Question 1 Given an integer array nums of 2n integers, group these integers into n pairs (a1, b1), (a2, b2),..., (an, bn) such that the sum of min(ai, bi) for all i is maximized. Return the maximized sum.

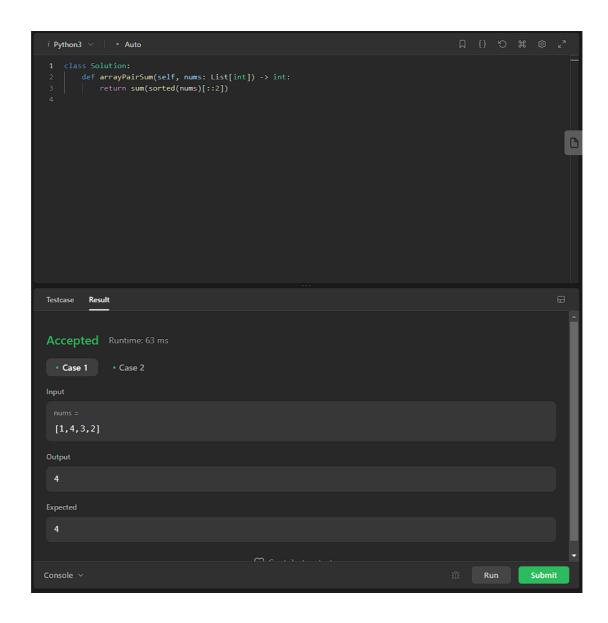**Example 1:**
Input: nums = [1,4,3,2]
Output: 4

`Approach` :

- Sort the input list nums in ascending order using the sorted function.
- Use slicing notation [::2] to select every second element starting from the first element of the sorted list. This creates a new list containing only the elements at even indices.
- Calculate the sum of the selected elements using the sum function.
- Return the sum as the result.

Time complexity: `O(n Log n)` --> the algorithm is dominated by the sorting step, resulting in O(n log n) time complexity
Space Complexity: `O(n)` --> the dominant factor in terms of space complexity is the creation of the sorted list, which requires O(n) additional space.

```
In [ ]: def arrayPairSum(nums):
            return sum(sorted(nums)[::2])
```

```
1   class Solution:
2       def arrayPairSum(self, nums: List[int]) -> int:
3           return sum(sorted(nums)[::2])
4
```

Testcase   **Result**

Accepted   Runtime: 63 ms

• Case 1   • Case 2

Input

nums =
[1,4,3,2]

Output

4

Expected

4

Console ˅   ⚉   Run   **Submit**

Question 2 Alice has n candies, where the ith candy is of type candyType[i]. Alice noticed that she started to gain weight, so she visited a doctor.

The doctor advised Alice to only eat n / 2 of the candies she has (n is always even). Alice likes her candies very much, and she wants to eat the maximum number of different types of candies while still following the doctor's advice.

Given the integer array candyType of length n, return the maximum number of different types of candies she can eat if she only eats n / 2 of them.

Example 1:
Input: candyType = [1,1,2,2,3,3]
Output: 3

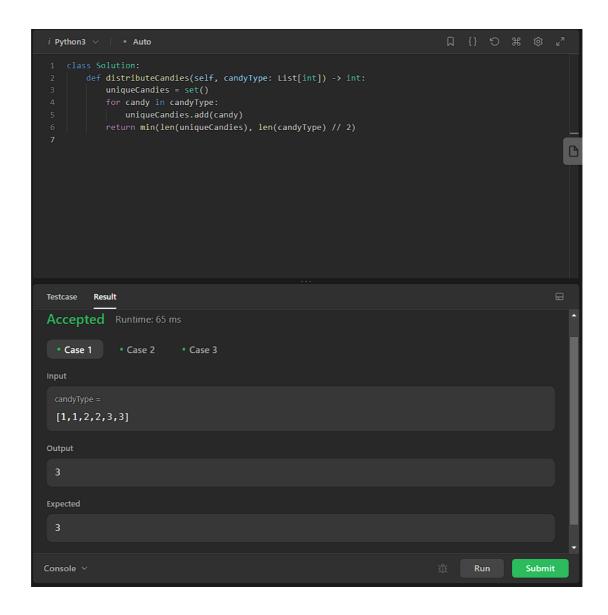*Explanation:* Alice can only eat 6 / 2 = 3 candies. Since there are only 3 types, she can eat one of each type.

