangement of the letters of one word to form another word. In other words, some permutations of string A must be the same as string B.

Input Format

The first line of input contains string A. The second line of input contains string B.

Output Format

Print "TRUE" if A and B are anagrams otherwise "FALSE".

Constraints

1 ≤ len(A), len(B) ≤ 104

Example Input

stop post

Output TRUE

1. Compress String

Given a string, compress the given string. See the example for more details.

Input Format

Input contains a string S, consisting of lowercase and uppercase characters.

Output Format

Print the compressed string.

Constraints

1 <= len(S) <= 100

Example Input aaabbbbhhheaaAsssssss

Output a3b4h3e1a2A1s7

Explanation

In the given string, a is repeating for 3 times - after compression a3. Similarly,

b is repeating for 4 times - b4 h is repeating for 3 times - h3 e is repeating for 1 time - e1 a is repeating for 2 times - a2 A is repeating for 1 time - A1 s is repeating 7 times - s7

1. Strings Rotation

Given two strings A and B of length N, check if string A is a rotation of string B.

Input Format

The first line of input contains N - the length of the strings. The second line contains A and B separated by a space.

Output Format

Print "yes" if A is a rotation of B, otherwise "no".

Constraints

1 <= N <= 500

'a' <= A[i], B[i] <= 'z'

Example Input 5

hello lohel

Output Yes

1. Prefix Suffix Equality

Given strings S and T. Print "Yes", if T is both a prefix and a suffix of S, otherwise "No".

Input Format

First and only line of input contains two strings separated by a space.

Output Format

Print "Yes", if T is both a prefix and a suffix of S, otherwise "No".

Constraints

1 <= len(S), len(T) <= 1000 'a' <= S[i], T[i] <= 'z'

Example Input smartinterviewssmart smart

Output Yes

1. Longest Prefix Suffix

Given a string, compute the length of the longest proper prefix which is same as the suffix of the given string.

Input Format

The input contains a string S, consisting of only lowercase characters.

Output Format

Print the length of the longest proper prefix which is the same as a suffix of the given string.

Constraints

1 <= len(S) <= 100

Example Input smartintsmart

Output

5

1. Morse Codes

Given an array of strings A, where each string can be formed by combining the Morse code representation of individual letters.

[".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-",".-..","--","-.","---",".--.","--

.-",".-.","...","-","..-","...-",".--","-..-","-.--","--.."]

The full table for the 26 letters of the English alphabet is given above. For example, "abc" can be written as ".--...-.-.", which is the concatenation of ".-", "-

...", and "-.-.". We will call such a concatenation the transformation of a word. Print the number of different transformations among all A we have.

Input Format

First line of input contains the size of the array N. The second line of input contains array of strings separated by spaces.

Output Format

Print the number of different transformations among all A we have. Constraints

1 ≤ A.length ≤ 100

1 ≤ A[i].length ≤ 12

A[i] consists of lowercase English letters. Example

Input 4

abc bze abnn xyz Output

3

Explanation "abc" -> ".--...-.-."

"bze" -> "-...--..."

"adek" -> ".--...-.-."

"xyz" -> "-..--.----.."

There are 3 different transformations: ".--...-.-." , "-...--..." and "-..--.----..".

1. Long Pressed Keys

Observing your friend as they type their name on the keyboard, you notice that occasionally a key might be held down longer, causing a character to appear multiple times. After examining the sequence of typed characters, determine whether it's possible that the typed sequence corresponds to your friend's name.

Print true if typed\_name corresponds to your friend\_name, otherwise print false.

Input Format

The first and only line of input contains two strings separated by space.

Output Format

Print true if typed\_name corresponds to your friend\_name, otherwise print false.

Constraints

1. ≤ len(friend\_name), len(typed\_name) ≤ 1000

Example Input

raju rrraaajjjjjjjjjjjjjjuuuu

Output True

1. String GCD

Given two strings, P and Q, your task is to find the largest common divisor string S, such that both P and Q are divisible by S. In other words, there exists a string 'S' for which P = S + S + ... + S and Q = S + S + ... + S. If such a string S exists, output the largest possible S, otherwise, print -1.

Note: A string X is divisible by string Y if and only if X can be obtained by concatenating Y with itself one or more times.

Input Format

The first line of input contains the string P. The Second line of input contains string Q.

Output Format

Print the largest string S.

Constraints

1 ≤ len(P), len(Q) ≤ 1000 'A' <= P[i],Q[i] <= 'Z'

Example Input ABABAB ABAB

Output AB

107 Basic Calculator

Given a mathematical expression consisting of integers, addition, subtraction,

multiplication, division, and modulo operators. Evaluate the expression and print the result.

Note:

Assume that it is a regular Calculator, where the expression will be simply evaluated from left to right, without following BODMAS Rule.

Assume that the division operator will give an integer value.

Assume that the modulo operator (say % M) will give a positive integer value in the range [0, M-1] Examples:

3 - 8 / 2 = -5 / 2 = -2

3 + 8 / 2 = 11 / 2 = 5

1. - 9 % 5 = -7 % 5 = 3

Input Format

The first line of input contains an integer N. The second line of input is a single line containing a mathematical expression. The expression will contain N integers, and (N-1) operators, separated by single spaces.

Output Format

Print a single integer, representing the result of the evaluated expression.

Constraints 2 <= N <= 8

1. <= Integer in the Expression <= 9

Example Input 6

8 + 2 \* 5 - 3 / 5 % 6

Output 3

Explanation

8 + 2 \* 5 - 3 / 5 % 6

= 10 \* 5 - 3 / 5 % 6

= 50 - 3 / 5 % 6

= 47 / 5 % 6

= 9 % 6

= 3

1. Dictionary Operations

You are tasked with implementing a program that manipulates an empty Dictionary based on a series of commands.

Input Format

The first line of input contains an integer N, indicating the number of commands

to follow. The next N lines contains any of the following commands:

insert K: Insert a key-value pair (K,1) into the Dictionary. If the key (K) already exists, increment the count associated with that key by 1.

search K: Search for a key (K) in the Dictionary. If the key exists, print True, otherwise print False.

delete K: Completely remove the key (K) from the Dictionary. remove K: If the count associated with the key (K) is greater than 1,

decrement the count by 1. If the count is 1, completely remove the key from the Dictionary.

get K: Retrieve the count associated with the key from the Dictionary. size: Gives the size of the Dictionary.

Output Format

For size and search command, print the result. For the remaining commands, print the sorted (K:V) of the Dictionary separated by spaces (see example for more details). If the Dictionary is empty, skip printing the Dictionary.

Constraints 1 <= N <= 50

1 <= K <= 100

For delete K, remove K and get K commands => K ∈ {Keys in the Dictionary} Example Input

10

insert 5

insert 10

insert 66

insert 66

remove 66

search 10

delete 10

insert 3

get 3 size

Output 5:1

5:1 10:1

5:1 10:1 66:1

5:1 10:1 66:2

5:1 10:1 66:1

True 5:1 66:1

3:1 5:1 66:1

1

3

1. HashSet Operations

Given two arrays of size N and M, insert them into two sets A and B. Implement the following operations in the same sequential order:

Union of A and B: Find the union of A and B, print the elements of the resulting set in sorted order.

Intersection of A and B: Find the intersection of A and B, print the elements of the resulting set in sorted order.

Symmetric Difference of A and B: Find the symmetric difference of A and B, print the elements of the resulting set in sorted order.

Check if A and B are disjoint sets: If yes, print true, otherwise, print false. Check if A is a subset of B: If yes, print true, otherwise, print false.

Check if A is a superset of B: If yes, print true, otherwise, print false.

Input Format

The first line of input contains N, the size of array1. The second line of input contains N elements of array1.

The third line of input contains M, the size of array2. The fourth line of input contains M elements of array2.

Output Format

For subset, superset, and disjoint operations print True or False. And for remaining all operations, if resulting set is not empty, print the sorted set separated by spaces, otherwise skip printing the set.

Constraints

1 <= N, M <= 50

1 <= array1[i], array2[i] <= 100

Example Input 4

9 6 8 7

3

9 9 6

Output 6 7 8 9

6 9

7 8

false false true

1. As the hotel manager, you have N guests to attend to, and each guest has a happiness value (Ci) that depends on when they are served. A guest’s unhappiness is determined by the difference between their happiness value (Ci) and the time (x) they are served, which is calculated as |Ci—x|.

Your goal is to find the minimum total unhappiness by serving all guests in any order you choose.

Please note that at a given time, only one guest can be served, and each guest takes exactly one unit of time to be served.

Here are the constraints:

1 <= N <= 10^3

1 <= Ci <= N

For example: Input: 4 2 2 3 3

Output: 2

Input: 4 1 1 1 1

Output: 6

1. Find the minimum possible absolute difference between the maximum and minimum sum of elements in 4 non-empty contiguous subsequences of an array A of N elements by making three cuts.

Constraints:

4 <= N <= 10^5

1 <= A[i] <= 10^4

Input:

10

10 71 84 33 6 47 23 25 52 64

Output: 36

1. You are given N people and K days, each day represented by an array of N numbers, indicating the order of the people who arrived at the theatre. Your task is to find the largest group of people who arrive in the same order for all K days.

Constraints:

1 <= N <= 1000

1 <= K <= 10

1 <= a[i][j] <= N

Sample Input:

N=4, K=3

Day 1: [1, 3, 2, 4]

Day 2: [1, 3, 2, 4]

Day 3: [1, 4, 3, 2]

Sample Output:

1. (people 1, 3, and 2 arrive in the same order for all K days)
2. You are given a string of length n consisting of numbers and question marks (# represents a mine, and the number represents the sum of the number of mines in the adjacent cells). Your task is to fill the question marks with either 0 or 1 in such a way that the string represents the correct sum of mines in the adjacent cells. Return the number of ways to fill the question marks.

Constraints:

1 <= n <= 10^5

Example Input:

??0?1?

Example Output:

2

Example explanation: Two possible ways to fill the question marks are #1001# and 00001#. Round-2 sample questions

1. Shyam has N various varieties of dishes lined up in a row: A1, A2,..., AN, where Ai signifies the type of the ith dish. He wants to select as many dishes as possible from the menu, but only if the following two requirements are satisfied:
   * He can select only one sort of dish.
   * Two adjacent dishes cannot be selected simultaneously.

Now the query is that Shyam wants to know which type of dish he should select so that he has the option of choosing the maximum number of dishes possible.

Example:

Given N = 9 and A = [1,2,2,1,2,1,1,1,1]

As we can see here, For type 1, Shyam has a maximum of four dishes to choose from. Selecting A1, A4, A7, and A9 is one of the possible ways of selecting four dishes of type 1.

For type 2, Shyam has a maximum of two dishes to choose from. One way is to select A3 and A5. In this situation, Shyam should go for type 1, which allows him to select comparatively more dishes.

Input Format

T, the number of test cases, appears on the first line. Then there are the test cases following T.

The first line of each test case contains a single integer N. And the N integers A1, A2,..., AN appears on the second line.

Constraints

1 <= T <= 10^3

1 <= N <= 10^3

1 <= Ai <= 10^3

Output Format

Print a single line for each test case containing one integer indicating the type of dish Shyam should select. Print the smallest response if there are multiple answers.

Sample Input:

3

5

1 2 2 1 2

6

1 1 1 1 1 1

8

1 2 2 2 3 4 2 1

Sample Output: 1

1

2

1. Sam is handed a rectangular piece of paper with the dimensions h\*w, where h denotes height and w denotes width. Sam wants to fold the paper in the fewest possible moves such that its dimensions are h1\*w1. The paper could only be folded parallel towards its edges, and the dimensions should be integers after folding. If h=8 and w=4, for example, fold the paper till it's h1, w1 = 6,1. First, fold the long edge 6

units to the side, resulting in a 6\*4 paper. And then, Fold the page in three folds along the width 2 units edge to achieve a 6\*1 page.

Input Format

First line contains an integer denotes h Seconds line contains an integer denotes w Third line contains an integer denotes h1

Fourth line contains an integer denotes w1

Constraints

1 <= h, w, h1, w1 <= 10^15

h1 <= h w1 <= w Output Format

Print the minimum moves required. Sample Input:

2

3

2

2

Sample Output: 1

Question 1 – The Lost Package

A delivery company is missing one package from a shipment of N packages. Each package has a unique tracking number from 1 to N, but one is missing.

The company needs to quickly identify the missing package to ensure timely delivery.

Parameters:

* T :: INTEGER — number of test cases (1 ≤ T ≤ 1000)
* N :: INTEGER — total number of packages (2 ≤ N ≤ 10⁶)
* A :: INTEGER ARRAY — tracking numbers of N-1 packages (1 ≤ A[i] ≤ N, all unique)

Input Format:

* First line: Integer T, number of test cases
* For each test case:
* First line: Integer N
* Second line: N-1 space-separated integers representing tracking numbers Output Format:
* For each test case, print a single integer — the missing tracking number

Example Cases:

Case 1:

Input:

1

5

1 2 3 5

Output:

4

Explanation:

The complete set is {1, 2, 3, 4, 5}. The number 4 is missing from the input list.

Case 2: Input: 1

3

2 3

Output:

1

Explanation:

The complete set is {1, 2, 3}. The number 1 is missing.

Case 3:

Input:

1

6

1 2 3 4 6

Output:

5

Explanation:

The full range is {1, 2, 3, 4, 5, 6}. Tracking number 5 is not present.

Question 2 – The Secret Message

A spy receives a coded message where words are scrambled. The message must be decrypted by reversing the order of words, while keeping the characters in each word intact.

The spy needs to decode the message before the enemy intercepts it.

Parameters:

* T :: INTEGER — number of test cases (1 ≤ T ≤ 1000)
* S :: STRING — the coded message (1 ≤ |S| ≤ 10⁵), consisting of words separated by single spaces

Input Format:

* First line: Integer T, number of test cases
* For each test case:
* One line containing a string S — the scrambled message Output Format:
* For each test case, print the decoded message with the words in reverse order.

Example Cases:

Case 1:

Input:

1

hello world1 Output:

world hello Explanation:

There are two words: "hello" and "world". Reversing their order gives: "world hello". Case 2 :

Input:

1

spy must decode message Output:

message decode must spy

Explanation:

The words are: ["spy", "must", "decode", "message"]. After reversing: ["message", "decode", "must", "spy"].

Case 3:

Input:

1

run now Output:

now run Explanation:

Two words: "run" and "now". Reversed gives: "now run".

Question 3 – The Bank Account Validator

A bank uses a checksum validation system to verify account numbers. Given an account number, check if it follows the correct checksum rule. If the checksum is incorrect, the account number is flagged for review.

Checksum Rule:

The account number is valid if the sum of its digits is divisible by 10. Parameters:

* T :: INTEGER — number of test cases (1 ≤ T ≤ 1000)
* A :: INTEGER — the account number (1 ≤ A ≤ 10¹⁸)

Input Format:

* First line: Integer T, number of test cases
* For each test case:
* One line containing an integer A — the account number Output Format:
* For each test case, print "Valid" if the account number passes the checksum, otherwise print "Invalid".

Example Cases:

Case 1:

Input:

1

12340

Output:

Valid Explanation:

Sum of digits = 1 + 2 + 3 + 4 + 0 = 10. 10 is divisible by 10, so it's Valid.

Case 2:

Input:

1

987654321

Output:

Invalid

Explanation:

Sum of digits = 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 45. 45 is not divisible by 10, so it's Invalid.

Case 3:

Input:

1

1111111111

Output:

Valid Explanation:

Sum of digits = 1×10 = 10 → divisible by 10 → Valid.

Question 4 – The Library Book Sorting

A librarian needs to sort books by title efficiently to help visitors find books faster. The sorting must be done in alphabetical order while handling large datasets.

The system should be optimized for quick retrieval. Parameters:

* T :: INTEGER — number of test cases (1 ≤ T ≤ 100)
* N :: INTEGER — number of books (1 ≤ N ≤ 10⁵)
* Titles :: STRING ARRAY — list of book titles (each title is a non-empty string of at most 100 characters)

Input Format:

* First line: Integer T, number of test cases
* For each test case:
* First line: Integer N, number of books
* Next N lines: each containing a string (book title) Output Format:
* For each test case, print the sorted titles — one per line Example Cases:

Case 1: Input: 1

3

Harry Potter

A Game of Thrones Cinderella Output:

A Game of Thrones Cinderella Harry Potter Explanation:

Sorted order of titles alphabetically is:

* A Game of Thrones
* Cinderella
* Harry Potter

Case 2:

Input:

1

4

Zoo Life Alpha Tales

Echoes of Time Braveheart Output:

Alpha Tales Braveheart Echoes of Time Zoo Life Explanation: Alphabetical sort yields the correct order shown above.

Case 3:

Input:

1

2

The Great Gatsby Animal Farm Output:

Animal Farm The Great Gatsby Explanation:

"A" comes before "T", so "Animal Farm" is listed before "The Great Gatsby".

Question 5 – The Restaurant Bill Splitter

A group of friends is dining at a restaurant and needs to split the bill fairly.

Given the total bill amount and the number of people, calculate how much each person should pay. The system should handle rounding errors efficiently.

Parameters:

* T :: INTEGER — number of test cases (1 ≤ T ≤ 1000)
* B :: INTEGER — total bill amount (1 ≤ B ≤ 10⁶)
* P :: INTEGER — number of people (1 ≤ P ≤ 100)

Input Format:

* First line: Integer T, number of test cases
* For each test case:
* One line containing two integers B and P Output Format:
* For each test case, print the amount each person should pay, rounded to 2 decimal places

Example Cases:

Case 1:

Input:

1

100 4

Output:

25.00

Explanation:

100 divided by 4 = 25.0 → each pays 25.00.

Case 2: Input: 1

99 3

Output:

33.00

Explanation:

99 divided by 3 = 33.0 → each pays 33.00.

Case 3:

Input:

1

250 6

Output:

41.67

Explanation:

250 ÷ 6 = 41.666... → rounded to 41.67

Question 6 – The Traffic Light Timer

A city traffic management system needs to simulate traffic light changes at fixed intervals.

The system must ensure smooth traffic flow by switching between red, yellow, and green lights in sequence:

Red → Green → Yellow → Red...

Given the durations for each light, simulate the sequence of light changes over a given time period.

Parameters:

* T :: INTEGER — number of test cases (1 ≤ T ≤ 100)
* R :: INTEGER — duration of red light (1 ≤ R ≤ 100)
* Y :: INTEGER — duration of yellow light (1 ≤ Y ≤ 100)
* G :: INTEGER — duration of green light (1 ≤ G ≤ 100)
* T\_total :: INTEGER — total time to simulate the light changes (1 ≤ T\_total ≤ 1000)

Input Format:

* First line: Integer T, number of test cases
* For each test case:
* One line containing four integers: R, Y, G, T\_total Output Format:
* For each test case, print a single line with space-separated names of lights (Red, Green, or Yellow) indicating which light is on at each second from 1 to T\_total.

Example Cases:

Case 1:

Input:

1

2 1 1 5

Output:

Red Red Green Yellow Red Explanation:

Cycle: Red(2s) → Green(1s) → Yellow(1s) → repeat Total time = 5 seconds

Lights sequence:

1-2 → Red

1. → Green
2. → Yellow
3. → Red

Case 2:

Input:

1

1 1 1 6

Output:

Red Green Yellow Red Green Yellow Explanation:

Each light lasts 1 second → cycle is 3 seconds Repeat cycle twice to get 6 total seconds

Case 3:

Input:

1

3 2 1 10

Output:

Red Red Red Green Yellow Red Red Red Green Yellow Explanation:

Cycle: Red(3s) → Green(1s) → Yellow(2s) → total cycle = 6 seconds Next cycle continues from second 7 to 10

Question 7 – The Movie Ticket Booking

A cinema hall has multiple available seats, and customers want to book the best seat.

The system should recommend the seat closest to the center for an optimal viewing experience.

Parameters:

* T :: INTEGER — number of test cases (1 ≤ T ≤ 100)
* N :: INTEGER — number of available seats (1 ≤ N ≤ 1000)
* Seats :: INTEGER ARRAY — seat numbers (positive integers)

Input Format:

* First line: Integer T, number of test cases
* For each test case:
* First line: Integer N, number of seats
* Second line: N space-separated integers denoting available seat positions Output Format:
* For each test case, print the seat number that is closest to the center position of all available seats.
* If there are multiple such seats, print the smallest seat number.

Example Cases:

Case 1:

Input:

1

5

10 20 30 40 50

Output:

30

Explanation:

Center = (min + max) / 2 = (10 + 50) / 2 = 30

30 is present → print 30.

Case 2:

Input:

1

6

1 2 3 7 8 9

Output:

3

Explanation:

Center = (1 + 9) / 2 = 5

Closest seats: 3 and 7 (both 2 away) → choose smaller = 3.

Case 3: Input: 1

4

13 17 20 25

Output:

17

Explanation:

Center = (13 + 25) / 2 = 19 Closest:

* 17 (distance 2)
* 20 (distance 1) → closer

So, output = 20

Question 8 – The Weather Forecast Analyzer

A meteorological department records temperature readings daily.

Given a set of temperature readings, find the highest and lowest temperatures recorded to help predict weather trends.

Parameters:

* T :: INTEGER — number of test cases (1 ≤ T ≤ 100)
* N :: INTEGER — number of temperature readings (1 ≤ N ≤ 1000)
* Temp :: INTEGER ARRAY — list of temperature readings (−50 ≤ Temp[i] ≤ 50)

Input Format:

* First line: Integer T, number of test cases
* For each test case:
* First line: Integer N, number of temperature readings
* Second line: N space-separated integers representing temperatures Output Format:
* For each test case, print two space-separated integers: highest temperature and lowest temperature

Example Cases:

Case 1:

Input:

1

5

10 20 -5 30 15

Output:

30 -5

Explanation:

Maximum = 30, Minimum = -5 → output 30 -5

Case 2:

Input:

1

4

0 0 0 0

Output:

0 0

Explanation:

All readings are the same. Both max and min = 0

Case 3:

Input:

1

6

-10 -20 -5 -30 -1 -25

Output:

-1 -30

Explanation:

Highest = -1, Lowest = -30 → output -1 -30

Question 9 – The Social Media Username Generator

A social media platform needs to generate unique usernames for new users.

The username should be based on the user's name and birth year, ensuring uniqueness by combining them.

Parameters:

* T :: INTEGER — number of test cases (1 ≤ T ≤ 1000)
* Name :: STRING — user's name (consisting of alphabets, length ≤ 100)
* Year :: INTEGER — birth year (1900 ≤ Year ≤ 2025)

Input Format:

* First line: Integer T, number of test cases
* For each test case:
* One line with string Name
* One line with integer Year Output Format:
* For each test case, print the generated username as: lowercase(name) + last two digits of year

Example Cases:

Case 1:

Input:

1

Alice 1995 Output:

alice95 Explanation:

Name in lowercase: alice

Last two digits of year 1995: 95 Username: alice95 Case 2:

Input:

1

Bob 2001

Output:

bob01 Explanation:

Name lowercase: bob

Last two digits of year 2001: 01 Username: bob01 Case 3:

Input:

1

Eve 1987

Output:

eve87 Explanation:

Name lowercase: eve

Year: 1987 last two digits: 87 Username: eve87

Question 10 – The Online Shopping Discount Calculator

An e-commerce website applies discounts to shopping carts based on predefined rules.

Given a list of item prices and a discount percentage, calculate the final price after applying the discount.

Parameters:

* T :: INTEGER — number of test cases (1 ≤ T ≤ 1000)
* N :: INTEGER — number of items (1 ≤ N ≤ 100)
* Prices :: INTEGER ARRAY — prices of items (positive integers)
* D :: INTEGER — discount percentage (1 ≤ D ≤ 50)

Input Format:

* First line: Integer T, number of test cases
* For each test case:
* One line: Integer N
* One line: N space-separated integers — prices of items
* One line: Integer D — discount percentage Output Format:
* For each test case, print the final price after applying the discount, rounded to 2 decimal places.

Example Cases:

Case 1:

Input:

1

3

100 200 300

10

Output:

540.00

Explanation:

Sum = 100+200+300 = 600

Discount = 10% → 600 \* 0.10 = 60

Final price = 600 - 60 = 540.00

Case 2 :

Input:

1

2

150 350

20

Output:

400.00

Explanation: Sum = 500

Discount 20% → 100 Final price = 400.00 Case 3:

Input:

1

5

50 50 50 50 50

5

Output: 237.50

Explanation:

Sum = 250

Discount 5% → 12.5 Final price = 237.50

Question 11 – The Bank Fraud Detection

Banks monitor transactions for suspicious activities to prevent fraud. Given a series of financial transactions, your task is to find the longest sequence of consecutive transactions that add up to a target suspicious sum K. This will help detect potential fraud patterns.

Parameters:

* N: number of transactions
* K: suspicious sum to detect
* Transactions: list of transaction amounts Input Format:

First line contains two integers N and K.

Second line contains N space-separated integers representing transaction amounts. Output Format:

Print the length of the longest sequence with sum K. Example Cases:

Example Case 1:

Input:

5 12

1. 4 2 8 1

Output:

3

Explanation:

The longest sequence with sum 12 is [2, 4, 2, 8], but only the subarray [4, 2, 6] adds exactly to 12;

length is 3.

Example Case 2:

Input:

6 15

1 2 3 4 5 6

Output:

5

Explanation:

The longest sequence with sum 15 is [1, 2, 3, 4, 5], length is 5.

Example Case 3: Input:

4 10

1 2 3 4

Output:

0

Explanation:

No consecutive sequence sums to 10, output is 0. Question 12 – The Hacker’s Challenge

A hacker is trying to decipher a password by generating the smallest lexicographical permutation of a given string. The password system shuffles characters randomly, making it hard to guess. Can you help the hacker find the smallest permutation possible?

Parameters:

* S: input string Input Format:

A single string S.

