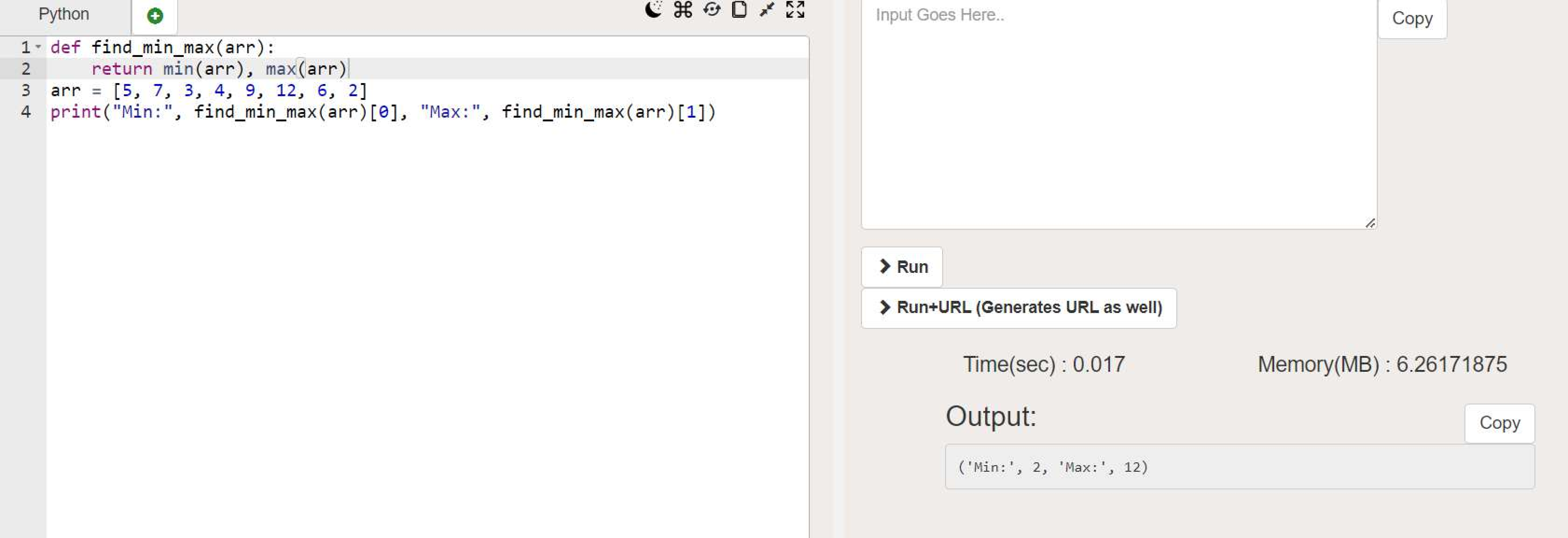


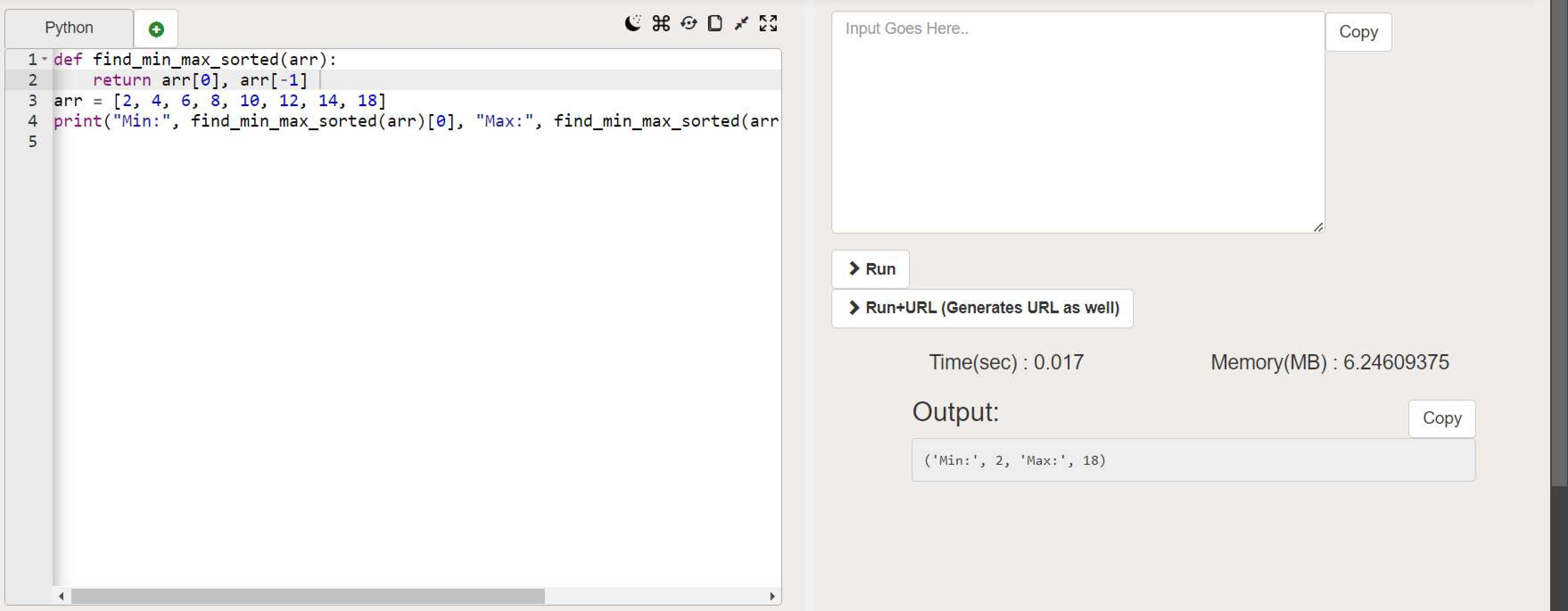
```python
def find_min_max(arr):
    return min(arr), max(arr)
arr = [5, 7, 3, 4, 9, 12, 6, 2]
print("Min:", find_min_max(arr)[0], "Max:", find_min_max(arr)[1])
```

Time(sec) : 0.017

