APPLY ALGORITHM DESIGN

Group 14

LESSON OUTLINE

01 Computational thinking

02 Applying algorithmic design

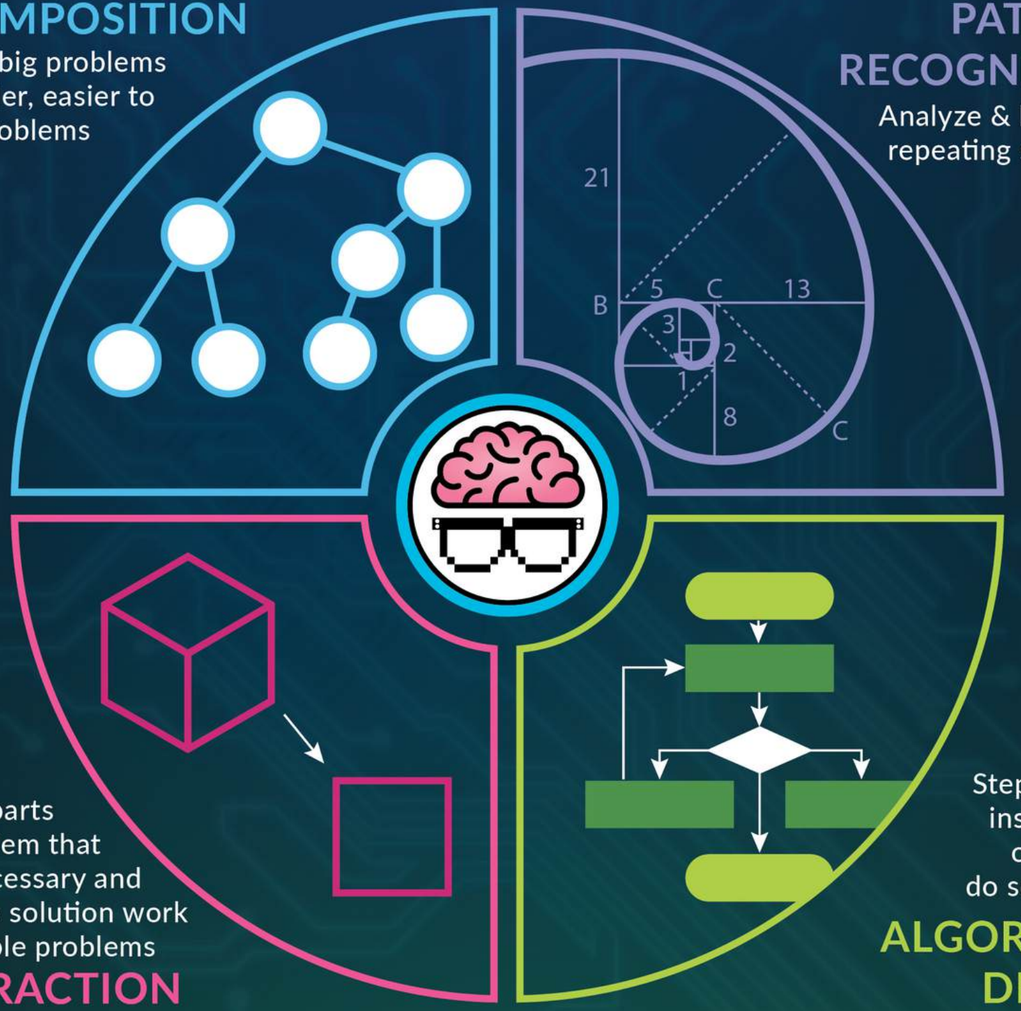
03 Quiz

WHAT IS COMPUTATIONAL THINKING?

COMPUTATIONAL THINKING

DECOMPOSITION
Breaking big problems into smaller, easier to mange problems

PATTERN RECOGNITION
Analyze & look for a repeating sequence



Remove parts of a problem that are unnecessary and make one solution work for multiple problems
ABSTRACTION

Step-by-Step instructions on how to do something
ALGORITHM DESIGN

6 COMPUTATIONAL THINKING SKILLS

01 Abstraction

02 Decomposition

03 Pattern Recognition

04 Algorithm Design

05 Evaluation

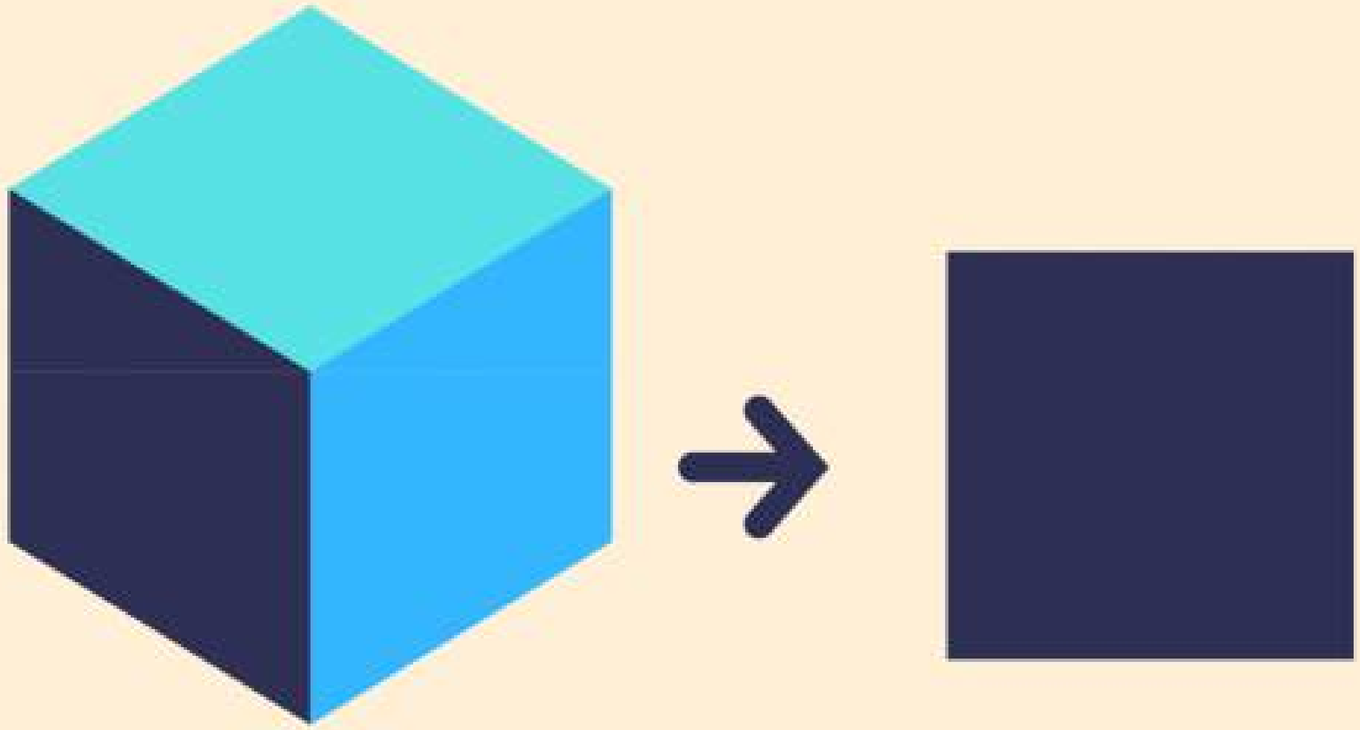
06 Testing

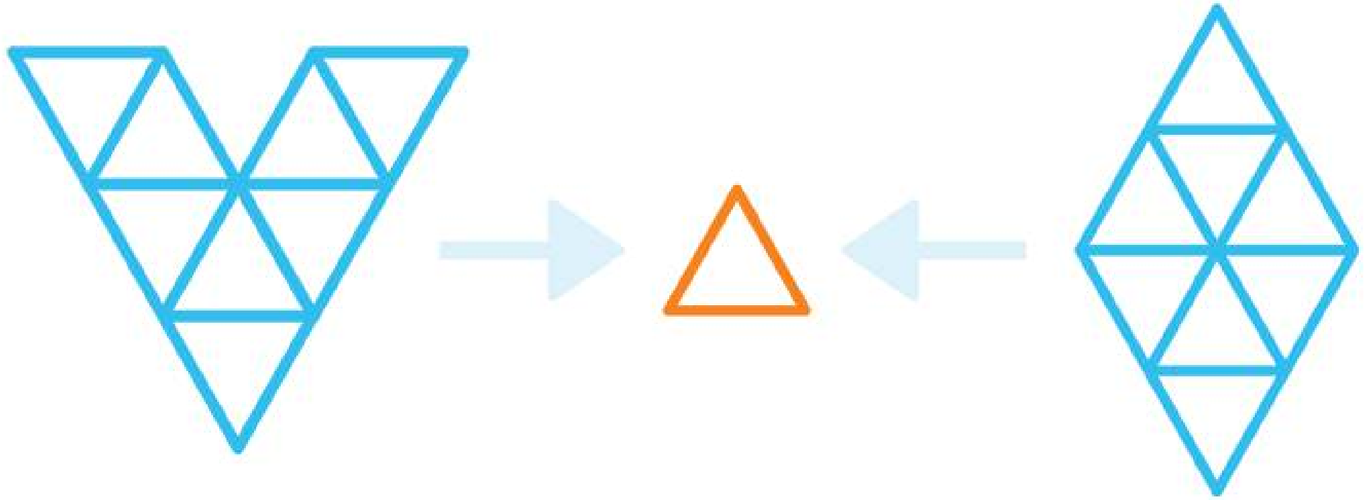
01 ABSTRACTION

Filter out unnecessary details, retain only important information.

