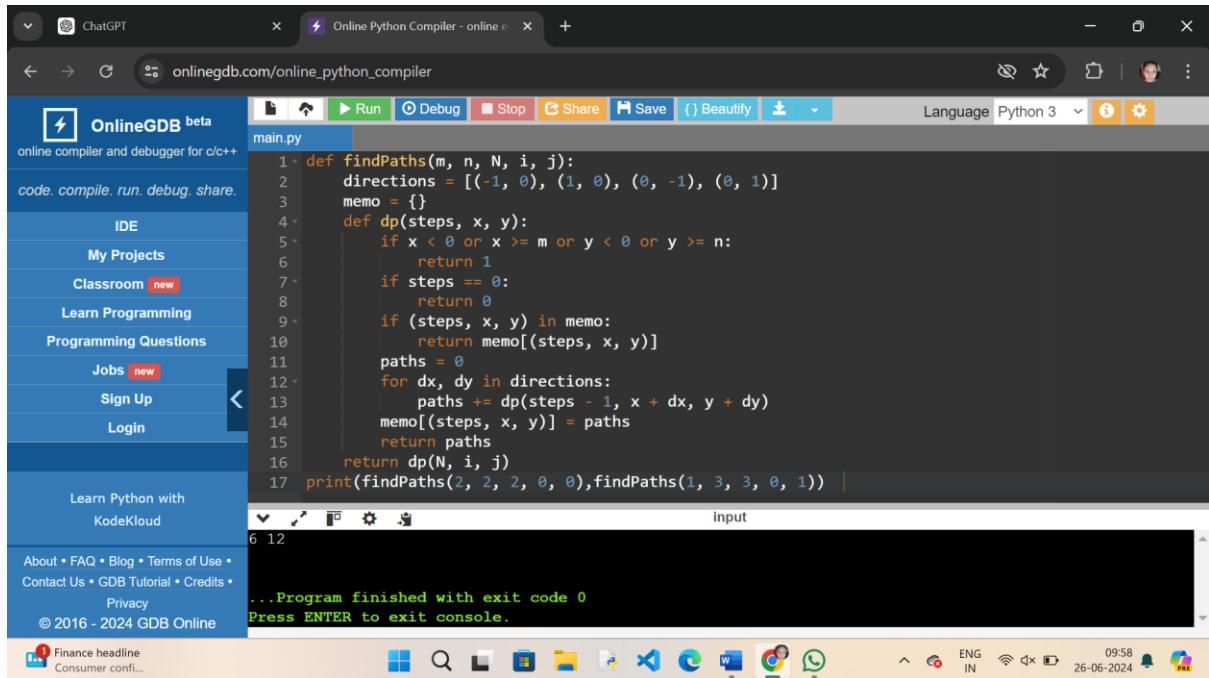


- Given an $m \times n$ grid and a ball at a starting cell, find the number of ways to move the ball out of the grid boundary in exactly N steps.

Example:

- Input: $m=2, n=2, N=2, i=0, j=0$ Output: 6
- Input: $m=1, n=3, N=3, i=0, j=1$ Output: 12



The screenshot shows the OnlineGDB beta IDE interface. On the left, there's a sidebar with various options like IDE, My Projects, Classroom, Learn Programming, Programming Questions, Jobs, Sign Up, and Login. The main area has tabs for 'main.py' and 'Output'. The code in 'main.py' is as follows:

```

1 def findPaths(m, n, N, i, j):
2     directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
3     memo = {}
4     def dp(steps, x, y):
5         if x < 0 or x >= m or y < 0 or y >= n:
6             return 1
7         if steps == 0:
8             return 0
9         if (steps, x, y) in memo:
10            return memo[(steps, x, y)]
11        paths = 0
12        for dx, dy in directions:
13            paths += dp(steps - 1, x + dx, y + dy)
14        memo[(steps, x, y)] = paths
15        return paths
16    return dp(N, i, j)
17 print(findPaths(2, 2, 2, 0, 0), findPaths(1, 3, 3, 0, 1))

```

The 'Output' tab shows the results of running the code: "...Program finished with exit code 0". The status bar at the bottom right indicates the date and time as 26-06-2024 09:58.

- You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are arranged in a circle. That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have security systems connected, and it will automatically contact the police if two adjacent houses were broken into on the same night.

Examples:

- (i) Input : nums = [2, 3, 2]

Output : The maximum money you can rob without alerting the police is 3(robbing house 1).

- (ii) Input : nums = [1, 2, 3, 1]

Output : The maximum money you can rob without alerting the police is 4 (robbing house 1 and house 3).

The screenshot shows the OnlineGDB beta IDE interface. On the left sidebar, there are links for IDE, My Projects, Classroom, Learn Programming, Programming Questions, Jobs, Sign Up, and Login. The main area displays a Python script named 'main.py' with the following code:

```

1 def rob(nums):
2     def rob_linear(houses):
3         prev, curr = 0, 0
4         for amount in houses:
5             prev, curr = curr, max(curr, prev + amount)
6         return curr
7     if not nums:
8         return 0
9     if len(nums) == 1:
10        return nums[0]
11    return max(rob_linear(nums[:-1]), rob_linear(nums[1:]))
12 print(rob([2, 3, 2]),rob([1, 2, 3, 1])) # Output: 4

```

The input field contains '3 4' and the output field shows the result of the program execution: '...Program finished with exit code 0'. The status bar at the bottom right indicates the date and time as 26-06-2024 10:02.