Memory(MB) : 6.26171875

Output:

```
('Min:', 2, 'Max:', 12)
```



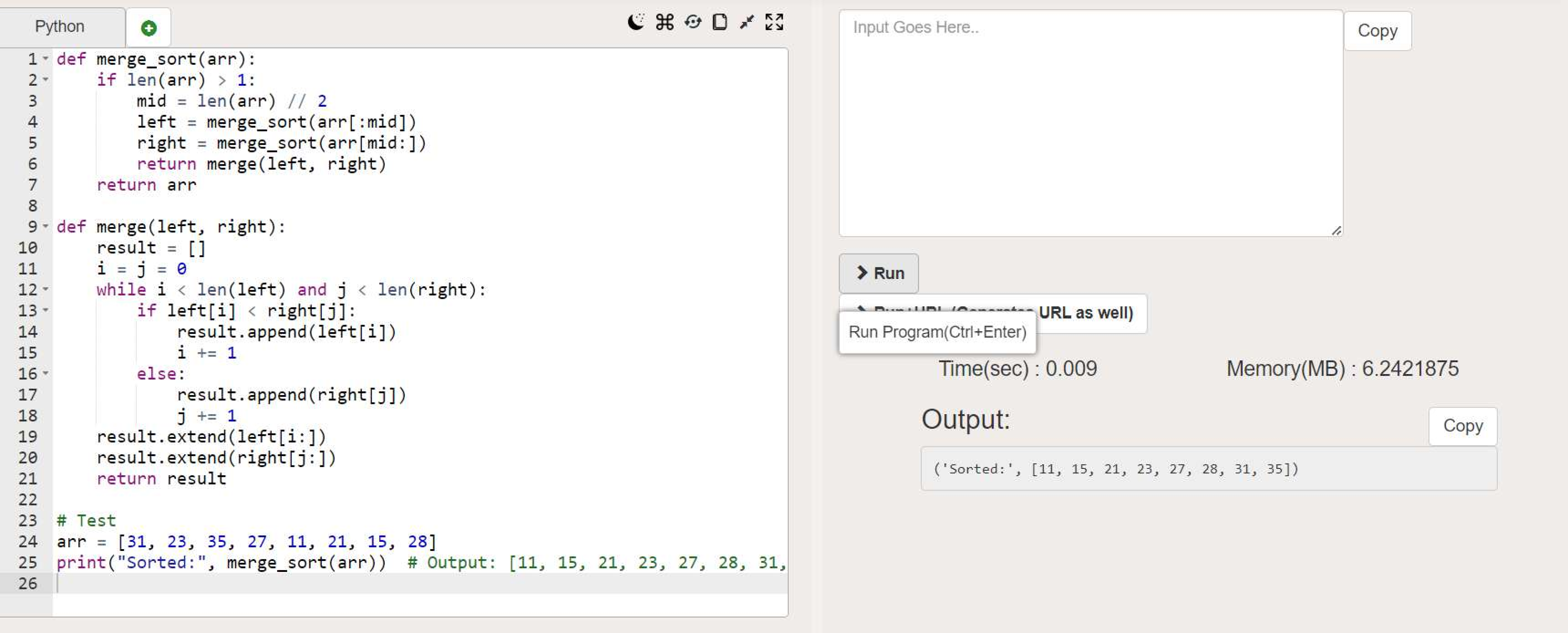
```python
def find_min_max_sorted(arr):
    return arr[0], arr[-1]
arr = [2, 4, 6, 8, 10, 12, 14, 18]
print("Min:", find_min_max_sorted(arr)[0], "Max:", find_min_max_sorted(arr
```