Output Format:

Print the smallest lexicographical permutation of S. Example Cases:

Example Case 1:

Input:

dcba Output:

abcd Explanation:

The smallest lex order permutation of 'dcba' is 'abcd'. Example Case 2:

Input: aabbcc Output: aabbcc

Explanation:

The string is already the smallest permutation. Example Case 3:

Input:

zyx Output:

xyz Explanation:

The smallest lex order permutation of 'zyx' is 'xyz'. Question 13 – The Warehouse Inventory Tracker

A warehouse manages thousands of items daily. Some items sell more frequently than others, and your task is to determine the most frequently sold product. By tracking sales patterns, inventory can be adjusted to meet customer demand effectively.

Parameters:

* N: number of items sold
* Items: list of item names sold Input Format:

First line contains integer N.

Second line contains N space-separated item names. Output Format:

Print the most frequently sold item. Example Cases:

Example Case 1:

Input:

6

apple banana apple orange banana apple Output:

apple Explanation:

'apple' sold 3 times, more than any other.

Example Case 2:

Input:

4

pen pen pencil pen Output:

pen Explanation:

'pen' sold 3 times, most frequent. Example Case 3:

Input:

5

chair table chair table chair Output:

chair Explanation:

'chair' sold 3 times, highest frequency. Question 14 – The Flight Route Optimizer

An airline wants to provide the shortest flight route between two cities to save fuel and reduce travel time. Given multiple flight routes, find the shortest path between two specified cities while ensuring safety and efficiency.

Parameters:

* N: number of cities
* M: number of flight routes
* A,B,C: direct flight from city A to B with cost C
* X,Y: source and destination cities Input Format:

First line contains two integers N and M. Next M lines contain three integers A,B,C. Last line contains two integers X and Y. Output Format:

Print the shortest cost from X to Y. Example Cases:

Example Case 1:

Input:

4 4

1 2 5

2 3 10

1. 4 3

1 4 20

1 4

Output:

18

Explanation:

Shortest path from 1 to 4 is 1->2->3->4 with cost 5+10+3=18.

Example Case 2:

Input:

3 3

1 2 4

2 3 6

1 3 15

1 3

Output:

10

Explanation:

Shortest path 1->2->3 costs 4+6=10, better than direct 15.

Example Case 3:

Input:

2 1

1 2 7

2 1

Output:

No path Explanation:

No flight from 2 to 1, output accordingly. Question 15 – The Smart Home Energy Saver

In a smart home, appliances consume electricity based on predefined schedules. To optimize energy consumption, you must determine the best time slots to turn appliances ON/OFF to minimize overall power usage while meeting household needs.

Parameters:

* N: number of appliances
* T: total time slots available
* S,E: appliance start and end time slots Input Format:

First line contains two integers N and T. Next N lines contain two integers S and E. Output Format: Print the optimized schedule minimizing energy consumption.

Example Cases:

Example Case 1:

Input:

3 24

1 5

6 10

11 15

Output:

Schedule with minimal overlap Explanation:

Turn on appliances in non-overlapping slots to save power. Example Case 2:

Input:

2 12

1 6

7 12

Output:

Schedule all appliances fully Explanation:

Appliances run sequentially without overlap.

Example Case 3:

Input:

1. 24

1 4

3 8

7 12

11 16

Output:

Optimized schedule minimizing conflicts Explanation:

Adjust start/end times to minimize peak consumption. Question 16 – The Online Auction System

An online auction platform needs a mechanism to determine the highest bid for an item in real-time. Each bidder places their bid, and the system must identify the highest valid bid before the auction closes.

Parameters:

* N: number of bids
* B,A: bidder B placing bid amount A Input Format:

First line contains integer N.

Next N lines contain two integers B and A. Output Format:

Print the highest bid amount and the bidder’s ID.

Example Cases:

Example Case 1:

Input:

3

1 100

2 150

3 120

Output:

150 2

Explanation:

Bidder 2 has the highest bid 150. Example Case 2:

Input:

5

1 200

2 180

3 210

4 190

1. 205

Output:

210 3

Explanation:

Bidder 3 has the highest bid 210. Example Case 3:

Input:

2

1 50

2 50

Output:

50 1

Explanation:

Both bid 50, bidder 1 wins as first highest bid. Question 17 – The Emergency Evacuation Plan

An emergency occurs in a large building, and an evacuation plan needs to be executed efficiently. Given a building layout with multiple rooms, your task is to find the shortest escape path from a given starting location to the nearest exit.

Parameters:

* N: number of rooms
* M: number of hallways
* A,B: hallway connecting room A to B
* X: starting room Input Format:

First line contains two integers N and M. Next M lines contain two integers A and B. Last line contains integer X.

Output Format:

Print the shortest distance to the nearest exit. Example Cases:

Example Case 1: Input:

5 5

1 2

2 3

3 4

4 5

1 5

1

Output:

1

Explanation:

From room 1, nearest exit is room 5 reachable in 1 step. Example Case 2:

Input:

4 3

1 2

2 3

3 4

2

Output:

2

Explanation:

Starting at room 2, nearest exit at room 4 is 2 steps away. Example Case 3:

Input:

3 2

1 2

2 3

3

Output:

0

Explanation:

Starting room 3 is already an exit, distance 0. Question 18 – The Stock Market Trend Analyzer

A stock analyst wants to identify long-term stock price trends. Given a list of stock prices recorded over N days, determine the longest consecutive increasing trend. This helps traders make informed investment decisions.

Parameters:

* N: number of days
* A[i]: stock price on day i Input Format:

First line contains integer N.

Second line contains N space-separated integers representing stock prices. Output Format:

Print the length of the longest consecutive increasing trend. Example Cases:

Example Case 1:

Input:

7

100 101 102 90 91 92 93

Output:

4

Explanation:

Longest increasing trend is days 4 to 7: 90, 91, 92, 93.

Example Case 2: Input:

5

5 4 3 2 1

Output:

1

Explanation:

No increasing trend, each day alone is length 1. Example Case 3:

Input:

6

1 2 3 4 5 6

Output:

6

Explanation:

Prices increase every day, length 6. Question 19 – The Music Playlist Generator

A music streaming platform wants to generate the most popular playlist based on listening history. Given a list of songs played, identify the top K most frequently played songs.

Parameters:

* N: number of song plays
* K: number of top songs to return
* Songs: list of song names played Input Format:

First line contains two integers N and K.

Second line contains N space-separated song names. Output Format:

Print K most frequently played songs. Example Cases:

Example Case 1:

Input:

7 2

song1 song2 song1 song3 song2 song1 song4 Output:

song1 song2 Explanation:

'song1' played 3 times, 'song2' played 2 times. Example Case 2:

Input:

5 3

a b a b c Output:

a b c Explanation:

'a' and 'b' played twice, 'c' once. Example Case 3:

Input:

4 1

a a b c Output:

a Explanation:

'a' is most played.

Question 20 – The Hospital Patient Queue Management

A hospital emergency room must prioritize patients based on severity. Each incoming patient is assigned a severity score, and the system must always serve the most critical patient first.

Parameters:

* N: number of patients
* P,S: patient ID P with severity score S Input Format:

First line contains integer N.

Next N lines contain two integers P and S. Output Format:

Print the patient ID with the highest severity. Example Cases:

Example Case 1:

Input:

3

101 5

102 10

103 7

Output:

102

Explanation:

Patient 102 has the highest severity 10. Example Case 2:

Input:

4

201 8

202 8

203 7

204 6

Output:

201

Explanation:

Tie in severity 8, earliest patient 201 chosen. Example Case 3:

Input:

2

301 3

302 4

Output:

302

Explanation:

Patient 302 has higher severity 4.

Sample 1

Today you decided to go to the gym. You currently have E energy. There are N exercises in the gym. Each of these exercises drains Ai amount of energy from your body.

You feel tired if your energy reaches 0 or below. Calculate the minimum number of exercises you have to perform such that you become tired. Every unique exercise can only be performed at most 2 times as others also have to use the machines.

If performing all the exercises does not make you feel tired, return

-1.

Parameters:

E :: INTEGER

The first line contains an integer, E, denoting the Energy. E :: 1 -> 10^5 N :: INTEGER

The next line contains an integer, N, denoting the number of exercises.

N :: 1 -> 10^5

A :: INTEGER ARRAY

Each line i of the N subsequent lines (where 0 ≤ i < N) contains an integer describing the amount of energy

drained by ith exercise.

A[i] :: 1 -> 10^5

Case#: 1 Input:

6

2

1

2

Output:

4

E = 6

Do 1st exercise 2 times Do 2nd exercise 2 times Hence, total exercise done 4. Case#: 2

Input:

10

2

1

2

Output:

-1

E = 10

By doing both the exercises 2 times you won’t feel tired. Case#: 3

Input:

2

3

1

5

2

Output:

1

E = 2

Use 3rd exercise 1 time. Hence, total exercise done 1.

Sample 2

There is a battle between heroes and villains going on. You have M heroes, all of them have the same health H. There are N villains, health of the ith villain is Vi.

When a hero, with health H battles a villain with health Vi, one of the three scenarios can happen: if H > Vi: The villain is defeated and the health of the hero is decreased by Vi

if H < Vi: The villain wins, his health is not affected and the hero is no longer able to fight. if H = Vi: Both of them are considered defeated and neither can fight.

The heroes start fighting villains one by one in the same order, first villain 1 then villain 2 and so on. It is might be possible that before defeating all the villains, all the heroes are defeated. Therefore, to ensure the victory of the heroes, you want to remove some villains from the front.

Your task is to find the minimum number of villains you need to remove from the front such that the victory of the heroes is guaranteed.

Note: If in the last battle, both the hero and villain are defeated and no more heroes or villains remain, it would still be considered a victory since all the villains are defeated.

Parameters:

N :: INTEGER

The first line contains an integer, N, denoting the number of villains N :: 1 -> 2\*10^5 M :: INTEGER

The next line contains an integer, M, denoting the number of heroes M :: 1 -> 2\*10^5 H :: INTEGER

The next line contains an integer, H, denoting the health of each of the heroes H :: 1 -> 10^9

array :: INTEGER ARRAY

Each line i of the N subsequent lines (where 0 ≤ i < N) contains an integer describing the health of each of the villains.

array[i] :: 1 -> 10^9 Case#: 1 Input:

4

4

3

3

1

3

3

Output:

0

[3, 1, 3, 3]. We have 4 heroes will health 3. The heroes 1 will fight villain 1. Both get defeated. The hero 2 fights villain 2. It wins the battle and now his health is 2. He fights the third villain and loses, the villain still has health 3. The hero 3 fights villain 3 and both get defeated. Hero 4 fights villain 4 and both get defeated. So no need to remove any villain.

Case#: 2 Input:

5

3

3

1

2

3

1

1

Output:

0

The fight will take place and hero 1 will defeat villain 1 and 2. Hero 2 will defeat villain 2. Hero 3 will defeat villain 3 and 4

Case#: 3 Input:

5

1

4

1

2

3

1

3

Output:

3

Only 1 hero is present with health 4. Since you can only remove villain from the front, you will have to remove the first 3 villains to ensure victory. The hero can fight the last 2 villain of health 1 and 3 respectively and win the battle.

External Document © 2024 Infosys Limited

Sample 3

You need to build a road in a rugged terrain. You know the sea level of each segment of the rugged terrain, i.e. the i-th segment is Li meters from sea level.

You needs to transform the terrain into a strictly downward sloping terrain for the road, i.e, for each i-th segment where 2 <= i

<= N, resultant Li-1 > Li. In order to do so, you employ a powerful digging team to help you dig and reduce the sea level of the segments. On day D, the team can reduce the sea level for each segments that you scheduled that day by 2D-1 meters each.

You are allowed to assign the team to dig on multiple segments and/or dig on the same segments for multiple days.

Your task is to find the minimum number of days needed to transform the terrain as per your requirements.

Parameters:

N :: INTEGER

The first line contains an integer, N, denoting the number of elements in L. N :: 1 -> 10^5

L :: INTEGER ARRAY

Each line i of the N subsequent lines (where 0 < i ≤ N) contains an integer describing Li, the sea level of the

i-th segment.

L[i] :: -10^9 -> 10^9

Case#: 1 Input:

2

3

3

Output:

1

We can dig on the 2nd segment, reducing it from 3-meter sea level to 2. Resulting in {3, 2} which is strictly decreasing.

Case#: 2 Input:

2

5

-3

Output:

0

It is already strictly decreasing before start. Case#: 3 Input:

4

-1

1

1

1

Output:

3

One of the possible way:

On day 1, we can dig on 1st and 4th segment, resulting in {-2, 1, 1, 0} On day 2, we can dig on 3rd and 4th segments, resulting in {-2, 1,

-1, -2}

On day 3, we can dig on 2nd, 3rd and 4th segments, resulting in

{-2, -3, -5, -6}

External Document © 2024 Infosys Limited

Sample 4

You are given an array of size N. You need to change this array into a mountain. By mountain we mean, the either ends of the array should have equal elements. Then as we move towards the

middle from both ends, the next element is just one more than the previous one. So it would have a peak in the middle and decreases if you go towards either end, just like a mountain.

Examples of mountains are [1, 2, 3, 2, 1] or [6, 7, 8, 8, 7, 6]. But

the array [1, 2, 4, 2, 1] is not a mountain because from 2 to 4 the

difference is 2. The array [1, 2, 3, 1] is also not a mountain because the elements 2 and 3 are not equal from both ends.

You need to find the minimum number of elements that should be changed in order to make the array a mountain. You can make the elements negative or zero as well.

Parameters:

N :: INTEGER

The first line contains an integer, N, denoting the number of elements in array.

N :: 1 -> 10^5

array :: INTEGER ARRAY

Each line i of the N subsequent lines (where 0 ≤ i < N) contains an integer describing i-th element of array. array[i] :: 1 -> 10^6 Case#: 1

Input:

5

1

2

3

4

5

Output:

2

array = [1, 2, 3, 4, 5] . We can change 4 and 5 to make it [1, 2, 3, 2, 1]

Case#: 2 Input:

9

1

1

1

2

3

2

1

1

1

Output:

4

array = [1, 1, 1, 2, 3, 2, 1, 1, 1]. We can change the array to [-1, 0, 1, 2,

3, 2, 1, 0, -1]

Case#: 3 Input:

6

3

3

4

4

5

5

Output:

3

array = [3, 3, 4, 4, 5, 5]. We can chage the array to [2, 3, 4, 4, 3, 2]

Sample 5

You have an interesting string S of length N. It is interesting because you can rearrange the characters of this string in any order. You want to cut this string into some contiguous pieces such that after cutting, all the pieces are equal to one another.

You can’t rearrange the characters in the cut pieces or join the pieces together. You want to make the

number of pieces as large as possible. What is the maximum number of pieces you can get?

Note: You can observe that you may not want to cut the string at all, therefore the number of pieces is 1. Hence, the answer always exists.

Parameters:

S :: STRING

The first line contains a string, S, denoting the string. len(S) :: 1 -> 2 \* 10^5 Case#: 1

Input:

zzzzz Output:

5

You can cut it into 5 pieces “z” + “z” + “z” + “z” + “z” Case#: 2

Input:

ababcc Output:

2

Rearrange the string as abcabc. you can cut it into “abc” + “abc”, hence the answer is 2

Case#: 3 Input: abccdcabacda Output:

2

Rearrange the string as “dcbaca” + “dcbaca”, the answer is 2. Sample 6

You are given an array A of size N.

You are allowed to choose at most one pair of elements such that distance (defined as the difference of their indices) is at most K and swap them.

Find the smallest lexicographical array possible after swapping. Notes:

An array x is lexicographically smaller than an array y if there exists an index i such that xi<yi, and xj=yj for

all 0≤j<i. Less formally, at the first index i in which they differ, xi<yi

Parameters:

N :: INTEGER

The first line contains an integer, N, denoting the number of elements in A. N :: 1 -> 10^5

A :: INTEGER ARRAY

Each line i of the N subsequent lines (where 0 ≤ i < N) contains an integer describing A[i].

A[i] :: 1 -> 10^5 K :: INTEGER

The next line contains an integer, K, denoting the upper bound on distance of index. K :: 1 -> N

Case#: 1 Input:

3

2

2

2

1

Output:

2

2

2

Here as all the array values are equal swapping will not change the final result. Case#: 2

Input:

5

5

4

3

2

1

3

Output:

2

4

3

5

1

Here A=[5,4,3,2,1] K=3, we can swap elements at index 0 and index 3 which makes A=[2,4,3,5,1]. Case#: 3

Input:

5

2

1

1

1

1

3

Output:

1

1

1

2

1

Here A=[2,1,1,1,1] K=3, we can swap elements at index 0 and index 3 which makes A=[1,1,1,2,1].

Sample 7

There is a restaurant that offers different types of dishes.

Your friend knows that there are exactly N dishes in the restaurant. The type of the dishes is described by an array Arr. This means that if two elements of Arr have the same value, then they are from the same type.

Your friend wants to eat as many dishes as possible. However, your friend will never order more than one serving of a particular dish.

It is given that the restaurant will not let you order the dishes of the same type more than once. Moreover, if you order A dishes in the restaurant your next order must contain (2\*A) dishes. It is also given that the restaurant does not accept orders containing more than one type of dishes in the same order.

Your friend can start eating with any number of dishes. Find the maximum number of dishes that your friend can eat.

Notes:

Your friend is a foodie and can eat any amount of food that is served to him by the restaurant. Parameters:

N :: INTEGER

The first line contains an integer, N, denoting the number of elements in Arr. N :: 1 -> 10^5

Arr :: INTEGER ARRAY

Each line i of the N subsequent lines (where 0 ≤ i < N) contains an integer describing Arr[i].

Arr[i] :: 1 -> 10^9

Case#: 1 Input:

5

1

2

4

2

3

Output:

3

N=5

A=[1,2,4,2,3]

For example, start with type 1 or 4, then double the amount by eating two of type 2 dishes. Case#: 2

Input:

7

2

2

1

1

1

1

1

Output:

6

N=7 A=[2,2,1,1,1,1,1]

Start with eating two dishes of type 2, then eat four dishes of type 1.

Note that you can’t start with one dish of type one, then two dishes of type 2, and get back to eat to a dish of size 1 again, your friend cannot eat the same type of dishes multiple times.

Case#: 3 Input:

4

1

1

1

1

Output:

4

N=4 A=[1,1,1,1]

Your friend can eat all 4 dishes in the first order.

Problem Statement:

You are given a grid of size n × m where some cells are known to be either light (0) or dark

1. Your task is to count the number of valid full configurations such that:
   * Every unknown cell is assigned either light (0) or dark (1).
   * The XOR of colors of all adjacent pairs of cells is even (i.e., total XOR is 0).

Two cells are adjacent if they share an edge.

Parameters

* + t: Number of test cases
  + For each test case:
* n: Number of rows
* m: Number of columns
* k: Number of known cells
* For k lines: xi, yi, ci representing a known cell at (xi, yi) with color ci

Input

2

3 3 5

1 1 0

1 2 1

2 2 0

3 2 1

3 3 0

2 2 4

1 1 0

1 2 1

2 1 1

2 2 0

Output

4

0

Given an array, delete the minimum number of elements to make it a "good" array. An array is good if:

* + Its length is even.
  + Every consecutive pair of elements has the same sum.

Parameters

* + T: Number of test cases
  + For each test case:
* N: Length of the array
* A: The array of integers Input

1

6

1 3 2 2 1 3

Output

0

You are given students' predictions for boarding on days. Vadim wants to ensure that at least one student

sees both of their predictions come true on consecutive days. Determine if it’s guaranteed.

Parameters

* + t: Number of test cases
  + For each test case:
* n: Number of students
* a: List of n integers (predicted days)

Input

2

3

1 2 3

4

5 5 5 5

Output

Yes No

You have n sticks of unique lengths 1 to n. Arrange them to satisfy a string s consisting of '<' and '>' characters.

Parameters

* + t: Number of test cases
  + For each test case:
* n: Number of sticks
* s: A string of length n-1 containing '<' or '>' Input

1

4

<><

Output

1 4 2 3

Given an array of integers, remove any number of elements (without rearranging) to maximize the number of non-overlapping contiguous subarrays with positive sum.

Parameters

* + T: Number of test cases
  + For each test case:
* N: Length of array
* A: Array of integers Input

2

5

1 -2 3 4 -1

4

-1 -2 -3 -4

Output

2

0

Given a string S, find the lexicographically largest subsequence possible by removing characters (order must be maintained).

Parameters

* + T: Number of test cases
  + For each test case:
* S: The input string

Input

2

abcbdab edcba

Output

ddb e

reach all other nodes directly or indirectly. Count how many nodes are essential.

Parameters

* + T: Number of test cases
  + For each test case:
* N, M: Number of nodes and edges
* M lines of edges u v

Input

1

4 4

1 2

2 3

3 4

1 3

Output

1

Parameters

* + T: Number of test cases
  + For each test case:
* N, M: Number of nodes and edges
* M lines of edges u v

Input

2

3 3

1 2

2 3

3 1

4 2

1 2

3 4

Output

Yes No

value x, find the maximum XOR between x and any value in u’s subtree.

Parameters

* + N: Number of nodes
  + A: List of N values
  + N-1 lines of edges
  + Q: Number of queries
  + Q lines of queries: u x

Input

|  |  |  |
| --- | --- | --- |
| 5 |  | |
| 1 | 2 3 | 4 5 |
| 1 | 2 |  |
| 1 | 3 |  |
| 2 | 4 |  |
| 2 | 5 |  |
| 3 |  |  |
| 1 | 1 |  |
| 2 | 6 |  |
| 3 | 7 |  |

Output

5

7

4

You are given an array of N integers. Answer Q queries: for each, find the K-th smallest number in subarray A[L...R].

Parameters

* + N: Length of array
  + A: The array
  + Q: Number of queries
  + Q queries: L R K

Input

5

1 5 2 6 3

3

1 3 2

2 5 3

1 5 4

Output

2

5

5

Neo wants to escape from the Matrix. In front of him are n buttons arranged in a row, each with an

integer weight a₁, a₂, …, aₙ. Neo is immobilized but can create and move clones. Each clone:

* + Can be created in front of any button.
  + Can move one position left or right.

Whenever a clone stands in front of an unpressed button, it presses it instantly. Buttons can only be pressed once. To escape, the sequence of button weights pressed must be non- increasing.

Find the minimum number of clones Neo needs to create to press all buttons in such an order.

Parameters

* + t: Number of test cases (1 ≤ t ≤ 10⁴)
  + For each test case:
* n: Number of buttons (1 ≤ n ≤ 2⋅10⁵)
* a₁, a₂, ..., aₙ: Integer weights of buttons (1 ≤ aᵢ ≤ 10⁹)

Input

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 |  | | | |
| 5 |
| 4 | 3 | 2 | 1 | 5 |
| 3 |  |  |  |  |
| 1 | 1 | 1 |  |  |
| 6 |  |  |  |  |
| 7 | 8 | 1 | 5 | 9 2 |
| 10 |  |  |  |  |
| 1 | 7 | 9 | 7 | 1 10 2 10 10 7 |

Output

2

1

2

3

Given a binary string s of length n, a grid g of size n x n is constructed such that row i is the result of flipping the i-th character of s. A good set S of size k consists of unique cell coordinates (x, y) such that:

* + Each g[x][y] = 0
  + All cells in S are part of a single connected region (adjacent in 4 directions) Determine the maximum size of such a good set for each test case.

Parameters

* + t: Number of test cases (1 ≤ t ≤ 10³)
  + For each test case:
* n: Length of binary string (1 ≤ n ≤ 2⋅10⁵)
* s: Binary string of length n

Input

6

3

000

4

0010

7

1011001

4

0001

2

11

1

0

Output

3

9

10

7

1

0

For each integer x from 1 to n, define the string S(x) as:

1. Compute x+1
2. Concatenate x and x+1 as strings
3. Sort the resulting string’s digits in non-decreasing order Determine how many distinct strings S(x) exist for x ∈ [1, n]. Parameters
   * t: Number of test cases (1 ≤ t ≤ 10⁴)
   * For each test case:

* n: Upper bound (1 ≤ n ≤ 10⁹ - 2) Input

2

42

1337

Output

42

948

You are given a final binary grid n × m. Balls can be pushed from the left (into rows) or top (into columns). Rules:

* + Balls move forward to the first empty cell.
  + If they hit another ball, the new ball stops and pushes the existing one.
  + No push is allowed if the row/column is full.

Determine if it’s possible to reach the given final configuration.

Parameters

* + t: Number of test cases (1 ≤ t ≤ 10⁴)
  + For each test case:
* Two integers n and m (1 ≤ n, m ≤ 50)
* n binary strings of length m representing the grid

Input

5

3 3

001

001

110

3 3

010

111

010

3 3

111

111

111

3 3

000

000

000

3 3

000

000

001

Output

YES YES YES YES NO

You have two lanes starting with 1 person each. You're given n pairs of gates. Each gate can be:

• + a: Add a people

* + x a: Multiply by a

Gained people can be added to either lane. Find the maximum total people after all operations.

Parameters

* + t: Number of test cases (1 ≤ t ≤ 10⁴)
  + For each test case:
* n: Number of gate pairs (1 ≤ n ≤ 30)
* n lines, each with two gate operations Input

|  |  |  |  |
| --- | --- | --- | --- |
| 4 |  | | |
| 3 |
| + | 4 | x | 2 |
| x | 3 | x | 3 |
| + | 7 | + | 4 |
| 4 |  |  |  |
| + | 9 | x | 2 |
| x | 2 | x | 3 |
| + | 9 | + | 10 |
| x | 2 | + | 1 |
| 4 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| x | 2 | + | 1 |
| + | 9 | + | 10 |
| x | 2 | x | 3 |
| + | 9 | x | 2 |
| 5 |  |  |  |
| x | 3 | x | 3 |
| x | 2 | x | 2 |
| + 21 + 2 |  |  |  |
| x 2 x 3 |  |  |  |
| + 41 x 3 |  |  |  |

Output

32

98

144

351

You are given a string s consisting of characters A and B. Your task is to split it into blocks of length 1 and 2 such that:

* + No more than a blocks equal to "A",
  + No more than b blocks equal to "B",
  + No more than ab blocks equal to "AB",
  + No more than ba blocks equal to "BA",
  + Blocks "AA" and "BB" are prohibited.

