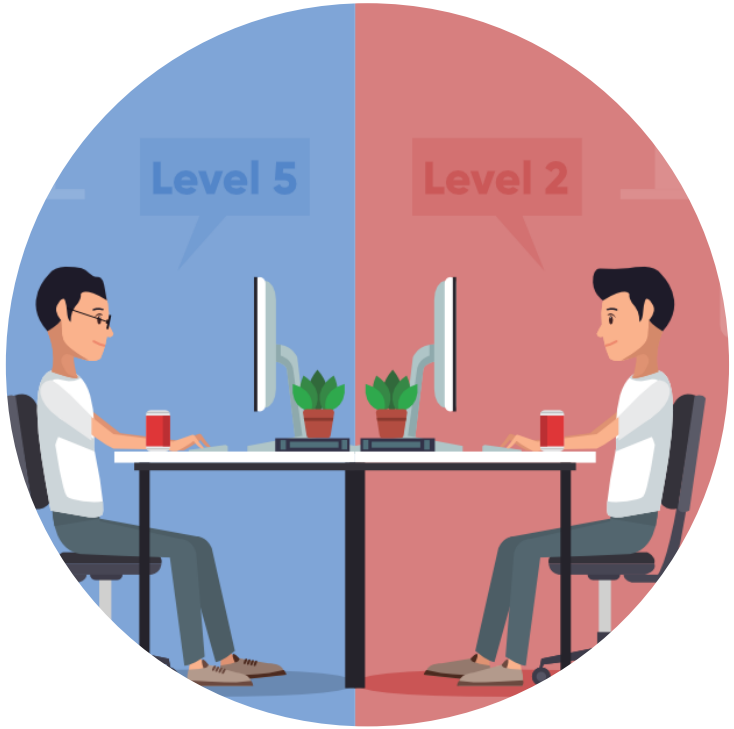
Competitive Programming Challenge



JAVA & DB

Question 1 (10) (50%)

Given an array of long integers (arr) and a number (num). Iterate through the elements in arr and double the value of num whenever an element equals num. arr can be reordered before the iteration to maximize the value of num. Find the maximum possible value of num.

Example

arr = [1, 2, 4, 11, 12, 8]

num = 2

Iterating through arr:

arr	num = 2
1	2
2	4
4	8
11	8
12	8
8	16

Question 2 (8) (40%)

A music player allows users to choose songs to play, but only in pairs and only pairs of songs with durations that add up to a multiple of 60 seconds (e.g., 60, 120, 180). Given a list of song durations, calculate the total number of different song pairs that can be chosen.

Example

n = 3

songs = [40, 20, 60]

One pair of songs can be chosen whose combined duration is a multiple of a whole minute (40 + 20 = 60) and the return value would be 1. While the third song is a single minute long, songs must be chosen in pairs.

Function Description

Develop the function playlist.

playlist has the following parameter(s):

- int songs[n]: array of integers representing song durations in seconds

Returns:

- int: the number of songs pairs that add up to a multiple of a minute

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq songs[i] \leq 1000$, where $0 \leq i < n$

Question 3 (6) (30%)

This challenge involves points in two and three dimensional space. The classes and methods to implement will store values for coordinates as well as calculate distances between points. The 2D and 3D distances between two points are calculated using the following formulae:

$$\text{2D distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$\text{3D distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Implement the classes and methods defined below:

A superclass named Point2D:

Instant Variable	**Functionality**
X	Stores the value of the x-coordinate.
Y	Stores the value of the y-coordinate.
Constructor	**Functionality**
Point2D(…)	A parameterized constructor that initializes the instance variables.
Methods	**Functionality**
double dist2D(Point2D p)	Calculates and returns the 2D distance between two points (the current Point2D object and Point2D parameter p).
void printDistance(double d)	Prints the 2D distance between two points as 2D distance = k, where k is distance d as a ceiling-rounded integer, on a new line.

Question 3 (Continue…)

A derived class named Point3D that extends Point2D:

Instant Variable	**Functionality**
Z	Stores the value of the z-coordinate.
Constructor	**Functionality**
Point3D(…)	A parameterized constructor that initializes the instance variables.
Methods	**Functionality**
double dist3D(Point3D p)	Calculates and returns the 3D distance between two points (the current Point3D object and Point3D parameter p).
void printDistance(double d)	Prints the 3D distance between two points as 3D distance = k, where k is distance d as a ceiling-rounded integer, on a new line.

A main method is provided in the locked portion of the editor. It parses six values representing point coordinates and calls the implemented constructors and methods. Here, x[1], y[1], and z[1] represent the coordinates of the first point, and x[2], y[2], and z[2] represent the coordinates of the second point. Note that printed output must exactly match the above for the test cases to pass.