Time(sec) : 0.017

Memory(MB) : 6.24609375

Output:

```
('Min:', 2, 'Max:', 18)
```

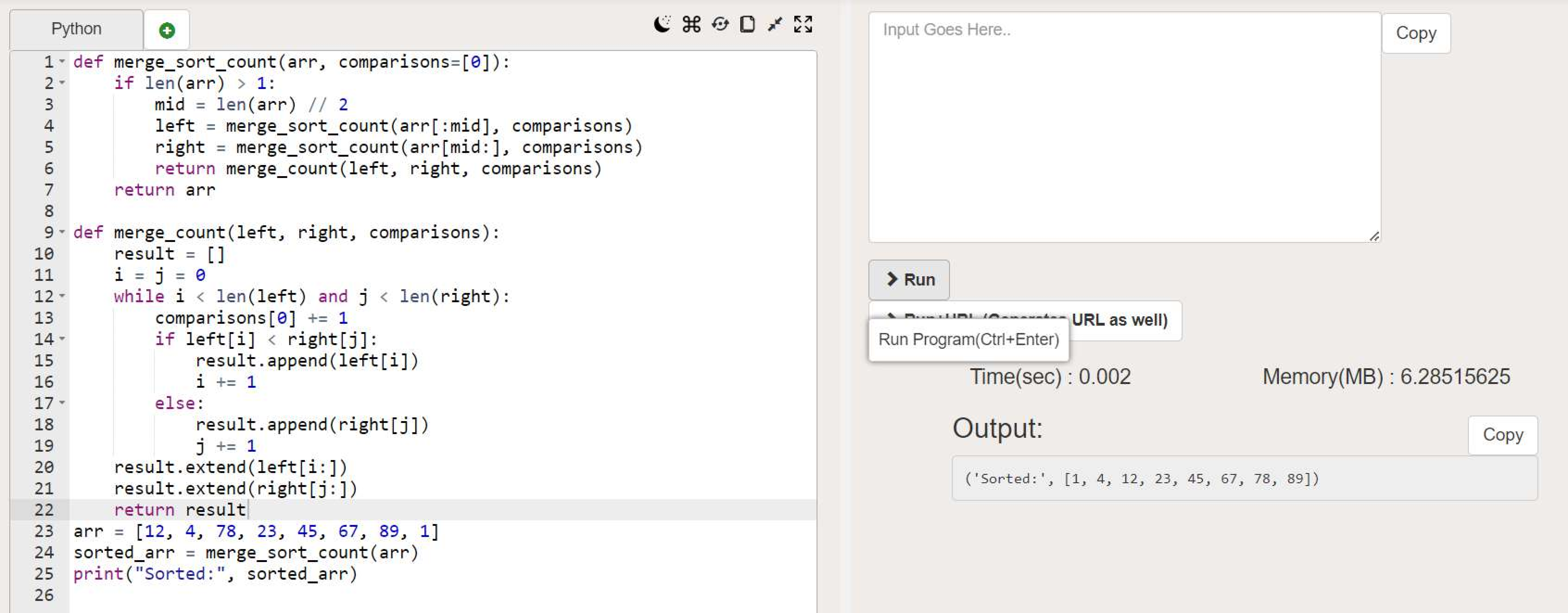


```python
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left = merge_sort(arr[:mid])
        right = merge_sort(arr[mid:])
        return merge(left, right)
    return arr

def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result

# Test
arr = [31, 23, 35, 27, 11, 21, 15, 28]
print("Sorted:", merge_sort(arr))  # Output: [11, 15, 21, 23, 27, 28, 31,