3. You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Examples:

- (i) Input: n=4 Output: 5
- (ii) Input: n=3 Output: 3

The screenshot shows the OnlineGDB beta IDE interface. On the left sidebar, there are links for IDE, My Projects, Classroom, Learn Programming, Programming Questions, Jobs, Sign Up, and Login. The main area displays a Python script named 'main.py' with the following code:

```

1 def climbStairs(n):
2     if n == 1:
3         return 1
4     if n == 2:
5         return 2
6     prev2, prev1 = 1, 2
7     for i in range(3, n + 1):
8         current = prev1 + prev2
9         prev2, prev1 = prev1, current
10    return prev1
11
12 print(climbStairs(4),climbStairs(3))
13

```

The input field contains '5 3' and the output field shows the result of the program execution: '...Program finished with exit code 0'. The status bar at the bottom right indicates the date and time as 26-06-2024 10:04.

4. A robot is located at the top-left corner of a $m \times n$ grid. The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid. How many possible unique paths are there?

Examples:

- (i) Input: m=7,n=3 Output: 28
- (ii) Input: m=3,n=2 Output: 3

The screenshot shows the OnlineGDB beta IDE interface. On the left, there's a sidebar with links like IDE, My Projects, Classroom, Learn Programming, Programming Questions, Jobs, Sign Up, and Login. The main area has tabs for 'main.py' and 'Input'. The code in 'main.py' is:

```
1 def uniquePaths(m, n):
2     dp = [[1] * n for _ in range(m)]
3     for i in range(1, m):
4         for j in range(1, n):
5             dp[i][j] = dp[i-1][j] + dp[i][j-1]
6     return dp[m-1][n-1]
7 print(uniquePaths(7, 3),uniquePaths(3, 2))
```

The 'Input' tab shows the command: `python main.py`. The output window displays the results of the program execution:

```
...Program finished with exit code 0
Press ENTER to exit console.
```

In a string S of lowercase letters, these letters form consecutive groups of the same character. For example, a string like $s = "abbxxxxzzy"$ has the groups "a", "bb", "xxxx", "z", and "yy". A group is identified by an interval $[start, end]$, where start and end denote the start and end indices (inclusive) of the group. In the above example, "xxxx" has the interval $[3,6]$. A group is considered large if it has 3 or more characters. Return the intervals of every large group sorted in increasing order by start index.

Example 1:

Input: $s = "abbxxxxzzy"$

Output: $[[3,6]]$

Explanation: "xxxx" is the only large group with start index 3 and end index 6.

Example 2:

Input: $s = "abc"$

Output: $[]$

Explanation: We have groups "a", "b", and "c", none of which are large groups.

```

OnlineGDB beta
online compiler and debugger for c/c++
code. compile. run. debug. share.

IDE
My Projects
Classroom new
Learn Programming
Programming Questions
Jobs new
Sign Up
Login

Learn Python with
KodeKloud

About • FAQ • Blog • Terms of Use •
Contact Us • GDB Tutorial • Credits •
Privacy
© 2016 - 2024 GDB Online

main.py
1- def largeGroupPositions(s):
2-     result = []
3-     n = len(s)
4-     i = 0
5-     while i < n:
6-         start = i
7-         while i < n - 1 and s[i] == s[i + 1]:
8-             i += 1
9-             if i - start + 1 >= 3:
10-                 result.append([start, i])
11-             i += 1
12-     return result
13- print(largeGroupPositions("abbxxxxzzy"),largeGroupPositions("abc"))
14-
15-

```

Input

```

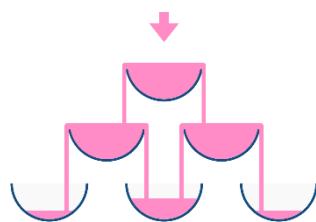
[[3, 6]] []
... Program finished with exit code 0
Press ENTER to exit console.

```

89°F Mostly cloudy

ENG IN 26-06-2024 10:10

- We stack glasses in a pyramid, where the first row has 1 glass, the second row has 2 glasses, and so on until the 100th row. Each glass holds one cup of champagne. Then, some champagne is poured into the first glass at the top. When the topmost glass is full, any excess liquid poured will fall equally to the glass immediately to the left and right of it. When those glasses become full, any excess champagne will fall equally to the left and right of those glasses, and so on. (A glass at the bottom row has its excess champagne fall on the floor.) For example, after one cup of champagne is poured, the top most glass is full. After two cups of champagne are poured, the two glasses on the second row are half full. After three cups of champagne are poured, those two cups become full - there are 3 full glasses total now. After four cups of champagne are poured, the third row has the middle glass half full, and the two outside glasses are a quarter full, as pictured below.