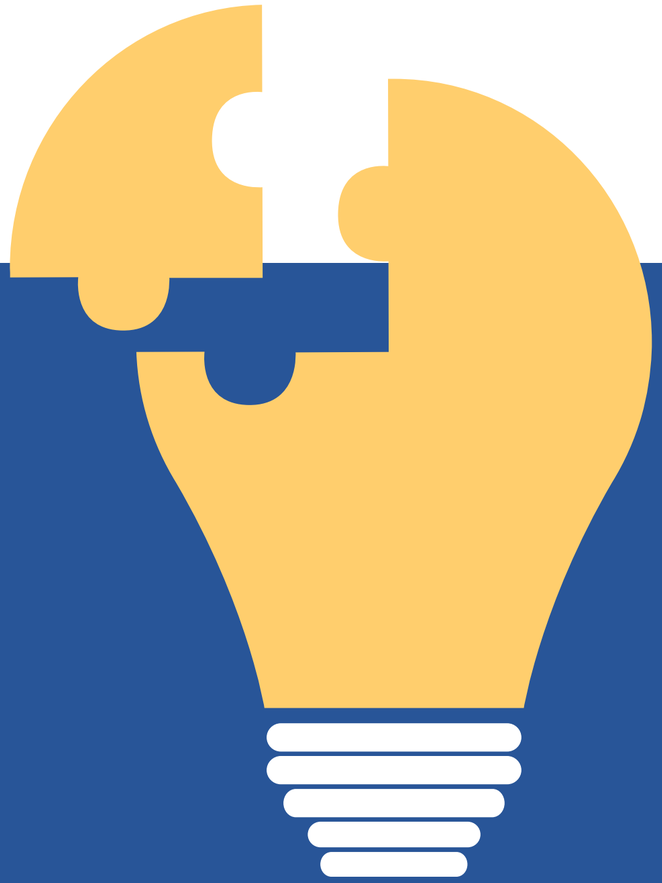
Helps reduce complexity and makes the problem easier to understand.

ABSTRACTION





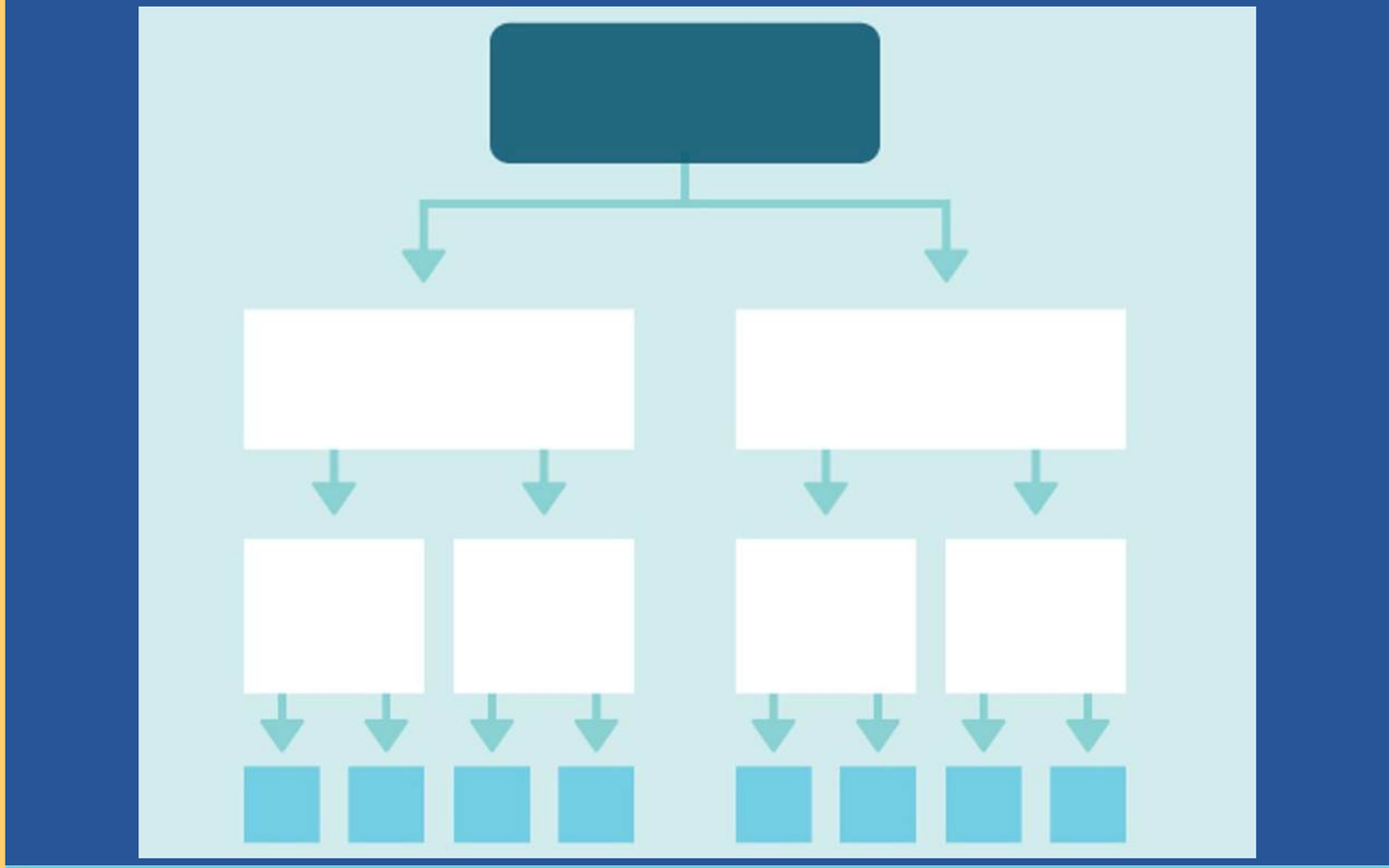
02 DECOMPOSITION

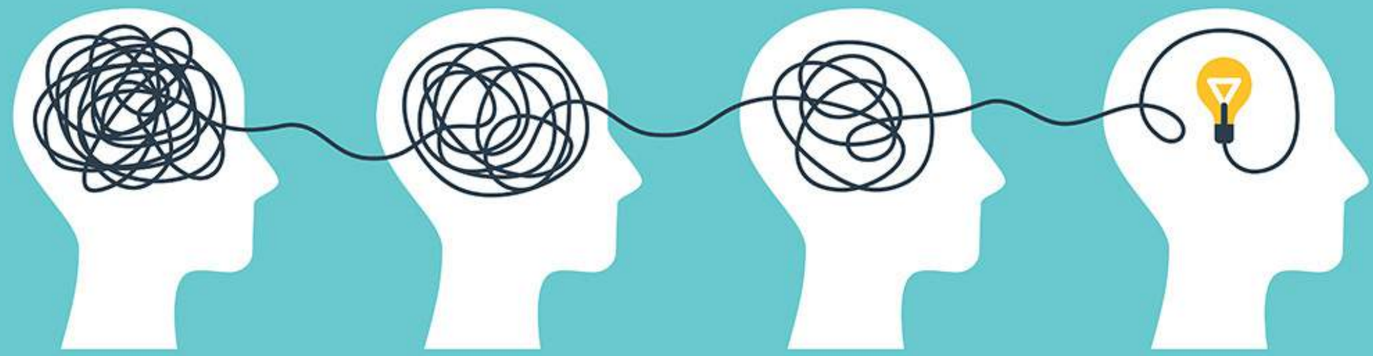


Break down a complex problem into smaller, more independent, and manageable parts.

Helps reduce overall complexity, making small sections easy to understand and solve.