`Approach` :

- Initialize an empty set unique_candies to store the unique types of candies.
- Iterate over each candy in the candyType array.
- Add each candy to the unique_candies set.
- After iterating through all the candies, the unique_candies set will contain only the unique types of candies.
- Calculate the minimum between the length of unique_candies and n/2 to get the maximum number of different types of candies Alice can eat.
- Return the minimum as the result.

Time complexity: `O(n)` --> The time complexity of this code is O(n), where n is the length of the candyType array.
Space Complexity: `O(n)` --> The space complexity is O(n), where n is the length of the candyType array.

In [ ]:
```python
def distributeCandies(candyType):
    uniqueCandies = set()
    for candy in candyType:
        uniqueCandies.add(candy)
    return min(len(uniqueCandies), len(candyType) // 2)
```

```
  1  class Solution:
  2      def distributeCandies(self, candyType: List[int]) -> int:
  3          uniqueCandies = set()
  4          for candy in candyType:
  5              uniqueCandies.add(candy)
  6          return min(len(uniqueCandies), len(candyType) // 2)
  7
```

Python3 ∨   • Auto

Testcase   Result

Accepted Runtime: 65 ms

• Case 1   • Case 2   • Case 3

Input

candyType =
[1,1,2,2,3,3]

Output

3

Expected

3

Console ∨   Run   Submit

Question 3 We define a harmonious array as an array where the difference between its maximum value and its minimum value is exactly 1.

Given an integer array nums, return the length of its longest harmonious subsequence among all its possible subsequences.

A subsequence of an array 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 1:
Input: nums = [1,3,2,2,5,2,3,7]
Output: 5

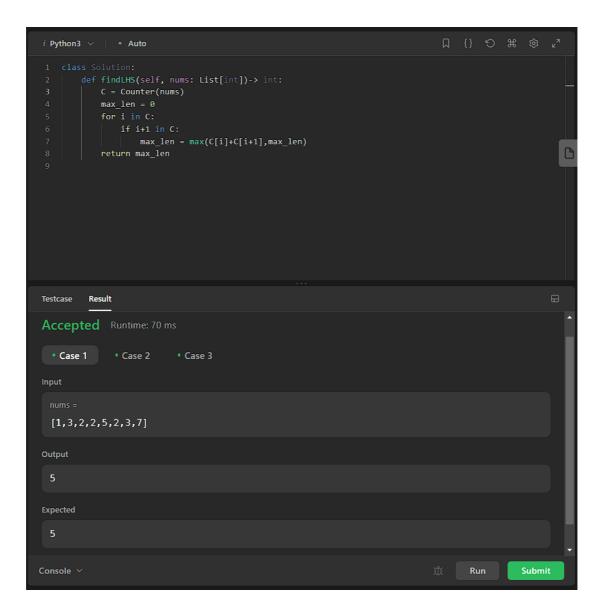*Explanation: The longest harmonious subsequence is [3,2,2,2,3].*

`Approach` :

- The Counter(nums) creates a Counter object C from the nums list, which counts the occurrences of each number in nums.
- The variable mx is initialized to 0, which will keep track of the maximum length of the harmonious subsequence.
- The for loop iterates over the keys in the C counter object.
- For each key i, it checks if i+1 exists in the C counter object.
- If i+1 exists in C, it compares the sum of the counts of i and i+1 with the current mx value and updates mx if necessary.
- After iterating over all keys, mx will contain the maximum length of the harmonious subsequence.
- Finally, the mx value is returned as the result.

Time complexity: `O(n)` --> This implementation is of O(n), where n is the length of the input list nums.
Space Complexity: `O(n)` --> The space complexity is O(n), where n is the length of the input list nums.

In [ ]:
```python
from collections import Counter

def findLHS(nums):
    C = Counter(nums)
    max_len = 0
    for i in C:
        if i+1 in C:
            max_len = max(C[i]+C[i+1],max_len)
    return max_len



# # Method - 2
# In this implementation, we manually create a dictionary called counts to count
# We iterate over the nums list and update the count for each element in the dic

def findLHS(nums):
    counts = {}
    for num in nums:
        if num in counts:
            counts[num] += 1
        else:
            counts[num] = 1

    max_length = 0
    for num in counts:
        if num + 1 in counts:
            length = counts[num] + counts[num + 1]
            max_length = max(max_length, length)

    return max_length
```