Constraints

$-128 \leq x,y,z \leq 127$

Question 4 (6) (30%)

There are two tables in a database of real estate owners. One has ownership information and the other has price information, in millions. An owner may own multiple houses, but a house will have only one owner.

Write a query to print the IDs of the owners who have at least 100 million worth of houses and own more than 1 house. The order of output does not matter. The result should be in the format: BUYER_ID TOTAL_WORTH

There are 2 tables: house, price.

Name	**Type**	**Description**
Table Name: house		
BUYER_ID	INTEGER	Unique buyer ID
HOUSE_ID	STRING	Unique house ID
Table Name: price		
HOUSE_ID	STRING	Unique house ID. The primary key.
PRICE	INTEGER	The price of the house.

Question 4 (Continue…)

Table Name: house	
BUYER_ID	HOUSE_ID
1	1001
2	1002
1	1003
3	1004
Table Name: price	
HOUSE_ID	PRICE
1001	60
1002	120
1003	40
1004	90

Sample Output

1 100

Explanation

1 has a total of (60 + 40) = 100 million worth houses and is included in the results.

2 has 120 million worth houses but has only 1 house.

3 has 90 million worth of houses.

Question 5 (5) (25%)

There will be a list of items in the form of a 2-dimensional string array where each element contains [name, relevance, price]. Given the sort column, the sort order (0: ascending, 1: descending), the number of items to be displayed on each page (except for the last page which may have fewer), and a page number, determine the list of item names in the specified page while respecting the item's order. Page numbering starts at 0.

Example

items = [['item1', '10', '15'], ['item2', '3', '4'], ['item3', '17', '8']]

sortParameter = 1

sortOrder = 0

itemsPerPage = 2

pageNumber = 1

n = 3 items

Sort them by (relevance: 1) in ascending order (items = [['item2', '3', '4'], ['item1', '10', '15'], ['item3', '17', '8']])

Display up to 2 items in each page

The page 0 contains 2 item names ['item2', 'item1'] and page 1 contains only 1 item name, so result = 'item3'

Question 5 (Continue…)

Function Description

Complete the function fetchItemsToDisplay in the editor below.

fetchItemsToDisplay has the following parameter(s):

- string items[n][3]: a 2D array of arrays of strings in the form [name, relevance, price] int sortParameter: the column of the items to sort on
- int sortOrder:0 = ascending and 1 = descending int itemsPerPage: the number of items per page
- int pageNumber: the page number to display item names

Returns:

string pageItems[m]: array of item names on the requested page in the order they are displayed

Constraints

- $1 \leq n < 10^5$
- $1 \leq m \leq n$
- $0 \leq relevance, price < 10^8$
- relevance and price are both integers $1 \leq itemsPerPage < 20$
- $0 \leq pageNumber < 10$

Question 6 (5) (25%)

Question Description

FC Codelona is trying to assemble a team from a roster of available players. They have a minimum number of players they want to sign, and each player needs to have a skill rating within a certain range. Given a list of players' skill levels with desired upper and lower bounds, determine how many teams can be created from the list.

Example

skills = [12, 4, 6, 13, 5, 10]

minPlayers = 3

minLevel = 4

maxLevel = 10

The list includes players with skill levels [12, 4, 6, 13, 5, 10].

They want to hire at least 3 players with skill levels between 4 and 10, inclusive. Four of the players with the following skill levels { 4, 6, 5,10} meet the criteria.

There are 5 ways to form a team of 3 players : {4, 5, 6}, {4, 6, 10}, {4, 5,10}, {5, 6, 10}, and {4, 5, 6, 10}.

Return 5.

Question 6 (Continue…)

Function Description:

Complete the function countTeams in the editor below.

countTeams has the following parameter(s):

- int skills[n]: an array of integers that represent the skill level per player
- int minPlayers: the minimum number of team members required
- int minLevel: the lower limit for skill level, inclusive
- int maxLevel: the upper limit for skill level, inclusive

Return:

- int: the total number of teams that can be formed per the criteria

Constraints

- $1 \leq n \leq 20$
- $1 \leq minPlayers \leq n$
- $1 \leq minLevel \leq maxLevel \leq 1000$
- $1 \leq skills[i] \leq 1000$

Question 7 (9) (45%)

Question Description:

Pat is an ordinary kid who works hard to be a great runner. As part of training, Pat must run sprints of different intervals on a straight trail. The trail has numbered markers that the coach uses as goals. Pat's coach provides a list of goals to reach in order. Each time Pat starts at, stops at, or passes a marker it is considered a visit. Determine the lowest numbered marker that is visited the most times during Pat's day of training.