DECOMPOSITION



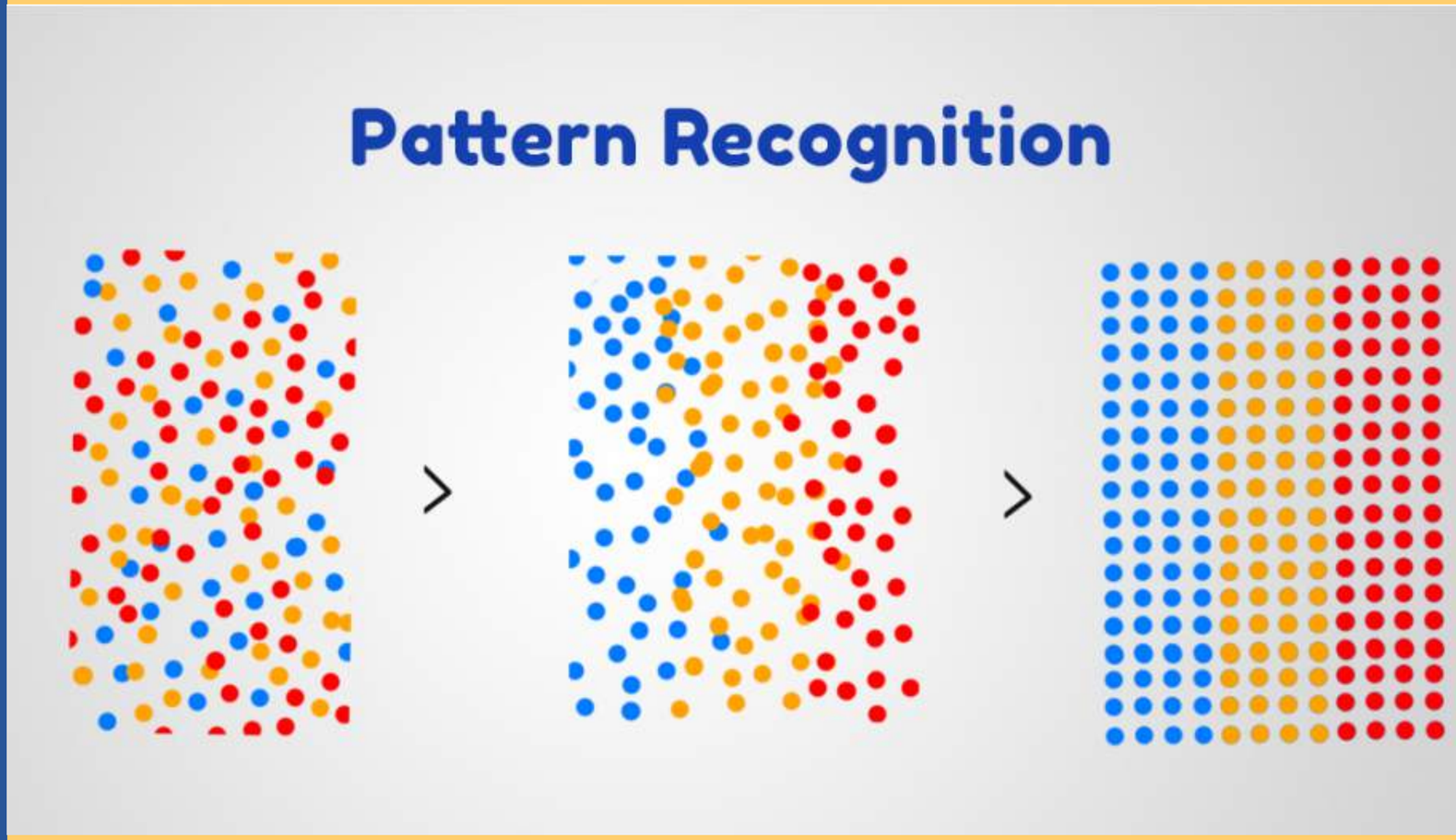


03 PATTERN RECOGNITION

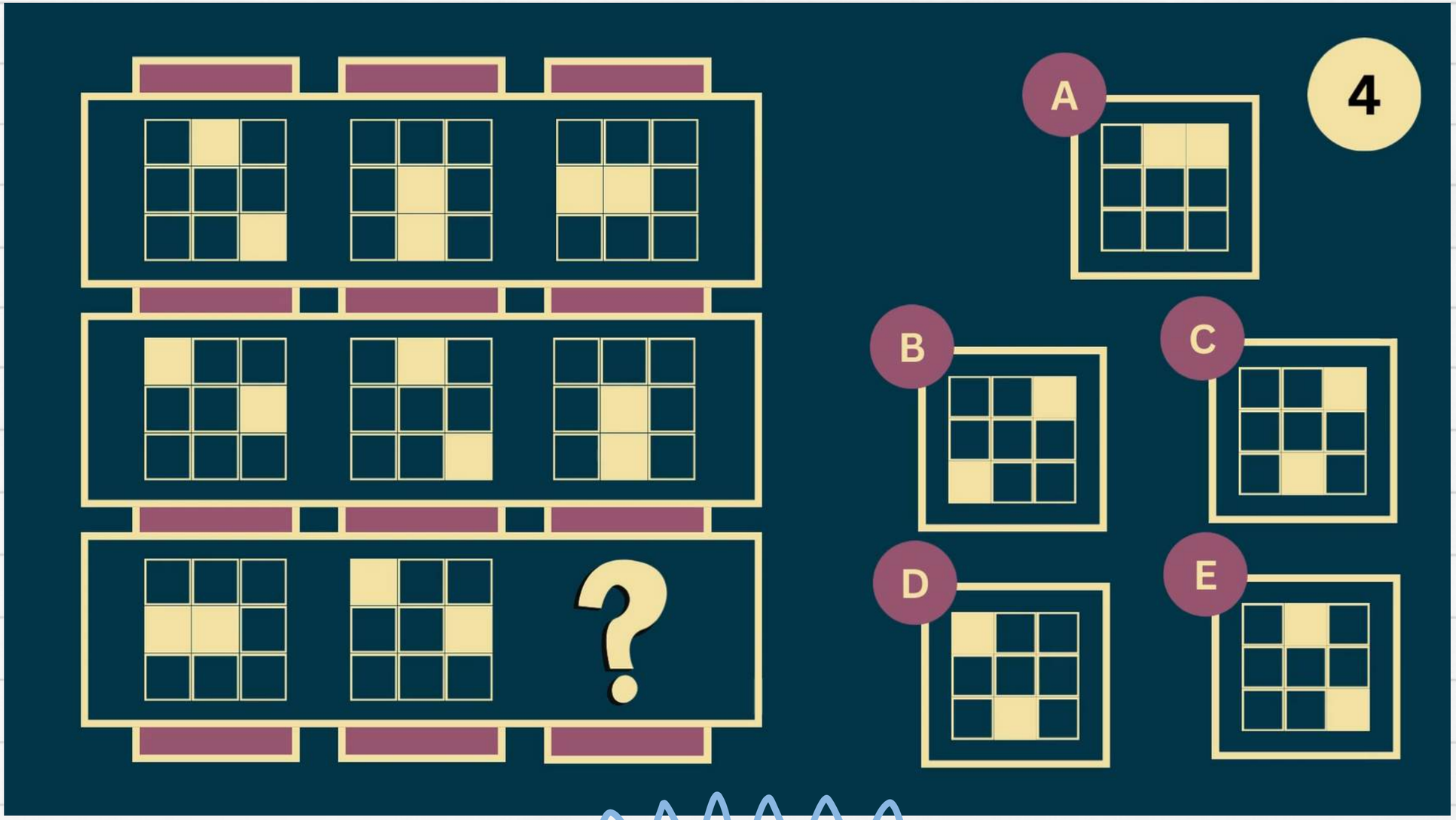
Ability to recognize patterns, trends, or patterns that repeat in data or similar problems