Each character of s must belong to exactly one block.

Parameters

* + t: Number of test cases
  + For each test case:
* s: A string of characters A and/or B
* a, b, ab, ba: Maximum allowed counts for "A", "B", "AB", and "BA" blocks respectively

Input

|  |  |
| --- | --- |
| 7  A |  |
| 0 0 | 10 10 |
| B |  |
| 0 1 | 0 0 |
| ABA |  |
| 0 0 | 1 1 |

ABBABAAB 5 5 0 0

ABABBAABBAAB 1 1 2 3 ABBBBAB

0 3 2 0

BAABBA 1 3 2 0

Output

NO YES NO YES YES YES NO

You have two bags. Initially, the first contains n numbers, the second is empty. You can:

1. Move any number from the first to the second.
2. Increase a number in the first by 1, if the same number exists in the second.

Can you make both bags identical?

Parameters

* + t: Number of test cases
  + For each test case:
* n: Number of elements (even number)
* a: Array of integers (length n)

Input

|  |  |
| --- | --- |
| 9 |  |
| 2 |  |
| 1 | 1 |
| 2 |  |
| 2 | 1 |
| 4 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 4 | 4 |  |  |  |
| 4 |  |  |  |  |  |
| 3 | 4 | 3 | 3 |  |  |
| 4 |  |  |  |  |  |
| 2 | 3 | 4 | 4 |  |  |
| 6 |  |  |  |  |  |
| 3 | 3 | 4 | 5 | 3 | 3 |
| 6 |  |  |  |  |  |
| 2 | 2 | 2 | 4 | 4 | 4 |
| 8 |  |  |  |  |  |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 4 |
| 10 |  |  |  |  |  |  |

9 9 9 10 10 10 10 10 10 10

Output

Yes No Yes Yes No Yes No Yes Yes

Given a non-zero array a of length n, you start with 0 coins. Repeatedly:

1. Choose index i (1 ≤ i ≤ n).
2. Gain |a[i]| coins.
3. If a[i] < 0, cut array from the beginning to a[i−1].
4. If a[i] > 0, cut array from a[i+1] to the end. Maximize your coin count.

Parameters

* + t: Number of test cases
  + For each test case:
* n: Array length
* a: Array of integers (non-zero)

Input

3

6

3 1 4 -1 -5 -9

6

-10 -3 -17 1 19 20

1

1

Output

23

40

1

There are n pupils with heights a[i]. The lineup is comfortable if a[i] \* 2 ≥ a[i+1] for all valid i.

Each pizza increases any pupil's height by 1. Distribute at most k pizzas to maximize max(a) while keeping the lineup comfortable.

Parameters

* + t: Number of test cases
  + For each test case:
* n, q: Number of pupils, number of queries

|  |  |  |
| --- | --- | --- |
| o | a: Initial heights |  |
|  | o q integers: k₁, ..., | kq: Number of pizzas for each query |
| Input |  |  |
| 3 |  |  |
| 2 1 |  |  |
| 10 20 |  |  |
| 10 |  |  |
| 6 7 |  |  |
| 3 1 2 | 4 5 6 |  |
| 1 |  |  |
| 2 |  |  |
| 4 |  |  |
| 8 |  |  |
| 16 |  |  |
| 32 |  |  |
| 64 |  |  |
| 10 4 |  |  |
| 1 2 4 | 8 16 32 64 128 256 512 |  |
| 10 |  |  |
| 100 |  |  |
| 1000 |  |  |
| 10000 |  |  |

Output

26

7 8 10 12 19 35 67

513 560 1011 10001

There are n lilypads, Alice and Bob start on different pads a and b.

On each turn, the frog must jump left or right if not occupied. Alice moves first. If a frog has no valid move, it loses. Determine if Alice can guarantee a win.

Parameters

* + t: Number of test cases
  + For each test case:
* n: Number of lilypads
* a, b: Starting positions of Alice and Bob Input

5

2 1 2

3 3 1

4 2 3

5 2 4

7 6 2

Output

NO YES NO YES YES

# Q1. Problem Statement

You are given a set of items, each with a specific weight and value. Your task is to determine the maximum value that can be obtained by selecting a combination of these items, considering a knapsack with a limited capacity. You are allowed to take fractions of items into the knapsack.

Write a program to implement the fractional knapsack problem.

# Input format:

The input consists of several lines, each representing an item.Each item is represented by two

integers separated by a space: the weight of the item followed by its value.The input terminates with a line containing a single integer -1. After the items, there is an integer representing the maximum weight capacity.

# Output format:

The output prints "The maximum value of the current list is: ", followed by a double value, representing the maximum value that can be obtained by selecting fractions of items to fit into the knapsack, rounded off to two decimal points.

# Code constraints:

1 ≤ weight and values ≤ 1000

1. ≤ capacity ≤ 100

# Sample test cases:

**Input 1:**

10 60

20 100

30 120

-1

50

# Output 1:

The maximum value of the current list is: 240.00

# Explanation:

We have 3 items:

Item 1: weight = 10, value = 60 → value/weight = 6.0

Item 2: weight = 20, value = 100 → value/weight = 5.0

Item 3: weight = 30, value = 120 → value/weight = 4.0 Capacity of knapsack = 50.

Take all of Item 1 (weight 10): Remaining capacity = 40, total value = 60.

Take all of Item 2 (weight 20): Remaining capacity = 20, total value = 160. Take 20/30 = 2/3 of Item 3:

Value = (2/3) × 120 = 80

Remaining capacity = 0, total value = 240. Thus, the maximum value is 240.00.

# Q2. Problem Statement

Charlie is the Physical Education trainer in a school. The students of a class are standing in a line. The students can have the same or different weights. He wants to make the sum of the weights of alternative students equal to the sum of the weights of other alternative students. But he can’t achieve this in a simple way.

First of all, he needs to figure out how many ways he can remove any student from the line and achieve his goal.

Can you help Charlie find a number of possible ways?

**Example Input:**

**5**

**6 1 7 2 1**

**Output:**

**1**

**Explanation:**

He can remove the second student from the line and now the order will be 6,7,2,1. And now he can easily achieve the goal. 6+2 = 7+1 = 8. So there is one possible way and the output is 1.

# Input format:

The first line of input denotes the size of an array, N.

The next line contains N space-separated integer elements of the array.

# Output format:

The output prints the number of possible ways Charlie can remove any student from the line to

achieve his goal.

**Code constraints:**

**1 < N < 10001**

**1 ≤ array[i] ≤ 100001 Sample test cases: Input:**

**5**

**6 1 7 2 1**

**Output: 1**

**Input: 5**

**2 2 2 2 2**

**Output:**

**5**

**Q3. Problem Statement**

You are given the time at which customers enter a barbershop and leave after the service. You need to find the minimum number of barbers required so that no customer waits for the service but is served instantly.

Use **greedy algorithm** to solve the given problem.

# Input format:

The first line of input is an integer value representing the total number of customers n. The second line is space-separated integer values representing the arrival time.

The third line is space-separated integer values representing the leaving time.

# Output format:

The output prints the number of barbers required.

# Code constraints:

1. ≤ n ≤ 5
2. ≤ arrival time & leaving time ≤ 10

# Example:

**Input:**

5

1 2 3 4 5

1. 3 4 6 7

# Output:

2

# Explanation:

All men arriving at times 1, 3, and 5 can be served by Barber 1. Here, Barber 2 serves all men who enter the shop at times 2 and 4, so another barber is needed. Therefore, a total of two people is required.

# Sample test cases:

**Input 1:**

5

1 2 3 4 5

2 3 4 6 7

# Output 1:

2

# Input 2:

2

1 3

2 4

# Output 2:

1

# Q4. Problem Statement

You are working on a smart detection tool for essays. Given a large document, you need to search for instances of a particular text using the Rabin-Karp algorithm.

Implement the **Rabin-Karp** algorithm to find and return all starting positions of the text in the document.

# Input format:

The first line consists of the document text.

The second line consists of the text to be searched.

# Output format:

The output prints the starting indices of all occurrences of the sentence in the document, separated by spaces.

# Code constraints:

The document text can hold up to 1000 characters. The text to be searched can hold up to 100 characters.

The text to be searched is case-sensitive.

# Sample test cases:

**Input:**

The quick brown fox jumps over the lazy dog. The quick brown fox is clever. quick brown fox

# Output:

4 49

# Input:

The early bird catches the worm. Persistence pays off in the end.

bird **Output:** 10

Q5. Problem Statement

Given a string S that consists of only alphanumeric characters and dashes. The string is separated into N + 1 groups by N dashes. Also given an integer K. We want to reformat the string S, such that each group contains exactly K characters, except for the first group, which could be shorter than K but still must contain at least one character. Furthermore, a dash must be inserted between two groups, and you should convert all lowercase letters to uppercase. Return the reformatted string.

# Input:

5F3Z-2e-9-w

4

# Output:

5F3Z-2E9W

# Explanation:

The string S has been split into two parts; each part has 4 characters.

**Note:** Two extra dashes are not needed and can be removed.

# Input:

2-5g-3-J

2

# Output:

2-5G-3J

# Explanation:

The string s has been split into three parts; each part has 2 characters except the first part as it could be shorter as mentioned above.

# Input format:

The first line contains a string **S** which represents the original string that needs to be reformatted. The string can contain alphanumeric characters and dashes (-).

The second line contains an integer **K** which represents the length of each group after reformatting.

# Output format:

The output is a single string that represents the reformatted version of the input string S, grouped

according to the integer K.

# Code constraints:

1 ≤ S.length() ≤ 100

S contains only alphanumeric characters and dashes (-). 1 ≤ K ≤ 100

# Sample test cases: Input 1:

2-5g-3-J

2

# Output:

2-5G-3J

# Input:

5F3Z-2e-9-w

4

# Output:

5F3Z-2E9W

# Q6. Problem Statement

Given a set of integers, the task is to divide it into two sets S1 and S2 such that the absolute difference between their sums is minimum.

If there is a set S with n elements, then if we assume Subset1 has m elements, Subset2 must have n- m elements and the value of abs(sum(Subset1) – sum(Subset2)) should be minimum.

To achieve this, use **dynamic programming** to determine the closest possible sum to half of the total sum of the set.

# Input format:

The first line contains an integer n, representing the number of elements in the array.

The second line contains n space-separated integers, representing the elements of the array.

# Output format:

The output prints a single integer representing the minimum difference between the sums of two subsets of the given array.

# Code constraints:

1 ≤ n ≤ 10

1. ≤ elements ≤ 100

# Example Input

4

1. 6 11 5

# Output

1

# Explanation:

Subset1 = {1, 5, 6}, sum of Subset1 = 12 Subset2 = {11}, sum of Subset2 = 11

# Sample test cases:

**Input 1:**

6

3 1 4 2 2 1

# Output 1:

1

# Input 2:

4

1 6 11 5

# Output 2:

1

# Q7. Problem Statement

Hannah is monitoring a live data stream and needs to identify if any segment of the stream is an anagram of a given pattern. An anagram is a rearrangement of the letters in the pattern.

She needs to implement the Rabin-Karp algorithm to detect and return all starting indices where an anagram of the pattern appears in the data stream. Help her with the program.

# Input format:

The first line consists of a string.

The second line consists of a string for which Hannah wants to find anagrams in the stream.

# Output format:

The output prints the starting indices of all segments in the stream where an anagram of the pattern

appears, separated by spaces.

Code constraints:

Both the stream and pattern contain only lowercase English letters.

# Sample test cases:

**Input 1:**

cbaebabacd abc

# Output 1:

Anagram found at index 0 Anagram found at index 6 **Input 2:**

xyzzyxzyxzyxzyxyz xyz

# Output 2:

Anagram found at index 0 Anagram found at index 3 Anagram found at index 4 Anagram found at index 5 Anagram found at index 6 Anagram found at index 7 Anagram found at index 8 Anagram found at index 9 Anagram found at index 10 Anagram found at index 11 Anagram found at index 12 Anagram found at index 14

# Q8. Problem Statement

Nira is studying the **N-Queens problem** and needs to write a program to solve it. The program should find a valid arrangement of queens on a size N x N board, ensuring that no two queens can attack each other. Once a solution is found, the program should display the board with the queens correctly placed.

# Input format:

The input consists of an integer **N**, representing the size of the chessboard (N x N).

# Output format:

The output prints the configuration of the chessboard, with queens placed so that no two queens threaten each other.

# Code constraints:

4 ≤ N ≤ 20 **Example Input:**

4

# Output:

0 0 1 0

1 0 0 0

0 0 0 1

0 1 0 0

**Explanation:** The output represents a valid arrangement of queens on the 4x4 chessboard, where 1 indicates the position of a queen and 0 indicates an empty cell.

In the first row, a queen is placed in the third column.

In the second row, a queen is placed in the first column. In the third row, a queen is placed in the fourth column. In the fourth row, a queen is placed in the second column.

This arrangement of queens on the board ensures that no two queens threaten each other.

# Sample test cases:

**Input:**

4

# Output:

0 0 1 0

1 0 0 0

0 0 0 1

0 1 0 0

# Input:

6

# Output:

0 0 0 1 0 0

1 0 0 0 0 0

0 0 0 0 1 0

0 1 0 0 0 0

0 0 0 0 0 1

0 0 1 0 0 0

# Q9. Problem Statement

In a bioinformatics project, you are given a long DNA sequence. You need to find occurrences of a specific gene sequence within this DNA strand. Use the **Rabin-Karp** algorithm to locate all occurrences of the gene sequence within the DNA strand and return their starting indices.

# Input format:

The first line consists of a string representing a long DNA sequence consisting of characters 'A', 'C', 'G', 'T'. The second line consists of the specific gene sequence to search for.

# Output format:

The output prints the starting indices of all occurrences of the sentence in the document,

separated by spaces.

# Code constraints:

The DNA strand string can hold up to 1000 characters.

The gene sequence to be searched can hold up to 100 characters. The gene sequence to be searched is case-sensitive.

# Sample test cases: Input 1:

AGCTTAGCTAGCTGCTAAGCTT AGCT

# Output:

0 5 9 17

**Input:** AGCTCAG CA

# Output:

4

Q10. Problem Statement

Given the dimension of a sequence of matrices in an array arr[], where the dimension of the ith matrix is (arr[i-1] \* arr[i]), the task is to find the most efficient way to multiply these matrices together such that the total number of element multiplications is minimum.

# Code constraints:

1 ≤ n ≤ 10

1. ≤ elements ≤ 100

# Example Input:

5

40 20 30 10 30

# Output:

26000

# Explanation:

There are 4 matrices of dimensions 40×20, 20×30,

30×10, 10×30.

Let the input 4 matrices be A, B, C, and D.

The minimum number of multiplications is obtained by

putting parenthesis in following way (A(BC))D. The minimum is 20\*30\*10 + 40\*20\*10 + 40\*10\*30 **Example 2**

# Input

5

1. 2 3 4 3

# Output:

30

# Explanation:

There are 4 matrices of dimensions 1×2, 2×3, 3×4,

4×3.

Let the input 4 matrices be A, B, C, and D.

The minimum number of multiplications is obtained by

putting parenthesis in following way ((AB)C)D. The minimum number is 1\*2\*3 + 1\*3\*4 + 1\*4\*3 = 30 **Input format:**

The first line contains an integer n, representing the number of matrices.

The second line contains n integers, where the ith integer represents the dimension of the matrix. **Output format:**

The output prints a single integer representing the minimum number of scalar multiplications needed to multiply the chain of matrices.

# Sample test cases: Input 1:

5

40 20 30 10 30

# Output 1:

26000

# Input 2:

5

1 2 3 4 3

# Output 2:

30

# Q11. Problem Statement

At a cybersecurity agency, around 100 encrypted messages come in for processing. Since the length of the messages is quite long, manual labor does not help much. Hence, you are appointed to create a program that helps in decoding the messages, given the following information:

Messages come in a binary array format, i.e., an array where the elements are 0 and 1. A meaningful message is formed if all the 1s are grouped together.

The number of swaps must be minimal to get a meaningful message.

For example. If an array is 1 0 0 1 0 1, then the minimum number of swaps is 1 so that all 1s come together, thereby forming 0 0 0 1 1 1.

# Input Format:

The first line has the number of elements in the binary array. The second line consists of the elements separated by a space.

# Output Format:

The output will display the minimum number of swaps required to group all 1s together, in the

following format:

Minimum number of swaps to group all ones together: <number\_of\_swaps>

# Code Constraints:

1 ≤ n ≤ 30

The elements of the array are either 0 or 1.

# Sample Input:

6

1 0 0 1 0 1

# Sample Output:

Minimum number of swaps to group all ones together: 1

# Sample Input:

4

1 0 0 1

# Sample Output:

Minimum number of swaps to group all ones together: 1

# Q12. Problem Statement

Siva is a math teacher. He conducted a class test for the students. To avoid mall practices in the test, he gave two sets of questions. To identify the answer sheets of different sets, he asked the students to mark them as 0 in their answer sheets if they had taken the first set of questions and 1 for the second set of questions. After collecting the answer sheets, he needs to separate the answer sheets into 0 and 1 Help Siva with this task.

# Input Format:

The first line of the input consists of the value of n. The next input is the array of elements separated by a space, which is either '0' or '1'.

# Output Format:

The output displays the arranged array so that 0s and 1s are separated.

Code Constraints:

1 ≤ n ≤ 20

1. ≤ each element ≤ 1

# Sample Input:

6

1. 0 0 1 1 1

# Sample Output:

0 0 1 1 1 1

# Sample Input:

5

1 0 0 1 0

# Sample Output:

0 0 0 1 1

# Q13. Problem Statement

Guru is teaching a math class where he asks his students to work on finding pairs of numbers that add up to a given sum. He has an array of numbers, and he wants to help his students find all pairs whose sum equals a specified target value. To make it more interesting, he gives the task as a challenge to his students. The task is to write a program that helps Guru's students efficiently find and print all such pairs in an array that sum up to the target value.

# Sample Input:

18

8

7 12 5 10 13 3 8 9

# Sample Output:

5 + 13 = 18

10 + 8 = 18

# Sample Input

100

10

20 50 30 80 45 60 75 40 20 60

# Sample Output:

20 + 80 = 100

60 + 40 = 100

**Note:** Print only unique value pairs in the order they first appear in the array. Once a pair of values has been printed, any other occurrences of the same pair (regardless of position) should be ignored.

# Input Format:

The first line of input consists of an integer sum, representing the target sum to be found.

The second line of input consists of an integer n, representing the number of elements in the array.

The third line of input consists of n space-separated integers, representing the elements of the array.

# Output Format:

The first line of output consists of all pairs (if any) whose sum is equal to the given value. Each output line should be in the format:

a + b = sum, where a and b are the numbers from the array that sum up to the given value

# Code Constraints:

1 ≤ n ≤ 105

103 ≤ targetSum ≤ 103

-103 ≤ arr[i] ≤ 103

# Q14. Problem Statement

Consider a fleet of soldiers from a country who are being assembled for a rehearsal session. The enemy country secretly surrounds them and has a special strategy to kill the soldiers in a particular

pattern. Assume that the soldiers are standing in a single straight line. The enemies will repeatedly scan through this line and kill soldiers who are all matching the given pattern.

Find the list of soldiers who are surviving at last or find if all of them are killed. Here soldiers are represented as alpha-numeric letters, irrespective of cases.

Implement the "FindRemainingSoldiers" class with the "defeatSoldiers(String soldiers, String

pattern)" method to find the left-out soldiers if any, else print "Defeat" as a result.

# Sample Input:

xAbcyAAbcbAbccz Abc

# Sample Output:

xyz

# Explanation:

soldiers: xAbcyAAbcbAbccz pattern: Abc Iteration:

0: xAbcyAAbcbAbccz

1: xyAAbcbAbccz

2: xyAbAbccz

3: xyAbcz

4: xyz

# Input Format:

The first line of input represents "a fleet of soldiers" (an alphanumeric string). The second line of input represents the kill "pattern" (alpha-numeric string).

# Output Format:

The first line of output prints the remaining sequence of soldiers as a string, or "Defeat" if all soldiers

have been killed.

# Code Constraints:

1. ≤ length of soldiers ≤ 100 1 ≤ length of pattern ≤ 100

# soldiers and pattern only consist of alphanumeric characters (0-9, a-z, A-Z).

Input 2:

AbAAbcbcc Abc **Output 2:**

Defeat

# Input 3:

AAbAbccc Abc **Output 3:**

Ac

# Q15. Problem Statement

Manoj is working as a ticket checker at Satyam Cinemas. Due to overcrowding, everyone is asked to form a queue. so people who came together got separated in the queue. People who come together are considered to be a single group. Always assume that the person of each group who is standing first in the queue holds the ticket. You have to check the ticket of each person and make sure all the members of his group enter the screen along with him.

# Sample Input:

abcaaubcc

# Sample Output:

aaabbcccu

# Explanation

Imagine **"abcaaubcc"** as a queue, and each alphabet represents a person standing in a queue. People of the same group are represented using the same alphabet. The first person you will be checking is **"a".** You have to allow all the people in group "**a".** Then the list will be like this: "**aaabcubcc".** The next person you will be checking is **"b"** Now the list will be updated as **"aaabbcucc".** After checking c list, it will be **"aaabbcccu".** Then after checking "**u**" the list will be the same as "**aaabbcccu**"

# Sample Input:

zaynzaakony

# Sample Output:

zzaaayynnko

# Explanation

The output for the given test case will be "zzaaayynnko". It groups the characters 'z', 'z', 'a', 'a', 'a', 'y', 'y', 'n', 'n', 'k', 'o' together, which represent the grouped people standing in the queue.

**Note:** People of the same group will be represented using the same alphabetic character. Also, only lowercase letters will be used.

# Input format:

The first line of input consists of a string **queue**, representing the people standing in the queue. Each character represents a person, and people in the same group are represented by the same character. **Output format:**

The output prints the characters in the order they would enter the hall after grouping all people of the same group together, maintaining their original order of appearance.

# Code constraints:

The string contains only lowercase letters (a-z).

1 ≤ length of string ≤ 100.

# Sample test cases:

**Input 1:**

abcaaubcc

# Output 1:

aaabbcccu

# Input 2:

zaynzaakony

# Output 2:

zzaaayynnko

# Q16. Problem Statement

Given a string **s** that consists of lowercase letters, return the length of the longest palindrome that can be built with those letters. Letters are case- sensitive. For example, "Aa" is not considered a palindrome here.

**Input:** s = "abccccdd"

# Output: 7

**Explanation:** One longest palindrome that can be built is "dccaccd", whose length is 7.

# Input format:

The input consists of a string.

# Output format:

The output prints the length of the longest palindrome that can be built with the given letters.

#### Code constraints:

The input string can be up to 50 characters long.

# Sample test cases:

**Input 1:**

abccccdd

# Output 1:

7

# Input 2:

madam

# Output 2:

5

# Input 3:

abc

# Output 3:

1

# Input 4:

Aa

# Output 4:

1

# Q17.Problem Statement

We are given n-platform and two main running railway track for both direction. Trains which needs to stop at your station must occupy one platform for their stoppage and the trains which need not to stop at your station will run away through either of main track without stopping. Now, each train has three value first arrival time, second departure time and third required platform number.

We are given m such trains you have to tell maximum number of train for which you can provide stoppage at your station.

Examples:

# Input: n = 3, m = 6

Train no.| Arrival Time |Dept. Time | Platform No.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | | 10:00 | | | 10:30 | | | 1 |
| 2 | | 10:10 | | | 10:30 | | | 1 |
| 3 | | 10:00 | | | 10:20 | | | 2 |
| 4 | | 10:30 | | | 12:30 | | | 2 |
| 5 | | 12:00 | | | 12:30 | | | 3 |
| 6 | | 09:00 | | | 10:05 | | | 1 |

**Output: Maximum Stopped Trains = 5 Explanation:** If train no. 1 will left to go without stoppage then 2 and 6 can easily be accommodated on platform 1. And 3 and 4 on platform 2 and 5 on

platform 3.

# Input: n = 1, m = 3

Train no.|Arrival Time|Dept. Time | Platform No.

1 | 10:00 | 10:30 | 1

2 | 11:10 | 11:30 | 1

3 | 12:00 | 12:20 | 1

**Output: Maximum Stopped Trains = 3 Explanation:** All three trains can be easily stopped at platform 1.

**Input format:** two integers **n** no of

platforms, **m** number of trains, and array trains[] as input parameter and

**Output format:** returns an integer.

# Code Constraints:

1 <= N <= 100

1 <= M <= 104

# Q18. Problem Statement

A new variety of rice has been brought in supermarket and being available for the first time, the quantity of this rice is limited. Each customer demands the rice in two different packaging of size a and size b. The sizes a and b are decided by staff as per the demand. Given the size of the packets a and b, the total quantity of rice available d and the number of customers n, find out maximum number of customers that can be satisfied with the given quantity of rice. Display the total number of customers that can be satisfied and the index of customers that can be satisfied.

# Note: If a customer orders 2 3, he requires 2 packets of size a and 3 packets of size b. Assume indexing of customers starts from 1.

**Input format:** The first line of input contains two integers n and d; next line contains two integers a and b. Next n lines contain two integers for each customer denoting total number of bags of size a and size b that customer requires.

**Output format:** Print the maximum number of customers that can be satisfied and in the next line print the space-separated indexes of satisfied customers.

# Input: n = 5, d = 5

a = 1, b = 1

2 0

3 2

4 4

10 0

0 1

# Output: 2

5 1

# Input: n = 6, d = 1000000000

a = 9999, b = 10000

10000 9998

10000 10000

10000 10000

70000 70000

10000 10000

10000 10000

# Output: 5

1 2 3 5 6

**Explanation:** In first example, the order of customers according to their demand is:

|  |  |
| --- | --- |
| Customer ID | Demand |
| 5 | 1 |
| 1 | 2 |
| 2 | 5 |

|  |  |
| --- | --- |
| 3 | 8 |
| 4 | 10 |

From this, it can easily be concluded that only customer 5 and customer 1 can be satisfied for total demand of 1 + 2 = 3. Rest of the customer cannot purchase the remaining rice, as their demand is greater than available amount.