Example

n = 5

sprints = [2, 4, 1, 3]

if the number of markers on the trail, n = 5, and assigned sprints = [2, 4, 1, 3], Pat first sprints from position 2 → 4. The next sprint is from position 4 → 1, and then 1 → 3. A marker numbered position p is considered to be visited each time Pat either starts or ends a sprint there and each time it is passed while sprinting. The total number of visits to each position in the example is calculated like so:

Total Visits Per Position					
Sprint	1	2	3	4	5
2 → 4		☺→	→	→☺	
4 → 1	☺←	←	←	←☺	
1 → 3	☺→	→	→☺		
Total Visits	2	3	3	2	0

Pat has visited markers 2 and 3 a total of 3 times each. Since 2 < 3, the lowest numbered marker that is visited the most times during Pat's day of training is 2.

Question 7 (Continue…)

Function Description:

Complete the function getMostVisited in the editor below.

getMostVisited has the following parameter(s):

- int n: an integer denoting the number of markers along the trail
- int sprints[m]: an array of integers denoting the sequence of markers to reach, beginning at the marker shown in sprints[0].

Returns:

int: an integer denoting Pat's most visited position on the trail after performing all m − 1 sprints. If there are multiple such answers, return the smallest one.

Constraints:

- $1 \leq n \leq 10^5$
- $2 \leq m \leq 10^5$
- $1 \leq sprints[i] \leq m$ (where $0 \leq i < m$) $sprints[i-1] \neq sprints[i]$ (where $0 < i < m$)

Question 8 (4) (20%)

Question Description:

There is a shop with old-style cash registers. Rather than scanning items and pulling the price from a database, the price of each item is typed in manually. This method sometimes leads to errors. Given a list of items and their correct prices, compare the prices to those entered when each item was sold. Determine the number of errors in selling prices.

Example:

products = ['eggs', 'milk', 'cheese’]

productPrices = [2.89, 3.29, 5.79]

productSold = ['eggs', 'eggs', 'cheese', 'milk']

soldPrice = [2.89, 2.99, 5.97, 3.29].

	Price		
Product	Actual	Expected	Error
eggs	2.89	2.89	
eggs	2.99	2.89	1
cheese	5.97	5.79	1
milk	3.29	3.29	

The second sale of eggs has a wrong price, as does the sale of cheese. There are 2 errors in pricing.

Question 8 (Continue…)

Function Description:

Complete the function priceCheck in the editor below.

priceCheck has the following parameter(s):

- string products[n]: each products[i] is the name of an item for sale string productPrices[n]: each productPrices[i] is the price of products[i]
- string productSold[m]: each productSold[j] is the name of a product sold
- float soldPrice[m]: each soldPrice[j] contains the sale price recorded for productSold[j].

Returns:

- int: the number of sale prices that were entered incorrectly

Constraints:

- $1 \leq n \leq 10^5$
- $1 \leq m \leq n$
- $1.00 \leq productPrices[i], soldPrice[j] \leq 100000.00$, where $0 \leq i < n$, and $0 \leq j < m$

Question 9 (7) (35%)

Question Description:

Ryan is movie obsessed and has collected a list of movie quality ratings. He wants to watch the largest contiguous list of movies with the highest cumulative ratings possible. To do this, he must calculate the sum of all contiguous subarrays in order to determine the maximum possible subarray sum.

For example, ratings are arr = [-1,3,4,-2,5,-7]. We can see that the highest value contiguous subarray runs from arr[1]-arr[4] and is 3 + 4 + -2 + 5 = 10.

Function Description:

Complete the function maximumSum in the editor below. It must return a long integer denoting the maximum sum for any contiguous subarray in arr.

maximumSum has the following parameter(s):

- arr[arr[0],...arr[n-1]]: an array of integers

Constraints:

- $1 \leq n \leq 10^6$
- $-10^7 \leq arr[i] \leq 10^7$

Question 10 (11) (55%)

Question Description:

Given an interface named "OnlineAccount" that models the account of popular online video streaming platforms, perform the operations listed below. The interface "OnlineAccount" consists of the basePrice, regularMoviePrice, and exclusiveMoviePrice.

In order to complete this challenge, you need to implement an incomplete class named "Account" which implements the "OnlineAccount" interface as well as the "Comparable<Account>" interface.

Class Account has two attributes to keep track of the number of movies watched:

1. Integer noOfRegularMovies
2. Integer noOfExclusiveMovies
3. String ownerName

Methods to complete for class Account:

1. Add a parameterized constructor that initializes the attributes ownerName, numberOfRegularMovies and numberOfExclusiveMovies.
2. double monthlyCost() => This method returns the monthly cost for the account. [Monthly Cost = base price + noOfRegularMovies*regularMoviePrice + noOfExclusiveMovies*exclusiveMoviePrice]
3. Override the compareTo method of the Comparable interface such that two accounts can be compared based on their monthly cost.
4. String toString() which returns => "Owner is [ownerName] and monthly cost is [monthlyCost] USD."

Question 11 (14) (70%)

Question Description:

A university maintains data on professors, departments, courses, and schedules in four tables:

DEPARTMENT, PROFESSOR, COURSE, and SCHEDULE.