Help reuse existing solutions or develop more general solutions

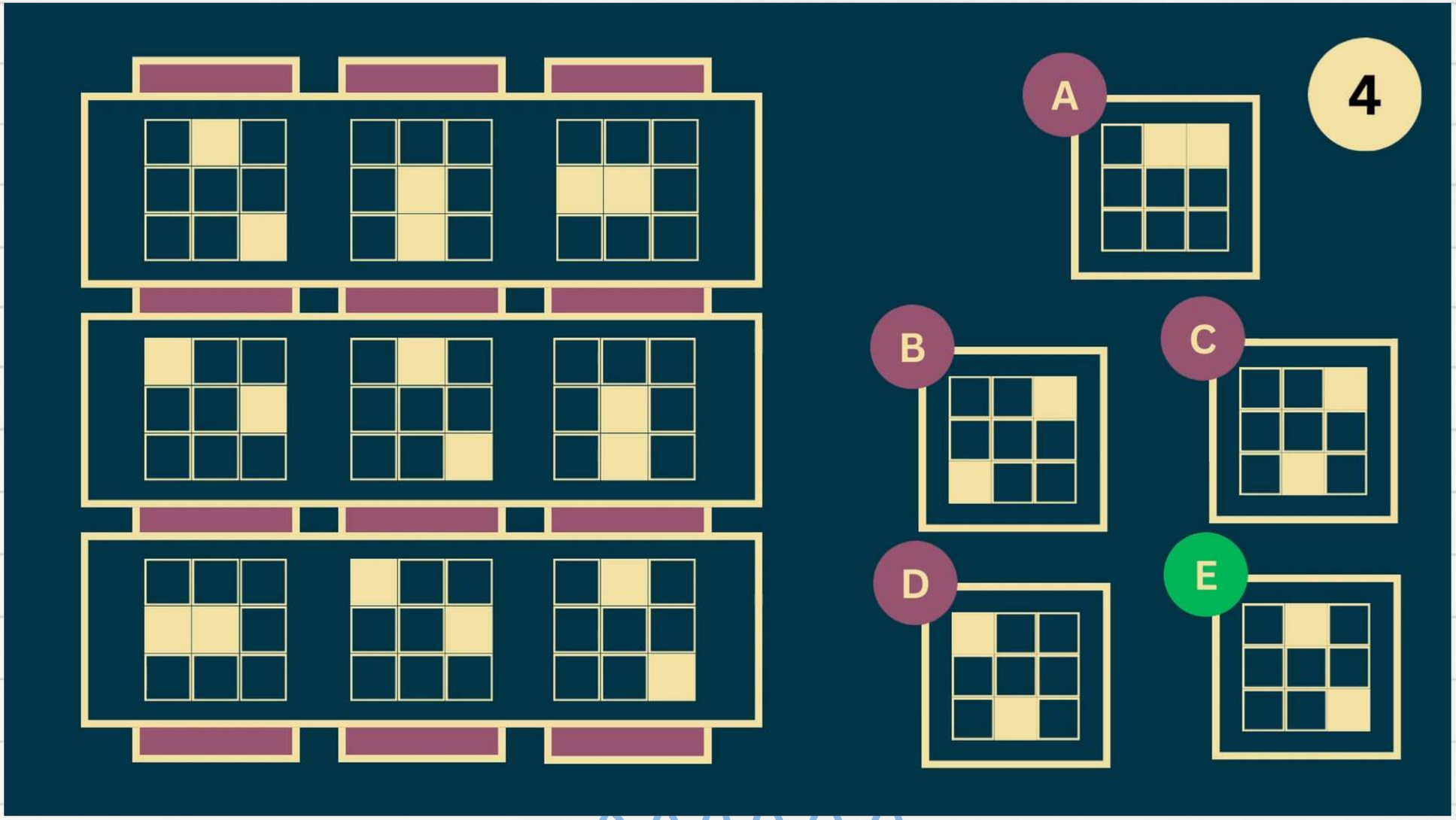
PATTERN RECOGNITION



PATTERN RECOGNITION



PATTERN RECOGNITION

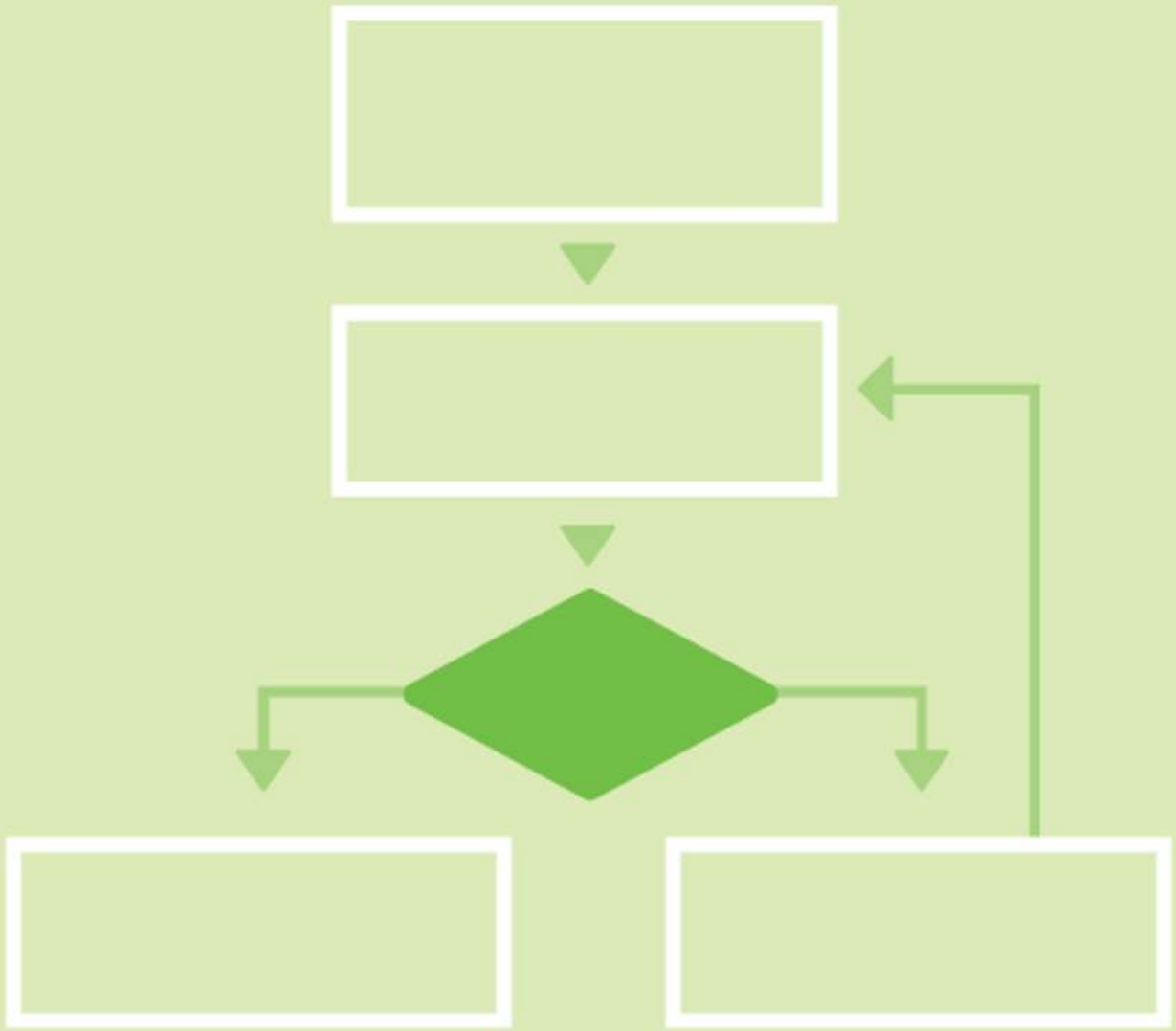


04 ALGORITHM DESIGN

A set of specific instructional steps to solve a problem, including defining logical, effective steps to solve specific problems

An algorithm needs to make sure that it is both accurate, time-efficient, and memory-efficient

ALGORITHM DESIGN



05 EVALUATION

The skill of examining and comparing solutions to select the most effective method based on criteria such as accuracy, performance, ...

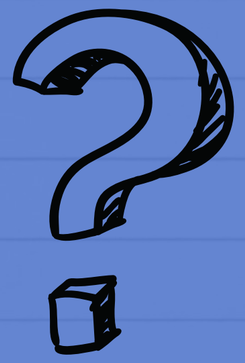
Enhance the quality of the solution and ensure feasibility and efficiency before implementation.

06 TESTING

The skill of assessing the accuracy and efficiency of a solution by applying specific test cases and analyzing the results

Ensure that the system or solution works as expected and detect errors or weaknesses to fix them promptly.