# Code Constraints:

1 <= n <= 10^5

(Up to 100,000 customers) 1 <= d <= 10^12

(Available rice quantity is very large, up to 1 trillion) 1 <= a, b <= 10^6 (Packet sizes a and b can be up to 1 million grams or units)

1. <= number of bags requested by a customer <= 10^6 Customer indexing is from 1 to n

Sum of rice needed by all customers will not exceed 10^18. The customer demands are always non-negative integers.

You can serve customers in any order to maximize number of customers.

# Q19. Problem Statement

You are given an array of ‘N’ distinct integers and an integer ‘X’ representing the target sum.

You have to tell the minimum number of elements you have to take to reach the target sum ‘X’.

# Note:

You have an infinite number of elements of each type.

# For example:

If N=3 and X=7 and array elements are [1,2,3].

Way 1 - You can take 4 elements [2, 2, 2, 1] as 2 + 2

+ 2 + 1 = 7.

Way 2 - You can take 3 elements [3, 3, 1] as 3 + 3 +

1 = 7.

Here, you can see in Way 2 we have used 3 coins to reach the target sum of 7. Hence the output is 3.

# Input Format:

The first line of input contains an Integer T denoting the number of test cases.

The first line of each test contains 2 space separated integers, N and X denoting the size of the array and given target sum.

The second line of each test contains contains N elements space separated denoting the elements of the array.

# Output format:

print a separate line integer output for each test case.

# Code Constraints:

1. <= T <= 10

1 <= N <= 15

1 <= nums[i] <= (2^31) - 1

1 <= X <= 10000

Note:All the elements of the “nums” array will be unique.

# Sample Input 1:

2

1. 7

1 2 3

1 0

1

# Sample output 1:

3

0

# Explanation For Sample Output 1:

For the first test case,

Way 1 - You can take 4 elements [2, 2, 2, 1] as 2 + 2

+ 2 + 1 = 7.

Way 2 - You can take 3 elements [3, 3, 1] as 3 + 3 +

1 = 7.

Here, you can see in Way 2 we have used 3 coins to reach the target sum of 7. Hence the output is 3.

For the second test case To reach X = 0, you don’t need to take any elements from the array. The sum is already 0, so the total number of elements used is 0.

# Sample Input 2:

2

3 4

12 1 3

1. 11
2. 1

# Sample output 2:

2

6

# Q20. Problem Statement

A **path** in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence **at most once**.

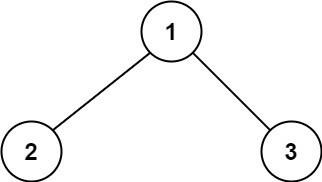
Note that the path does not need to pass through the root.

The **path sum** of a path is the sum of the node's values in the path.

Given the root of a binary tree, return *the maximum* ***path sum*** *of any* ***non-empty*** *path*.

**Input format:** First line: Space-separated list of integers representing node values in **level-order traversal**. If a node is absent (null), use "null" in its place.

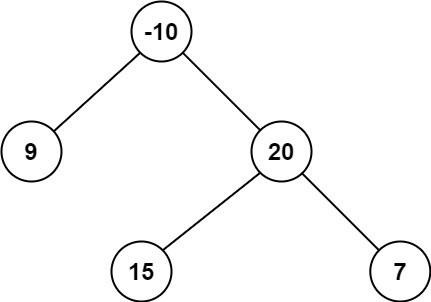
**Output format:** single integer — the maximum path sum from any non-empty path.



**Input:** root = [1,2,3]

# Output: 6

**Explanation:** The optimal path is 2 -> 1 -> 3 with a path sum of 2 + 1 + 3 = 6.



**Input:** root = [-10,9,20,null,null,15,7]

# Output: 42

**Explanation:** The optimal path is 15 -> 20 -> 7 with a path sum of 15 + 20 + 7 = 42.

# Code Constraints:

The number of nodes in the tree is in the range [1, 3 \* 104].

-1000 <= Node.val <= 1000

# Q21. Problem Statement

Given an array arr of n integers, where arr[i] represents price of the stock on the ith day. Determine the **maximum profit** achievable by buying and selling the stock at most once. The stock should be purchased before selling it, and both actions cannot occur on the same day.

# Input Format:

First line: An integer n, the size of the array (number of days).

Second line: n space-separated integers representing arr[i], the stock prices on each day.

# Output Format:

A single integer: the maximum profit achievable.

If no profit is possible, print 0

**Input:** arr = [10, 7, 5, 8, 11, 9]

# Output: 6

**Explanation:** Buy on day 3 (price = 5) and sell on day 5 (price = 11), profit = 11 - 5 = 6.

**Input:** arr = [5, 4, 3, 2, 1]

# Output: 0

**Explanation:** In this case, no transactions are made. Therefore, the maximum profit remains 0.

# Code Constraints:

1 <= n<= 105

0 <= arr[i] <= 106

# Q22. Problem Statement

The three stacks s1, s2 and s3, each containing positive integers, are given. The task is to find the maximum possible equal sum that can be achieved by removing elements from the top of the stacks. Elements can be removed from the top of each stack, but the final sum of the remaining elements in all three stacks must be the same. The goal is to determine the maximum possible equal sum that can be achieved after removing elements.

Note: The stacks are represented as arrays, where the first index of the array corresponds to the top element of the stack.

**Input Format:** First line contains 3 integer inputs representing the size of the stack 1,stack 2 and stack 3 respectively.

Second third and fourth lines represent the space separated integers representing the stack elements of stack 1 stack 2 and stack 3 respectively.

# Output Format:

A single integer: the maximum equal sum achievable after removing some top elements.

# Input:

1. 4 2
2. 2 3

1 1 2 3

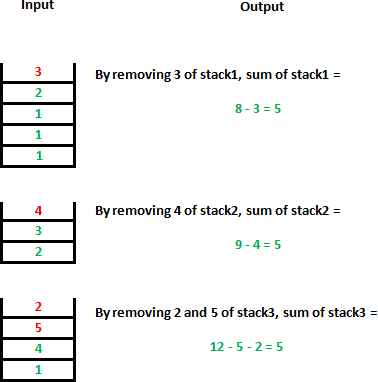
1 4

# Output:

5

# Explanation:

We can pop 1 element from the 1st stack, and 2 elements from the 2nd stack. Now remaining elements yield the equal sum of the three stacks, that is 5.



# Code Constraints:

1 <= n1, n2, n3 <= 104

(where n1, n2, n3 are the number of elements in stacks s1, s2, and s3 respectively) 1 <= element <= 105

(elements are positive integers)

Stacks may have different sizes (n1 ≠ n2 ≠ n3).

You can only remove elements from the top (start of the array). At any time, you can remove zero or more elements from any stack. The stacks must be non- empty initially (i.e., each has at least 1 element). Final sum must be equal for all three stacks after removals. You need to maximize the common sum after removals. If it's not possible to achieve any common sum, the result must be 0.

# Q23. Problem Statement

Given a weighted, undirected, and connected graph with V vertices and E edges, your task is to find the sum of the weights of the edges in the Minimum Spanning Tree (MST) of the graph. The graph is represented by an adjacency list, where each element adj[i] is a vector containing vector of integers. Each vector represents an edge, with the first integer denoting the endpoint of the edge and the second integer denoting the weight of the edge.

# Input Format:

First line: Two integers V and E — the number of vertices and edges.

Next E lines: Each line contains three integers u v w, meaning there is an undirected edge between vertex u and vertex v with weight w.

# Output Format:

A single integer: the sum of the weights of all edges in the Minimum Spanning Tree (MST).

# A diagram of a triangle with green circles and numbers AI-generated content may be incorrect.Input:

1. 3

0 1 5

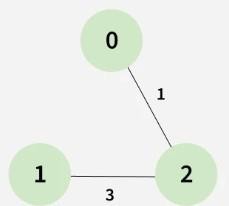
1 2 3

0 2 1

# Output:

4

Explaination: The Spanning Tree resulting in a weight of 4 is shown below.



# Code Constraints:

2 ≤ V ≤ 1000

V-1 ≤ E ≤ (V\*(V-1))/2

1 ≤ w ≤ 1000

The graph is connected and doesn't contain self- loops & multiple edges.

# Q24: Problem Statement

You are given the arrival times **arr[]** and departure times **dep[]** of all trains that arrive at a railway station on the same day. Your task is to determine the minimum number of platforms required at the station to ensure that no train is kept waiting.

At any given time, the same platform cannot be used for both the arrival of one train and the departure of another. Therefore, when two trains arrive at the same time, or when one arrives before another departs, additional platforms are required to accommodate both trains.

# Input format:

First line: An integer n, representing the number of trains.

Second line: n integers, representing the arrival times arr[] of the trains. Third line: n integers, representing the departure times dep[] of the trains.

All times are given in 24-hour format without a colon (e.g., 9:00 AM is 900, 11:30 AM is 1130, etc.).

# Output Format:

A single integer: minimum number of platforms needed so that no train waits**.**

# Sample Test cases:

**Input**: 6

900 940 950 1100 1500 1800

910 1200 1120 1130 1900 2000

**Output**: 3

**Explanation**: There are three trains during the time 9:40 to 12:00. So we need a minimum of 3 platforms. **Input**: arr[] = [900, 1235, 1100], dep[] = [1000, 1240,

1200]

**Output**: 1

**Explanation**: All train times are mutually exclusive. So we need only one platform.

# Input

3

1000 935 1100

1200 1240 1130

**Output**: 3

**Explanation**: All 3 trains have to be there from 11:00 to 11:30 Code Constraints:

1≤ number of trains ≤ 50000 0000 ≤ arr[i] ≤ dep[i] ≤ 2359

Note: Time intervals are in the 24-hour format(HHMM) , where the first two characters represent hour (between 00 to 23 ) and the last two characters represent minutes (this will be

<= 59 and

>= 0).

# Q25. Problem Statement

Given 2 arrays **deadline[],** and **profit[],** which represent a set of jobs, where each job is associated with a **deadline**, and a **profit**. Each job takes 1 unit of time to complete, and only one job can be scheduled at a time. You will earn the profit associated with a job only if it is completed by its deadline.

Your task is to find:

The **maximum number of jobs** that can be completed within their deadlines. The **total maximum profit** earned by completing those jobs.

# Input Format:

First line: Single integer n (number of jobs). Second line: n space-separated integers representing the deadline[] array.

Third line: n space-separated integers representing the profit[] array.

# Output Format:

Output two integers separated by a space: First integer: Maximum number of jobs done. Second integer: Maximum profit earned.

# Sample test cases:

**Input:** 4

1. 1 1 1

20 10 40 30

Output:

2 60

**Explanation:** Job1 and Job3 can be done with maximum profit of 60 (20+40).

**Input:**

**5**

**2 1 2 1 1**

**100 19 27 25 15**

**Output:**

**2 127**

**Explanation:** Job1 and Job3 can be done with maximum profit of 127 (100+27).

# Code Constraints:

1 ≤ deadline.size() == profit.size() ≤ 105 1 ≤ deadline[i] ≤ deadline.size() 1 ≤ profit[i] ≤ 500

# Q26. Problem Statement

A celebrity is a person who is known to all but does not know anyone at a party. A party is being organized by some people. A square matrix mat[][] (n\*n) is used to represent people at the party such that if an element of row i and column j is set to 1 it means ith person knows jth person. You need to return the index of the celebrity in the party, if the celebrity does not exist, return -1.

Note: Follow 0-based indexing.

# Input Format:

The first line contains an integer n, the number of people at the party (size of the matrix). The next n lines contain n integers, where each integer is either 0 or 1, representing the matrix mat[][].

# Output Format:

Print the index of the celebrity if one exists. Otherwise, print -1 if no celebrity exists. Examples:

# Input:

3

1 1 0

0 1 0

0 1 1

# Output: 1

Explanation: 0th and 2nd person both know 1st person. Therefore, 1 is the celebrity person.

# Input:

2

1 1

1 1

**Output**: -1

Explanation: Since both the people at the party know each other. Hence none of them is a celebrity person.

# Input:

1

1

# Output: 0

**Code Constraints:**

1 <= mat.size()<= 1000

0 <= mat[i][j]<= 1

mat[i][i] == 1

# Q27. Problem Statement

You are given an weighted directed graph, represented by an adjacency matrix, dist[][] of size n x n, where dist[i][j] represents the weight of the edge from node i to node j. If there is no direct edge, dist[i][j] is set to a large value (i.e., 108) to represent infinity. The graph may contain negative edge weights, but it does not contain any negative weight cycles.

Your task is to find the shortest distance between every pair of nodes i and j in the graph.

# Input Format:

The first line contains an integer n, the number of nodes in the graph (size of the matrix).

The next n lines contain n integers, where each integer represents the weight of the edge between nodes. A value of 108 indicates that there is no direct edge between the nodes (infinity).

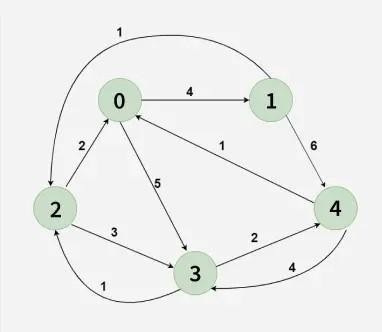
Note: Modify the distances for every pair in place. Output Format:

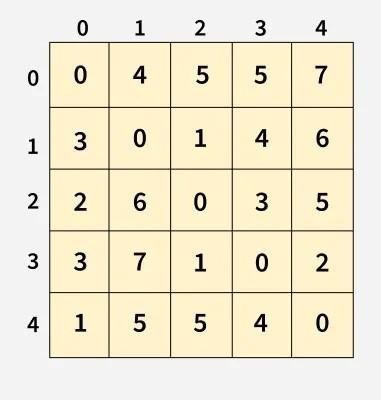
Print the updated adjacency matrix after applying the Floyd-Warshall algorithm, which gives the shortest distances between every pair of nodes in the graph.

**Input:** dist[][] = [[0, 4, 108, 5, 108], [108, 0, 1, 108,

6], [2, 108, 0, 3, 108], [108, 108, 1, 0, 2], [1, 108, 108,

4, 0]]





**Output:** [[0, 4, 5, 5, 7], [3, 0, 1, 4, 6], [2, 6, 0, 3, 5],

[3, 7, 1, 0, 2], [1, 5, 5, 4, 0]]

**Explanation:** Each cell dist[i][j] in the output shows the shortest distance from node i to node j, computed by considering all possible intermediate nodes.

# Constraints:

1 ≤ dist.size() ≤ 100

-1000 ≤ dist[i][j] ≤ 1000

# Q28. Problem Statement

Assume you are an awesome parent of some children and want to give your children some cookies. But, you should give each child at most one cookie. Each child **i** has a greed factor **greed[i]**, which is the minimum size of cookie that the child will be content with and each cookie **j** has a size **cookie[j]**. If **cookie[j] >= greed[i]**, we can assign the cookie **j** to **j** to the child **ith**, and the child **i** will be content. Find the maximum number of child that can be content.

# Input Format:

The first line contains two integers n and m, representing the number of children and the number of cookies.

The second line contains n integers, representing the greed factor of each child (greed[i]). The third line contains m integers, representing the size of each cookie (cookie[j]).

# Output Format:

**Example test cases:**

**Input:** 3 2

1 2 3

1 1

# Output: 1

**Explanation:** You can only assign cookie 0 or 1 to child 0.

# Input:

2 3

1 2

1 2 3

# Output: 2

**Explanation:** You can assign cookie 0 to child 0 and cookie 1 to child 1.

# Constraints:

1 ≤ greed.size() ≤ 105

1 ≤ cookie.size() ≤ 105

1 ≤ greed[i] , cookie[i] ≤ 109

# Q29. Problem Statement

Given an array of integers of size **N** and a number **K**., You must modify array **arr[]** exactly **K** number of times. Here modify array means in each operation you can replace any array element either **arr[i]** by **- arr[i]** or **-arr[i]** by **arr[i]**. You need to perform this operation in such a way that after K operations, the sum of the array must be maximum.

# Input Format:

The first line contains two integers N and K, where N is the number of elements in the array

and K is the number of modifications you must perform.

The second line contains N integers representing the array arr[].

# Output Format:

A single integer representing the maximum sum of the array after exactly K modifications.

# Example test cases:

**Input:**

1. 1

1 2 -3 4 5

# Output:

15

# Explanation:

We have k=1 so we can change -3 to 3 and sum all the elements to produce 15 as output.

# Input:

10 5

1. -2 5 -4 5 -12 5 5 5 20

# Output:

68

# Explanation:

Here we have k=5 so we turn -2, -4, -12 to 2, 4, and 12 respectively. Since we have performed 3 operations so k is now 2. To getnmaximum sum of array we can turn positive turned 2 into negative and then positive again so k is 0. Now sum is

5+5+4+5+12+5+5+5+20+2 = 68

# Constraints:

1 ≤ N,K ≤ 105

-109 ≤ Ai ≤ 109

# Q30. Problem Statement

There are N Mice and N holes that are placed in a straight line. Each hole can accomodate only 1 mouse. The positions of Mice are denoted by array A and the position of holes are denoted by array B.

A mouse can stay at his position, move one step right from x to x + 1, or move one step left from x to x −

1. Any of these moves consumes 1 minute.

Assign mice to holes so that the time when the last mouse gets inside a hole is minimized.

# Input Format:

First argument is an integer array A. Second argument is an integer array B. **Output Format:**

Return an integer denoting the minimum time when the last mouse gets inside the holes.

# Example test cases:

**Input:**

-4 2 3

0 -2 4

# Output:

2

# Explanation:

Assign the mouse at position (-4 to -2), (2 to 0) and

(3 to 4). The number of moves required will be 2, 2 and 1 respectively. So, the time taken will be 2.

# Input:

-2

6

# Output:

4

# Explanation:

Assign the mouse at position -2 to -6. The number of moves required will be 4. So, the time taken will be 4.

# Q31. Problem Statement

As it is Tushar's Birthday on March 1st, he decided to throw a party to all his friends at TGI

Fridays in Pune. Given are the eating capacity of each friend, the filling capacity of each dish and the cost of each dish. A friend is satisfied if the sum of the filling capacity of dishes he ate is equal to his capacity. Find the minimum cost such that all of Tushar's friends are satisfied (reached their eating capacity).

NOTE:

Each dish is supposed to be eaten by only one person. Sharing is not allowed. Each friend can take any dish an unlimited number of times.

There always exists a dish with a filling capacity of 1 so that a solution always exists.

# Input Format:

Friends: A: List of integers denoting eating capacity of friends separated by space. Capacity: B: List of integers denoting filling capacity of each type of dish.

Cost: C: List of integers denoting cost of each type of dish.

# Output Format:

A single integer representing the minimum cost to satisfy all the friends

# Example Input:

1. 6

1 3

1. 3

# Example Output:

14

**Explanation:** The first friend will take 1st and 2nd dish, the second friend will take 2nd dish twice. Thus, total cost = (5+3)+(3\*2)= 14

# Code Constraints:

1. <= Capacity of friend <= 1000 1 <= No. of friends <= 1000 1 <= No. of dishes <= 1000

# Q32. Problem Statement

Given an integer **A** you have to find the **number of ways** to fill a 3 x A board with 2 x 1 dominoes.

Return the answer modulo **109 + 7**.

# Input Format:

First and only argument is an integer **A**. **Output Format:**

Return an integer denoting the **number of ways** to fill a 3 x A board with 2 x 1 dominoes with modulo **109 + 7**.

# Example Test case:

Input:

2

Output:

3

Explanation:

Following are all the 3 possible ways to fill up a **3 x 2** board.



Input:

1

Output:

0

Explanation: Not a even a single way exists to completely fill the grid of size **3 x 1**.

Problem 1

In the land of Numerica, the Council of Integers faced a grand challenge issued by the wise Mathmagician:

"Rearrange yourselves in such a way that, when read side by side, you form the largest number possible.

Only then will you prove you might and earn the title of the Great Sequence!"

The integers need your help to determine the correct order to maximize their combined value. (Return a single 0, if the answer is 0.)

Parameters:

T :: INTEGER

The number of test cases.

T :: 1 → 100

N :: INTEGER

The number of integers in the array. N :: 1 → 1000

A :: INTEGER ARRAY

The N integers separated by spaces.

Each A[i] :: 1 → 10⁶

Case#: 1 Input:

1

5

101 21 92 97 203

Output:

97 92 21 203 101

Explanation:

The goal is to arrange the integers so their concatenation forms the largest possible number. E = [101, 21, 92, 97, 203]

Sorted with custom comparator (xy > yx), we get: 97, 92, 21, 203, 101 Resulting string: 979221203101 is the largest.

Case#: 2 Input:

1

4

0 0 0 0

Output:

0

Explanation:

All elements are zero. Any permutation gives 0000, which simplifies to a single 0. Case#: 3

Input:

1

6

8 90 89 908 898 9

Output:

9 90 908 89 898 8

Explanation:

Using the same custom sort logic:

Compare combinations like 90890 vs 90908, and reorder accordingly. Final sorted order: 9, 90, 908, 898, 89, 8

Resulting number: 9 90908898988 is the largest possible. Problem 2

The Professor has discovered that a vast amount of gold is hidden beneath the Bank of Spain. To execute a perfect heist, he first needs to calculate the total gold stored inside.

The gold is stored across a grid floor of size N x N, where each cell represents a room. The room number is defined as the sum of its row and column indices.

The amount of gold in a room is calculated as the absolute difference between the sum of even digits and the sum of odd digits in the room number.

You must compute the total gold units stored across all rooms in the bank for each test case. Parameters:

T :: INTEGER

The number of test cases.

T :: 1 → 10⁵

N :: INTEGER

The size of the grid (N x N).

N :: 1 → 10⁶

Case#: 1 Input:

1

1

Output:

2

Grid size: 1x1

Room (1,1) → Room number = 2

Even digits = 2, Odd digits = 0 → |2 - 0| = 2 Total = 2

Case#: 2 Input:

1

2

Output:

12

Grid size: 2x2 → Rooms: (1,1) → 2 → |2 - 0| = 2

(1,2) → 3 → |0 - 3| = 3

(2,1) → 3 → |0 - 3| = 3

(2,2) → 4 → |4 - 0| = 4

Sum = 2 + 3 + 3 + 4 = 12

Case#: 3 Input:

1

3

Output:

36

Grid size: 3x3 → Room numbers:

(1,1)=2, (1,2)=3, (1,3)=4

(2,1)=3, (2,2)=4, (2,3)=5

(3,1)=4, (3,2)=5, (3,3)=6

Gold units per room:

2, 3, 4, 3, 4, 5, 4, 5, 6

Total = 36

Problem 3

You are a cricket analyst preparing a presentation for team owners. You have access to N match scores of a player.

To highlight the player's performance, you want to display the longest increasing subsequence of these scores — this means a sequence that is strictly increasing and not necessarily from consecutive matches.

Find and return the length of this longest increasing subsequence (LIS) for each test case.

Winning the owners’ approval depends on this.

Parameters:

T :: INTEGER

The number of test cases.

T :: 2 → 9

N :: INTEGER

The number of scores.

N :: 2 → 999

scores :: INTEGER ARRAY

N space-separated integers denoting match scores.

Each score :: 1 → 299

Case#: 1 Input:

1

6

10 22 9 33 21 50

Output:

4

One possible longest increasing subsequence is: 10, 22, 33, 50

Length = 4

Case#: 2 Input:

1

5

5 4 3 2 1

Output:

1

All scores are decreasing. Any single score is a valid LIS of length 1. So the answer is 1.

Case#: 3 Input:

1

8

1 3 2 5 4 7 6 9

Output:

5

A longest increasing subsequence: 1, 2, 4, 6, 9

Length = 5

Problem 4

There are N people in a meeting. These N people are sitting in a circle. Each person has already shaken hands with their two adjacent neighbors (left and right).

Now, every person must shake hands with all the remaining people except the adjacent ones. No two persons shake hands more than once.

Find the number of additional handshakes required so that every pair of non-adjacent people shakes hands exactly once.

Parameters:

N :: INTEGER

The number of people sitting in a circle.

N :: 1 → 10^4

Case#: 1 Input:

2

Output:

0

Each person has already shaken hands with their only neighbor. No other person remains to shake hands with.

So, 0 handshakes are needed. Case#: 2

Input:

4

Output:

2

Total possible handshakes without restriction = C(N, 2) = 6 Initial handshakes with neighbors in a circle = N = 4 Remaining handshakes = 6 - 4 = 2

Case#: 3

Input:

6

Output:

9

Total possible handshakes = C(6, 2) = 15

Handshakes already done (one per neighbor per person, divided by 2 to avoid double-counting): 6 Remaining = 15 - 6 = 9

Problem 5

You are given an array of integers V1, V2, ..., VN,

Your task is to count the number of non-decreasing subarrays. A subarray V[i, j] is considered non-decreasing if:

Vi, Vi+1, ..., Vj

Each individual element is trivially a non-decreasing subarray.

You are required to compute this count for each test case and output it. Parameters:

T :: INTEGER

The number of test cases.

T :: 1 → 10

N :: INTEGER

The number of elements in the array.

N :: 1 → 10⁵

V :: INTEGER ARRAY

The array elements.

Each element Vᵢ :: 1 → 10⁹

Case#: 1

Input:

1

4

1 4 2 3

Output:

6

All valid non-decreasing subarrays:

(1), (1 4), (4), (2), (2 3), (3)

Total = 6

Case#: 2 Input:

1

1

5

Output:

1

Only one subarray possible: (5), which is non-decreasing. Total = 1

Case#: 3 Input:

1

5

1. 2 2 2 2

Output:

15

All elements are equal → every subarray is non-decreasing. Total subarrays = N \* (N + 1) / 2 = 5 \* 6 / 2 = 15

Problem 6

A string is called an "ab string" if it is composed of characters 'a' and 'b' and follows either of these patterns:

All 'a' characters come before all 'b' characters: "aa...abbb...b" All 'b' characters come before all 'a' characters: "bb...baaa...a" You are given a string S consisting only of 'a' and 'b'.