`With Method - 1`

```python
class Solution:
    def findLHS(self, nums: List[int])-> int:
        C = Counter(nums)
        max_len = 0
        for i in C:
            if i+1 in C:
                max_len = max(C[i]+C[i+1],max_len)
        return max_len
```

Testcase | Result

Accepted Runtime: 70 ms

• Case 1 • Case 2 • Case 3

Input

nums =
[1,3,2,2,5,2,3,7]

Output

5

Expected

5

Console ⌄

Run Submit

With Method - 2

```python
class Solution:
    def findLHS(self, nums: List[int])-> int:
        counts = {}
        for num in nums:
            if num in counts:
                counts[num] += 1
            else:
                counts[num] = 1

        max_length = 0
        for num in counts:
            if num + 1 in counts:
                length = counts[num] + counts[num + 1]
                max_length = max(max_length, length)

        return max_length
```

Testcase **Result**

Accepted Runtime: 85 ms

• Case 1   • Case 2   • Case 3

Input

nums =
[1,3,2,2,5,2,3,7]

Output

5

Expected

5

Console ∨   Run   Submit

Question 4

You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in adjacent plots. Given an integer array flowerbed containing 0's and 1's, where 0 means empty and 1 means not empty, and an integer n, return true if n new flowers can be planted in the flowerbed without violating the no-adjacent-flowers rule and false otherwise.

Example 1:
Input: flowerbed = [1,0,0,0,1], n = 1
Output: true

`Approach` :

- Initialize a variable count to 0 to keep track of the planted flowers count.
- Iterate over the flowerbed array from index 0 to the last index.
- Check if the current position is 0 and both its adjacent positions (if they exist) are also 0.
- If the conditions are met, increment count by 1 and update the current position to 1 to mark it as planted.
- After iterating over the flowerbed array, check if the count is equal to the target n. If it is, return true.
- If we haven't planted enough flowers, return false.

Time complexity: `O(n)` --> The time complexity of the function is O(n), where n is the length of the flowerbed array.

Space complexity: `O(1)` --> The space complexity of the function is O(1), as it uses a constant amount of additional space.

In [ ]:
```python
def canPlaceFlowers(flowerbed, n):
    count = 0
    length = len(flowerbed)
    for i in range(length):
        if flowerbed[i] == 0 and (i == 0 or flowerbed[i-1] == 0) and (i == lengt
            count += 1
            flowerbed[i] = 1
        if count >= n:
            return True
    return False
```

i Python3 ∨   • Auto

```python
1   class Solution:
2       def canPlaceFlowers(self, flowerbed: List[int], n: int) -> bool:
3           count = 0
4           length = len(flowerbed)
5           for i in range(length):
6               if flowerbed[i] == 0 and (i == 0 or flowerbed[i-1] == 0) and (i == length-1 or flowerbed[i+1] == 0):
7                   count += 1
8                   flowerbed[i] = 1
9               if count >= n:
10                  return True
11          return False
12
```

Testcase   **Result**

Accepted   Runtime: 47 ms

• Case 1   • Case 2

Input

flowerbed =
[1,0,0,0,1]

n =
1

Output

true

Expected

true

Console ∨   Run   Submit

Question 5

Given an integer array nums, find three numbers whose product is maximum and return the maximum product.
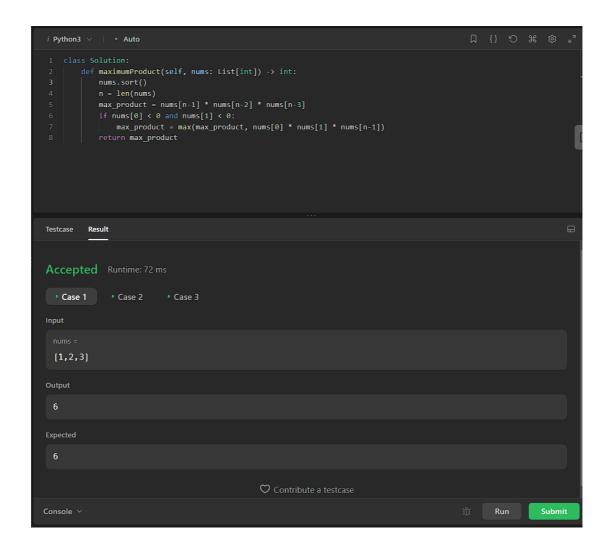
Example 1:

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

`Approach` :

- Sort the nums array in ascending order.
- Calculate the product of the first two elements and the last element, and store it as max_product.
- If the last element is negative, consider the possibility of multiplying two negative numbers. Calculate the product of the first two elements and the last element
- (instead of the last element as calculated in the previous step), and update max_product if this product is greater.
- Return max_product as the result.