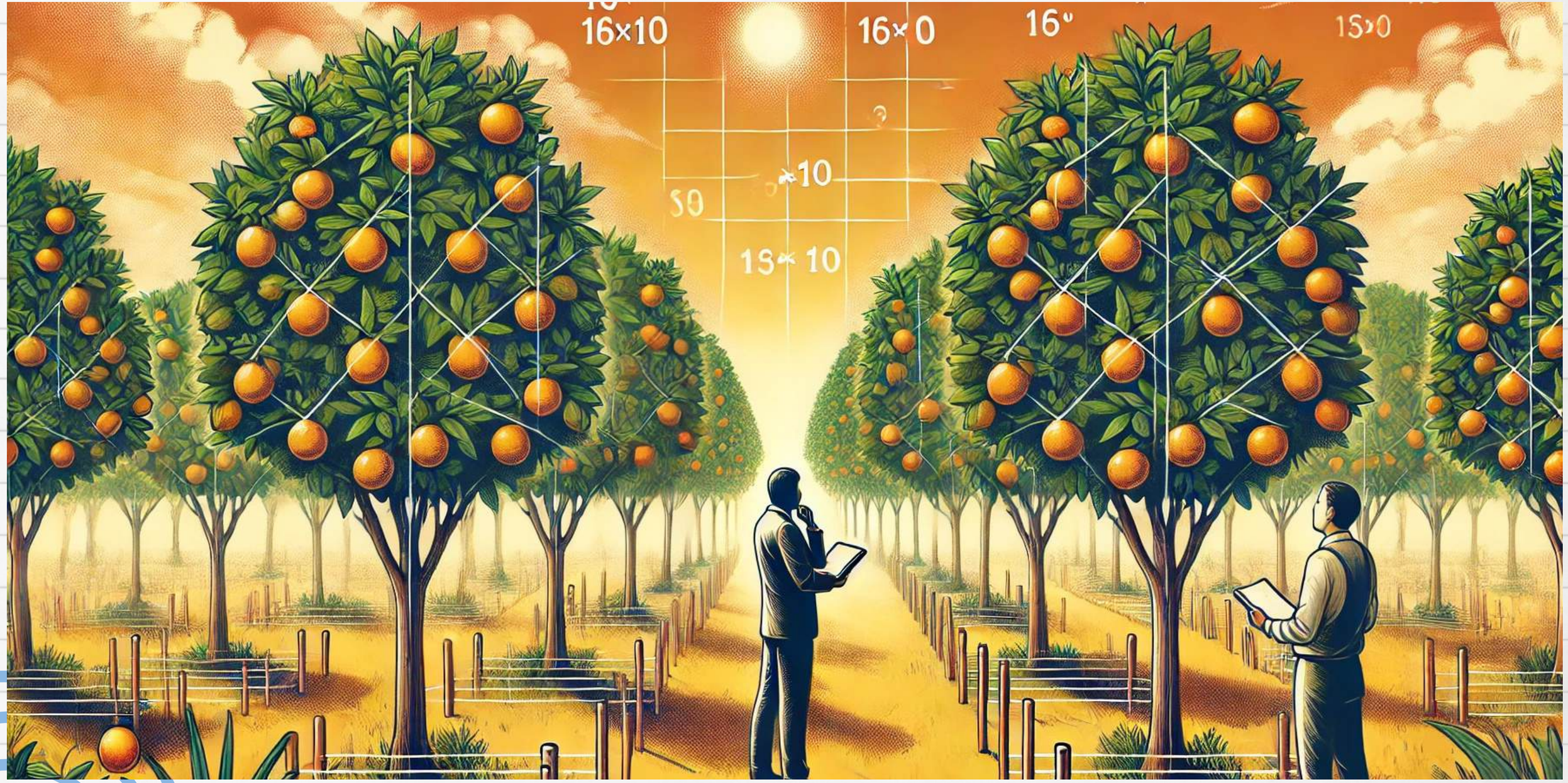
APPLYING ALGORITHMIC DESIGN



Name some algorithm design techniques you have learned.

- Brute force
- Backtracking
- Brand and Bound
- Greedy
- Divide, Decrease, Transform and Conquer
- Dynamic programing
- Distributed algorithms
- Approximation algorithms
- ....

PROBLEM 1: ORANGE



PROBLEM 1: ORANGE

Lee's garden has N orange trees arranged in a row, each tree having A_i oranges Lee wants to pick oranges so that the total number of oranges received is at least K and the average number of oranges per tree is maximized. Please help Lee find a contiguous sequence of trees that meets the above requirements

Input:

- The first line contains two positive integers N, K ($1 \le N \le 100000$, $0 \le K \le 10^9$)
- The second line contains N integers ($1 \le A_i \le 10^9$)

Output: A unique integer is the result of the problem (the result rounded down to an integer)

Example:

Input	Output
5 6 1 5 2 4 3	3

SOLUTION

01 Abstraction

- Input: Array A consisting of n integers and a number K.
- Output: The length of the found subarray.
- Find the subarray with the largest average value that still satisfies the sum being no less than K

Example: What is the output?

Input	Output
5 10 2 3 5 8 1	6

Explanation?

SOLUTION

02 Decomposition & Pattern Recognition

The problem of searching and optimizing in an array

- Finding a subarray with a constrained sum
- Find the minimum value **(Binary Search combined with calculation)**
- Maintain a value through local optimization without needing to check the global solution **(Greedy)**
- Quickly calculate the sum of subarrays **(sum prefix array)**

Can use divide and conquer or dynamic programming for this problem?

SOLUTION

03 Algorithm Design

- Calculate **the prefix sum S[i]**: $S[i] = A[1] + A[2] + \ldots + A[i]$
- For the segment [l, r], the average value: $Avg(l, r) = \frac{Sum(l, r)}{r - l + 1}$

Brute force:

Iterate through each contiguous subarray, then calculate and check $Avg(l, r) \geq p$ and $Sum(l, r) \geq K$

What is the complexity for Brute force?

The total number of contiguous subarrays: $O(n^2)$

Calculating the sum: O(1)

→ Complexity: $O(n^2)$

03 Algorithm Design

✨ Optimal algorithm:

1. Transforming the problem:

- Condition Avg(l, r) ≥ p is equivalent to: S[r] - S[l - 1] ≥ p.(r - l + 1)
- Simplifying: S[r] - p.r ≥ S[l - 1] - p.(l - 1)
- The smaller the above expression, the easier it is for the substring to satisfy the condition.

2. Transform $S[r]-p\cdot r \geq Min$ where $Min = \min(S[l-1]-p\cdot(l-1))$ is maintained using **Greedy**

3. Binary search on p: Find the largest p for which a satisfying substring exists.

CODE

```python
def check_avg(n, k, arr, p):
    prefix_sum = [0] * (n + 1)
    min_prefix = float('inf')
    j = 0
    for i in range(1, n + 1):
        prefix_sum[i] = prefix_sum[i - 1] + arr[i - 1]
        while j < i and prefix_sum[i] - prefix_sum[j] >= k:
            min_prefix = min(min_prefix, prefix_sum[j] - j*p)
            j += 1
        if prefix_sum[i] - p*i >= min_prefix:
            return True
    return False

def max_average_subarray(n, k, arr):
    left, right = min(arr), max(arr)

    while left <= right:
        mid = (left + right) // 2
        if check_avg(n, k, arr, mid):
            res = mid
            left = mid + 1
        else:
            right = mid - 1
    return res
```

Complexity?

04 Evaluation

- Binary search: $O(logn)$
- Check for each p: O(n)
- Update min: O(1)
- Complexity: $O(n \log n)$
- Space complexity: O(n)

05 Testing

- Boundary cases: n is small, all elements are the same, ...
- Equivalence class partitioning: K = 0, ...
- Positive test cases: valid value and appropriate K, ...
- Performance test cases: N and K are large, ...
- ...

PROBLEM 1: ORANGE

Some practical examples:

- Optimize product selection when shopping
- Find an effective rest period
- ....

PROBLEM 2: HIGH PLACE

PROBLEM 2: HIGH PLACE

A city wants to build X massage parlors, Y karaoke, and Z bars. This city has a total of (X + Y + Z) plots of land, and each plot will have different profits for each type. Calculate the maximum profit this city can achieve.

Input:

- The first line contains three positive integers X, Y, Z (X + Y + Z ≤ 10^5)
  - the number of massage parlors, karaoke bars, and bars to be built.
- The next X + Y + Z lines, each consisting of three integers A_i, B_i, C_i $(0 \leq A_i, B_i, C_i \leq 10^9)$, respectively represent the profit obtained from building a massage, karaoke, and bar on the i-th plot of land.

Output: A single integer is the maximum profit.

PROBLEM 2: HIGH PLACE

Example:

Input	Output
3 2 1 0 3 2 1 4 9 5 3 2 7 5 9 4 8 9 3 0 4	35

Explanation: Build a massage on plots 3, 4, 6; build a karaoke on plots 1, 5; build a bar on plot 2.

Total profit is: (5 + 7 + 3) + (3 + 8) + 9 = 35

SOLUTION

01 Abstraction

- **Input:** 3 integers X, Y, Z. Arrays A, B, and C with lengths corresponding to X + Y + Z.
- **Output:** The largest sum.
- Take X elements from array A, Y elements from array B, and Z elements from array C so that the sum is the largest.

SOLUTION

Example: What is the output?

Input	Output
2 2 2 1 2 3 2 1 3 3 1 2 4 5 6 5 4 3 6 7 8	26

Explanation: Build a massage on plots 2, 3; karaoke on plots 4, 6; bar on plots 1, 5.

Total profit is: (2 + 3) + (5 + 7) + (3 + 3) = 26

02 Decomposition & Pattern Recognition

- Find the optimal solution: select the optimal subsequence with three groups, each group having a fixed size (X, Y, Z)
- Choosing the best piece of land for each establishment can be solved independently from the other establishment.
- **Local optimal selection:** At each step, we can choose the plot with the highest profit for each type of establishment. Leading to **global optimal solution**

Which algorithm can we use?