You are allowed to remove any number of characters (possibly zero) from the string.

Your task is to determine the maximum possible length of a valid ab string that can be obtained after such removals.

Parameters:

T :: INTEGER

The number of test cases.

T :: 1 → 100 S :: STRING

A string of characters 'a' and 'b'.

Length |S| :: 1 → 10⁶

Case#: 1 Input:

1

abaababbbaaaabaaab Output:

14

We can remove characters at indices 2, 5, 14, and 18 to get: aa...aaabbbb...b

Resulting string is of the form "aaaaabbbbb", which is valid and of length 14. Case#: 2

Input:

1

bab Output:

3

The string is already valid in form "bb...baaa...a" No characters need to be removed.

Maximum length = 3 Case#: 3

Input:

1

bbbbababb Output:

8

Remove one 'a' in the middle to get:

bbbbbbbb or bbbbbabb → which is valid (bbbbbbbb)

Problem 7

A factory has a set of machines, each assigned a specific task. The tasks are scheduled in the order they arrive. If a machine is idle, it begins working on the next available task immediately.

You are given the time required to complete each task.

Your task is to compute and print the completion time for each machine's task, assuming each begins execution immediately after the previous one completes.

Parameters:

T :: INTEGER

The number of test cases. T :: 1 → 10

N :: INTEGER

Number of tasks in a test case.

N :: 1 → 10⁵

T[] :: INTEGER ARRAY

The time required for each task.

Each T[i] :: 1 → 10⁴

Case#: 1 Input:

1

3

4 5 7

Output:

4 9 16

Explanation:

* Task 1 finishes at time 4
* Task 2 starts at 4, finishes at 4+5 = 9
* Task 3 starts at 9, finishes at 9+7 = 16

Case#: 2 Input:

1

5

2 3 1 4 6

Output:

2 5 6 10 16

* Task 1: 2
* Task 2: 2+3 = 5
* Task 3: 5+1 = 6
* Task 4: 6+4 = 10
* Task 5: 10+6 = 16

Case#: 3 Input:

1

4

1 1 1 1

Output:

1 2 3 4

Each task takes 1 unit time and starts right after the previous one.

Problem 8

You are given a string s which contains lowercase and uppercase characters along with the asterisk (\*) character.

In one operation:

You choose a star \* in the string.

You remove the closest non-star character to its left, and also remove the star itself.

You must perform this operation for every star in the string, exactly once per star, and print the final resulting string after all stars and the respective characters are removed.

It is guaranteed that:

The operation is always possible for each star. The resulting string is always unique.

Parameters:

T :: INTEGER

Number of test cases.

T :: 1 → 100

S :: STRING

A string consisting of uppercase/lowercase letters and \* characters.

Length :: 1 → 1000

Case#: 1 Input:

1

Dinesh\*\*reddy\*\*\*\* Output:

Diner

* 'h' is removed with '\*'
* 's' is removed with '\*'
* 'y' is removed with '\*'
* 'd' is removed with '\*'
* 'd' is removed with '\*'
* 'e' is removed with '\*' Resulting string: Diner

Case#: 2 Input:

1

abcdef\*\* Output:

abcd

* Remove 'f' and '' → "ade"
* Remove 'e' and '\*' → "ad"

Case#: 3 Input:

1

ABCDEFG

Output:

ABCDEFG

* There are no \* characters, so no operations are performed.
* The string remains as it is.

Problem 9

You are given an array arr of integers indexed from 0. Your task is to find the greatest common divisor (GCD) of the smallest and largest numbers in the array.

The GCD of two numbers is the largest positive integer that divides both numbers without leaving a remainder.

Parameters:

n :: INTEGER

The number of elements in the array.

n :: 1 → 10⁵

arr :: INTEGER ARRAY

Each line i of the array contains an integer value.

arr[i] :: 1 → 10⁹

Case#: 1 Input:

5

2 4 6 8 10

Output:

2

* Minimum: 2
* Maximum: 10
* GCD(2, 10) = 2

Case#: 2 Input:

4

15 25 35 45

Output:

15

* Minimum: 15
* Maximum: 45
* GCD(15, 45) = 15

Case#: 3 Input:

6

17 34 51 68 85 102

Output:

17

* Minimum: 17
* Maximum: 102
* GCD(17, 102) = 17

Problem 10

Leo and his friend Max are playing a game with two stacks of nuts. Initially, the stacks have A and B nuts respectively. They take turns, starting with Leo. The rules are as follows:

On a turn, a player:

Eats all the nuts from one of the stacks.

Splits the other stack into two non-empty stacks.

If the remaining stack has only 1 nut, it cannot be split, and the player loses. Determine the winner assuming both players play optimally.

Parameters:

T :: INTEGER

The number of test cases.

T :: 1 → 1000

A, B :: INTEGER

Each test case contains two integers, the count of nuts in the two stacks.

A, B :: 1 → 10⁴

Case#: 1 Input:

1

1 1

Output:

Max

Leo starts, but both stacks have only 1 nut. If Leo eats one stack, the other cannot be split. So Leo loses.

Case#: 2 Input:

1

1 2

Output:

Leo

Leo eats stack with 1 nut, splits the 2-nut stack into (1, 1). Now Max has no valid move, so Leo wins. Case#: 3

Input:

1

4 5

Output:

Leo

Leo has many possible valid moves (e.g., eat 4 and split 5 into 2,3). With optimal strategy, Leo forces Max into a losing position.

Problem 11

Rahul and Sameer are seated next to each other during an exam and want to share answers. There are p different sets of questions numbered from 0 to p−1, assigned based on each student’s roll number using the formula (roll number - 1) % p. They can only share answers if assigned the same set of questions. Given Rahul’s roll number A and Sameer’s roll number B, find how many different values of p allow them to have the same set of questions.

Parameters:

T :: INTEGER

Number of test cases. 1 ≤ T ≤ 100

A :: INTEGER

Rahul’s roll number. 1 ≤ A ≤ 10^8

B :: INTEGER

Sameer’s roll number. 1 ≤ B ≤ 10^8

Case#: 1 Input:

1

2 8

Output:

4

They get the same set if (A−1) % p == (B−1) % p → p divides |(B−A)| = |8−2| = 6.

Number of divisors of 6 = 4 (1, 2, 3, 6). Hence, output is 4.

Case#: 2 Input:

1

15 1

Output:

4

p divides |1−15| = 14. Divisors of 14 are (1, 2, 7, 14). Hence, output is 4.

Case#: 3 Input:

1

10 5

Output:

2

p divides |5−10| = 5. Divisors are (1, 5). Hence, output is 2.

Problem 12

Aman loves chocolates and there are N chocolates in a bowl. Aman can only take m chocolates such that m is a power of 2 and m ≤ N. The goal is to find the maximum number of chocolates Aman can take.

Parameters:

T :: INTEGER

Number of test cases. 1 ≤ T ≤ 100

N :: INTEGER

Initial number of chocolates in the bowl. 1 ≤ N ≤ 10^5

Case#: 1 Input:

1

1

Output:

1

The maximum power of 2 less than or equal to 1 is 1. Case#: 2

Input:

1

25

Output:

16

Powers of 2 ≤ 25 are 1, 2, 4, 8, 16. The maximum is 16.

Case#: 3 Input:

1

59

Output:

32

Powers of 2 ≤ 59 are 1, 2, 4, 8, 16, 32. The maximum is 32.

Problem 13

You are given two positive integers A and B. Your task is to find and print the sum of all prime numbers between A and B (inclusive).

Parameters:

T :: INTEGER

Number of test cases. 1 ≤ T ≤ 10

A :: INTEGER

Start of the range. 1 ≤ A ≤ 10^6

B :: INTEGER

End of the range. 1 ≤ B ≤ 10^6

Case#: 1 Input:

1

18 35

Output:

102

Prime numbers between 18 and 35 are: 19, 23, 29, 31

Sum = 19 + 23 + 29 + 31 = 102

Case#: 2 Input:

1

1 20

Output:

77

Prime numbers between 1 and 20 are: 2, 3, 5, 7, 11, 13, 17, 19

Sum = 2 + 3 + 5 + 7 + 11 + 13 + 17 + 19 = 77

Case#: 3 Input:

1

10 10

Output:

0

No prime numbers between 10 and 10 (since 10 is not prime). Sum = 0

Problem 14

In a desolate, frozen wasteland, Alice and her family find themselves trapped inside a deserted outpost with no escape. The only lifeline they have is a stockpile of frozen meals that have miraculously survived the apocalyptic conditions.

In this icy realm, sustenance comes in the form of frozen meals, and each family member requires a daily intake of X meals to stave off starvation. However, the twist lies in their ability to endure beyond the depletion of their food supply.

After the last meal is consumed, the family can endure for an additional D days before succumbing to the harsh environment.

Given the scarcity of resources, the pressing question is: for how many days can Alice and her family brave the frozen desert?

Your task is to compute and output, for each test case, the total number of days Alice and her family can endure in this frozen desert.

Parameters:

T :: INTEGER

[The number of scenarios to be evaluated. 1≤T≤1051 ≤ T ≤ 10^5]

N :: INTEGER

[Total number of frozen meals available. 0≤N<5000 ≤ N < 500]

X :: INTEGER

[Daily requirement per family member. 1≤X≤101 ≤ X ≤ 10]

D :: INTEGER

[Number of days the family can endure without sustenance after meals run out. 0≤D<200 ≤ D < 20]

Case#: 1 Input:

245 4 9

Output:

70

Total meals = 245 Daily need = 4

They survive 245 // 4 = 61 days with food They survive +9 days without food

Total = 61 + 9 = 70

Case#: 2 Input:

405 2 0

Output:

202

With 405 meals and a daily need of 2, they survive 405 // 2 = 202 days. Since D = 0, no extra days.

Total = 202

Case#: 3 Input:

188 3 5

Output:

67

Meals = 188, Daily requirement = 3

They survive 188 // 3 = 62 days with food Then +5 days without food

Total = 62 + 5 = 67

Problem 15

There are three students: Krishna, Ram, and Vishnu. Each is assigned a positive integer roll number: a, b, and c. You are given min(a, b), min(b, c), and min(c, a). Determine if there exist positive integers a, b, and c satisfying these three values.

Your goal is to output "YES" if such three integers exist, and "NO" otherwise. Parameters:

T :: INTEGER

Numberoftestcases.(1≤T≤1000)Number of test cases. (1 ≤ T ≤ 1000)

minAB :: INTEGER

Minimumof‘a‘and‘b‘.(1≤minAB≤10)Minimum of `a` and `b`. (1 ≤ minAB ≤ 10)

minBC :: INTEGER

Minimumof‘b‘and‘c‘.(1≤minBC≤10)Minimum of `b` and `c`. (1 ≤ minBC ≤ 10)

minCA :: INTEGER

Minimumof‘c‘and‘a‘.(1≤minCA≤10)Minimum of `c` and `a`. (1 ≤ minCA ≤ 10)

Case#: 1 Input:

4 9 4

Output:

YES

Values a=9, b=4, c=9 satisfy all minimums. Case#: 2

Input:

1. 4 5

Output:

NO

No triple satisfies all three mins together. Case#: 3

Input:

5 10 5

Output:

YES

Values a=10, b=5, c=10 match all conditions.

Problem 16

San Te is starting the 37th Chamber of Shaolin to teach women self-defence. A woman is eligible for training if her age is between 10 and 60 (inclusive). Given the ages of women in the village, determine how many are eligible.

Your goal is to output the number of eligible women for each test case. Parameters:

T :: INTEGER

Numberoftestcases.(1≤T≤20)Number of test cases. (1 ≤ T ≤ 20)

N :: INTEGER

Numberofwomeninthetestcase.(1≤N≤100)Number of women in the test case. (1 ≤ N ≤ 100)

A1, A2, ..., AN :: INTEGER ARRAY

Agesofthewomen.Each1≤Ai≤100Ages of the women. Each 1 ≤ Ai ≤ 100

Case#: 1 Input:

6

53 39 4 10 37 32

Output:

5

All except 4 are between 10 and 60.

Case#: 2 Input:

3

3 52 1

Output:

1

Only 52 is eligible.

Case#: 3

Input:

5

9 10 60 61 30

Output:

3

Only 10, 60, and 30 fall within the eligible range.

Problem 17

You are given a string of lowercase English letters. Construct a new string based on the following rules:

1. Characters with higher frequency come first.
2. After placing a character once, reduce its count by one and repeat the process.
3. If two characters have the same frequency, choose the lexicographically smaller character first. Your goal is to output the rearranged string following these rules.

Parameters:

t :: INTEGER

Numberoftestcases.(1≤t≤100)Number of test cases. (1 ≤ t ≤ 100)

len :: INTEGER

Lengthoftheinputstring.(1≤len≤100)Length of the input string. (1 ≤ len ≤ 100)

string :: STRING

Thelowercasealphabetstringtoberearranged.The lowercase alphabet string to be rearranged. Case#: 1

Input:

13

abcdccbacbcbc Output:

ccbcbcabcabcd

‘c’ appears most, followed by ‘b’, then ‘a’, and finally ‘d’.

Case#: 2 Input:

47

sjdfvhwebuyuuuuuuuuuuuunndcvshhhhhhhvuiwoqwjeoqpo Output: uuuuuuhuhuhuhuhuhouvwdehjnoqsuvwbcdefhijnopqsuvwy

‘u’ appears most, followed by ‘h’, then sorted by frequency and lexicographical order.

Case#: 3 Input:

8

aabbccdd Output:

abcdabcd

Each character has equal frequency. Sorted lexicographically.

Problem 18

007 is preparing for an ambush mission. Out of N available weapons, each with a certain attack rating, he can select any number of weapons as long as their total attack rating does not exceed a specified limit K. His goal is to maximize the number of weapons he can carry without violating the total attack rating limit.

Your goal is to output the maximum number of weapons 007 can carry within the given constraint. Parameters:

T :: INTEGER

Numberoftestcases(1≤T≤10)Number of test cases (1 ≤ T ≤ 10)

N :: INTEGER

Numberofweaponsavailable(1≤N≤105)Number of weapons available (1 ≤ N ≤ 10⁵)

K :: INTEGER

Maximumallowedtotalattackrating(1≤K≤109)Maximum allowed total attack rating (1 ≤ K ≤ 10⁹)

attack\_ratings :: LIST[INTEGER]

ListofattackratingsfortheNweapons(1≤attackrating≤109)List of attack ratings for the N weapons (1 ≤ attack\_rating ≤ 10⁹)

Case#: 1 Input:

1

1. 3

4 5 2 1

Output:

2

Selected weapons: 2 and 1 → total = 3

(Max possible within K) Case#: 2

Input:

1

1. 15

10 2 3 4 5

Output:

4

Selected: 2, 3, 4, 5 → total = 14

(Max number of weapons without exceeding 15) Case#: 3

Input:

1

3 2

3 4 5

Output:

0

No weapon with attack rating ≤ 2.

Problem 19

Nikola Tesla wants to study N subjects numbered from 1 to N. He is given M prerequisite pairs [a, b], meaning subject a must be completed before subject b. Determine if it is possible for Tesla to study all subjects following these prerequisite conditions.

Parameters:

T :: INTEGER

Numberoftestcases(1≤T≤10)Number of test cases (1 ≤ T ≤ 10)

N :: INTEGER

Numberofsubjects(1≤N≤105)Number of subjects (1 ≤ N ≤ 10^5)

M :: INTEGER

Numberofprerequisitepairs(1≤M≤105)Number of prerequisite pairs (1 ≤ M ≤ 10^5)

pairs :: LIST of pairs [a, b]

Eachpairindicatessubject‘a‘mustbedonebefore‘b‘(1≤a,b≤N)Each pair indicates subject `a` must be done before `b` (1 ≤ a,b ≤ N)

Case#: 1 Input:

1

2 1

1 2

Output:

Yes

Prerequisite order is possible, so Tesla can study all subjects. Case#: 2

Input:

1

3 3

1 2

2 3

3 1

Output:

No

There is a cycle in prerequisites, so Tesla cannot complete all subjects. Case#: 3

Input:

1

4 3

1 2

2 3

3 4

Output:

Yes

Linear prerequisite order allows studying all subjects.

Problem 20

A number is unlucky if it contains the substring "13". All other numbers are lucky. Given n, find how many lucky numbers of length n digits can be formed. Leading zeros are allowed.

Parameters:

T :: INTEGER

Numberoftestcases(1≤T≤10)Number of test cases (1 ≤ T ≤ 10)

N :: INTEGER

Lengthofthenumber(1≤N≤10000)Length of the number (1 ≤ N ≤ 10000)

Case#: 1 Input:

2

1

2

Output:

10

99

Explanation:

* For n=1, all digits 0-9 are lucky (no "13" possible). So 10 lucky numbers.
* For n=2, total numbers = 100; exclude numbers with "13" as substring → 100 - 1 = 99. Case#: 2

Input:

1

3

Output:

989

Explanation:

Count all 3-digit numbers (with leading zeros) excluding those containing substring "13". Case#: 3

Input:

1

4

Output:

9801

Explanation:

Count all 4-digit numbers without substring "13". Note: Outputs are modulo 10^9+7.

Problem 21

Tribonacci Numbers: We all know about Fibonacci numbers where Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2), where Fibonacci(n) = n where 0 <= n <= 1. Tribonacci numbers are those numbers whose value is equal to the sum of the previous 3 numbers of the same sequence, whereas in Fibonacci it's just 2. So, given the N value, you are supposed to print the Nth Tribonacci number.

NOTE :tribonnaci(n) = n where 0 <= n <= 2.The value can be a huge value, So print the answer in mod 10^9+7

Parameters:

T :: INTEGER

Numberoftestcases(1≤T≤10)Number of test cases (1 ≤ T ≤ 10)

N :: INTEGER

IndexofTribonaccinumbertofind(0≤N≤500)Index of Tribonacci number to find (0 ≤ N ≤ 500)

Case#: 1 Input:

4

1

2

3

4

Output:

1

2

3

6

* Tribonacci(1) = 1
* Tribonacci(2) = 2
* Tribonacci(3) = Tribonacci(2)+Tribonacci(1)+Tribonacci(0) = 2+1+0 = 3
* Tribonacci(4) = Tribonacci(3)+Tribonacci(2)+Tribonacci(1) = 3+2+1 = 6 Case#: 2

Input:

1

0

Output:

0

Tribonacci(0) = 0

Case#: 3

Input:

1

5

Output:

11

Tribonacci(5) = Tribonacci(4)+Tribonacci(3)+Tribonacci(2) = 6+3+2 = 11

Problem 22

Climb steps:You are on the ground level basically not on any step. In one instance you can either climb 1 or 2 steps. Suppose you want to reach the Nth step.Print the total number of ways you can reach your desired location.NOTE:The answer can be a large value, so print it in mod 10^9+7 format.

Print the total number of ways you can reach the Nth step under mod 10^9+7 Parameters:

T :: INTEGER

Numberoftestcases(1≤T≤10)Number of test cases (1 ≤ T ≤ 10)

N :: INTEGER

Targetstepnumber(1≤N≤10,000)Target step number (1 ≤ N ≤ 10,000)

Case#: 1 Input:

2

2

3

Output:

2

3

* For N=2: Ways = {1+1, 2} = 2
* For N=3: Ways = {1+1+1, 1+2, 2+1} = 3

Case#: 2 Input:

1

1

Output:

1

Only one way to reach step 1 (a single step). Case#: 3

Input:

1

4

Output:

5

Ways to reach step 4 are 5 in total (1+1+1+1, 1+1+2, 1+2+1, 2+1+1, 2+2).

Problem 23

Ganesh Chaturthi is near, given n entries of donations where each entry has a group number (1 to 10) and the amount of money donated by a person, print the total amount collected by each group from 1 to 10.

Parameters:

n :: INTEGER

NumberofdonationentriesNumber of donation entries For each entry:

group\_number :: INTEGER

Groupnumber(1≤groupnumber≤10)Group number (1 ≤ group\_number ≤ 10)

money :: INTEGER

AmountdonatedbythepersonAmount donated by the person Case#: 1

Input:

5

1 12

1. 4
2. 4

2 56

1 22

Output:

1 34

2 60

3 4

1. 0
2. 0
3. 0
4. 0
5. 0
6. 0

10 0

Sum donations group-wise, print all groups 1 through 10. Case#: 2

Input:

3

10 100

10 200

1 50

Output:

1 50

2 0

3 0

4 0

5 0

6 0

7 0

8 0

9 0

10 300

Case#: 3 Input:

1

5 75

Output:

1 0

2 0

3 0

4 0

5 75

6 0

7 0

8 0

9 0

10 0

Problem 24

Since her childhood Goldy loves playing with toys.On a fine day she went to a toy shopping mall with her father and brought some set of toys of her choice to her home.Now she wants to arrange them in such a ways that all the toys with same size should be at one place and while arranging, the count of toys also matters i.e the size with maximum number of toys will be placed first and then with second maximum number of toys and so on until all the toys get arranged. If number of toys of two or more sizes are equal then arrange the toys with smaller size first. As Goldy is very confused about how to arrange, she asked your help to do so. Output the toys arranged accordingly.

Parameters:

N :: INTEGER

TotalnumberoftoysTotal number of toys A :: ARRAY[INTEGER]

SizesofthetoysSizes of the toys Case#: 1

Input:

8

2 5 2 8 5 6 8 8

Output:

8 8 8 2 2 5 5 6

Count: 8 -> 3, 2 -> 2, 5 -> 2, 6 -> 1

Order: 8 (3 times), 2 (2 times), 5 (2 times, but 5 > 2 so after 2), 6 (1 time)

Case#: 2 Input:

6

1 1 2 2 3 3

Output:

1 1 2 2 3 3

Counts equal (2 each), sorted by size ascending. Case#: 3

Input:

7

4 4 4 1 2 2 3

Output:

4 4 4 2 2 1 3

Count: 4->3, 2->2, 1->1, 3->1

Order by count descending, tie by size ascending.

Problem 25

Yashwanth was so hungry so he went to KFC to eat, while he was eating his mobile suddenly fell down from his hand and broke. so, now he cannot pay online. Now he has to pay hand cash. He has many 2000,500,200,100,50,20,10,5,2,1 coins. He wants to pay the bill with fewer coins. So, he is asking for your help to find it.

Note: He should pay with fewer coins

Find the minimum number of coins needed. Parameters:

T :: INTEGER

NumberoftestcasesNumber of test cases N :: INTEGER

AmounttopayforeachtestcaseAmount to pay for each test case Case#: 1

Input:

5

547

673

556

9989

2550

Output:

5

6

4

5

3

Explanation for 547: 5001 + 202 + 51 + 21 = 5 coins

Case#: 2 Input:

3

1235

4222

489

Output:

6

5

8

Explanation for 1235: 2000(0), 500(2), 200(1), 100(0), 50(0), 20(1), 10(1), 5(1), 2(0), 1(0) = 6 coins

Case#: 3 Input:

2

45454

87778

Output:

29

52

Large numbers broken down greedily by largest coin first

Problem 26

Given an array A, count the number of inversion pairs (j, k) such that j < k and A[j] > A[k]. Parameters:

n :: INTEGER

NumberofelementsinthearrayNumber of elements in the array A :: ARRAY of INTEGER

ArrayelementsArray elements Case#: 1

Input:

5

1

20

6

4

5

Output:

5

Inversions are (20,6), (20,4), (20,5), (6,4), (6,5)

Case#: 2 Input:

4

4

3

2

1

Output:

6

All pairs form inversions: (4,3), (4,2), (4,1), (3,2), (3,1), (2,1)

Case#: 3 Input:

3

1

2

3

Output:

0

No inversions as array is sorted in ascending order.

Problem 27

Given an encrypted string where characters are followed by their frequency, find the Nth character in the decrypted string. If N is larger than the length of the decrypted string, print -1.

Parameters:

s :: STRING

Encryptedstringintheformatcharacter+frequencyEncrypted string in the format character + frequency

N :: INTEGER

Indexofcharactertofindinthedecryptedstring(1−based)Index of character to find in the decrypted

string (1-based) Case#: 1

Input:

a1b1c3d23 5

Output:

c

Decrypted string = "abcccddddddddddddddddddddddd", 5th character is 'c' Case#: 2

Input:

a1b1c3 10

Output:

-1

Decrypted string = "abccc", 10th character does not exist Case#: 3

Input:

x2y10z1 12

Output:

y

Decrypted string = "xxyyyyyyyyyyz", 12th character is 'y'

Problem 28

A factory has a set of machines, each assigned a specific task. The tasks are scheduled in the order they arrive. If a machine is idle, it begins working on the next available task immediately.

You are given the time required to complete each task.

Your task is to compute and print the completion time for each machine's task, assuming each begins execution immediately after the previous one completes.

Parameters:

T :: INTEGER

The number of test cases. T :: 1 → 10

N :: INTEGER

Number of tasks in a test case.

N :: 1 → 10⁵

T[] :: INTEGER ARRAY

The time required for each task.

Each T[i] :: 1 → 10⁴

Case#: 1 Input:

1

3

4 5 7

Output:

4 9 16

Explanation:

* Task 1 finishes at time 4
* Task 2 starts at 4, finishes at 4+5 = 9
* Task 3 starts at 9, finishes at 9+7 = 16

Case#: 2 Input:

1

5

1. 3 1 4 6

Output:

2 5 6 10 16

* Task 1: 2
* Task 2: 2+3 = 5
* Task 3: 5+1 = 6
* Task 4: 6+4 = 10
* Task 5: 10+6 = 16

Case#: 3 Input:

1

4

1 1 1 1

Output:

1 2 3 4

Each task takes 1 unit time and starts right after the previous one.

Problem 29

MTV is given an integer sequence of N numbers. But he only likes natural numbers. So he removes some numbers from the given sequence of N numbers so that the formed sequence is contiguous natural number sequence starting from 1.

Let us say there are K numbers after removing some numbers from initial sequence, MTV will be satisfied if, for each integer i (1 ≤ i ≤ K) , the i-th of those numbers from the left is equal to i. See sample test cases for better understanding.