Now after pouring some non-negative integer cups of champagne, return how full the j^{th} glass in the i^{th} row is (both i and j are 0-indexed.)

Example 1:

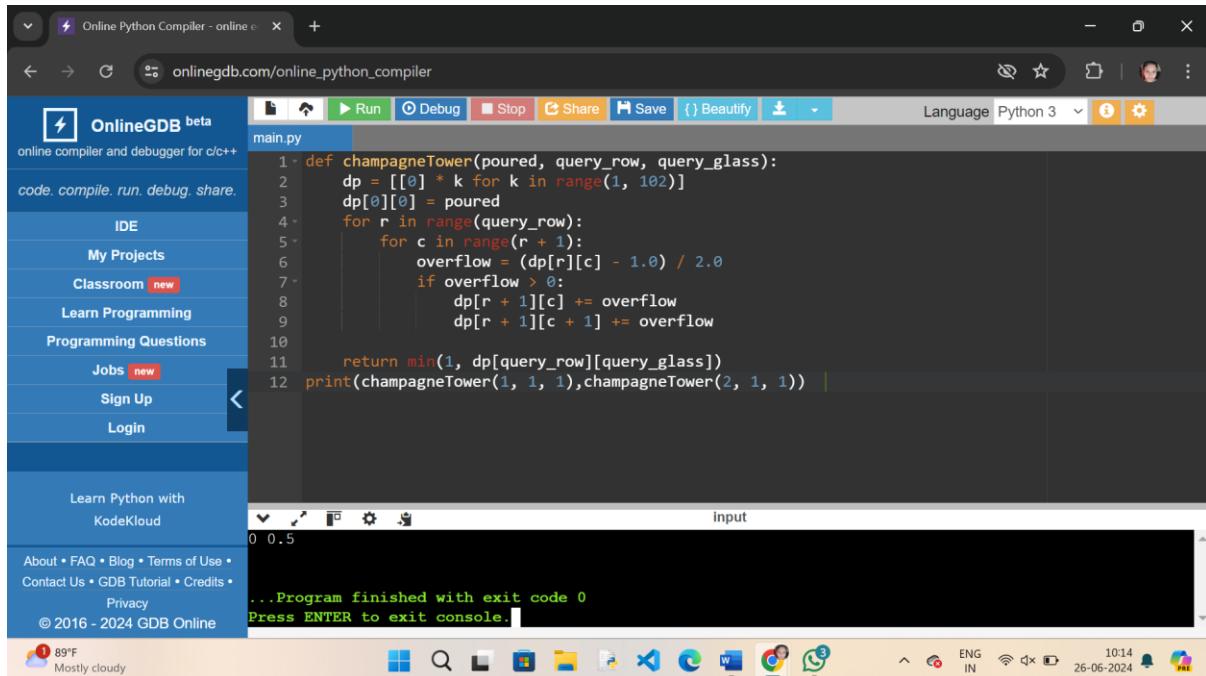
Input: poured = 1, query_row = 1, query_glass = 1
Output: 0.00000

Explanation: We poured 1 cup of champagne to the top glass of the tower (which is indexed as (0, 0)). There will be no excess liquid so all the glasses under the top glass will remain empty.

Example 2:

Input: poured = 2, query_row = 1, query_glass = 1
Output: 0.50000

Explanation: We poured 2 cups of champagne to the top glass of the tower (which is indexed as (0, 0)). There is one cup of excess liquid. The glass indexed as (1, 0) and the glass indexed as (1, 1) will share the excess liquid equally, and each will get half cup of champagne.



The screenshot shows the OnlineGDB beta interface. On the left, there's a sidebar with links like 'IDE', 'My Projects', 'Classroom', 'Learn Programming', 'Programming Questions', 'Jobs', 'Sign Up', and 'Login'. The main area has tabs for 'main.py' and 'output'. The code in 'main.py' is:

```

1 def champagneTower(poured, query_row, query_glass):
2     dp = [[0] * k for k in range(1, 102)]
3     dp[0][0] = poured
4     for r in range(query_row):
5         for c in range(r + 1):
6             overflow = (dp[r][c] - 1.0) / 2.0
7             if overflow > 0:
8                 dp[r + 1][c] += overflow
9                 dp[r + 1][c + 1] += overflow
10
11     return min(1, dp[query_row][query_glass])
12 print(champagneTower(1, 1, 1),champagneTower(2, 1, 1))

```

The 'output' tab shows the console output:

```

0 0.5
...Program finished with exit code 0
Press ENTER to exit console.

```

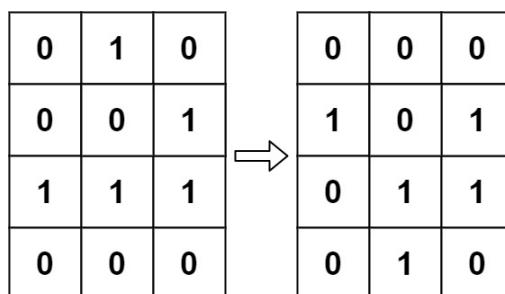
5. "The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970." The board is made up of an $m \times n$ grid of cells, where each cell has an initial state: live (represented by a 1) or dead (represented by a 0). Each cell interacts with its eight neighbors (horizontal, vertical, diagonal) using the following four rules

Any live cell with fewer than two live neighbors dies as if caused by under-population.

