

**1. 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.**

Aim:

To write a program to read the same forward and backward in a string is a palindrome

Algorithm:

**Step 1:** Start

**Step 2:** Read the array of strings words

**Step 3:** For each string word in words, do the following:

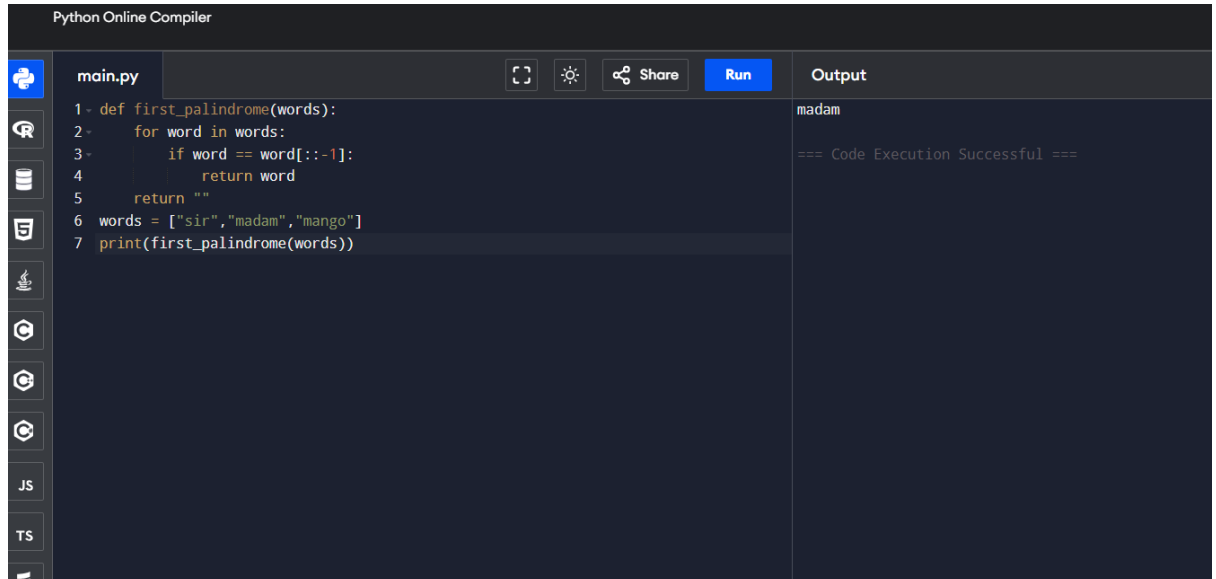
- a. Reverse the string  $\rightarrow \text{rev} = \text{word}[::-1]$
- b. Compare the original string with its reverse

If word == rev, then

Return word (This is the first palindrome)

**Step 4:** If no palindromic string is found after checking all words, return ""

**Step 5:** Stop



The screenshot shows a Python Online Compiler interface. The code in the editor is as follows:

```
main.py
1- def first_palindrome(words):
2-     for word in words:
3-         if word == word[::-1]:
4-             return word
5-     return ""
6 words = ["sir", "madam", "mango"]
7 print(first_palindrome(words))
```

The output on the right shows the result of the execution:

```
madam
=== Code Execution Successful ===
```

Result :

The program has been executed successfully

**2. 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].**

**Aim:**

To find how many elements from nums1 exist in nums2 and how many elements from nums2 exist in nums1, and return both counts as [answer1, answer2].

**Algorithm:**

**Step 1:** Start

**Step 2:** Read arrays nums1 and nums2

**Step 3:** Initialize answer1 = 0 and answer2 = 0

**Step 4:**

For each element x in nums1:

If x exists in nums2, increment answer1 by 1

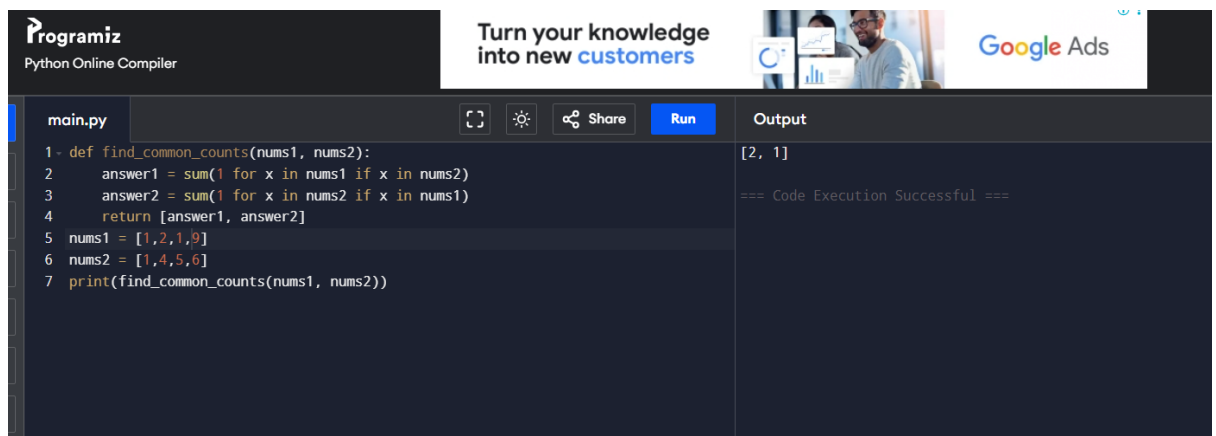
**Step 5:**

For each element x in nums2:

If x exists in nums1, increment answer2 by 1

**Step 6:** Return [answer1, answer2]

**Step 7:** Stop



The screenshot shows a web-based Python IDE. The top bar includes the 'Programiz' logo, the text 'Python Online Compiler', and a Google Ads banner. The main area is divided into a code editor and an output console. The code editor contains the following Python code:

```
1- def find_common_counts(nums1, nums2):
2     answer1 = sum(1 for x in nums1 if x in nums2)
3     answer2 = sum(1 for x in nums2 if x in nums1)
4     return [answer1, answer2]
5 nums1 = [1,2,1,9]
6 nums2 = [1,4,5,6]
7 print(find_common_counts(nums1, nums2))
```

The output console on the right shows the result of the execution: [2, 1]. Below the output, it says '=== Code Execution Successful ==='.

**Result:**

The program has been executed successfully

**3. You are given a 0-indexed integer array nums. The distinct count of a subarray of nums is defined as: Let  $\text{nums}[i..j]$  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  $\text{nums}[i..j]$  is called the distinct count of  $\text{nums}[i..j]$ . 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.**

Aim:

To find the sum of squares of the number of distinct elements in all possible subarrays of a given integer array.

Algorithm:

Step 1: Start

Step 2: Initialize total = 0

Step 3: For each index i from 0 to n-1:

a. Create an empty set seen

b. For each index j from i to n-1:


i. Add  $\text{nums}[j]$  to seen

ii. Find  $\text{count} = \text{len}(\text{seen})$

iii. Add  $\text{count}^2$  to total

Step 4: After all subarrays are processed, return total

Step 5: Stop



```
main.py
1- def sum_of_squares_of_distinct_counts(nums):
2-     n = len(nums)
3-     total = 0
4-     for i in range(n):
5-         seen = set()
6-         for j in range(i, n):
7-             seen.add(nums[j])
8-             count = len(seen)
9-             total += count ** 2
10-    return total
11- nums = [4,5,8]
12- print(sum_of_squares_of_distinct_counts(nums))

Output
310
=== Code Execution Successful ===
```

Result :

The program has been executed successfully

4. 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 `nums[i] == nums[j]` and  $(i * j)$  is divisible by `k`.

Aim:

To count the number of index pairs  $(i, j)$  where `nums[i]` equals `nums[j]` and the product  $(i * j)$  is divisible by a given integer `k`.