Print the minimum number of numbers that MTV has to remove from the initial sequence(i.e, K should be maximized). If no sequence can be formed the print -1.

Parameters:

N :: INTEGER

Number of integers in the sequence (1 ≤ N ≤ 10^5)

arr :: ARRAY of INTEGER

The sequence of N integers Case#: 1

Input:

3

2 1 2

Output:

1

Removing the first element gives [1, 2], which forms a natural sequence starting from 1. So minimum removals = 1.

Case#: 2 Input:

5

1 3 2 4 5

Output:

0

The sequence already contains [1,2,3,4,5] in order, so no removal needed. Case#: 3

Input:

4

2 3 4 5

Output:

-1

There's no 1 in the sequence, so no natural sequence starting from 1 can be formed.

Problem 30

You are given an integer array A. Your task is to calculate the sum of absolute difference of indices of first and last occurrence for every integer that is present in array A.

Formally, if element x occurs m times in the array at indices B1, B2, B3, ...,Bm, then the answer for x

will be Bm−B1 if array B is sorted.

You are required to calculate the sum of the answer for every such x that occurs in the array. Output this sum for each test case.

Parameters:

T :: INTEGER

Number of test cases (1 ≤ T ≤ 1000)

N :: INTEGER

Number of elements in the array for a test case (1 ≤ N ≤ 200000)

A[i] :: INTEGER

Element of the array (1 ≤ A[i] ≤ 1e9). Total N across all test cases ≤ 200000.

Case#: 1 Input:

1

5

1 2 3 3 2

Output:

4

* 1 appears once → 0
* 2 appears at indices 1 and 4 → 4 - 1 = 3
* 3 appears at indices 2 and 3 → 3 - 2 = 1

Total = 0 + 3 + 1 = 4

Case#: 2 Input:

1

6

5 5 5 5 5 5

Output:

5

Only 5 appears at indices 0 to 5 → 5 - 0 = 5

Case#: 3 Input:

1

4

10 20 30 40

Output:

0

All elements are unique, so the first and last index are same → all differences = 0 → 0.

Easy

1. Collecting Baskets

Problem Statement:

There is a long stretch of market stalls. Each stall has a certain number of fruit baskets. You start at the first stall and can only move forward. At every step, you are allowed to pick up baskets from that stall—but there’s a twist.

You can only collect baskets in increasing order of quantity (strictly). That means, if the last basket you collected had X baskets, you can only pick a basket with more than X baskets next.

You want to know the maximum number of baskets you can collect following this rule.

Input Format:

* + First line contains an integer N, the number of stalls.
  + Second line contains N integers separated by space, where each integer A[i] denotes the number of baskets at the i-th stall.

Output Format:

* + A single integer — the maximum number of baskets you can collect in increasing order.

Sample Input 1:

6

1 3 2 5 4 6

Sample Output 1:

4

Explanation:

One of the longest increasing sequences: 1 → 2 → 4 → 6. Total baskets = 4.

Sample Input 2:

5

5 4 3 2 1

Sample Output 2:

1

Explanation:

Only one basket can be collected at most, since all are in decreasing order.

Test Cases Input 1

1

10

Output 1

1

Input 2

3

1. 3 3

Output 2

1

Input 3

6

1 2 3 4 5 6

Output 3

6

Input 4

5

7 7 7 7 7

Output 4

1

Input 5

8

10 22 9 33 21 50 41 60

Output 5

5

Input 6

6

1 3 2 4 3 5

Output 6

4

Input 7

10

100 90 80 70 60 50 40 30 20 10

Output 7

1

Input 8

7

2 4 6 8 10 12 14

Output 8

7

Code

def max\_baskets(arr):

from bisect import bisect\_left

# This will store the smallest tail of all increasing subsequences # with different lengths.

tails = []

for basket in arr:

pos = bisect\_left(tails, basket) if pos == len(tails):

tails.append(basket) else:

tails[pos] = basket

return len(tails)

# Read input

n = int(input())

arr = list(map(int, input().split()))

# Print output print(max\_baskets(arr))

1. Ticket Counter Time

Problem Statement:

There’s a single ticket counter and a line of people waiting. Each person needs a specific amount of

time to be served, which is represented by an array T.

The counter serves people in order, but here's the rule: the counter works in rounds. In each round, the person at the front gets served for 1 time unit, then goes to the back of the queue if their remaining time is more than 0.

You are given the time required for each person, and you are asked to calculate how many time units it will take before the K-th person (0-indexed) in the original queue completes their service.

Input Format:

* First line contains two integers N (number of people) and K (index of target person).
* Second line contains N space-separated integers, where T[i] is the total time required for the i-th person.

Output Format:

* A single integer — total time taken until the K-th person completes service.

Sample Input 1:

1. 2

1 2 3 4

Sample Output 1:

8

Explanation:

[1,2,3,4] → [2,3,4] after 1 (P0 done)

[2,3,4,1] → [3,4,1] after 1 (P1: 1 left)

[3,4,1,1] → [4,1,1] after 1 (P2: 2 left)

[4,1,1,2] → [1,1,2] after 1 (P3: 3 left)

[1,1,2,3] → [1,2,3] after 1 (P1 done)

[2,3] → [3,2] after 1 (P2: 1 left)

[2,1] → [1,1] after 1 (P3: 1 left)

[1] → P2 done after 1 more

Sample Input 2:

1. 0

5 1 1 1 1

Sample Output 2:

9

Test Cases Input 1

1 0

1

Output 1

1

Input 2

3 2

1 1 1

Output 2

3

Input 3

3 0

3 3 3

Output 3

7

Input 4

5 4

1 1 1 1 5

Output 4

9

Input 5

1. 2

3 3 1 3 3 3

Output 5

3

Input 6

5 2

2 2 2 2 2

Output 6

8

Input 7

5 2

10 2 3 1 1

Output 7

10

Input 8

4 3

1 1 1 5

Output 8

8

Code

from collections import deque

def ticket\_counter\_time(n, k, times):

queue = deque([(i, t) for i, t in enumerate(times)]) total\_time = 0

while queue:

idx, time\_left = queue.popleft() time\_left -= 1

total\_time += 1

if time\_left == 0: if idx == k:

return total\_time else:

queue.append((idx, time\_left))

return -1 # Should never reach here if input is valid

# Read input

n, k = map(int, input().split())

times = list(map(int, input().split()))

# Output result print(ticket\_counter\_time(n, k, times))

1. Repeating Pairs

Problem Statement:

You are given a string S. Your task is to count how many distinct pairs of characters occur more than once as consecutive characters in the string.

A "pair" means two characters next to each other. Pairs are considered the same if they have the same characters in the same order (e.g., ab and ab are the same, ab and ba are different).

Input Format:

* + A single line containing a string S.

Output Format:

* + Print a single integer — the number of distinct character pairs that appear more than once as consecutive characters.

Sample Input 1: ababcabc

Sample Output 1:

2

Explanation: Consecutive pairs:

ab, ba, ab, bc, ca, ab, bc ab occurs 3 times

bc occurs 2 times

Output = 2 (pairs ab and bc)

Sample Input 2: aaaa

Sample Output 2:

1

Test Cases Input 1 abcdef Output 1

0

Input 2

a

Output 2

0

Input 3 abababa Output 3

2

Input 4 abcabcabc Output 4

3

Input 5 aaaaaa Output 5

1

Input 6 abcdefg Output 6

0

Input 7 abababab Output 7

2

Input 8 z

Output 8

0

Code

from collections import defaultdict

def count\_repeating\_pairs(s): pair\_count = defaultdict(int)

for i in range(len(s) - 1):

pair = s[i] + s[i + 1] pair\_count[pair] += 1

# Count pairs that appear more than once

return sum(1 for count in pair\_count.values() if count > 1)

# Read input

s = input().strip()

# Output result print(count\_repeating\_pairs(s))

1. Most Frequent Character After Cleanup

Problem Statement:

You are given a string S that may contain:

* + Lowercase letters (a-z)
  + Uppercase letters (A-Z)
  + Digits (0-9)
  + Special characters (punctuation, symbols, whitespace, etc.) Your task is to:

1. Remove all non-alphabetic characters (digits, special characters, whitespace).
2. Ignore case — that is, treat A and a as the same.
3. Find the most frequent alphabetic character in the cleaned string.

If multiple characters have the same maximum frequency, return the one that comes first in alphabetical order.

Input Format:

* + A single line containing a string S.

Output Format:

* + A single lowercase letter — the most frequent alphabetic character (after cleanup).

Sample Input 1:

Hello, World! 123

Sample Output 1:

l Explanation:

Cleaned string: helloworld → Frequencies: h:1, e:1, l:3, o:2, w:1, r:1, d:1

Most frequent: l

Sample Input 2: A@aaBBccC1234# Sample Output 2:

a

Test Cases Input 1 aaBBccC Output 1 c

Input 2 aaBBccCaa Output 2

a

Input 3 AAAAaaaaAAAaaa!!!@@@ Output 3

a

Input 4 xxYYzz!! Output 4 x

Input 5 QWERTYQWERTY

Output 5 e

Input 6

!!!@@@###bbb%%%^^^ Output 6

b

Input 7 aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ1234567890!@#$ Output 7

a

Input 8 z

Output 8 z

Code

from collections import defaultdict

def most\_frequent\_letter(s): freq = defaultdict(int)

for char in s:

if char.isalpha(): # Only consider alphabetic characters char = char.lower()

freq[char] += 1

if not freq:

return "No alphabetic characters" # No alphabetic characters

# Find the character with highest frequency, break ties by alphabetical order max\_freq = max(freq.values())

result = min([ch for ch in freq if freq[ch] == max\_freq]) print(result)

# Read input

s = input().strip() most\_frequent\_letter(s)

1. Lonely Light

Problem Statement:

In a row of street lights, only one light is faulty — it flickers only when it receives power an odd number of times.

You are given an array of integers, where each integer represents the ID of a street light that received power. Every light gets powered an even number of times — except for one faulty light that was powered an odd number of times.

Your task is to find the ID of that faulty street light.

Input Format:

* + First line contains an integer N — the number of entries.
  + Second line contains N space-separated integers representing the light IDs that received power.

Output Format:

* + A single integer — the ID of the faulty light.

Sample Input 1:

5

1 2 3 2 1

Sample Output 1:

3

Explanation:

1. appears twice, 2 appears twice, 3 appears once → faulty = 3

Sample Input 2:

7

5 7 5 4 7 9 4

Sample Output 2:

9

Test Cases Input 1

1

42

Output 1

42

Input 2

7

4 5 6 5 4 7 7

Output 2

6

Input 3

3

10 20 10

Output 3

20

Input 4

9

8 3 8 6 3 2 2 5 5

Output 4

6

Input 5

11

9 10 10 12 13 9 13 15 15 12 11

Output 5

11

Input 6

7

11 12 13 11 12 13 14

Output 6

14

Input 7

5

5 6 7 6 5

Output 7

7

Input 8

3

1000 2000 1000

Output 8

2000

Code

def find\_faulty\_light(arr): result = 0

for num in arr:

result ^= num # XOR cancels out pairs return result

# Read input

n = int(input())

arr = list(map(int, input().split()))

# Output result print(find\_faulty\_light(arr))

1. Sum of Array Pairs

Problem Statement:

You are given an array of integers. Your task is to find all unique pairs of integers in the array whose sum is equal to a given target value.

Input Format:

* + The first line contains an integer n (1 ≤ n ≤ 1000), the number of elements in the array.
  + The second line contains n space-separated integers, representing the elements of the array.
  + The third line contains an integer target, representing the sum you need to find pairs for.

Output Format:

* + Output all unique pairs of integers whose sum equals the given target.
  + Each pair should be printed on a new line, with the two integers separated by a space.
  + If no such pairs exist, print No pairs found.

Sample Input 1:

6

1 4 2 3 3 5

6

Sample Output 1:

1 5

1. 4
2. 3

Sample Input 2:

5

10 20 30 40 50

100

Sample Output 2: No pairs found

Test Cases Input 1

1

5

10

Output 1

No pairs found

Input 2

4

2 2 2 2

4

Output 2

2 2

Input 3

8

3 1 4 5 6 2 7 9

10

Output 3

1 9

3 7

1. 6

Input 4

4

1 2 3 4

10

Output 4

No pairs found

Input 5

6

1. 1 3 7 2 4

8

Output 5

1 7

3 5

Input 6

5

-1 -2 3 4 5

3

Output 6

-2 5

-1 4

Input 7

6

3 3 3 3 3 3

6

Output 7

3 3

Input 8

1

5

10

Output 8

No pairs found

Code

def find\_pairs(arr, target):

seen = set() # To keep track of elements we've already visited result = set() # To store unique pairs

for num in arr:

complement = target - num if complement in seen:

# To ensure pairs are added in a sorted order (for uniqueness) result.add(tuple(sorted((num, complement))))

seen.add(num)

if result:

for pair in sorted(result):

print(pair[0], pair[1]) else:

print("No pairs found")

# Reading input

n = int(input()) # Number of elements in the array

arr = list(map(int, input().split())) # The array of integers target = int(input()) # The target sum

# Calling the function to find and print pairs find\_pairs(arr, target)

1. Count Set Bits in an Integer

Problem Statement:

Given a non-negative integer n, your task is to count the number of set bits (1s) in its binary representation.

Input Format:

* + A single line containing the integer n (0 ≤ n ≤ 10⁹)

Output Format:

* + Print the number of set bits in n.

Sample Input 1:

5

Sample Output 1:

2

Explanation:

Binary of 5 = 101, which has two 1s.

Sample Input 2:

15

Sample Output 2:

4

Explanation:

Binary of 15 = 1111, which has four 1s.

Test Cases Input 1

0

Output 1

0

Input 2

64

Output 2

1

Input 3

255

Output 3

8

Input 4

254

Output 4

7

Input 5

164

Output 5

3

Input 6

128

Output 6

1

Input 7

42

Output 7

3

Input 8

1024

Output 8

1

Code

def count\_set\_bits(n): count = 0

while n:

n &= (n - 1) # clears the lowest set bit count += 1

print(count)

# Input parsing n = int(input())

count\_set\_bits(n)

1. Weighted Digit Power Sum

Problem Statement:

A number is considered digit-power-weighted if the sum of its digits raised to their position from left to right, multiplied by their position, results in a unique score.

Given a positive integer n, calculate its weighted digit power sum using the formula:

For a number with digits d₁ d₂ ... dₖ, compute: d₁^1 \* 1 + d₂^2 \* 2 + d₃^3 \* 3 + ... + dₖ^k \* k Return the final value of this computation.

Input Format:

* + A single line containing the integer n (0 ≤ n ≤ 10⁹)

Output Format:

A single integer – the weighted digit power sum.

Sample Input 1:

321

Sample Output 1:

14

Explanation:

Digits: 3 2 1

* + 3^1 \* 1 = 3
  + 2^2 \* 2 = 4 \* 2 = 8
  + 1^3 \* 3 = 1 \* 3 = 3

Total = 3 + 8 + 3 = 14

Sample Input 2:

245

Sample Output 2:

409

Explanation:

* + 2^1 × 1 = 2
  + 4^2 × 2 = 16 × 2 = 32
  + 5^3 × 3 = 125 × 3 = 375

Sum = 2 + 32 + 375 = 409

Corrected Output: 409

Test Cases Input 1

7

Output 1

7

Input 2

111

Output 2

6

Input 3

1

Output 3

1

Input 4

222

Output 4

34

Input 5

123

Output 5

90

Input 6

321

Output 6

14

Input 7

406

Output 7

652

Input 8

105

Output 8

376

Code

def weighted\_digit\_power\_sum(n): s = str(n)

total = 0

for i, ch in enumerate(s): digit = int(ch)

power = digit \*\* (i + 1) total += power \* (i + 1)

print(total)

# Read input

n = int(input()) weighted\_digit\_power\_sum(n)

1. Chef’s Energy Boost Plan

Problem Statement:

Chef Rahul has recently taken a pledge to improve his health. Every morning, he drinks different fruit smoothies lined up in a long queue at his smoothie bar. Each smoothie has a certain energy value, which can be positive (if it boosts his energy) or negative (if it's a detox smoothie that lowers it).

Chef wants to figure out which k-day stretch of smoothies gives him the maximum total energy. He can only pick a consecutive sequence of k smoothies.

You must help Chef find the maximum energy he can gain in any stretch of k consecutive days.

Input Format:

* + First line: Two integers n (number of days/smoothies) and k (length of stretch)
  + Second line: n space-separated integers representing the energy values of the smoothies.

Output Format:

A single integer – the maximum total energy Chef can gain from any k consecutive smoothies.

Sample Input 1:

1. 6

1 2 3 4 5 6

Sample Output 1:

21

Sample Input 2:

5 2

-1 -2 -3 -4 -5

Sample Output 2:

-3

Test Cases Input 1

1 1

5

Output 1

5

Input 2

5 3

1 2 3 4 5

Output 2

12

Input 3

6 2

-1 -2 -3 -4 -5 -6

Output 3

-3

Input 4

1. 4

3 -1 2 -5 4 6 -2

Output 4

7

Input 5

1. 3

1 -1 2 -2 3 -3 4 -4

Output 5

4

Input 6

6 3

10 9 8 -1 -2 -3

Output 6

27

Input 7

6 2

-1 -2 -3 4 5 6

Output 7

11

Input 8

1. 5

2 2 2 2 2 2 2 2 2

Output 8

10

Code

def max\_energy\_window(n, k, energy): window\_sum = sum(energy[:k]) max\_sum = window\_sum

for i in range(k, n):

window\_sum += energy[i] - energy[i - k] max\_sum = max(max\_sum, window\_sum)

return max\_sum

# Read input

n, k = map(int, input().split())

energy = list(map(int, input().split())) print(max\_energy\_window(n, k, energy))

1. Lucky Locker Code Problem Statement:

An ancient locker in a treasure vault opens only when you enter a "Lucky Code". A number is considered lucky if the sum of its digits is a multiple of 4.

You're given a number N. Your task is to find the smallest integer greater than or equal to N which is a lucky code.

Input Format:

* + One line containing an integer N (1 ≤ N ≤ 10⁶)

Output Format:

* + A single integer – the smallest lucky code ≥ N.

Sample Input 1:

100

Sample Output 1:

103

Explanation:

Sum of digits of 100 → 1 → not a multiple of 4

101 → 1 + 0 + 1 = 2 → No

102 → 1 + 0 + 2 = 3 → No

103 → 1 + 0 + 3 = 4 → Correct

Sample Input 2:

1234

Sample Output 2:

1236

Test Cases Input 1

1

Output 1

4

Input 2

16

Output 2

17

Input 3

9999

Output 3

9999

Input 4

256

Output 4

259

Input 5

1006

Output 5

1007

Input 6

2017

Output 6

2019

Input 7

10

Output 7

13

Input 8

45

Output 8

48

Code

def is\_lucky(n):

return sum(int(d) for d in str(n)) % 4 == 0

def find\_lucky\_code(n): while not is\_lucky(n):

n += 1

return n

# Read input

n = int(input()) print(find\_lucky\_code(n))

1. The River Crossing - Find a Pair of Rocks with a Specific Problem Statement:

In a small village by the river, the villagers are crossing a stream using rocks. The rocks are placed at varying distances from each other along the riverbank. A young boy wants to cross the river, but he is only allowed to jump between rocks that are at a specific distance apart. You are tasked with helping the boy identify all the pairs of rocks where the difference between their positions equals a given distance.

Input Format:

* + The first line contains an integer n (1 ≤ n ≤ 1000), the number of rocks placed along the river.
  + The second line contains n integers rocks[0], rocks[1], ..., rocks[n-1] (1 ≤ rocks[i] ≤ 10^6),

representing the positions of the rocks in sorted order.

* + The third line contains an integer target\_distance (1 ≤ target\_distance ≤ 10^6), the distance

between the rocks the boy can jump.

Output Format:

* + For each pair of rocks whose difference in positions equals the target\_distance, print the pair on a new line as (rock1, rock2).
  + If no such pairs exist, print No pairs found.

Sample Input 1:

6

1 3 5 7 9 11

2

Sample Output 1:

(1, 3)

(3, 5)

(5, 7)

(7, 9)

(9, 11)

Sample Input 2:

5

2 4 6 8 10

5

Sample Output 2: No pairs found

Test Cases Input 1

5

2 4 6 8 10

3

Output 1

No pairs found Input 2

4

1 3 5 7

4

Output 2

(1, 5)

(3, 7)

Input 3

6

1 3 5 7 9 11

2

Output 3

(1, 3)

(3, 5)

(5, 7)

(7, 9)

(9, 11)

Input 4

7

1 2 3 4 5 6 7

1

Output 4

(1, 2)

(2, 3)

(3, 4)

(4, 5)

(5, 6)

(6, 7)

Input 5

7

5 1 10 8 6 4 3

2

Output 5

No pairs found

Input 6

5

10 15 20 25 30

5

Output 6

(10, 15)

(15, 20)

(20, 25)

(25, 30)

Input 7

6

1 2 3 4 5 6

1

Output 7

(1, 2)

(2, 3)

(3, 4)

(4, 5)

(5, 6)

Input 8

1

100

5

Output 8

No pairs found

Code

def find\_pairs\_with\_distance(rocks, target\_distance): left = 0

right = 1 pairs = []

while right < len(rocks):

current\_diff = rocks[right] - rocks[left]

if current\_diff == target\_distance: pairs.append((rocks[left], rocks[right])) left += 1

right = left + 1 # Reset the right pointer to one step ahead of the left elif current\_diff < target\_distance:

right += 1 else:

left += 1

if left == right: right += 1

if not pairs:

print("No pairs found") else:

for pair in pairs:

print(f"({pair[0]}, {pair[1]})")

# Example usage

if name == " main ": # Sample Input 1

n = int(input()) # Number of rocks

rocks = list(map(int, input().split())) # List of rock positions target\_distance = int(input()) # The target distance

# Call the function to find pairs find\_pairs\_with\_distance(rocks, target\_distance)

1. Undo the Typos Problem Statement:

Sara is writing her college essay in a simple text editor that only supports typing lowercase alphabets and undoing the last letter using the backspace key (represented by #). She types a sequence of characters, and for every #, the last typed character (if any) gets removed.

Can you help her figure out the final text after all typing and backspaces?

Input Format:

* + A single line string s (1 ≤ |s| ≤ 10^5), consisting of lowercase letters and the # character. Output Format:
  + A single line containing the final string after processing all characters.

Sample Input 1: abc#d##

Sample Output 1: a

Sample Input 2: a#bc##d

Sample Output 2: d

Test Cases Input 1 ##abc Output 1 abc

Input 2 hello Output 2 hello

Input 3 abc###xyz Output 3 xyz

Input 4 helloworld Output 4 helloworld

Input 5 a#b#c#d#e Output 5

e

Input 6 abc##de# Output 6 Ad

Input 7 a#bc#d#efg#h#

Output 7 bef

Input 8 a#b#c#d#e#f#z Output 8

z

Code

def process\_typing(s):

stack = [] for ch in s:

if ch == '#':

if stack:

stack.pop() else:

stack.append(ch) return ''.join(stack)

# Example usage:

if name == " main ": s = input().strip() print(process\_typing(s))

1. Balanced Boxes Problem Statement:

A warehouse robot is stacking boxes represented by opening [ and closing ] brackets. It follows a sequence of commands, where:

* + [ means it places a box on the stack.
  + ] means it removes the top box (only if there is one).

Your job is to determine whether the robot has stacked and unstacked the boxes correctly. That is, every opening [ must have a corresponding closing ], and the order must be valid.

Input Format:

* + A single line string s consisting only of the characters [ and ].
  + Length: 1 ≤ |s| ≤ 10^5

Output Format:

* + Print "Balanced" if all boxes are stacked and unstacked properly.
  + Otherwise, print "Unbalanced".

Sample Input 1: [[]]

Sample Output 1: Balanced

Sample Input 2:

][][

Sample Output 2: Unbalanced

Test Cases Input 1

[]

Output 1 Balanced

Input 2 [[[]]]

Output 2 Balanced

Input 3

][

Output 3 Unbalanced

Input 4 [[[]]

Output 4 Unbalanced

Input 5 []][[]

Output 5 Unbalanced

Input 6 [[[[[]]]]]

Output 6 Balanced

Input 7 [][[[[]]]]

Output 7 Balanced

Input 8

]]]]

Output 8

Unbalanced

Code

def is\_balanced\_boxes(s): stack = []

for ch in s:

if ch == '[':

stack.append(ch) elif ch == ']':

if not stack:

return "Unbalanced" stack.pop()

return "Balanced" if not stack else "Unbalanced"

# Example usage

if name == " main ": s = input().strip() print(is\_balanced\_boxes(s))

Medium

1. The Thief of Time Problem Statement:

In the ancient kingdom of Chronosia, there exists a legendary thief named Kael who can steal time from time crystals. Each time crystal has a time value and a steal time (how long it takes to steal from it). Kael has T minutes before the royal guards catch him. His goal is to maximize the amount of time he can steal from the available crystals before he gets caught.

Each crystal can only be stolen once and stealing must be completed fully to obtain the time value.

Kael decides to steal from the most "efficient" crystals first, where efficiency is defined as the highest time value per unit of steal time. Help Kael make the optimal choice.

Input Format:

* + The first line contains an integer N — the number of time crystals.
  + The second line contains an integer T — the total time Kael has before the guards arrive.
  + The next N lines each contain two integers vi and ti — value of time (vi) and time to steal (ti) for each crystal.

Output Format:

* + Print a single integer — the maximum total value of time Kael can steal before getting caught.