Time complexity: `O(n Log n)` --> This is because the function sorts the array using the sort method, which has a time complexity of O(n log n).
Space complexity: `O(1)` --> The space complexity of the function is O(1), as it uses a constant amount of additional space.

```python
In [ ]: def maximumProduct(nums):
    nums.sort()
    n = len(nums)
    max_product = nums[n-1] * nums[n-2] * nums[n-3]
    if nums[0] < 0 and nums[1] < 0:
        max_product = max(max_product, nums[0] * nums[1] * nums[n-1])
    return max_product
```

```
1   class Solution:
2       def maximumProduct(self, nums: List[int]) -> int:
3           nums.sort()
4           n = len(nums)
5           max_product = nums[n-1] * nums[n-2] * nums[n-3]
6           if nums[0] < 0 and nums[1] < 0:
7               max_product = max(max_product, nums[0] * nums[1] * nums[n-1])
8           return max_product
```

Testcase   **Result**

Accepted   Runtime: 72 ms

• Case 1   • Case 2   • Case 3

Input

```
nums =
[1,2,3]
```

Output

```
6
```

Expected

```
6
```

♡ Contribute a testcase

Console ∨   Run   **Submit**

Question 6

Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.

You must write an algorithm with O(log n) runtime complexity.

Input: nums = [-1,0,3,5,9,12], target = 9
Output: 4

Explanation: 9 exists in nums and its index is 4

`Approach` :

- Initialize two pointers, left and right, to the start and end indices of the nums array, respectively.
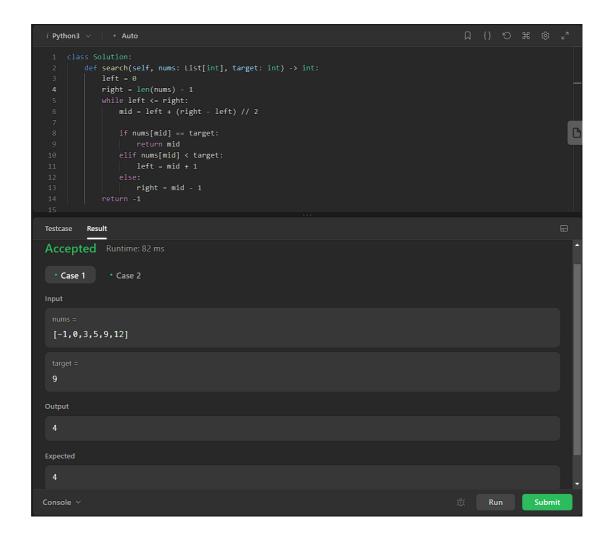
- While left is less than or equal to right, perform the following steps:
  - Calculate the middle index as mid using the formula: mid = left + (right - left) // 2.
  - If the element at mid is equal to the target, return mid as the index of the target.
  - If the element at mid is greater than the target, update right to mid - 1 to search in the left half of the array.
  - If the element at mid is less than the target, update left to mid + 1 to search in the right half of the array.
- If the target is not found after the entire binary search process, return -1.

Time complexity: `O(log n)` --> The time complexity of the function using binary search is O(log n), where n is the length of the input array nums.

Space complexity: `O(1)` --> The space complexity of the function is O(1), as it uses a constant amount of additional space.

```python
In [ ]: def search(nums, target):
    left = 0
    right = len(nums) - 1

    while left <= right:
        mid = left + (right - left) // 2

        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1
```

```python
1   class Solution:
2       def search(self, nums: List[int], target: int) -> int:
3           left = 0
4           right = len(nums) - 1
5           while left <= right:
6               mid = left + (right - left) // 2
7
8               if nums[mid] == target:
9                   return mid
10              elif nums[mid] < target:
11                  left = mid + 1
12              else:
13                  right = mid - 1
14          return -1
15
```

Testcase   Result

Accepted   Runtime: 82 ms

• Case 1   • Case 2

Input

nums =
[-1,0,3,5,9,12]

target =
9

Output

4

Expected

4

Console ∨   Run   Submit

Question 7 An array is monotonic if it is either monotone increasing or monotone decreasing.