Algorithm:

Step 1: Start

Step 2: Initialize `count = 0`

Step 3: Loop `i` from 0 to `n - 1`

    Loop `j` from `i + 1` to `n - 1`

        If both conditions are true:

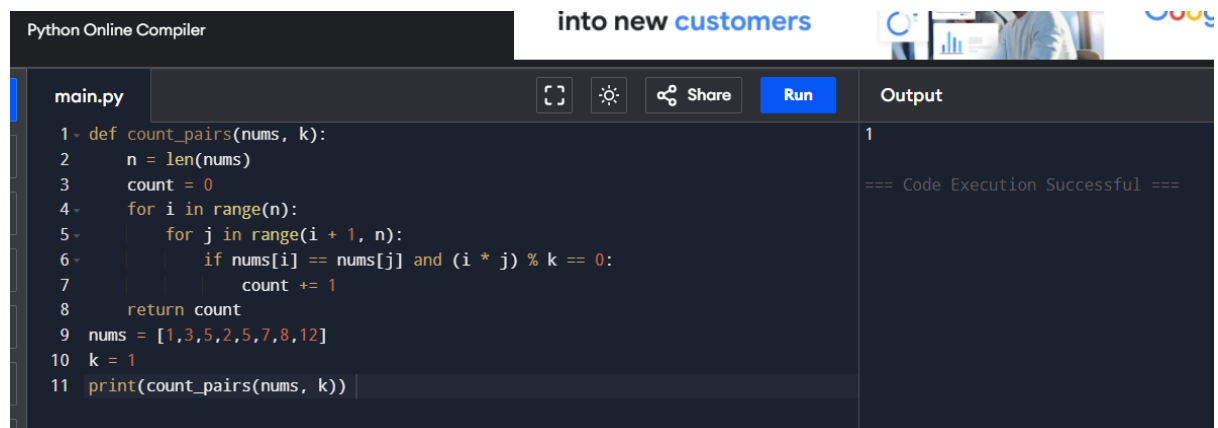
            - `nums[i] == nums[j]`

            -  $(i * j) \% k == 0$

        Then increment `count` by 1

Step 4: After checking all pairs, return `count`

Step 5: Stop



```
Python Online Compiler
into new customers

main.py
1 def count_pairs(nums, k):
2     n = len(nums)
3     count = 0
4     for i in range(n):
5         for j in range(i + 1, n):
6             if nums[i] == nums[j] and (i * j) % k == 0:
7                 count += 1
8     return count
9 nums = [1, 3, 5, 2, 5, 7, 8, 12]
10 k = 1
11 print(count_pairs(nums, k))

Output
1
=== Code Execution Successful ===
```

Result :

The program has been executed successfully

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

Aim:

To write the program for the below test cases with least time complexity.

Algorithm:

Step 1: Start

Step 2: Create an empty dictionary indices to group indices by the values in nums.

Step 3: For each index i and element num in nums:

Add i to the list indices[num].

Step 4: Initialize count = 0.

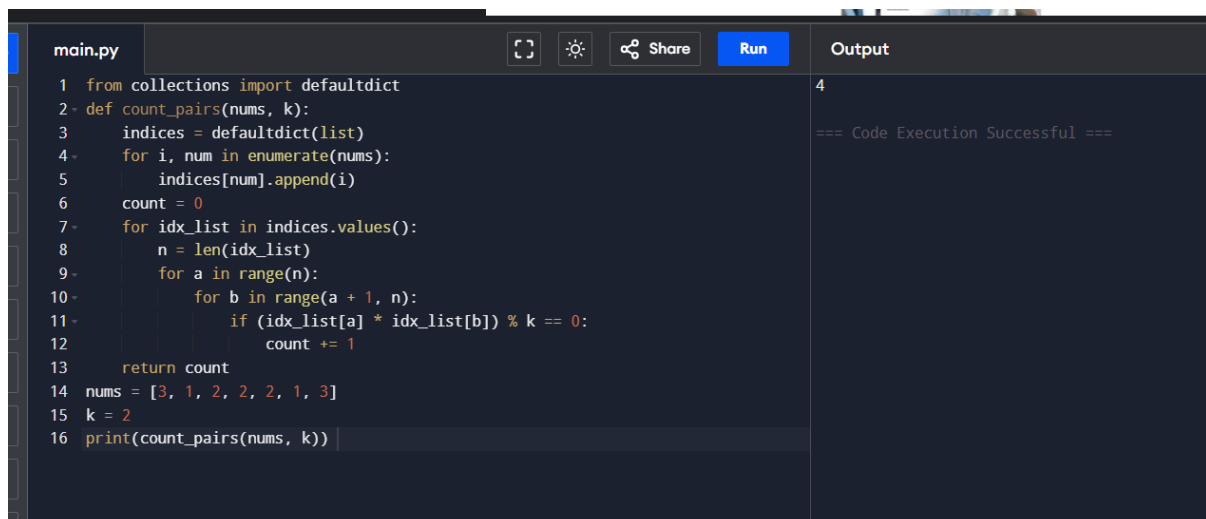
Step 5: For each list of indices idx\_list in the dictionary indices:

For every pair of indices (a, b) in idx\_list where a < b:

If  $(a * b) \% k == 0$ , increment count by 1.

Step 6: After checking all pairs, return count.

Step 7: Stop



The screenshot shows a code editor with a file named 'main.py'. The code defines a function 'count\_pairs(nums, k)' that uses a defaultdict to group indices by value. It then iterates through these groups to count pairs (a, b) such that (a \* b) % k == 0. The test case uses nums = [3, 1, 2, 2, 2, 1, 3] and k = 2. The output panel shows the result '4' and a message '=== Code Execution Successful ==='.

```
1 from collections import defaultdict
2 def count_pairs(nums, k):
3     indices = defaultdict(list)
4     for i, num in enumerate(nums):
5         indices[num].append(i)
6     count = 0
7     for idx_list in indices.values():
8         n = len(idx_list)
9         for a in range(n):
10            for b in range(a + 1, n):
11                if (idx_list[a] * idx_list[b]) % k == 0:
12                    count += 1
13    return count
14 nums = [3, 1, 2, 2, 2, 1, 3]
15 k = 2
16 print(count_pairs(nums, k))
```

4

=== Code Execution Successful ===

Result:

The program has been executed successfully

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.

Aim:

To sort a list of numbers efficiently and then find the maximum element from the sorted list using minimal time complexity.

Algorithms:

Step 1: Start

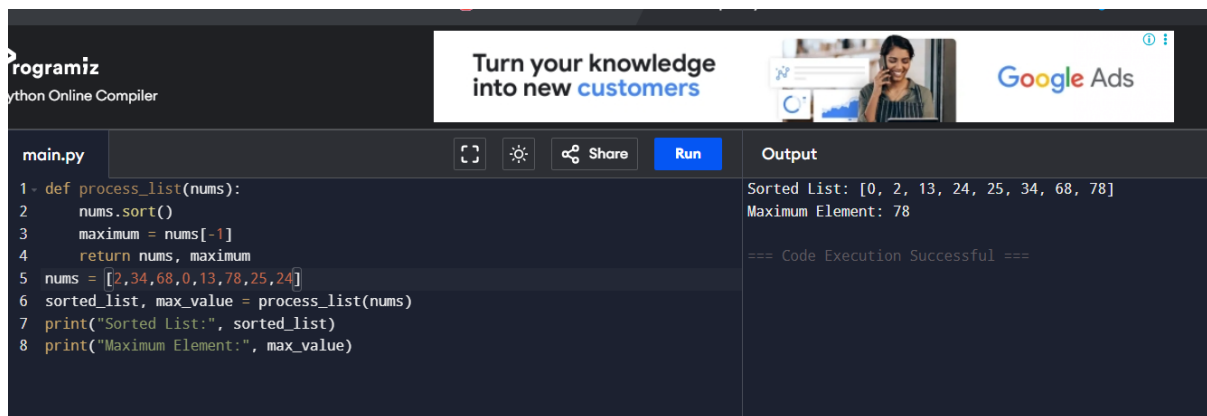
Step 2: Read the list nums

Step 3: Sort the list using an efficient sorting algorithm (Python uses Timsort, which is  $O(n \log n)$ )