Greedy algorithm

- Moreover, no decision changes are needed once a choice has been made
- Optimize profits: use **sorting, priority queues**

SOLUTION

03 Algorithm Design

1. Transforming the problem:

- Initial total profit:
- Set profit (Ai, Bi, Ci) to (0, Bi - Ai, Ci - Ai)

2. Sort and group:

- Sort by (Bi - Ci) in descending order.
- Assume X = 0, select the first Y values B and Z values C.
- Return to the problem when X <> 0: Divide k plots of land:
  - First choosing A and B
  - Then the remaining X + Y + Z - k, choosing A and C

SOLUTION

03 Algorithm Design

3. Optimize profits:

- With k initial values, select Y of the largest B values and (k - Y) A values.
- The remaining values are to select Z of the largest C values and (X + Y - k) A values.
- Use a priority queue to calculate the total profit from selecting B (**total profit B**) and selecting C (**total profit C**).

Result: Maximum total profit = sumA + **total profit B** + **total profit C**

PSEUDOCODE

```
Function solve():
    Read integers n and m from input
    Read array x from input and prepend 0
    Read array y from input and prepend 0

    # Initialize 2D prefix sum array a with zeros
    Initialize a as a 2D array of size (n + 1) x (m + 1) with all elements as 0

    For i from 1 to n:
        Initialize curr_sum to 0
        Read row from input

        For j from 1 to m:
            Update curr_sum by adding row[j-1]
            Update a[i][j] to curr_sum + a[i-1][j]
```

PSEUDOCODE

```
Initialize res to 0
# Cache frequently computed values
For x1 from 1 to n:
    For x2 from x1 to n:
        Set d to x[x2] - x[x1]
        Initialize Min to infinity
        # Precompute values outside the loop
        Set prev_rows to a[x1-1]
        Set curr_rows to a[x2]

        Initialize val to 0
        For j from 1 to m:
            Update Min to the minimum of Min and val - d * y[j]
            # Optimize val computation using cached values
            Update val to curr_rows[j] - curr_rows[0] - prev_rows[j] + prev_rows[0]
            Set curr_val to val - d * y[j] - Min
            If curr_val > res:
                Update res to curr_val
Return res
```

Complexity?

SOLUTION

04 Evaluation

- **Sort** the profit array: $O((X+Y+Z) \cdot \log(X+Y+Z))$
- Using a **heap**: $O((X+Y+Z) \cdot \log(X+Y+Z))$

→ **Complexity:** $O((X+Y+Z) \cdot \log(X+Y+Z))$

- Space complexity: $O(X+Y+Z)$

05 Testing

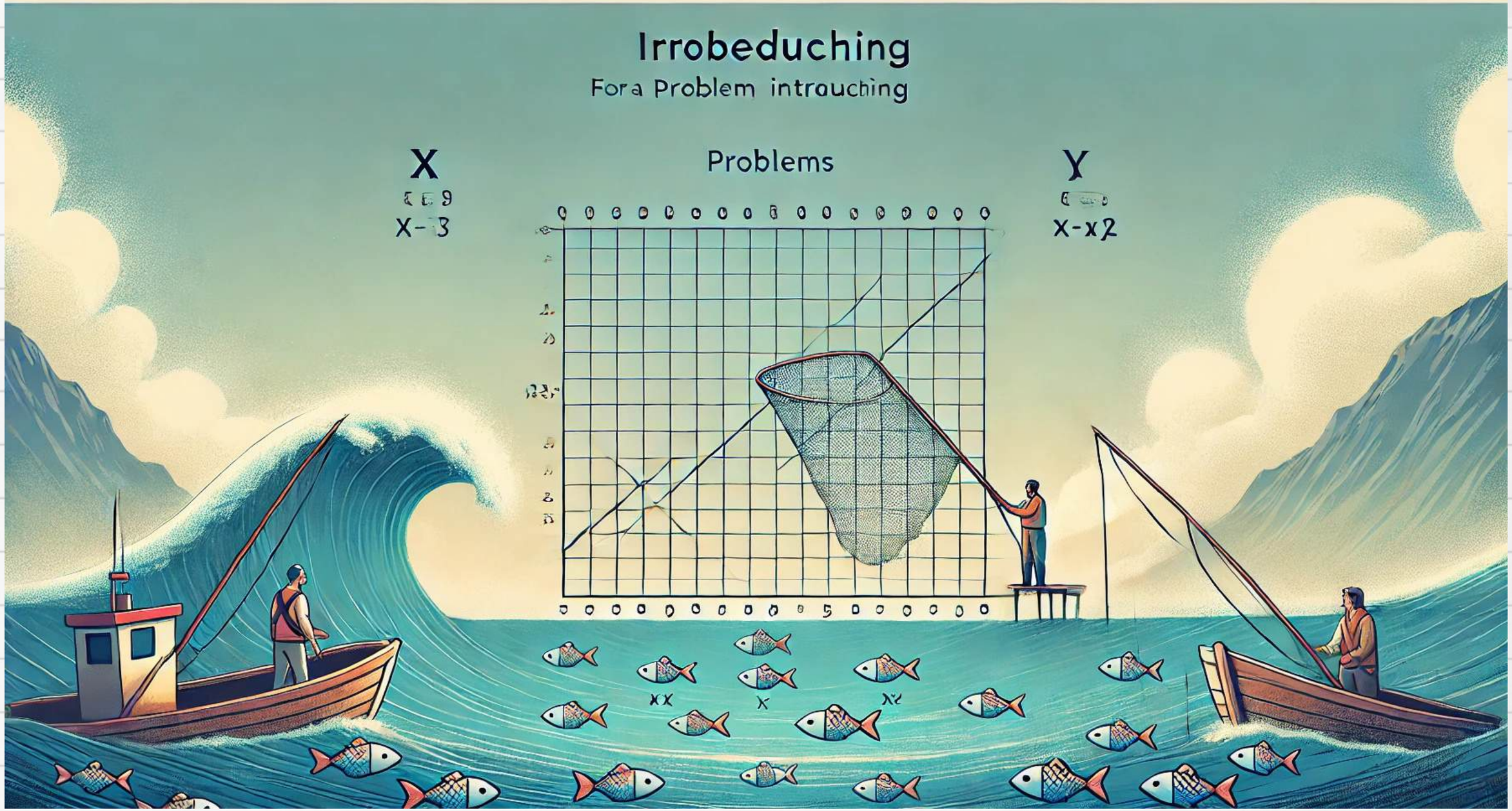
- Boundary cases: X, Y, Z have values of 0 or 1, ...
- Positive test cases.
- Performance test cases: X, Y, Z are large, ...
- Special test case: All plot profits are equal.

PROBLEM 2: HIGH PLACE

Some practical examples:

- Company resource allocation
- Agricultural land use planning
- Function allocation in hotels
- ...

PROBLEM 3: FISHING



PROBLEM 3: FISHING

A sea area is described by the coordinate system Oxy. There are N * M fish currently living in the sea area, with the fish at position (i, j) having coordinates (X(i), Y(j)) and a value of A(i,j) (with 1 ≤ i ≤ N, 1 ≤ j ≤ M). A fisherman is preparing to catch fish in the sea. To catch fish, the fisherman will spread a rectangular net with edges parallel to the coordinate axes (note that the area of the net can be zero). If we denote the coordinates of the bottom-left corner of the net as (x1, y1) and the coordinates of the top-right corner as (x2, y2) (x1 ≤ x2, y1 ≤ y2), then the profit of the fisherman will be equal to the total value of the fish within the net (including the edges of the net) minus the area of the net. Specifically, the profit is calculated using the following formula:

$$\left(\sum_{\substack{X_i \in [x_1, x_2] \\ Y_j \in [y_1, y_2]}} A_{i,j}\right) - (x_2 - x_1)(y_2 - y_1)$$

Find the maximum profit that the fisherman can achieve.