An array nums is monotone increasing if for all i <= j, nums[i] <= nums[j]. An array nums is monotone decreasing if for all i <= j, nums[i] >= nums[j].

Given an integer array nums, return true if the given array is monotonic, or false otherwise.
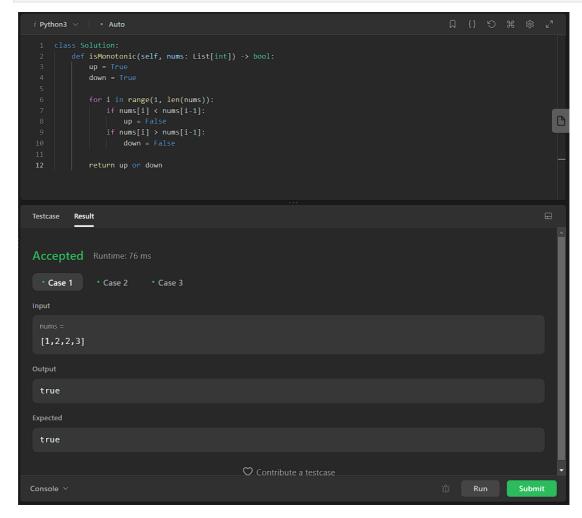
Example 1:
Input: nums = [1,2,2,3]
Output: true

`Approach` :

- Initialize two boolean variables, isIncreasing and isDecreasing, as True.

- Iterate over the nums array from index 1 to the last index.
- If the current element is less than the previous element, set isIncreasing to False.
- If the current element is greater than the previous element, set isDecreasing to False.
- After iterating over the array, if either isIncreasing or isDecreasing is True, return True (as the array is monotonic).
- If both isIncreasing and isDecreasing are False, return False (as the array is neither monotone increasing nor monotone decreasing).

Time complexity: `O(n)` --> The time complexity of the isMonotonic function is O(n), where n is the length of the input array nums.

Space complexity: `O(1)` --> The space complexity of the function is O(1), as it uses a constant amount of additional space.

In [ ]:
```python
def isMonotonic(nums):
    up = True
    down = True

    for i in range(1, len(nums)):
        if nums[i] < nums[i-1]:
            up = False
        if nums[i] > nums[i-1]:
            down = False

    return up or down
```

```python
i Python3 ∨    • Auto

1    class Solution:
2        def isMonotonic(self, nums: List[int]) -> bool:
3            up = True
4            down = True
5
6            for i in range(1, len(nums)):
7                if nums[i] < nums[i-1]:
8                    up = False
9                if nums[i] > nums[i-1]:
10                   down = False
11
12           return up or down
```

Testcase   Result

Accepted   Runtime: 76 ms

• Case 1   • Case 2   • Case 3

Input

nums =
[1,2,2,3]

Output

true

Expected

true

♡ Contribute a testcase

Console ∨   Run   Submit

Question 8

You are given an integer array nums and an integer k.

In one operation, you can choose any index i where 0 <= i < nums.length and change nums[i] to nums[i] + x where x is an integer from the range [-k, k]. You can apply this operation at most once for each index i.

The score of nums is the difference between the maximum and minimum elements in nums.

Return the minimum score of nums after applying the mentioned operation at most once for each index in it.

Example 1:

Input: nums = [1], k = 0 *Output:* 0

`Explanation` : The score is max(nums) - min(nums) = 1 - 1 = 0.

`Approach` :

- Find the maximum and minimum value in the list A using the max() and min() function..
- Calculate the difference between the maximum and minimum values of A.
- Subtract twice the value of K from the difference.
- Take the maximum between 0 and the result obtained in the previous step.
- Return the maximum value as the result.

Time complexity: `O(n)` --> The max() and min() functions each have a time complexity of O(n) because they need to compare each element in the list.
Space complexity: `O(1)` --> The space complexity of the function is O(1), as it uses a constant amount of additional space.

```
In [ ]: def smallestRangeI(A, K):
            return max(0, max(A) - min(A) - 2*K)
```

```
1  class Solution:
2      def smallestRangeI(self, A: List[int], K: int) -> int:
3          return max(0, max(A) - min(A) - 2*K)
```

Testcase   **Result**

Accepted   Runtime: 45 ms

• Case 1   • Case 2   • Case 3

Input

nums =
[1]

k =
0

Output

0

Expected

Console ⌄   Run   Submit