Sample Input 1:

4

10

60 2

100 4

120 6

30 1

Sample Output 1:

190

Sample Input 2:

3

5

50 3

70 4

30 2

Sample Output 2:

70

Test Cases

Input 1

0

100

Output 1

0

Input 2

3

0

100 1

200 2

300 3

Output 2

0

Input 3

3

1000000000

10000 10000

10000 10000

10000 10000

Output 3

30000

Input 4

5

15

50 5

60 4

70 6

80 3

30 2

Output 4

240

Input 5

4

10

100 5

200 10

150 7

50 2

Output 5

200

Input 6

3

4

10 5

100 4

5 1

Output 6

100

Input 7

3

1

100 2

200 3

300 4

Output 7

0

Input 8

3

6

100 2

150 2

120 2

Output 8

370

Code

def max\_stolen\_time\_value(n, t, crystals): # Sort by value per time descending

crystals.sort(key=lambda x: x[0] / x[1], reverse=True)

total\_value = 0 remaining\_time = t

for value, steal\_time in crystals:

if steal\_time <= remaining\_time:

total\_value += value remaining\_time -= steal\_time

else:

continue

return total\_value

# Input reading

if name == " main ": n = int(input())

t = int(input())

crystals = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_stolen\_time\_value(n, t, crystals))

1. The Thief of Time

Problem Statement:

There’s a single ticket counter and a line of people waiting. Each person needs a specific amount of

time to be served, which is represented by an array T.

The counter serves people in order, but here's the rule: the counter works in rounds. In each round, the person at the front gets served for 1 time unit, then goes to the back of the queue if their remaining time is more than 0.

You are given the time required for each person, and you are asked to calculate how many time units it will take before the K-th person (0-indexed) in the original queue completes their service.

Input Format:

* First line contains two integers N (number of people) and K (index of target person).
* Second line contains N space-separated integers, where T[i] is the total time required for the i-th person.

Output Format:

* A single integer — total time taken until the K-th person completes service.

Sample Input 1:

4 2

1 2 3 4

Sample Output 1:

8

Explanation:

[1,2,3,4] → [2,3,4] after 1 (P0 done)

[2,3,4,1] → [3,4,1] after 1 (P1: 1 left)

[3,4,1,1] → [4,1,1] after 1 (P2: 2 left)

[4,1,1,2] → [1,1,2] after 1 (P3: 3 left)

[1,1,2,3] → [1,2,3] after 1 (P1 done)

[2,3] → [3,2] after 1 (P2: 1 left)

[2,1] → [1,1] after 1 (P3: 1 left)

[1] → P2 done after 1 more

Sample Input 2:

5 0

5 1 1 1 1

Sample Output 2:

9

Test Cases Input 1

1 0

1

Output 1

1

Input 2

3 2

1 1 1

Output 2

3

Input 3

3 0

3 3 3

Output 3

7

Input 4

5 4

1 1 1 1 5

Output 4

9

Input 5

6 2

3 3 1 3 3 3

Output 5

3

Input 6

5 2

2 2 2 2 2

Output 6

8

Input 7

5 2

10 2 3 1 1

Output 7

10

Input 8

4 3

1 1 1 5

Output 8

8

Code

def max\_stolen\_time\_value(n, t, crystals): # Sort by value per time descending

crystals.sort(key=lambda x: x[0] / x[1], reverse=True)

total\_value = 0 remaining\_time = t

for value, steal\_time in crystals:

if steal\_time <= remaining\_time:

total\_value += value remaining\_time -= steal\_time

else:

continue

return total\_value

# Input reading

if name == " main ": n = int(input())

t = int(input())

crystals = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_stolen\_time\_value(n, t, crystals))

1. Festival of Lanterns

Problem Statement:

In the kingdom of Lumora, every year a lantern festival is held where participants light floating lanterns along a river. Each lantern must be lit and launched at a specific time interval, and each one brings a certain amount of joy to the crowd.

The Grand Organizer can only allow one lantern to be launched at any given second (due to narrow river constraints). You're given N lanterns, each with:

* + A deadline (the latest second it can be launched),
  + A joy value.

Your task is to select the subset of lanterns to maximize total joy, ensuring no two lanterns are launched at the same time, and each is launched on or before its deadline.

This is a classic Greedy + Priority Queue problem (variant of Job Scheduling with Deadlines).

Input Format:

* + First line: Integer N — number of lanterns.
  + Next N lines: Each line has two integers:
* di — deadline of the i-th lantern (1 ≤ di ≤ 10^5)
* ji — joy value of the i-th lantern

Output Format:

* + One integer — maximum total joy that can be obtained.

Sample Input 1:

5

2 100

1 50

2 200

1 20

3 300

Sample Output 1:

600

Explanation:

We can launch 3 lanterns (at most 1 per second):

* + Choose lanterns with highest joy but that can meet their deadline:
* Launch 300 at time 3
* Launch 200 at time 2
* Launch 100 at time 1

Total Joy = 300 + 200 + 100 = 600

Sample Input 2:

4

1 40

2 50

2 30

1 70

Sample Output 2:

120

Test Cases Input 1

1

1 9999

Output 1

9999

Input 2

4

1 10

1 20

1 5

1 100

Output 2

100

Input 3

3

1 100

2 200

3 300

Output 3

600

Input 4

6

1 5

2 20

2 30

2 25

3 50

3 90

Output 4

170

Input 5

4

1 10

2 10

3 10

4 10

Output 5

40

Input 6

3

5 100

6 200

1. 300

Output 6

600

Input 7

5

2 10

2 20

2 30

2 40

2 50

Output 7

90

Input 8

5

1 10000

1 9000

1 8000

1 7000

1 6000

Output 8

10000

Code

import heapq

def max\_total\_joy(n, lanterns): # Sort lanterns by deadline

lanterns.sort(key=lambda x: x[0]) min\_heap = []

for deadline, joy in lanterns:

heapq.heappush(min\_heap, joy)

# If we have more lanterns than we can schedule by this deadline, remove the least joyful one if len(min\_heap) > deadline:

heapq.heappop(min\_heap)

return sum(min\_heap)

# Input Reading

if name == " main ": n = int(input())

lanterns = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_total\_joy(n, lanterns))

1. The Merchant's Caravan Problem Statement:

A wealthy merchant is heading to the annual Golden Dunes Market, carrying items of different values and weights. His caravan can only carry a maximum weight W due to desert travel constraints.

He wants to maximize his profit by selecting the subset of items that can be carried without exceeding the total weight limit.

This is a 0/1 Knapsack-like greedy problem, but the twist is:

He can take fractions of an item if needed — i.e., this is the Fractional Knapsack Problem, solved greedily by value per unit weight.

Input Format:

* + First line: Two integers N and W
* N: number of items
* W: maximum weight capacity
  + Next N lines: Each line contains two integers:
* value and weight of the item

Output Format:

* + A single floating-point number — the maximum total value the merchant can carry.
  + Round off your answer to 2 decimal places.

Sample Input 1:

3 50

60 10

100 20

120 30

Sample Output 1:

240.00

Explanation:

Sort by value/weight:

* + 60/10 = 6.0
  + 100/20 = 5.0
  + 120/30 = 4.0

Pick:

* + 60 (10)
  + 100 (20)
  + Remaining capacity: 20 → take 2/3 of item worth 120 → value = 80

Total = 60 + 100 + 80 = 240.00

Sample Input 2:

1 1

100 2

Sample Output 2:

50.00

Test Cases Input 1

2 30

100 10

120 30

Output 1

220.00

Input 2

3 60

60 10

100 20

120 30

Output 2

280.00

Input 3

3 10

60 10

100 20

120 30

Output 3

60.00

Input 4

2 5

100 10

120 30

Output 4

50.00

Input 5

4 10

10 1

20 2

30 3

40 4

Output 5

100.00

Input 6

3 50

500 50

400 40

300 30

Output 6

500.00

Input 7

1 2

100 4

Output 7

50.00

Input 8

5 100

10 10

20 20

30 30

40 40

50 50

Output 8

100.00

Code

def max\_value(n, w, items):

items.sort(key=lambda x: x[0]/x[1], reverse=True) total\_value = 0.0

for value, weight in items:

if w == 0:

break

if weight <= w:

total\_value += value w -= weight

else:

total\_value += value \* (w / weight) w = 0

return round(total\_value, 2)

# Driver code

if name == " main ":

n, w = map(int, input().split())

items = [tuple(map(int, input().split())) for \_ in range(n)] print(f"{max\_value(n, w, items):.2f}")

1. The Band Booking Problem

Problem Statement:

A music event organizer wants to schedule bands for a music marathon. Each band has a performance start time, end time, and a popularity score (which is equal to the number of tickets they help sell).

Due to logistical constraints, no two bands can perform at overlapping times. The goal is to select a subset of non-overlapping bands to maximize total popularity.

Input Format:

* + First line: Single integer N (number of bands)
  + Next N lines: Three integers per line:

o start\_time end\_time popularity

Output Format:

* + A single integer: the maximum total popularity achievable with non-overlapping performances

Sample Input 1:

4

1 3 50

2 5 60

4 6 70

1. 7 30

Sample Output 1:

150

Explanation:

Take band 1 (1-3) → pop 50

Skip band 2 (2-5) – overlaps

Take band 3 (4-6) → pop 70

Take band 4 (6-7) → pop 30

Total = 50 + 70 + 30 = 150

Sample Input 2:

4

1 3 50

2 5 60

4 6 70

6 7 30

Sample Output 2:

150

Test Cases Input 1

3

1 2 20

3 4 40

5 6 60

Output 1

120

Input 2

3

1 5 10

2 6 30

3 7 20

Output 2

30

Input 3

4

1 3 50

3 5 50

5 7 50

2 8 200

Output 3

200

Input 4

3

1 10 100

2 3 20

4 5 30

Output 4

100

Input 5

5

1 3 20

3 6 30

6 9 40

9 12 50

12 15 60

Output 5

200

Input 6

4

1 5 100

10 15 150

20 25 200

30 35 250

Output 6

700

Input 7

5

1 10 100

1 2 30

2 3 30

3 4 30

4 5 30

Output 7

120

Input 8

6

1 3 50

3 5 60

5 7 70

1. 9 80

9 11 90

11 13 100

Output 8

450

Code

import bisect

def max\_popularity(bands):

bands.sort(key=lambda x: x[1]) # Sort by end time end\_times = [band[1] for band in bands]

dp = [0] \* (len(bands) + 1)

for i in range(1, len(bands)+1):

start, end, pop = bands[i-1]

# Find last band that ends before current starts idx = bisect.bisect\_right(end\_times, start) - 1 dp[i] = max(dp[i-1], dp[idx+1] + pop)

return dp[-1]

# Driver

if name == " main ": n = int(input())

bands = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_popularity(bands))

1. The Conference Room Dilemma

Problem Statement:

You are given N meetings, each with a start time and end time. All meetings must be scheduled in conference rooms, and no two overlapping meetings can be in the same room.

Your task is to find the minimum number of rooms required to accommodate all meetings without overlap.

Input Format:

* + First line: Integer N (number of meetings)
  + Next N lines: Two integers — start end

Output Format:

* + A single integer — minimum number of rooms required

Sample Input 1:

4

0 30

5 10

15 20

25 35

Sample Output 1:

2

Sample Input 2:

3

1 2

3 4

5 6

Sample Output 2:

1

Test Cases Input 1

4

1 10

1 10

1 10

1 10

Output 1

4

Input 2

5

1 4

2 5

3 6

6 7

7 8

Output 2

3

Input 3

4

1 2

2 3

3 4

4 5

Output 3

1

Input 4

3

1 10

2 10

3 10

Output 4

3

Input 5

5

1 10

2 3

3 4

4 5

5 6

Output 5

2

Input 6

6

1 5

2 6

5 7

6 8

7 9

1. 10

Output 6

2

Input 7

4

1 5

2 5

3 5

4 5

Output 7

4

Input 8

6

10 20

30 40

50 60

15 25

35 45

55 65

Output 8

2

Code

def min\_rooms(meetings):

start\_times = sorted([s for s, e in meetings]) end\_times = sorted([e for s, e in meetings])

start\_ptr = end\_ptr = 0 current\_rooms = max\_rooms = 0

while start\_ptr < len(meetings):

if start\_times[start\_ptr] < end\_times[end\_ptr]: current\_rooms += 1

max\_rooms = max(max\_rooms, current\_rooms) start\_ptr += 1

else:

current\_rooms -= 1

end\_ptr += 1

return max\_rooms

# Driver

if name == " main ": n = int(input())

meetings = [tuple(map(int, input().split())) for \_ in range(n)] print(min\_rooms(meetings))

1. The Coupon Collector

Problem Statement:

You are at a store that offers coupons on N items. For each item, you are given:

* + The price of the item.
  + The discount if a coupon is used.

But… you only have K coupons, and each coupon can be applied to only one item.

Your task is to minimize the total cost of buying all items by using at most K coupons, where each coupon gives the full discount on one item.

Input Format:

* + First line: Two integers N and K
  + Next N lines: Two integers price discount

Output Format:

* + A single integer — minimum total cost after using coupons.

Sample Input 1:

5 2

100 20

200 50

150 30

120 60

80 10

Sample Output 1:

540

Explanation:

Use coupons on items with the maximum discount: 60 and 50

* + Prices: [100, 200, 150, 120, 80] → Total = 650
  + Apply coupons to items with discount 60 and 50 → Save 110
  + Final cost = 650 - 110 = 540

Sample Input 2:

3 0

100 50

200 100

150 75

Sample Output 2:

450

Test Cases Input 1

3 3

100 10

100 20

100 30

Output 1

240

Input 2

2 5

300 100

400 150

Output 2

450

Input 3

4 2

100 0

200 0

300 0

400 0

Output 3

1000

Input 4

3 2

100 100

200 200

300 300

Output 4

100

Input 5

1 0

999 888

Output 5

999

Input 6

128

Output 6

1

Input 7

6 3

100 5

100 10

100 15

100 20

100 25

100 30

Output 7

525

Input 8

6 3

100 5

100 10

100 15

100 20

100 25

100 20

Output 8

535

Code

def min\_total\_cost(n, k, items):

total\_price = sum(price for price, \_ in items) discounts = sorted((d for \_, d in items), reverse=True) total\_discount = sum(discounts[:k])

return total\_price - total\_discount

# Driver

if name == " main ":

n, k = map(int, input().split())

items = [tuple(map(int, input().split())) for \_ in range(n)] print(min\_total\_cost(n, k, items))

1. Penalty-Free Submissions

Problem Statement:

You are a competitive programmer participating in a contest with N problems. Each problem has:

* + deadline: the latest time you can submit it without penalty.
  + penalty: the penalty incurred if you miss the deadline (if not done at all or submitted late).

Each problem takes exactly 1 unit of time to solve, and you can only solve one problem per unit time.

Your goal is to maximize the total penalty avoided, i.e., pick problems in such a way that you solve the highest-penalty problems within their deadlines.

Input Format:

* + First line: Integer N – number of problems
  + Next N lines: Two integers deadline and penalty for each problem

Output Format:

A single integer – maximum total penalty you can avoid

Sample Input 1:

5

2 100

1 19

2 27

1 25

3 15

Sample Output 1:

142

Sample Input 2:

4

1 50

1 40

1 30

1 20

Sample Output 2:

50

Test Cases Input 1

3

1 10

2 20

3 30

Output 1

60

Input 2

4

1 10

1 20

1 30

1 40

Output 2

40

Input 3

5

1 5

2 50

3 15

2 20

3 25

Output 3

95

Input 4

6

2 10

2 20

2 30

2 40

2 50

2 60

Output 4

110

Input 5

3

10 100

1. 90

8 80

Output 5

270

Input 6

6

1 100

2 10

1 30

3 20

2 90

3 80

Output 6

270

Input 7

4

1 50

2 50

3 50

2 50

Output 7

150

Input 8

5

10000 1000000

9000 900000

8000 800000

7000 700000

6000 600000

Output 8

4000000

Code

def max\_penalty\_avoided(n, tasks):

tasks.sort(key=lambda x: -x[1]) # Sort by penalty descending max\_deadline = max(d for d, \_ in tasks)

slots = [False] \* (max\_deadline + 1) total = 0

for deadline, penalty in tasks: for t in range(deadline, 0, -1):

if not slots[t]:

slots[t] = True total += penalty break

return total

# Driver

if name == " main ": n = int(input())

tasks = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_penalty\_avoided(n, tasks))

1. Minimum Rooms Required

Problem Statement:

You are given N events, each with a start time and an end time. Your task is to determine the minimum number of rooms required to schedule all events without any overlap in a single room.

Each room can hold only one event at a time. Events are considered overlapping if one starts before another ends (i.e., inclusive).

Input Format:

* + First line: Integer N – number of events
  + Next N lines: Two integers start and end for each event (inclusive)

Output Format:

A single integer – minimum number of rooms required.

Sample Input 1:

5

0 30

5 10

15 20

35 45

25 40

Sample Output 1:

2

Sample Input 2:

4

0 5

6 10

11 15

16 20

Sample Output 2:

1

Test Cases Input 1

4

1 10

2 9

3 8

4 7

Output 1

4

Input 2

6

1 5

2 6

5 9

7 10

10 13

12 15

Output 2

3

Input 3

5

1 100

2 3

4 5

6 7

1. 9

Output 3

2

Input 4

1

0 1

Output 4

1

Input 5

4

0 1

2 3

4 5

6 7

Output 5

1

Input 6

3

1 1

1 1

1 1

Output 6

3

Input 7

3

1 2

2 3

3 4

Output 7

2

Input 8

8

0 10

10 20

20 30

5 15

15 25

25 35

30 40

35 45

Output 8

3

Code

import heapq

def min\_rooms(events): events.sort() # Sort by start time min\_heap = []

for start, end in events:

if min\_heap and min\_heap[0] < start: heapq.heappop(min\_heap)

heapq.heappush(min\_heap, end)

return len(min\_heap)

# Driver

if name == " main ": n = int(input())

events = [tuple(map(int, input().split())) for \_ in range(n)] print(min\_rooms(events))

1. Deadline-Focused Task Execution Problem Statement:

You are given N tasks. Each task takes a certain amount of time to complete and has a deadline. You can execute the tasks one after another, starting from time 0. Your goal is to complete the maximum number of tasks such that each task is finished on or before its deadline.

You can only work on one task at a time, and you must choose tasks wisely (greedy!) so you can finish as many as possible.

Input Format:

* + First line: Integer N – number of tasks
  + Next N lines: Two integers duration and deadline for each task Output Format:
  + A single integer – maximum number of tasks that can be completed on or before their deadlines

Sample Input 1:

5

3 9

2 8

1 9

2 15

5 12

Sample Output 1:

5

Sample Input 2:

3

2 5

1 3

2 6

Sample Output 2:

3

Test Cases Input 1

4

2 4

3 5

4 6

10 7

Output 1

2

Input 2

4

1 1

2 3

3 6

4 10

Output 2

4

Input 3

5

1 5

1 6

1 7

1 8

1 9

Output 3

5

Input 4

3

3 2

4 3

5 4

Output 4

0

Input 5

5

6 12

1 3

2 4

1 5

3 6

Output 5

4

Input 6

4

3 10

2 10

1 10

4 10

Output 6

4

Input 7

6

5 10

3 8

2 5

1 3

6 12

4 11

Output 7

4

Input 8

7

10 100

20 100

30 100

40 100

15 100

5 100

25 100

Output 8

5

Code

import heapq

def max\_tasks(tasks):

tasks.sort(key=lambda x: x[1]) # sort by deadline total\_time = 0

durations = []

for duration, deadline in tasks:

total\_time += duration heapq.heappush(durations, -duration) if total\_time > deadline:

total\_time += heapq.heappop(durations) # remove longest task

return len(durations)

# Driver

if name == " main ": n = int(input())

tasks = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_tasks(tasks))

1. Charging Station Scheduling Problem Statement:

You're managing a charging station with K charging ports. There are N electric cars that want to charge. Each car arrives at a specific time and needs a specific duration to fully charge.

Your goal is to maximize the number of cars that can be charged, assuming:

* + A car can only start charging at or after its arrival time.
  + A port is immediately available after the last car using it finishes.
  + You can charge up to K cars simultaneously at any time.

Use a greedy algorithm to schedule cars so that the maximum number can finish charging.

Input Format:

* + First line: Two integers N (number of cars), K (number of ports)
  + Next N lines: Two integers arrival\_time and duration for each car

Output Format:

* + A single integer – the maximum number of cars that can be charged

Sample Input 1:

5 2

1 4

2 3

3 2

10 1

5 2

Sample Output 1:

4

Sample Input 2:

3 1

1 2

3 2

5 2

Sample Output 2:

3

Test Cases Input 1

4 2

1 2

3 2

5 2

7 2

Output 1

4

Input 2

4 1

1 5

2 5

3 5

4 5

Output 2

1

Input 3

3 3

1 5

1 5

1 5

Output 3

3

Input 4

5 2

1 4

2 3

3 2

10 1

5 2

Output 4

4

Input 5

4 2

1 2

2 2

10 1

11 1

Output 5

4

Input 6

6 2

1 3

2 2

3 2

4 2

5 2

6 2

Output 6

5

Input 7

10 2

1 2

1 2

1 2

1 2

1 2

3 2

3 2

3 2

5 2

5 2

Output 7

6

Input 8

6 2

1 1

2 1

3 1

4 1

5 1

6 1

Output 8

6

Code

import heapq

def max\_charged\_cars(cars, k):

# Sort cars by end time (arrival + duration) cars.sort(key=lambda x: x[0] + x[1])

ports = [] # Min-heap of end times count = 0

for arrival, duration in cars:

# Free up all ports where charging is done while ports and ports[0] <= arrival:

heapq.heappop(ports) if len(ports) < k:

heapq.heappush(ports, arrival + duration) count += 1

return count

# Driver

if name == " main ":

n, k = map(int, input().split())

cars = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_charged\_cars(cars, k))

1. Minimum Coins to Pay Problem Statement:

You are given an amount N and a list of M coin denominations. Your task is to find the minimum number of coins needed to make up that amount using the available denominations. You can use each coin type an unlimited number of times.

Input Format:

* + First line: Two integers N (target amount), M (number of denominations)
  + Second line: M integers – the coin denominations. Output Format:
  + A single integer – the minimum number of coins needed to make N

Sample Input 1:

49 6

1 2 5 10 20 50

Sample Output 1:

5

Explanation:

Use largest coins first:

* + 20 + 20 = 40
  + 5 = 45
  + 2 = 47
  + 1 + 1 = 49
  + Total: 5 coins

Sample Input 2:

49 6

1 2 5 10 20 50

Sample Output 2:

5

Test Cases Input 1

100 4

1 5 10 100

Output 1

1

Input 2

37 5

1 5 10 20 25

Output 2

4

Input 3

7 1

1

Output 3

7

Input 4

18 3

2 5 10

Output 4

3

Input 5

18 4

1 2 5 10

Output 5

4

Input 6

999 6

1 2 5 10 20 100

Output 6

17

Input 7

15 6

1 1 5 5 10 10

Output 7

2

Input 8

7 2

2 4

Output 8

2

Code

def min\_coins\_to\_pay(n, coins): coins.sort(reverse=True) count = 0

for coin in coins:

while n >= coin: n -= coin count += 1

return count

# Driver

if name == " main ":

n, m = map(int, input().split()) coins = list(map(int, input().split())) print(min\_coins\_to\_pay(n, coins))

1. Job Sequencing for Maximum Profit Problem Statement:

You are given N tasks. Each task has a deadline and a reward. You can only do one task per day, and each task takes exactly one day. If a task is not completed on or before its deadline, you miss the reward.

Your goal is to maximize the total reward by selecting and scheduling tasks greedily.

Input Format:

* + First line: An integer N (number of tasks)
  + Next N lines: Two integers D and R per line (deadline and reward) Output Format:

A single integer – maximum total reward you can earn

Sample Input 1:

5

2 50

1 10

2 20

1 30

3 40

Sample Output 1:

120

Sample Input 2:

4

1 10

1 20

1 30

1 40

Sample Output 2:

40

Test Cases Input 1

4

1 10

2 20

3 30

4 40

Output 1

100

Input 2

6

2 50

2 60

2 70

2 40

2 30

2 20

Output 2

130

Input 3

3

3 15

2 10

1 20

Output 3

45

Input 4

5

1 10

2 10

3 10

2 10

1 10

Output 4

30

Input 5

5

1 100

2 10

2 10

3 10

3 10

Output 5

120

Input 6

1

1 1000

Output 6

1000

Input 7

4

10 100

11 200

12 300

13 400

Output 7

1000

Input 8

6

1 10

1 20

1 30

1 40

1 50

1 60

Output 8

60

Code

def find(parent, x): if parent[x] != x:

parent[x] = find(parent, parent[x])

return parent[x]

def union(parent, x, y): parent[find(parent, x)] = find(parent, y)

def max\_total\_reward(tasks):

tasks.sort(key=lambda x: -x[1]) # Sort by reward descending max\_deadline = max(d for d, \_ in tasks)

parent = [i for i in range(max\_deadline + 2)]

total\_reward = 0 for d, r in tasks:

available\_day = find(parent, d) if available\_day > 0:

total\_reward += r

union(parent, available\_day, available\_day - 1) return total\_reward

# Driver

if name == " main ": n = int(input())

tasks = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_total\_reward(tasks))

Hard

1. Magical Paths in the Infinite Forest

Problem Statement:

The Infinite Forest is a mystical place represented as a 2D grid. You start at the top-left cell (0, 0) and want to reach the bottom-right cell (N-1, M-1). Each cell contains a magical cost, which you must pay to enter that cell.

You can only move right or down at any point in time. But there's a twist.

There are some rare teleport tiles in the grid (identified by a value of -1). From a teleport tile, you can jump to any other teleport tile for free, without paying the cost of the destination teleport tile (but you must still have passed into the teleport tile you're jumping from).

Your goal is to find the minimum magical cost to travel from the start to the destination, considering both normal and teleportation moves.

Input Format:

* The first line contains two integers N and M — the dimensions of the forest.
* The next N lines contain M space-separated integers. Each cell contains:
* A positive integer denoting the cost of entering that cell, or
* -1 denoting a teleport tile.

Output Format:

Print a single integer — the minimum cost to reach (N-1, M-1) from (0, 0).