PROBLEM 3: FISHING

Input:

- The first line: two integers N and M ($1 \le N, M \le 400$).
- The second line: a sequence of N integers $X_1, X_2, \ldots, X_N (|X_i| \le 10^7, X_{i-1} < X_i)$
- The third line: a sequence of M integers $Y_1, Y_2, \ldots, Y_M (|Y_i| \le 10^7, Y_{i-1} < Y_i)$
- The next N lines, the i-th line consists of M integers $A_{i,1}, \ldots, A_{i,M} (0 \le A_{i,j} \le 10^9)$

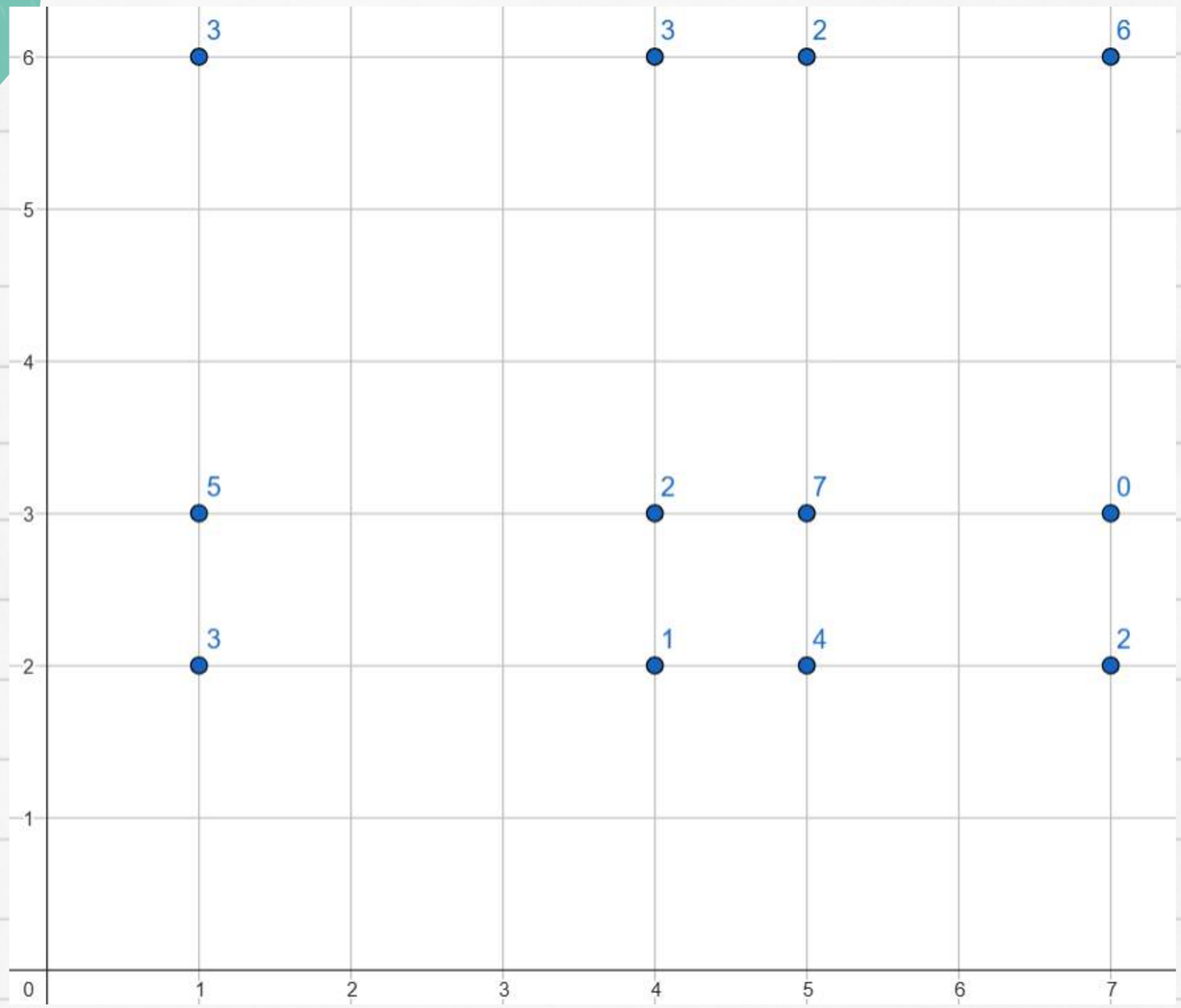
Output: The maximum profit achieved. .

Input	Output
4 3 1 4 5 7 2 3 6 3 5 3 1 2 3 4 7 2 2 0 6	18

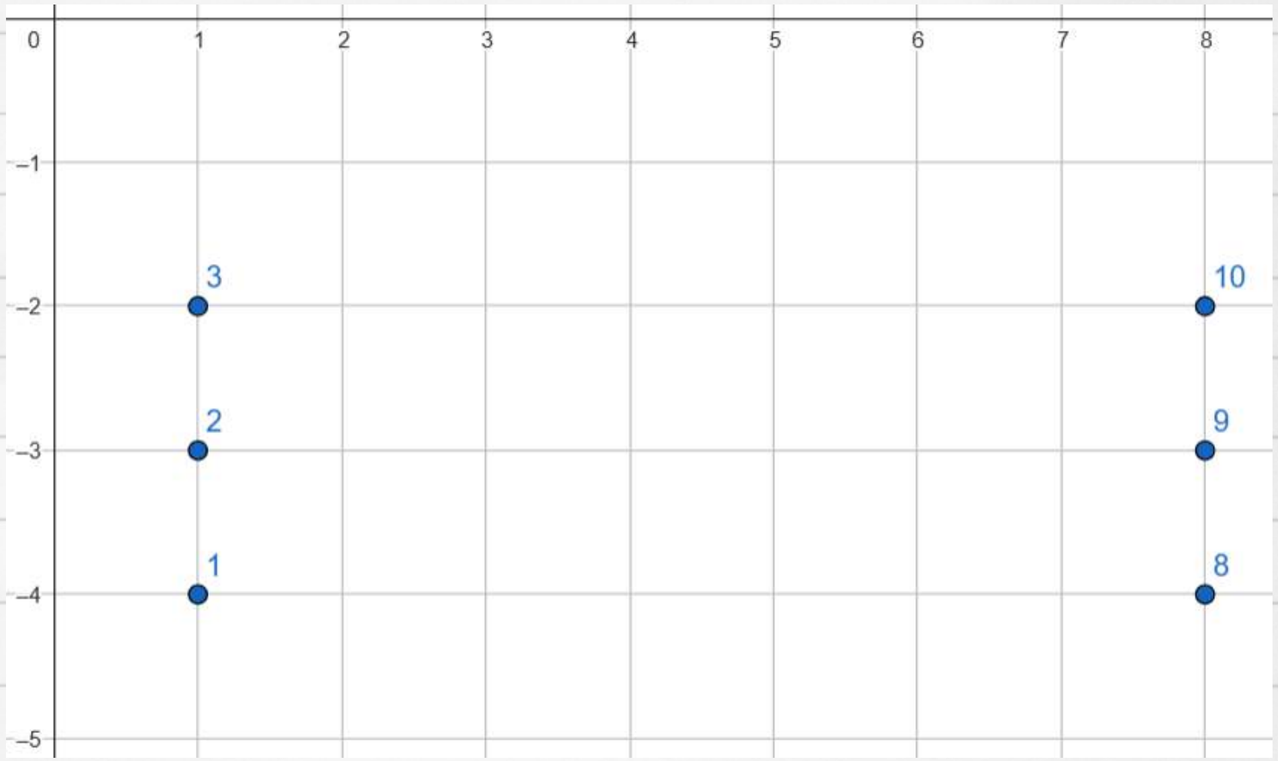
Input	Output
2 3 1 8 -4 -3 -2 1 2 3 8 9 10	27

PROBLEM 3: FISHING

Example 1:



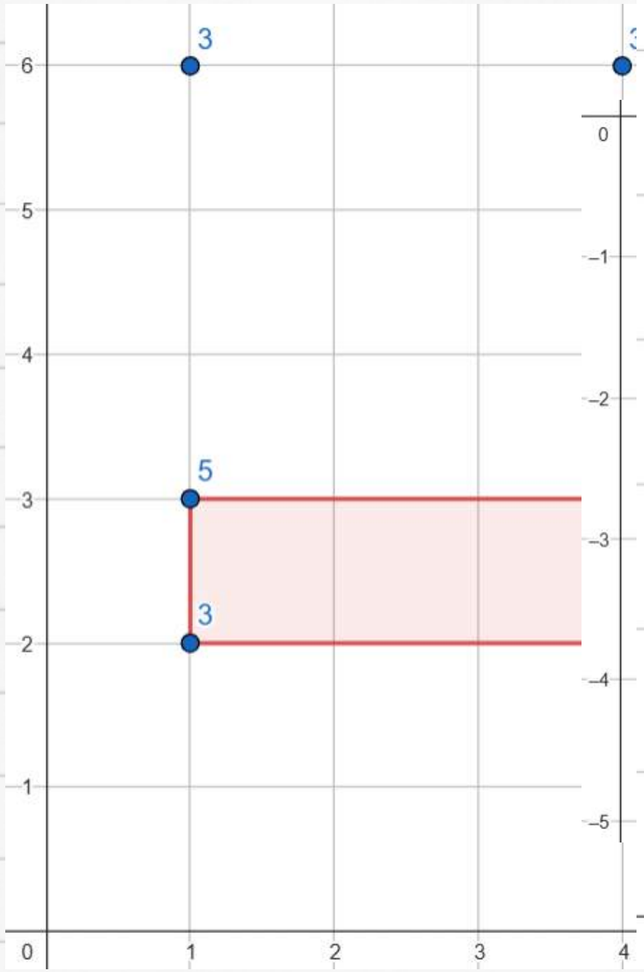
Example 2:



How is the profit calculated?