Step 4: Retrieve the last element of the sorted list as the maximum value

Step 5: Return the sorted list and the maximum element

Step 6: Stop

The image shows a screenshot of a web-based Python IDE. At the top, there's a header with 'Programiz Python Online Compiler' on the left, a central banner that says 'Turn your knowledge into new customers', and a 'Google Ads' logo on the right. Below the header, the interface is split into two main sections. The left section is a code editor with a dark background, showing a file named 'main.py'. It contains the following Python code:

```
1- def process_list(nums):
2     nums.sort()
3     maximum = nums[-1]
4     return nums, maximum
5
6 nums = [2,34,68,0,13,78,25,24]
7 sorted_list, max_value = process_list(nums)
8 print("Sorted List:", sorted_list)
9 print("Maximum Element:", max_value)
```

There are icons for full screen, settings, share, and a 'Run' button. The right section is titled 'Output' and displays the results of the code execution:

```
Sorted List: [0, 2, 13, 24, 25, 34, 68, 78]
Maximum Element: 78

=== Code Execution Successful ===
```

Result:

The program has been executed successfully

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?

Aim:

To create a new list containing only the unique elements from a given list and analyse its time and space efficiency.

Algorithms:

Step 1: Start

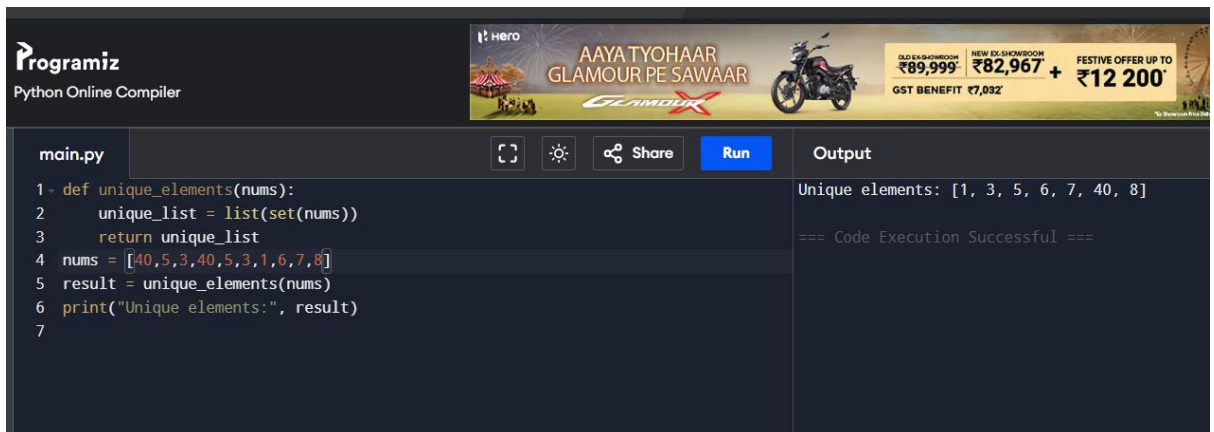
Step 2: Read the input list nums

Step 3: Convert nums to a set to remove duplicates (set(nums))

Step 4: Convert the set back to a list (list(set(nums)))

Step 5: Return the new list of unique elements

Step 6: Stop

The image is a screenshot of the Programiz Python Online Compiler interface. At the top, there is a banner for Hero motorcycles with the text 'AAYA TYOHAAR GLAMOUR PE SAWAAR' and price details. Below the banner, the compiler's logo 'Programiz Python Online Compiler' is visible on the left. The main area is divided into two panels. The left panel, titled 'main.py', contains the following Python code:

```
1 def unique_elements(nums):
2     unique_list = list(set(nums))
3     return unique_list
4 nums = [40,5,3,40,5,3,1,6,7,8]
5 result = unique_elements(nums)
6 print("Unique elements:", result)
7
```

The right panel, titled 'Output', shows the result of the code execution: 'Unique elements: [1, 3, 5, 6, 7, 40, 8]' followed by '=== Code Execution Successful ==='. Above the output panel are buttons for 'Share' and 'Run'.