Sample Input 1:

4 4

1 3 -1 4

2 8 5 9

4 -1 3 1

6 7 2 5

Sample Output 1:

13

Sample Input 2:

3 3

1 -1 3

2 4 -1

1 1 1

Sample Output 2:

2

Test Cases Input 1

3 3

1 2 3

4 5 6

7 8 9

Output 1

21

Input 2

3 3

1 -1 3

2 100 -1

1 1 1

Output 2

2

Input 3

4 4

1 2 -1 9

8 9 9 9

-1 9 9 9

1. 9 9 1

Output 3

31

Input 4

3 3

-1 3 1

2 -1 2

3 4 1

Output 4

3

Input 5

3 3

1 2 3

4 5 6

7 8 -1

Output 5

12

Input 6

3 3

1 2 3

4 -1 6

7 8 9

Output 6

18

Input 7

4 4

1 -1 -1 1

1 100 100 1

1 100 100 1

1 1 1 1

Output 7

5

Input 8

4 4

1 99 99 99

-1 99 99 99

99 99 99 -1

99 99 99 1

Output 8

2

Code import sys

import heapq

def min\_magic\_cost(grid, n, m): INF = float('inf')

dp = [[INF] \* m for \_ in range(n)]

dp[0][0] = grid[0][0] if grid[0][0] != -1 else 0

# Collect teleport tiles teleports = []

for i in range(n):

for j in range(m):

if grid[i][j] == -1: teleports.append((i, j))

# Priority queue: (cost, x, y) heap = [(dp[0][0], 0, 0)]

visited\_teleport = False

while heap:

cost, x, y = heapq.heappop(heap)

if cost > dp[x][y]: continue

# Normal right/down moves

for dx, dy in [(1, 0), (0, 1)]:

nx, ny = x + dx, y + dy

if 0 <= nx < n and 0 <= ny < m:

enter\_cost = 0 if grid[nx][ny] == -1 else grid[nx][ny] if dp[nx][ny] > cost + enter\_cost:

dp[nx][ny] = cost + enter\_cost heapq.heappush(heap, (dp[nx][ny], nx, ny))

# If on teleport, update all other teleports if grid[x][y] == -1 and not visited\_teleport:

for tx, ty in teleports:

if (tx, ty) != (x, y):

if dp[tx][ty] > cost:

dp[tx][ty] = cost

heapq.heappush(heap, (dp[tx][ty], tx, ty)) visited\_teleport = True # only need one full teleport sync

return dp[n-1][m-1]

# Reading input

n, m = map(int, input().split())

grid = [list(map(int, input().split())) for \_ in range(n)]

print(min\_magic\_cost(grid, n, m))

1. Longest Alternating Subsequence with Jumps

Problem Statement:

You are given an array of integers A of length N. A subsequence S of A is called a "Jumpy Alternating Subsequence" if:

1. It consists of elements from A in increasing index order (i.e., it's a subsequence).
2. The difference between consecutive elements in S must strictly alternate in sign.
3. The absolute difference between any two consecutive elements must be at least K. Your task is to find the length of the longest such subsequence.

Input Format:

* + First line: Two integers N and K — the length of the array and the minimum absolute jump size.
  + Second line: N space-separated integers — the array A.

Output Format:

* + A single integer — the length of the longest jumpy alternating subsequence.

Sample Input 1:

6 2

1 7 4 9 2 5

Sample Output 1:

6

Explanation:

One such longest sequence is [1, 7, 4, 9, 2, 5] with diffs [+6, -3, +5, -7, +3].

Sample Input 2:

5 3

1 2 3 4 5

Sample Output 2:

2

Test Cases Input 1

6 2

1 7 4 9 2 5

Output 1

6

Input 2

5 1

3 3 3 3 3

Output 2

1

Input 3

5 10

1 2 3 4 5

Output 3

1

Input 4

7 0

1 17 5 10 13 15 10

Output 4

5

Input 5

7 3

10 20 5 25 1 30 0

Output 5

7

Input 6

4 2

1 1 1 4

Output 6

2

Input 7

5 4

10 5 1 -3 -10

Output 7

2

Input 8

10 1

1 3 2 4 3 5 4 6 5 7

Output 8

10

Code

def longest\_jumpy\_alternating\_subsequence(arr, k): n = len(arr)

dp\_up = [1] \* n # ending with upward jump dp\_down = [1] \* n # ending with downward jump

for i in range(1, n):

for j in range(i):

diff = arr[i] - arr[j] if abs(diff) >= k:

if diff > 0:

dp\_up[i] = max(dp\_up[i], dp\_down[j] + 1) elif diff < 0:

dp\_down[i] = max(dp\_down[i], dp\_up[j] + 1)

return max(max(dp\_up), max(dp\_down))

# Read input

n, k = map(int, input().split())

arr = list(map(int, input().split()))

print(longest\_jumpy\_alternating\_subsequence(arr, k))

1. Minimum Sum Subsequence with Non-Adjacent Elements

Problem Statement:

You're given an array of integers arr of length n. You need to select a subsequence such that:

1. No two adjacent elements in the original array can be picked (i.e., if you pick arr[i], you cannot pick arr[i+1]).
2. The sum of the selected elements is minimized.
3. You must pick at least one element.

Write a program to compute the minimum possible sum under these rules.

Input Format:

* + First line contains a single integer n – the size of the array.
  + Second line contains n space-separated integers – the elements of the array. Output Format:
  + A single integer: Minimum number of button presses required to type the message.

Sample Input 1:

5

3 2 5 10 7

Sample Output 1:

2

Sample Input 2:

1

5

Sample Output 2:

5

Test Cases Input 1

2

10 2

Output 1

2

Input 2

6

-5 -10 -3 -1 -6 -9

Output 2

-20

Input 3

7

1 100 2 100 3 100 4

Output 3

1

Input 4

5

4 5 1 5 4

Output 4

4

Input 5

6

7 7 7 7 7 7

Output 5

7

Input 6

3

5 100

6 200

7 300

Output 6

600

Input 7

5

8 7 -100 6 9

Output 7

-92

Input 8

6

10 9 8 7 6 5

Output 8

9

Code

def min\_non\_adjacent\_sum(arr): n = len(arr)

if n == 1:

return arr[0]

dp = [0] \* n dp[0] = arr[0]

dp[1] = min(arr[0], arr[1])

for i in range(2, n):

dp[i] = min(dp[i - 1], dp[i - 2] + arr[i])

return dp[-1]

# Input

n = int(input())

arr = list(map(int, input().split())) print(min\_non\_adjacent\_sum(arr))

1. Maximize Length of Chain of Pairs Problem Statement:

You are given n pairs of integers. Each pair represents a directed link (a, b) such that a < b.

You can form a chain by choosing some pairs such that for each consecutive pair (a, b) and (c, d) in the chain, b < c.

Your task is to select the maximum number of pairs you can chain together following this rule.

Input Format:

* + First line: An integer n — number of pairs.
  + Next n lines: Each line has two integers a and b Output Format:

A single integer — the maximum number of pairs that can be chained.

Sample Input 1:

4

5 24

15 25

27 40

50 60

Sample Output 1:

3

Sample Input 2:

5

1 2

2 3

3 4

4 5

5 6

Sample Output 2:

3

Test Cases Input 1

5

1 2

2 3

3 4

4 5

5 6

Output 1

3

Input 2

3

1 10

2 3

3 4

Output 2

1

Input 3

1

0 1

Output 3

1

Input 4

2

1 5

2 6

Output 4

1

Input 5

6

1 2

2 3

4 5

6 7

8 9

10 11

Output 5

5

Input 6

6

5 10

1 4

7 11

13 20

6 8

21 25

Output 6

4

Input 7

0

Output 7

0

Input 8

7

-5 -3

-1 1

2 3

4 6

6 8

9 12

13 15

Output 8

6

Code

def max\_chain\_length(pairs): # Sort pairs by second value

pairs.sort(key=lambda x: x[1])

count = 0

current\_end = float('-inf')

for a, b in pairs:

if a > current\_end:

count += 1 current\_end = b

return count

# Input

n = int(input())

pairs = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_chain\_length(pairs))

1. Split Array into k Subarrays with Minimum Max Sum

Problem Statement:

You are given an array of n integers and an integer k.

Your task is to split the array into k non-empty continuous subarrays, such that the maximum sum among these k subarrays is minimized.

Return that minimum possible maximum subarray sum.

Input Format:

* + First line: Two integers n and k
  + Second line: n space-separated integers

Output Format:

A single integer — the minimum possible value of the largest subarray sum when the array is split into k parts.

Sample Input 1:

5 2

7 2 5 10 8

Sample Output 1:

18

Sample Input 2:

4 2

1 2 3 4

Sample Output 2:

6

Test Cases Input 1

1 1

100

Output 1

100

Input 2

6 3

1 4 4 1 3 2

Output 2

5

Input 3

6 4

1 4 4 1 3 22 8 200

Output 3

5

Input 4

10 5

1 1 1 1 1 1 1 1 1 10

Output 4

10

Input 5

5

1 3 20

3 6 30

6 9 40

9 12 50

12 15 60

Output 5

200

Input 6

7 3

2 3 1 2 4 3 1

Output 6

6

Input 7

5 5

1 2 3 4 5

Output 7

5

Input 8

6 1

1 2 3 4 5 6

Output 8

21

Code

def can\_split(arr, k, max\_allowed): count = 1

current\_sum = 0 for num in arr:

if current\_sum + num > max\_allowed: count += 1

current\_sum = num else:

current\_sum += num return count <= k

def split\_array\_min\_largest\_sum(arr, k): low = max(arr)

high = sum(arr) result = high

while low <= high:

mid = (low + high) // 2 if can\_split(arr, k, mid):

result = mid high = mid - 1

else:

low = mid + 1

return result

# Input

n, k = map(int, input().split())

arr = list(map(int, input().split())) print(split\_array\_min\_largest\_sum(arr, k))

1. Minimum Cuts for Palindrome Partitioning

Problem Statement:

You are given a string s. Your task is to partition the string such that every substring of the partition is a palindrome.

Return the minimum number of cuts needed to make all parts of the string palindromes.

Input Format:

* + First line: A single string s (lowercase English letters only)

Output Format:

* + A single integer — the minimum number of cuts needed.

Sample Input 1: aab

Sample Output 1:

1

Sample Input 2: a

Sample Output 2:

0

Test Cases Input 1 abc Output 1

2

Input 2 abba Output 2

0

Input 3 racecar Output 3

0

Input 4 noonabbad Output 4

2

Input 5 aaaaaa Output 5

0

Input 6 abacdc Output 6

1

Input 7 Banana Output 7

1

Input 8 Civicracecar Output 8

1

Code

def min\_cut\_palindrome(s):

n = len(s)

is\_palindrome = [[False]\*n for \_ in range(n)]

# Precompute palindromes for i in range(n):

is\_palindrome[i][i] = True for i in range(n-1):

is\_palindrome[i][i+1] = (s[i] == s[i+1]) for length in range(3, n+1):

for i in range(n - length + 1):

j = i + length - 1

is\_palindrome[i][j] = (s[i] == s[j]) and is\_palindrome[i+1][j-1]

# DP to calculate min cuts dp = [0] \* n

for i in range(n):

if is\_palindrome[0][i]:

dp[i] = 0 else:

dp[i] = float('inf') for j in range(i):

if is\_palindrome[j+1][i]: dp[i] = min(dp[i], dp[j] + 1)

return dp[-1]

# Input

s = input().strip()

print(min\_cut\_palindrome(s))

1. Longest Zigzag Subsequence

Problem Statement:

Given an array of integers, you must find the length of the longest subsequence where the differences between consecutive elements strictly alternate between positive and negative.

A subsequence is not necessarily contiguous. The first difference can be either positive or negative.

Input Format:

* + First line: An integer n — the number of elements in the array.
  + Second line: n space-separated integers.

Output Format:

* + A single integer — the length of the longest zigzag subsequence.

Sample Input 1:

6

1 7 4 9 2 5

Sample Output 1:

6

Sample Input 2:

5

1 4 7 2 5

Sample Output 2:

4

Test Cases Input 1

1

10

Output 1

1

Input 2

3

1 1 1

Output 2

1

Input 3

4

1 2 3 4

Output 3

2

Input 4

4

4 3 2 1

Output 4

2

Input 5

7

70 55 13 2 99 2 80

Output 5

5

Input 6

8

3 3 3 3 3 3 3 3

Output 6

1

Input 7

10

10 22 9 33 49 50 31 60 4 70

Output 7

8

Input 8

5

1 3 2 4 3

Output 8

5

Code

def longest\_zigzag\_subsequence(arr): n = len(arr)

if n == 0:

return 0

up = [1] \* n down = [1] \* n

for i in range(1, n):

for j in range(i):

if arr[i] > arr[j]:

up[i] = max(up[i], down[j] + 1) elif arr[i] < arr[j]:

down[i] = max(down[i], up[j] + 1)

return max(max(up), max(down))

# Input

n = int(input())

arr = list(map(int, input().split())) print(longest\_zigzag\_subsequence(arr))

1. Maximum Sum of Non-Overlapping Intervals

Problem Statement:

You're given n intervals, each with a start time, end time, and a value. Your goal is to select a subset of non-overlapping intervals such that the sum of their values is maximized.

Input Format:

* + First line: An integer n — the number of intervals.
  + Next n lines: Each line contains three integers: start end value

Output Format:

A single integer — the maximum sum of values for non-overlapping intervals.

Sample Input 1:

4

1 3 50

3 5 20

6 19 100

2 100 200

Sample Output 1:

200

Sample Input 2:

3

1 2 10

2 3 20

3 4 30

Sample Output 2:

60

Test Cases Input 1

3

1 4 10

2 5 20

3 6 30

Output 1

30

Input 2

5

1 10 100

2 3 10

3 4 20

4 5 30

5 6 40

Output 2

100

Input 3

2

1 10 100

11 20 200

Output 3

300

Input 4

3

1 5 20

6 10 30

5 6 10

Output 4

60

Input 5

3

1 2 100

2 3 100

3 4 100

Output 5

300

Input 6

1

1 1000000000 5000

Output 6

5000

Input 7

4

1 3 10

2 4 20

3 5 30

4 6 40

Output 7

60

Input 8

5

1 2 1

2 3 2

3 4 3

4 5 4

5 6 5

Output 8

15

Code

import bisect

def max\_value\_non\_overlapping(intervals): intervals.sort(key=lambda x: x[1])

n = len(intervals)

# Prepare end times for binary search ends = [interval[1] for interval in intervals] dp = [0] \* (n + 1)

for i in range(1, n + 1):

start, end, value = intervals[i - 1]

# Find the last interval that doesn't overlap idx = bisect.bisect\_right(ends, start) - 1 dp[i] = max(dp[i - 1], dp[idx + 1] + value)

return dp[n]

# Input

n = int(input())

intervals = [tuple(map(int, input().split())) for \_ in range(n)] print(max\_value\_non\_overlapping(intervals))

1. Count Beautiful Numbers

Problem Statement:

A number is called beautiful if the sum of its digits is divisible by k.

You are given two integers L and R, and a value k. Count how many numbers between L and R (inclusive) are beautiful.

Input Format:

* + A single line with three integers: L R k

Output Format:

A single integer — the count of beautiful numbers in the range [L, R].

Sample Input 1:

1 100 10

Sample Output 1:

9

Sample Input 2:

1 20 5

Sample Output 2:

3

Test Cases Input 1

10 99 9

Output 1

10

Input 2

1 1000 10

Output 2

99

Input 3

100 200 15

Output 3

5

Input 4

123456 654321 7

Output 4

75845

Input 5

1 1000000 1

Output 5

1000000

Input 6

1 1000000 99

Output 6

0

Input 7

1 1 1

Output 7

1

Input 8

9999999999999999 10000000000000000 9

Output 8

1

Code

from functools import lru\_cache def count\_beautiful(n, k):

digits = list(map(int, str(n)))

@lru\_cache(None)

def dp(pos, sum\_mod\_k, tight): if pos == len(digits):

return 1 if sum\_mod\_k == 0 else 0

limit = digits[pos] if tight else 9 total = 0

for d in range(0, limit + 1):

total += dp( pos + 1,

(sum\_mod\_k + d) % k, tight and (d == limit)

)

return total

return dp(0, 0, True)

def solve(L, R, k):

return count\_beautiful(R, k) - count\_beautiful(L - 1, k)

# Input

L, R, k = map(int, input().split()) print(solve(L, R, k))

1. Minimum Incompatibility Partition Problem Statement:

You're given an array of n integers and an integer k. You need to partition the array into k subsets of equal size. Each subset must contain unique elements only (no duplicates). The incompatibility of a subset is the difference between its maximum and minimum elements.

Your goal is to minimize the sum of incompatibilities across all subsets. If it's not possible to partition the array as required, return -1.

Input Format:

* + First line: Two integers n and k
  + Second line: n integers representing the array

Output Format:

* + A single integer — the minimum sum of incompatibilities or -1 if not possible. Sample Input 1:

8 4

1 2 1 4 3 3 5 6

Sample Output 1:

5

Sample Input 2:

6 3

1 2 3 4 5 6

Sample Output 2:

3

Test Cases Input 1

5 2

1 2 3 4 5

Output 1

-1

Input 2

8 4

1 2 1 4 3 3 5 6

Output 2

5

Input 3

6 2

1 1 1 2 2 2

Output 3

-1

Input 4

4 2

5 5 7 7

Output 4

4

Input 5

7 2

1 2 3 4 5 6 7

Output 5

-1

Input 6

6 3

3 1 6 2 4 5

Output 6

3

Input 7

4 2

8 8 8 8

Output 7

-1

Input 8

8 4

10 20 30 40 50 60 70 80

Output 8

40

Code

from itertools import combinations from collections import defaultdict import sys

def min\_incompatibility(nums, k): from functools import lru\_cache

n = len(nums) size = n // k

# If total elements can't be evenly split into k subsets if n % k != 0:

return -1

count = defaultdict(int) for num in nums:

count[num] += 1

if count[num] > k: # More than k duplicates - cannot split uniquely return -1

all\_masks = [] mask\_cost = {}

# Try all combinations of size `size`

for combo in combinations(range(n), size): seen = set()

subset = []

for i in combo:

if nums[i] in seen:

break seen.add(nums[i]) subset.append(nums[i])

else:

mask = 0

for i in combo:

mask |= 1 << i

mask\_cost[mask] = max(subset) - min(subset) all\_masks.append(mask)

dp = [sys.maxsize] \* (1 << n) dp[0] = 0

for mask in range(1 << n):

if bin(mask).count('1') % size != 0:

continue

for sub in all\_masks:

if (mask & sub) == 0:

next\_mask = mask | sub

dp[next\_mask] = min(dp[next\_mask], dp[mask] + mask\_cost[sub])

result = dp[(1 << n) - 1]

return result if result != sys.maxsize else -1

# Input Reading

if name == " main ":

n, k = map(int, input().split())

nums = list(map(int, input().split())) print(min\_incompatibility(nums, k))

1. Minimum Cost to Make Palindromes Problem Statement:

You are given a string s of lowercase English letters. You want to partition s into k non-empty, non- overlapping contiguous substrings such that each substring is a palindrome. You are allowed to change characters in the string. Changing one character costs 1 unit.

Your goal is to partition the string into exactly k palindromic substrings with the minimum total cost of character changes required.

Return the minimum cost.

Input Format:

* + First line: a string s of length n (1 ≤ n ≤ 100).
  + Second line: integer k (1 ≤ k ≤ n)

Output Format:

* + A single integer, the minimum cost to partition the string into exactly k palindromic substrings.

Sample Input 1: abc

2

Sample Output 1:

1

Sample Input 2: aabbc

3

Sample Output 2:

0

Test Cases Input 1

a 1

Output 1

0

Input 2 abc

1

Output 2

1

Input 3 aaaa

4

Output 3

0

Input 4 aaaaaa 2

Output 4

0

Input 5 abcdef 1

Output 5

3

Input 6 racecar 3

Output 6

0

Input 7 abcdef 2

Output 7

2

Input 8 abcdcba 3

Output 8

0

Code

def min\_change\_to\_palindrome(s, i, j): cost = 0

while i < j:

if s[i] != s[j]:

cost += 1

i += 1

j -= 1

return cost

def min\_cost\_partition(s, k): n = len(s)

# Precompute cost to convert s[i:j+1] to palindrome cost = [[0]\*n for \_ in range(n)]

for i in range(n):

for j in range(i, n):

cost[i][j] = min\_change\_to\_palindrome(s, i, j)

dp = [[float('inf')] \* (k+1) for \_ in range(n+1)]

dp[0][0] = 0

for i in range(1, n+1): # i chars taken for t in range(1, k+1): # t parts

for j in range(t-1, i): # try breaking between j and i dp[i][t] = min(dp[i][t], dp[j][t-1] + cost[j][i-1])

return dp[n][k]

# Input Reading

if name == " main ": s = input().strip()

k = int(input()) print(min\_cost\_partition(s, k))

1. Maximum Sum of Non-Adjacent Equal Substrings Problem Statement:

Given a string s of lowercase English letters, you are allowed to select substrings such that:

* + Each selected substring has all the same characters (e.g., "aaa" is valid, "aab" is not).
  + No two selected substrings overlap or are adjacent.
  + For each selected substring, you gain a score equal to the square of its length.

Your task is to select a subset of such substrings (non-overlapping and non-adjacent) to maximize the total score.

Input Format:

* + A single string s (1 ≤ |s| ≤ 10^5) consisting of lowercase English letters.

Output Format:

* + An integer representing the maximum total score.

Sample Input 1: aaabbaaa Sample Output 1:

18

Sample Input 2: ababa

Sample Output 2:

3

Test Cases Input 1 aaaaa Output 1

25

Input 2 aabbaabb Output 2

8

Input 3 aabbaa Output 3

8

Input 4 abcde Output 4

3

Input 5 aaabbbcccaaaddd Output 5

27

Input 6 aabaaabaaa Output 6

22

Input 7 a

Output 7

1

Input 8 aaabbaaccdddeeffggghh Output 8

31

Code

def max\_non\_adjacent\_block\_score(s): n = len(s)

blocks = [] i = 0

# Group into blocks of same characters while i < n:

j = i

while j < n and s[j] == s[i]: j += 1

blocks.append((s[i], j - i)) i = j

m = len(blocks) dp = [0] \* (m + 1)

for i in range(1, m + 1):

char, length = blocks[i - 1]

dp[i] = max(dp[i], dp[i - 1]) # Skip if i >= 2:

dp[i] = max(dp[i], dp[i - 2] + length \* length) # Take else:

dp[i] = max(dp[i], length \* length)

return dp[m]

# Input Reading

if name == " main ": s = input().strip()

print(max\_non\_adjacent\_block\_score(s))

1. Longest Balanced Binary Substring After K Flips Problem Statement:

You are given a binary string s (consisting of only '0' and '1') and an integer k.

You are allowed to flip at most k characters in the string (i.e., change '0' to '1' or '1' to '0').

Your task is to determine the length of the longest balanced substring you can obtain after at most k flips, where a balanced substring is defined as a substring with an equal number of 0s and 1s.

Input Format:

* + First line: a string s of length n (1 ≤ n ≤ 100).
  + Second line: integer k (1 ≤ k ≤ n)

Output Format:

A single integer – length of longest balanced substring after at most k flips

Sample Input 1:

110001

1

Sample Output 1:

6

Sample Input 2:

1111

2

Sample Output 2:

0

Test Cases

Input 1

1111

0

Output 1

0

Input 2

000111

1

Output 2

4

Input 3

00000

3

Output 3

0

Input 4

101010

0

Output 4

0

Input 5

101100110101

2

Output 5

10

Input 6

1

1 1000

Output 6

1000

Input 7

1000000001

1

Output 7

2

Input 8

111000

0

Output 8

0

Code

def longest\_balanced\_after\_k\_flips(s: str, k: int) -> int: n = len(s)

max\_len = 0

count = {0: 0, 1: 0}

left = 0

for right in range(n):

count[int(s[right])] += 1

# Flip the minority if needed to make counts equal while abs(count[0] - count[1]) > 2 \* k:

count[int(s[left])] -= 1

left += 1

total = right - left + 1

max\_balanced = total - abs(count[0] - count[1]) max\_len = max(max\_len, max\_balanced)

return max\_len

# Input

if name == " main ": s = input().strip()

k = int(input()) print(longest\_balanced\_after\_k\_flips(s, k))

1. Minimum Jumps to Make Array Strictly Increasing Problem Statement:

You are given an array of integers arr of size n. You can jump from index i to any index j such that:

* + j > i
  + arr[j] > arr[i]

You need to find the minimum number of jumps required to reach the end of the array starting from index 0 (i.e., arr[0] to arr[n-1]), such that at each step the next value is strictly greater than the current.

If it's not possible to reach the end, print -1.

Input Format:

* + First line: An integer N
  + Next line: N integers Output Format:

A single integer – Minimum number of jumps or -1

Sample Input 1:

6

1 3 2 4 6 8

Sample Output 1:

1

Sample Input 2:

5

5 4 3 2 1

Sample Output 2:

-1

Test Cases Input 1

4

1 2 3 4

Output 1

1

Input 2

1

10

Output 2

0

Input 3

7

1 100 1 101 2 102 103

Output 3

1

Input 4

8

10 20 10 30 10 40 10 50

Output 4

1

Input 5

5

1 1 1 1 2

Output 5

1

Input 6

10

5 6 1 2 3 4 5 6 7 8

Output 6

1

Input 7

5

1 3 5 7 9

Output 7

1

Input 8

6

1 2 1 2 1 2

Output 8

1

Code

from collections import deque, defaultdict

def min\_jumps\_strictly\_increasing(arr): n = len(arr)

if n == 1:

return 0

graph = defaultdict(list)

# Preprocess: Build graph of possible jumps for i in range(n):

for j in range(i + 1, n):

if arr[j] > arr[i]: graph[i].append(j)

# BFS

visited = [False] \* n queue = deque()

queue.append((0, 0)) # (index, jump count) visited[0] = True

while queue:

idx, jumps = queue.popleft() for neighbor in graph[idx]:

if not visited[neighbor]:

if neighbor == n - 1:

return jumps + 1 visited[neighbor] = True

queue.append((neighbor, jumps + 1))

return -1

# Driver Code

if name == " main ":

n = int(input())

arr = list(map(int, input().split())) print(min\_jumps\_strictly\_increasing(arr))