1. Any live cell with two or three live neighbors lives on to the next generation.
2. Any live cell with more than three live neighbors dies, as if by over-population.
3. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

The next state is created by applying the above rules simultaneously to every cell in the current state, where births and deaths occur simultaneously. Given the current state of the $m \times n$ grid board, return *the next state*.

Example 1:



The diagram illustrates a 4x3 grid of cells transitioning to the next state. The initial state (left) is:

0	1	0
0	0	1
1	1	1
0	0	0

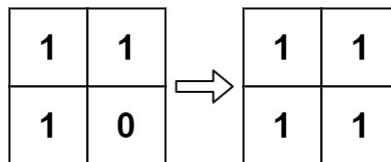
An arrow points to the next state (right):

0	0	0
1	0	1
0	1	1
0	1	0

Input: board = [[0,1,0],[0,0,1],[1,1,1],[0,0,0]]

Output: [[0,0,0],[1,0,1],[0,1,1],[0,1,0]]

Example 2:



Input: board = [[1,1],[1,0]]

Output: [[1,1],[1,1]]

Programiz Python Online Compiler

```
main.py
1 def game_of_life(board):
2     m, n = len(board), len(board[0])
3     neighbors = [(-1, -1), (-1, 0), (-1, 1),
4                  (0, -1),          (0, 1),
5                  (1, -1), (1, 0), (1, 1)]
6     for row in range(m):
7         for col in range(n):
8             live_neighbors = 0
9             for neighbor in neighbors:
10                r, c = row + neighbor[0], col + neighbor[1]
11                if (0 <= r < m) and (0 <= c < n) and abs(board[r][c]) == 1:
12                    live_neighbors += 1
13                if board[row][col] == 1 and (live_neighbors < 2 or live_neighbors >
14                                              3):
15                    board[row][col] = 2 # Live to dead
16                if board[row][col] == 0 and live_neighbors == 3:
17                    board[row][col] = -1 # Dead to live
18    for row in range(m):
19        for col in range(n):
20            if board[row][col] == 2:
21                board[row][col] = 0
22            if board[row][col] == -1:
23                board[row][col] = 1
24    return board
25 print(game_of_life([[0,1,0],[0,0,1],[1,1,1],[0,0,0]])) # Output: [[0,0,0],[1,0,1],[0,1,1],[0,1,0]]
```

Output

```
[[0, 0, 0], [0, 0, 1], [0, 1, 1], [0, 0, 0]]
[[1, 1], [1, 1]]
*** Code Execution Successful ***
```

32°C Mostly cloudy

Search

10:41 26-06-2024 ENG IN

6. You have an algorithm that process a list of numbers. It firsts sorts the list using an efficient sorting algorithm and then finds the maximum element in sorted list. Write the code for the same.

Test Cases

1. Empty List

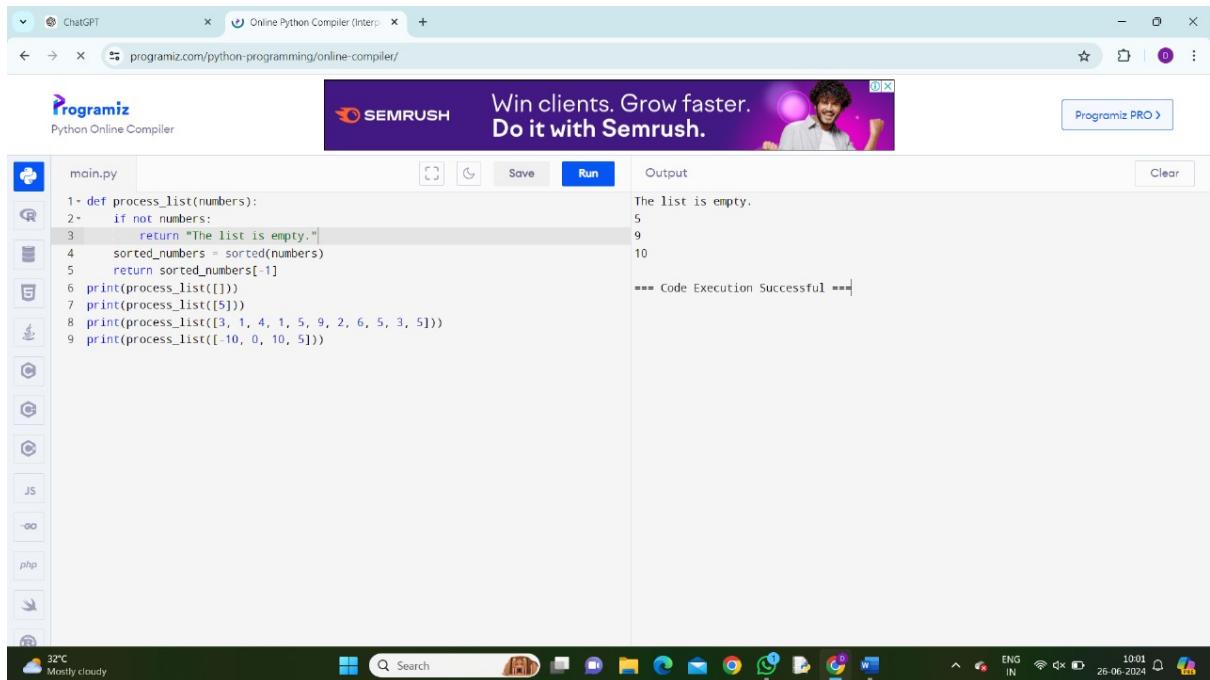
1. Input: []
2. Expected Output: None or an appropriate message indicating that the list is empty.