Result:

The program has been executed successfully

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

Aim:

To sort an array of integers using the Bubble Sort technique and analyse its time complexity in Big-O notation.

Algorithms:

**Step 1:** Start

**Step 2:** Read the input array arr of size n

**Step 3:** Repeat for i from 0 to n-1

a. Set swapped = False

b. For j from 0 to n - i - 2:

If  $\text{arr}[j] > \text{arr}[j + 1]$ , then swap them and set swapped = True

c. If no swaps were made in this pass, stop early (array is sorted)

**Step 4:** Return the sorted array

**Step 5:** Stop



```
main.py  [Icons]  Share  Run  Output
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 - i - 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 nums = [9,8,7,6,5,4,3,2,1]
13 sorted_nums = bubble_sort(nums)
14 print("Sorted Array:", sorted_nums)
15
```

Sorted Array: [1, 2, 3, 4, 5, 6, 7, 8, 9]  
=== Code Execution Successful ===

Result:

The program has been executed successfully



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

Aim:

To determine whether a given number exists in a sorted array using the binary search technique and analyze its time complexity ( $O(\log n)$ )

Algorithms:

Step 1: Start

Step 2: Initialize  $low = 0$ ,  $high = n - 1$

Step 3: Repeat while  $low \leq high$

a. Compute  $mid = (low + high) // 2$

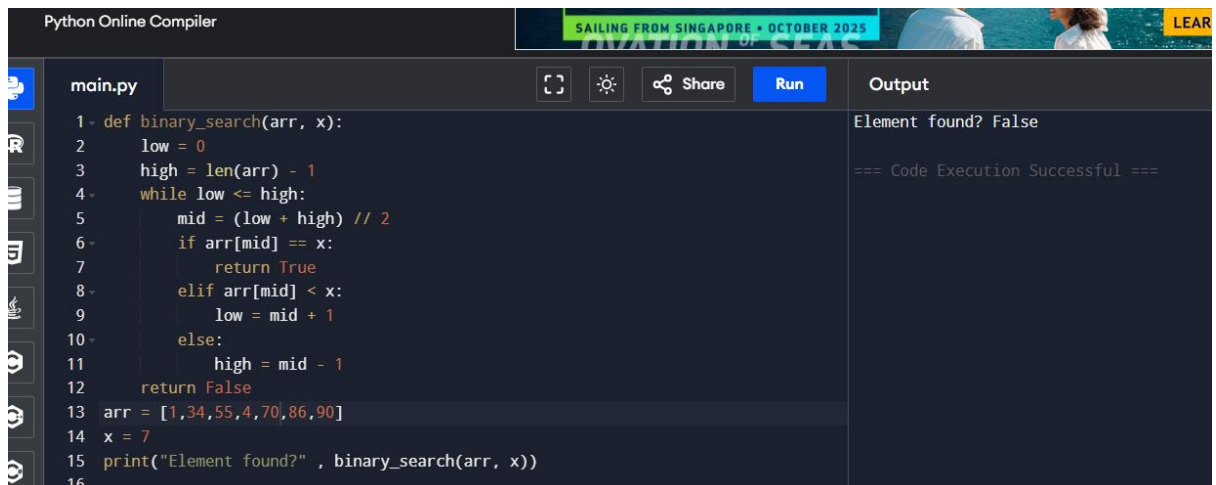
b. If  $arr[mid] == x$ , return True

c. If  $arr[mid] < x$ , set  $low = mid + 1$

d. Else, set  $high = mid - 1$

Step 4: If loop ends, return False (element not found)

Step 5: Stop



The screenshot displays a Python Online Compiler interface. The main editor shows a file named `main.py` containing a binary search function and its execution. The code is as follows:

```
1 def binary_search(arr, x):
2     low = 0
3     high = len(arr) - 1
4     while low <= high:
5         mid = (low + high) // 2
6         if arr[mid] == x:
7             return True
8         elif arr[mid] < x:
9             low = mid + 1
10        else:
11            high = mid - 1
12    return False
13 arr = [1,34,55,4,70,86,90]
14 x = 7
15 print("Element found?" , binary_search(arr, x))
16
```

The output panel on the right shows the result of the execution: "Element found? False" and "=== Code Execution Successful ===".