Write a query to print the names of professors with the names of the courses they teach (or have taught) outside of their department. Each row in the results must be distinct (i.e., a professor teaching the same course over multiple semesters should only appear once), but the results can be in any order. Output should contain two columns: PROFESSOR.NAME, COURSE.NAME.

Name	Type	Description
PROFESSOR		
ID	Integer	A professor’s ID in the inclusive range [1, 1000]. This is a primary key
NAME	String	A professor's name. This field contains between 1 and 100 characters (inclusive).
DEPARTMENT_ID	Integer	A professor's department ID. This is a foreign key to DEPARTMEN T.ID
SALARY	Integer	A professor's salary in the inclusive range [5000, 40000].
DEPARTMENT		
ID	Integer	A department ID in the inclusive range [1, 1000]. This is a primary key.
NAME	String	A department name. This field contains between 1 and 100 characters (inclusive)

Question 11 (Continue…)

Question Description:

Name	Type	Description
COURSE		
ID	Integer	A course ID in the inclusive range [1, 1000]. This is a primary key.
NAME	String	A course name. This field contains between 1 and 100 characters (inclusive).
DEPARTMENT_ID	Integer	A course's department ID. This is a foreign key to DEPARTMENT_ID.
CREDITS	Integer	The number of credits allocated to the course in the inclusive range [1, 10].
SCHEDULE		
PROFESSOR_ID	Integer	The ID of the professor teaching the course. This is a foreign key to PROFESSOR_ID.
COURSE_ID	Integer	The course's ID number. This is a foreign key to COURSE_ID.
SEMESTER	Integer	A semester ID in the inclusive range [1, 6].
YEAR	Integer	A calendar year in the inclusive range [2000, 2017].

Output Format:

Each distinct row of results must contain the name of a professor followed by the name of a course they taught outside of their department in the format:

PROFESSOR.NAME COURSE.NAME

Question 12 (6) (30%)

Question Description:

Given an integer k and a list of integers, count the number of distinct valid pairs of integers (a, b) in the list for which a + k = b. Two pairs of integers (a, b) and (c, d) are considered distinct if at least one element of (a, b) does not also belong to (c, d).

Example:

n = 4

numbers = [1, 1, 1, 2]

k = 1

This array has two different valid pairs: (1, 1) and (1, 2). For k = 1, there is only 1 valid pair which satisfies

a + k = b: the pair (a, b) = (1, 2).

Function Description:

Complete the function countPairs in the editor below.

countPairs has the following parameter(s):

- int numbers[n]: array of integers
- int k: target difference

Returns:

- int: number of valid (a, b) pairs in the numbers array that have a difference of k

Constraints:

- $2 \leq n \leq 2 \times 10^5$
- $0 \leq numbers[i] \leq 10^9$, where $0 \leq i < n$
- $0 \leq k \leq 10^9$

Question 13 (6) (30%)

Question Description:

A climber is trying to reach a flag that is some height above the ground. In the attempt to reach the flag, the climber can make any number of jumps up the rock wall where the flag is mounted. Movements can only be made up the wall, and the climber must end at exactly the height of the flag.

There are 2 types of jumps:

1. A jump of height 1.
2. A jump of height bigJump.

Determine the minimum number of jumps it will take the climber to reach the flag's exact height.

Example:

flagHeight = 8

bigJump = 3

The climber starts at height 0, takes two jumps of height bigJump and two of height 1 to reach exactly 8 units in 4 jumps.

Function Description:

Complete the function jumps in the editor below.

jumps has the following parameter(s):

- int flagHeight: an integer, the flag height
- int bigJump: an integer, the height of the second type of jump

Returns:

int: an integer, the minimum number of jumps necessary

Constraints:

$1 \leq bigJump, flagHeight \leq 10^9$

Question 14 (3) (15%)

Question Description:

Implement the following classes:

1. abstract class Employee with the following methods:

- abstract void setSalary(int salary) method
- abstract int getSalary() method
- abstract void setGrade(String grade) method (grade of the employee in the organization)
- abstract String getGrade() method
- void label() method which prints "Employee's data:\n" (Concrete method, implementation is hidden from end user)

2. class Engineer extending class Employee:

- private attribute int salary
- private attribute String grade
- implement the setter and getter methods from the parent class to set and get attributes (salary and grade)

3. class Manager extending class Employee:

- private attribute int salary
- private attribute String grade
- implement the setter and getter methods from the parent class to set and get attributes (salary and grade)

Note: The code stub handles input and calls the methods

Input Format For Custom Testing:

The first line contains an integer, n, that denotes the number of employees to be instantiated.

Each line i of the n subsequent lines (where $0 \leq i < n$) contains 3 variables. TYPE_OF_EMPLOYEE GRADE SALARY