2. Single Element List

1. Input: [5]
2. Expected Output: 5

3. All Elements are the Same

1. Input: [3, 3, 3, 3, 3]
2. Expected Output: 3



```
main.py
1- def process_list(numbers):
2-     if not numbers:
3-         return "The list is empty."
4-     sorted_numbers = sorted(numbers)
5-     return sorted_numbers[1]
6 print(process_list([]))
7 print(process_list([5]))
8 print(process_list([3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]))
9 print(process_list([-10, 0, 10, 5]))
```

The list is empty.
5
9
10
==== Code Execution Successful ===

7. Write a program that takes an input list of n numbers and creates a new list containing only the unique elements from the original list. What is the space complexity of the algorithm?

Test Cases

Some Duplicate Elements

- Input: [3, 7, 3, 5, 2, 5, 9, 2]
- Expected Output: [3, 7, 5, 2, 9] (Order may vary based on the algorithm used)

Negative and Positive Numbers

- Input: [-1, 2, -1, 3, 2, -2]
- Expected Output: [-1, 2, 3, -2] (Order may vary)

List with Large Numbers

- Input: [1000000, 999999, 1000000]
- Expected Output: [1000000, 999999]

The screenshot shows a browser window with the URL programiz.com/python-programming/online-compiler/. The main content area displays a Python script named `main.py` containing a function to find unique elements in a list. The output window shows the execution results, including the unique elements from three different lists and a success message.

```

main.py
1- def unique_elements(numbers):
2     unique_set = set()
3     unique_list = []
4
5     for number in numbers:
6         if number not in unique_set:
7             unique_set.add(number)
8             unique_list.append(number)
9
10    return unique_list
11 print(unique_elements([3, 7, 5, 2, 9]))
12 print(unique_elements([-1, 2, -1, 3, 2, -2]))
13 print(unique_elements([1000000, 999999, 1000000]))
```

Output

```
[3, 7, 5, 2, 9]
[-1, 2, 3, -2]
[1000000, 999999]

== Code Execution Successful ==
```

Sort an array of integers using the bubble sort technique. Analyze its time complexity using Big-O notation. Write the code

The screenshot shows a browser window with the URL programiz.com/python-programming/online-compiler/. The main content area displays a Python script named `main.py` containing a bubble sort implementation. The output window shows the sorted arrays for four different input lists and a success message.

```

main.py
1- def bubble_sort(arr):
2     n = len(arr)
3     for i in range(n):
4         swapped = False
5         for j in range(0, n-1):
6             if arr[j] > arr[j+1]:
7                 arr[j], arr[j+1] = arr[j+1], arr[j]
8                 swapped = True
9         if not swapped:
10            break
11    return arr
12 print(bubble_sort([64, 34, 25, 12, 22, 11, 90]))
13 print(bubble_sort([5, 1, 4, 2, 8]))
14 print(bubble_sort([3, 7, 3, 5, 2, 5, 9, 2]))
15 print(bubble_sort([-1, 2, -1, 3, 2, -2]))
16 print(bubble_sort([1000000, 999999, 1000000]))
```

Output

```
[11, 12, 22, 25, 34, 64, 90]
[1, 2, 4, 5, 8]
[2, 2, 3, 3, 5, 7, 9]
[-2, -1, -1, 2, 2, 3]
[999999, 1000000, 1000000]

== Code Execution Successful ==
```

Checks if a given number x exists in a sorted array arr using binary search. Analyze its time complexity using Big-O notation.

Test Case:

Example $X=\{ 3,4,6,-9,10,8,9,30 \}$ KEY=10

Output: Element 10 is found at position 5

Example $X=\{ 3,4,6,-9,10,8,9,30 \}$ KEY=100

Output : Element 100 is not found

The screenshot shows a browser window with the URL programiz.com/python-programming/online-compiler/. The page title is "Online Python Compiler (Interp)". The code editor contains a Python script named `main.py` with the following content:

```

1 def binary_search(arr, key):
2     low, high = 0, len(arr) - 1
3
4     while low <= high:
5         mid = (low + high) // 2
6         if arr[mid] == key:
7             return mid
8         elif arr[mid] < key:
9             low = mid + 1
10        else:
11            high = mid - 1
12
13    return -1
14 def search_result(arr, key):
15     result = binary_search(arr, key)
16     if result != -1:
17         return f"Element {key} is found at position {result}."
18     else:
19         return f"Element {key} is not found."
20 X = [3, 4, 6, -9, 10, 8, 9, 30]
21 X.sort()
22 print(search_result(X, 10))
23 print(search_result(X, 100))

```

The output window shows the results of the execution:

```

Element 10 is found at position 6.
Element 100 is not found.

== Code Execution Successful ==

```

The desktop taskbar at the bottom shows various application icons and system status.