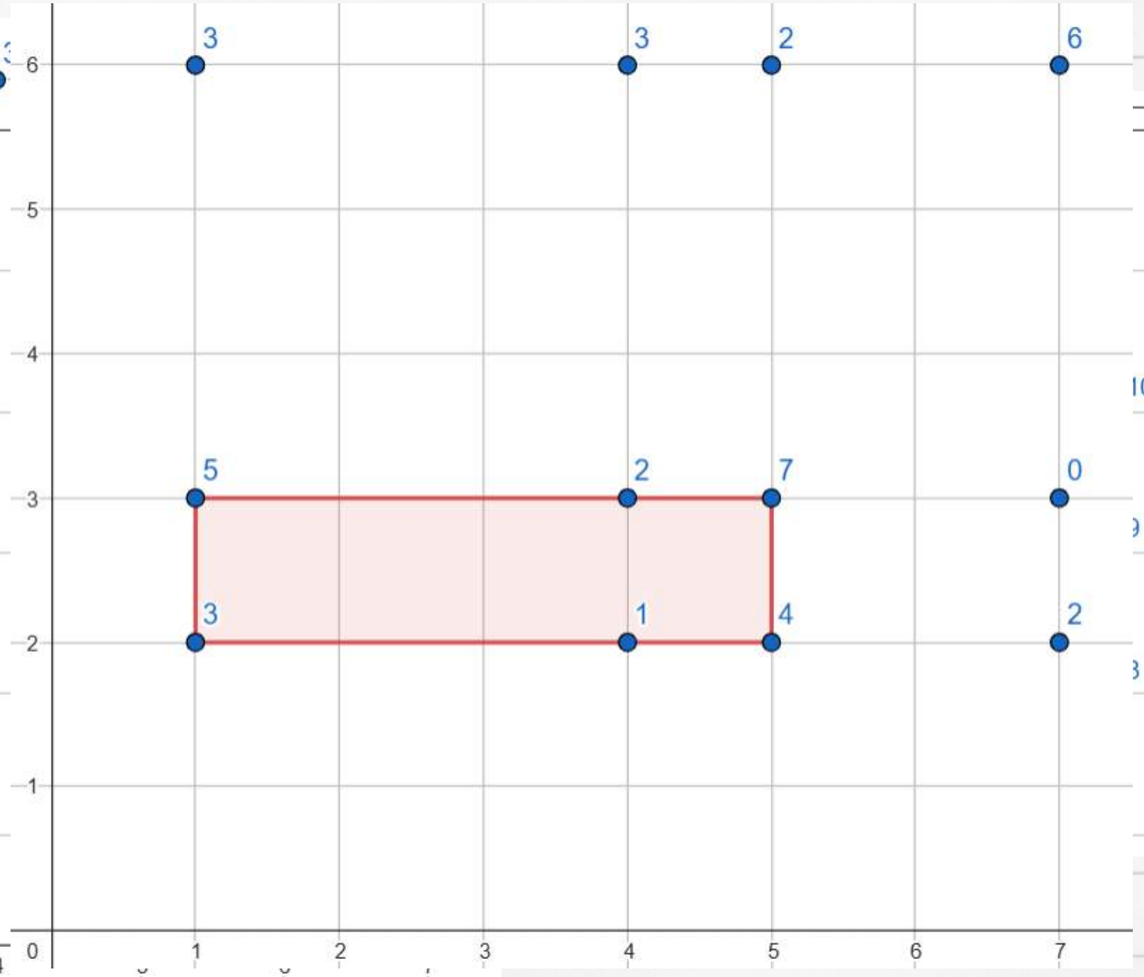
PROBLEM 3: FISHING

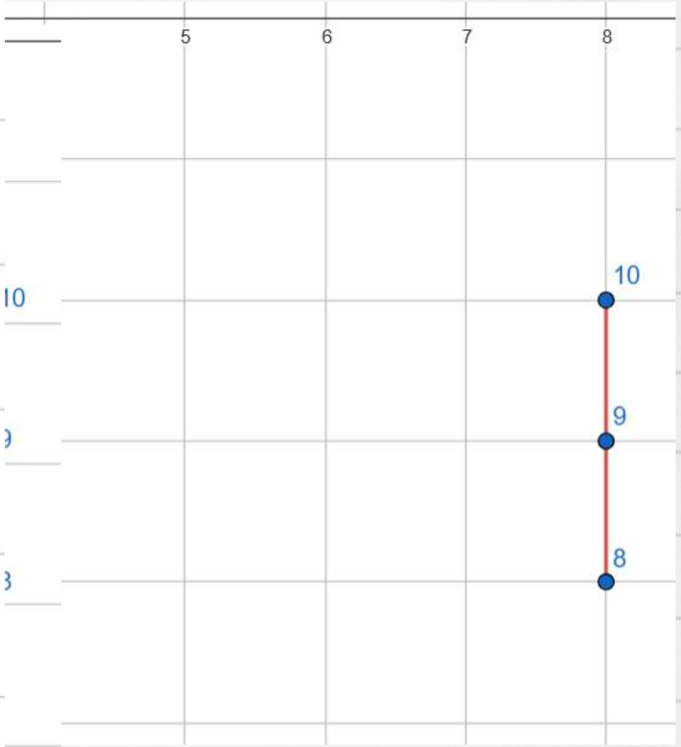
Example 1:



Profit: (3 + 5 + 1 + 2 + 4 + 7) - (5 - 1) * (3 - 2) = 18

Example 2:





Profit: (8 + 9 + 10) - (8 - 8) * (-2 - (-4)) = 27

SOLUTION

01 Abstraction

- **Input:** A network of coordinate points and values.
- **Output:** The maximum value of the expression.
- Find a rectangle with the bottom-left corner at (x1, y1) and the top-right corner at (x2, y2) such that the profit calculation expression is maximized.

SOLUTION

02 Decomposition & Pattern Recognition

- Noticing that if we fix the x-coordinates (x1, x2) and iterate through the y-coordinates, the problem becomes finding the subarray with the maximum sum.
- Use a **2D prefix sum array** to calculate the given expression, combined with **brute force**.
- Optimize the linear search, similar to the "Maximum Subarray Sum" problem.

03 Algorithm Design

- Calculate the 2D prefix sum array:

$$F(i,j) = A(i,j) + F(i-1,j) + F(i,j-1) - F(i-1,j-1)$$

- Profit:

$$S(x_1,y_1,x_2,y_2) = F(x_2,y_2) - F(x_2,y_1-1) - F(x_1-1,y_2) + F(x_1-1,y_1-1) - (x_2-x_1)\cdot(y_2-y_1)$$

- For each pair (x1, x2), choose a pair of coordinate (y1, y2) to maximize the profit.
- Calculate D[j] = S(x1, 1, x2, j)
- When selecting the rectangle (x1, y1, x2, y2):

$$Profit = D(y_2) - D(y_1) + S(x_1,y_1,x_2,y_1)$$

- Iterate through y2 and maintain the minimum value of D(y1) to maximize profit.

CODE

```python
def solve():
    n, m = map(int, input().split())
    x = [0] + list(map(int, input().split()))
    y = [0] + list(map(int, input().split()))

    # Optimize 2D Prefix Sum Calculation
    a = [[0] * (m + 1) for _ in range(n + 1)]
    for i in range(1, n + 1):
        curr_sum = 0
        row = list(map(int, input().split()))
        for j in range(1, m + 1):
            curr_sum += row[j-1]
            a[i][j] = curr_sum + a[i-1][j]
```

CODE

```
    res = 0
    # Cache frequently computed values.
    for x1 in range(1, n + 1):
        for x2 in range(x1, n + 1):
            d = x[x2] - x[x1]
            Min = float('inf')
            # Precompute the invariant parts within the loop.
            prev_rows = a[x1-1]
            curr_rows = a[x2]

            val = 0
            for j in range(1, m + 1):
                Min = min(Min, val - d * y[j])
                # Optimize the calculation of val by using cached values.
                val = curr_rows[j] - curr_rows[0] - prev_rows[j] + prev_rows[0]
                curr_val = val - d * y[j] - Min
                if curr_val > res:
                    res = curr_val
    return res
```

Complexity?

SOLUTION

04 Evaluation

- Iterate through all pairs (x1, x2): $O(N^2)$
- Find the optimal subarray for each pair: O(M)

→ **Complexity:** $O(N^2 \cdot M)$

- Space complexity: O(N.M)

05 Testing

- Boundary cases: a matrix of all zeros, ...
- Positive test cases, ...
- Performance test cases: A matrix containing large values A[i], or with a large size M × N.

PROBLEM 3: FISHING

Some practical examples:

- Optimization in resource extraction
- Selection of agricultural production areas to maximize crop yield.
- Analysis of customer concentration areas in marketing
- Optimal routing for warehouses or factories

QUIZ

Link: https://create.kahoot.it/share/cs112-quiz-on-tap/854b8e85-4507-4c2e-b6a6-220226469ae9

THANK YOU