Result:

The program has been executed successfully

10. 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.

Aim:

To sort an array of integers in ascending order using Heap Sort without built-in functions, achieving  $O(n \log n)$  time complexity and  $O(1)$  space complexity.

Algorithms:

**Step 1:** Start

**Step 2:** Build a **max heap** from the array

**Step 3:** Repeat until the heap size is 1

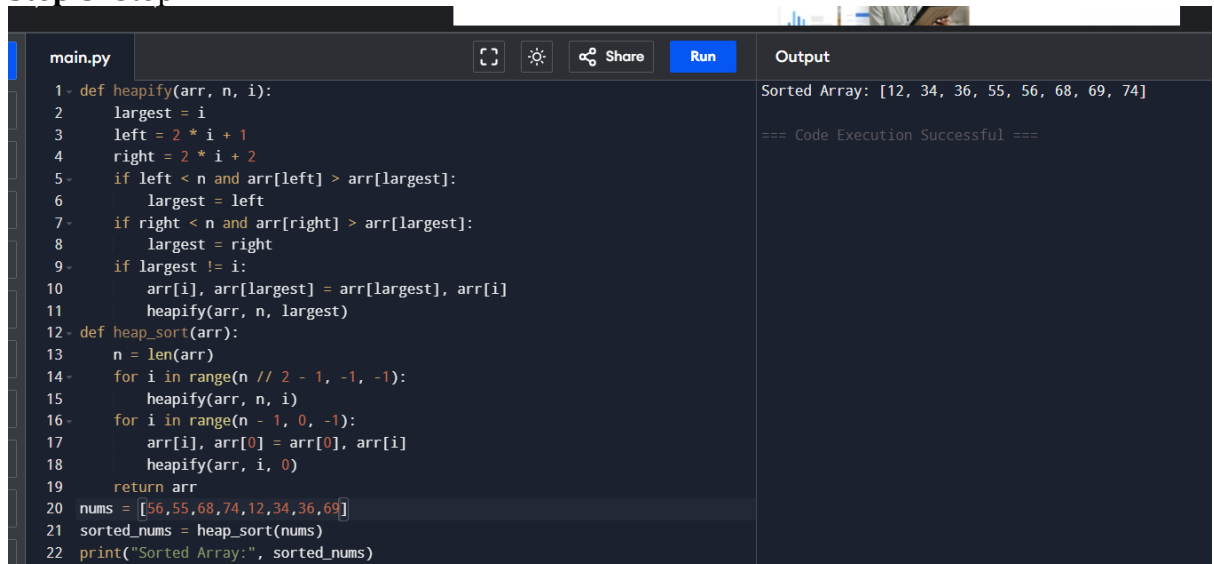
a. Swap the root (maximum element) with the last element

b. Reduce heap size by 1

c. Heapify the root to maintain the heap property

**Step 4:** The array is now sorted in ascending order

**Step 5:** Stop



```
main.py  [Icons]  Run  Output
1 def heapify(arr, n, i):
2     largest = i
3     left = 2 * i + 1
4     right = 2 * i + 2
5     if left < n and arr[left] > arr[largest]:
6         largest = left
7     if right < n and arr[right] > arr[largest]:
8         largest = right
9     if largest != i:
10        arr[i], arr[largest] = arr[largest], arr[i]
11        heapify(arr, n, largest)
12 def heap_sort(arr):
13     n = len(arr)
14     for i in range(n // 2 - 1, -1, -1):
15         heapify(arr, n, i)
16     for i in range(n - 1, 0, -1):
17         arr[i], arr[0] = arr[0], arr[i]
18         heapify(arr, i, 0)
19     return arr
20 nums = [56, 55, 68, 74, 12, 34, 36, 69]
21 sorted_nums = heap_sort(nums)
22 print("Sorted Array:", sorted_nums)
```

Sorted Array: [12, 34, 36, 55, 56, 68, 69, 74]

=== Code Execution Successful ===

Result:

The program has been executed successfully