Given an array of integers nums, sort the array in ascending order and return it. You must solve the problem without using any built-in functions in $O(n \log n)$ time complexity and with the smallest space complexity possible.

The screenshot shows a browser window with the URL programiz.com/python-programming/online-compiler/. The page title is "Online Python Compiler (Interp)". The code editor contains a Python script named `main.py` with the following content:

```

1 def merge_sort(nums):
2     if len(nums) <= 1:
3         return nums
4     mid = len(nums) // 2
5     left_half = merge_sort(nums[:mid])
6     right_half = merge_sort(nums[mid:])
7     return merge(left_half, right_half)
8 def merge(left, right):
9     sorted_list = []
10    i = j = 0
11    while i < len(left) and j < len(right):
12        if left[i] < right[j]:
13            sorted_list.append(left[i])
14            i += 1
15        else:
16            sorted_list.append(right[j])
17            j += 1
18    sorted_list.extend(left[i:])
19    sorted_list.extend(right[j:])
20    return sorted_list
21 print(merge_sort([3, 4, 6, -9, 10, 8, 9, 30]))
22 print(merge_sort([-64, 34, 25, 12, 22, 11, 90]))
23 print(merge_sort([5, 1, 4, 2, 8]))
24 print(merge_sort(["z", "a", "d", "c", "b"]))
25 print(merge_sort([-1, 2, -1, 3, 2, -2]))
26 print(merge_sort([-1000000, 999999, 1000000]))

```

The output window shows the results of the execution:

```

Element 10 is found at position 6.
Element 100 is not found.

== Code Execution Successful ==

```

The desktop taskbar at the bottom shows various application icons and system status.

Given an array of strings words, return the first palindromic string in the array. If there is no such string, return an empty string "". A string is palindromic if it reads the same forward and backward.

Example 1:

Input: words = ["abc", "car", "ada", "racecar", "cool"]

Output: "ada"

Explanation: The first string that is palindromic is "ada".

Note that "racecar" is also palindromic, but it is not the first.

Example 2:

Input: words = ["notapalindrome", "racecar"]

Output: "racecar"

Explanation: The first and only string that is palindromic is "racecar".

The screenshot shows a browser window with the URL <https://www.programiz.com/python-programming/online-compiler/>. The page title is "Online Python Compiler (Interpreted)". The main content area is titled "Programiz Python Online Compiler". It displays a Python script named "main.py" with the following code:

```
main.py
1 def first_palindromic_string(words):
2     for word in words:
3         if word == word[::-1]: # Check if the word is a palindrome
4             return word
5     return ""
6
7 # Example usage:
8 words1 = ["abc", "car", "ada", "racecar", "cool"]
9 words2 = ["notapalindrome", "racecar"]
10
11 print(first_palindromic_string(words1)) # Output: "ada"
12 print(first_palindromic_string(words2)) # Output: "racecar"
13
```

To the right of the code editor is the "Output" panel, which shows the results of running the code:

```
ada
racecar
== Code Execution Successful ==|
```

The browser's address bar shows the same URL. The taskbar at the bottom of the screen includes icons for various applications like ChatGPT, Gmail, YouTube, GitHub, and others.

You are given two integer arrays nums1 and nums2 of sizes n and m, respectively. Calculate the following values: answer1 : the number of indices i such that nums1[i] exists in nums2. answer2 : the number of indices i such that nums2[i] exists in nums1. Return [answer1,answer2].

Example 1:

Input: nums1 = [2,3,2], nums2 = [1,2]

Output: [2,1]

Explanation:

Example 2:

Input: nums1 = [4,3,2,3,1], nums2 = [2,2,5,2,3,6]

Output: [3,4]

Explanation:

The elements at indices 1, 2, and 3 in nums1 exist in nums2 as well. So answer1 is 3.

The elements at indices 0, 1, 3, and 4 in nums2 exist in nums1. So answer2 is 4.

The screenshot shows a browser window with the URL <https://www.programiz.com/python-programming/online-compiler/>. The main content is the Programiz Python Online Compiler interface. On the left, there is a code editor with a file named 'main.py' containing Python code. The code defines a function 'calculate_answers' that takes two sets of numbers and returns their distinct counts. The output window on the right shows the results: [2, 1] and [3, 4], followed by the message 'Code Execution Successful'. Below the code editor, there are icons for various programming languages: Python, C, C++, Java, JavaScript, PHP, and Swift. The system tray at the bottom indicates the date as 26-06-2024.

```

main.py
1 def calculate_answers(nums1, nums2):
2     set_nums1 = set(nums1)
3     set_nums2 = set(nums2)
4
5     answer1 = sum(1 for num in nums1 if num in set_nums2)
6     answer2 = sum(1 for num in nums2 if num in set_nums1)
7
8     return [answer1, answer2]
9
10 # Example usage:
11 nums1_1 = [2, 3, 2]
12 nums2_1 = [1, 2]
13 nums1_2 = [4, 3, 2, 3, 1]
14 nums2_2 = [2, 2, 5, 2, 3, 6]
15
16 print(calculate_answers(nums1_1, nums2_1)) # Output: [2, 1]
17 print(calculate_answers(nums1_2, nums2_2)) # Output: [3, 4]
18

```

You are given a 0-indexed integer array `nums`. The distinct count of a subarray of `nums` is defined as: Let `nums[i..j]` be a subarray of `nums` consisting of all the indices from `i` to `j` such that $0 \leq i \leq j < \text{nums.length}$. Then the number of distinct values in `nums[i..j]` is called the distinct count of `nums[i..j]`. Return the sum of the squares of distinct counts of all subarrays of `nums`. A subarray is a contiguous non-empty sequence of elements within an array.

Example 1:

Input: `nums = [1,2,1]`

Output: 15

Explanation: Six possible subarrays are:

[1]: 1 distinct value

[2]: 1 distinct value

[1]: 1 distinct value

[1,2]: 2 distinct values

[2,1]: 2 distinct values

[1,2,1]: 2 distinct values

The sum of the squares of the distinct counts in all subarrays is equal to $12 + 12 + 12 + 22 + 22 + 22 = 15$.

Example 2:

Input: `nums = [1,1]`

Output: 3

Explanation: Three possible subarrays are:

[1]: 1 distinct value

[1]: 1 distinct value

[1,1]: 1 distinct value

The sum of the squares of the distinct counts in all subarrays is equal to $12 + 12 + 12 = 3$.

The screenshot shows a browser window with the URL <https://www.programiz.com/python-programming/online-compiler/>. The main content is the Programiz Python Online Compiler. On the left, there's a code editor with a file named 'main.py' containing Python code. On the right, there's an 'Output' panel showing the results of running the code. The output is:

```
15
3
== Code Execution Successful ==|
```

Given a 0-indexed integer array `nums` of length `n` and an integer `k`, return *the number of pairs* (i, j) where $0 \leq i < j < n$, such that $\text{nums}[i] == \text{nums}[j]$ and $(i * j)$ is divisible by `k`.

Example 1:

Input: `nums = [3,1,2,2,2,1,3]`, `k = 2`

Output: 4

Explanation:

There are 4 pairs that meet all the requirements:

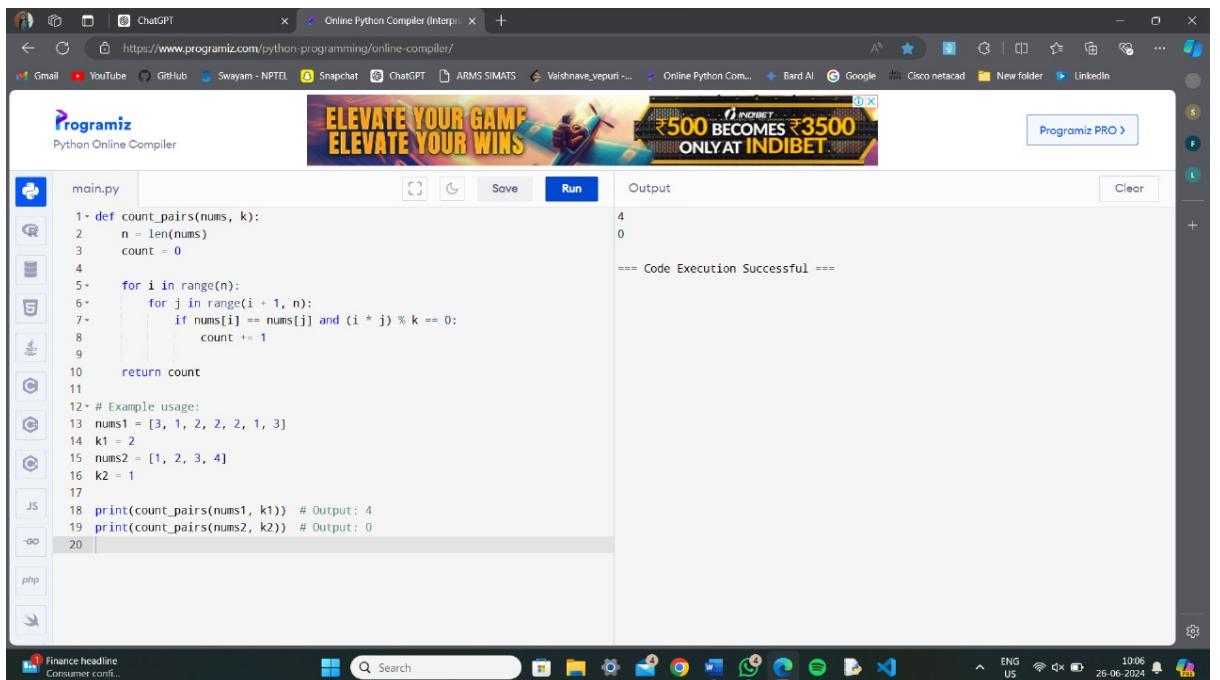
- `nums[0] == nums[6]`, and $0 * 6 == 0$, which is divisible by 2.
- `nums[2] == nums[3]`, and $2 * 3 == 6$, which is divisible by 2.
- `nums[2] == nums[4]`, and $2 * 4 == 8$, which is divisible by 2.
- `nums[3] == nums[4]`, and $3 * 4 == 12$, which is divisible by 2.

Example 2:

Input: `nums = [1,2,3,4]`, `k = 1`

Output: 0

Explanation: Since no value in `nums` is repeated, there are no pairs (i,j) that meet all the requirements.



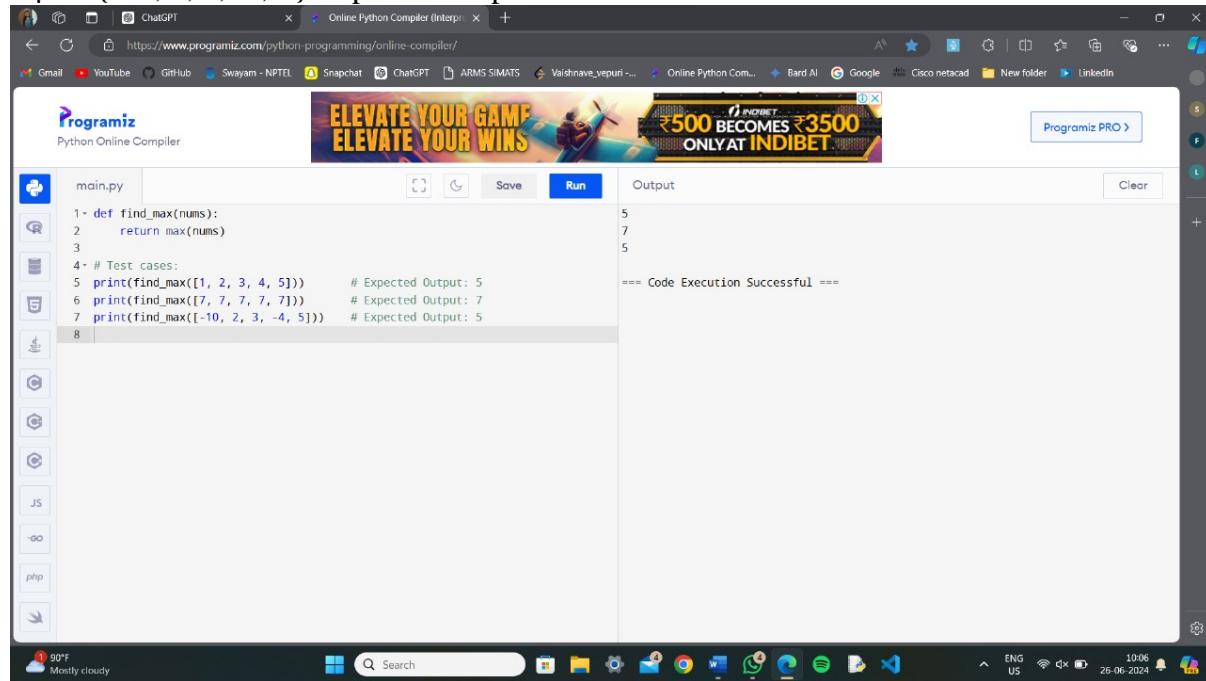
```
main.py
1 def count_pairs(nums, k):
2     n = len(nums)
3     count = 0
4
5     for i in range(n):
6         for j in range(i + 1, n):
7             if nums[i] == nums[j] and (i * j) % k == 0:
8                 count += 1
9
10    return count
11
12 # Example usage:
13 nums1 = [3, 1, 2, 2, 1, 3]
14 k1 = 2
15 nums2 = [1, 2, 3, 4]
16 k2 = 1
17
18 print(count_pairs(nums1, k1)) # Output: 4
19 print(count_pairs(nums2, k2)) # Output: 0
20
```

Output
4
0
== Code Execution Successful ==

Write a program FOR THE BELOW TEST CASES with least time complexity

Test Cases: -

- 1) Input: {1, 2, 3, 4, 5} Expected Output: 5
- 2) Input: {7, 7, 7, 7, 7} Expected Output: 7
- 3) Input: {-10, 2, 3, -4, 5} Expected Output: 5



```
main.py
1 def find_max(nums):
2     return max(nums)
3
4 # Test cases:
5 print(find_max([1, 2, 3, 4, 5]))      # Expected Output: 5
6 print(find_max([7, 7, 7, 7, 7]))      # Expected Output: 7
7 print(find_max([-10, 2, 3, -4, 5]))   # Expected Output: 5
8
```

Output
5
7
5
== Code Execution Successful ==