```

Time(sec) : 0.009 Memory(MB) : 6.2421875

Output:

```
('Sorted:', [11, 15, 21, 23, 27, 28, 31, 35])
```

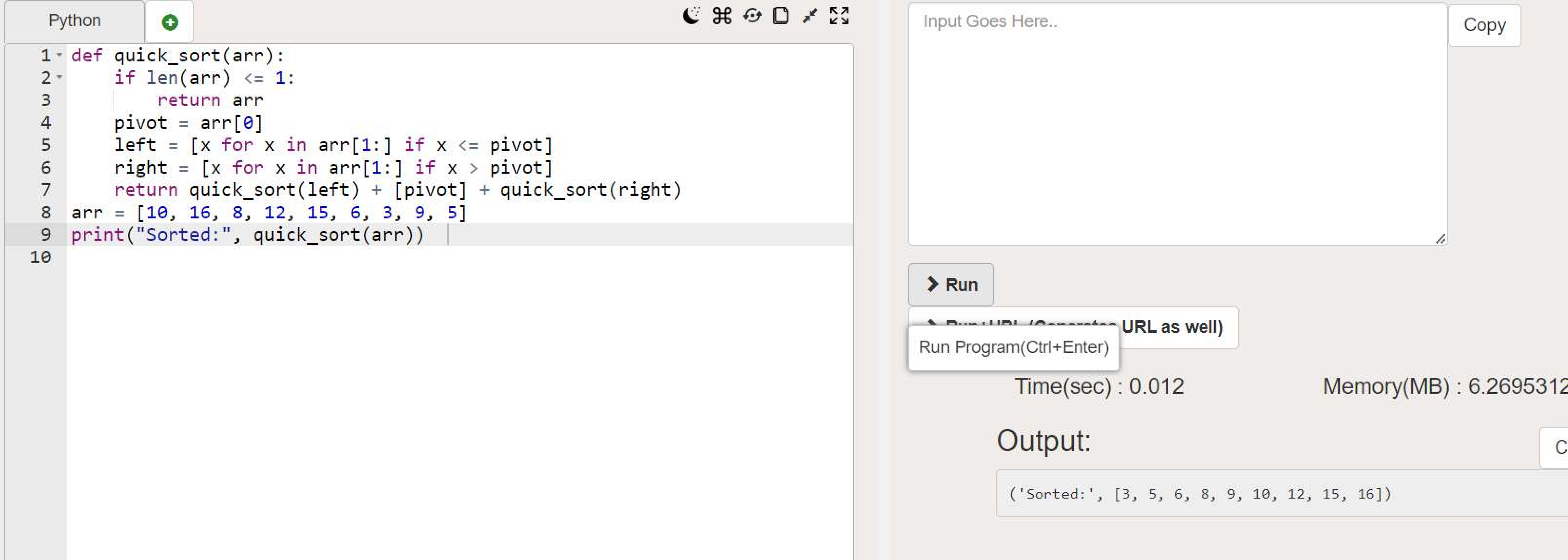


```python
def merge_sort_count(arr, comparisons=[0]):
    if len(arr) > 1:
        mid = len(arr) // 2
        left = merge_sort_count(arr[:mid], comparisons)
        right = merge_sort_count(arr[mid:], comparisons)
        return merge_count(left, right, comparisons)
    return arr

def merge_count(left, right, comparisons):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        comparisons[0] += 1
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result
arr = [12, 4, 78, 23, 45, 67, 89, 1]
sorted_arr = merge_sort_count(arr)
print("Sorted:", sorted_arr)
```

Time(sec) : 0.002    Memory(MB) : 6.28515625

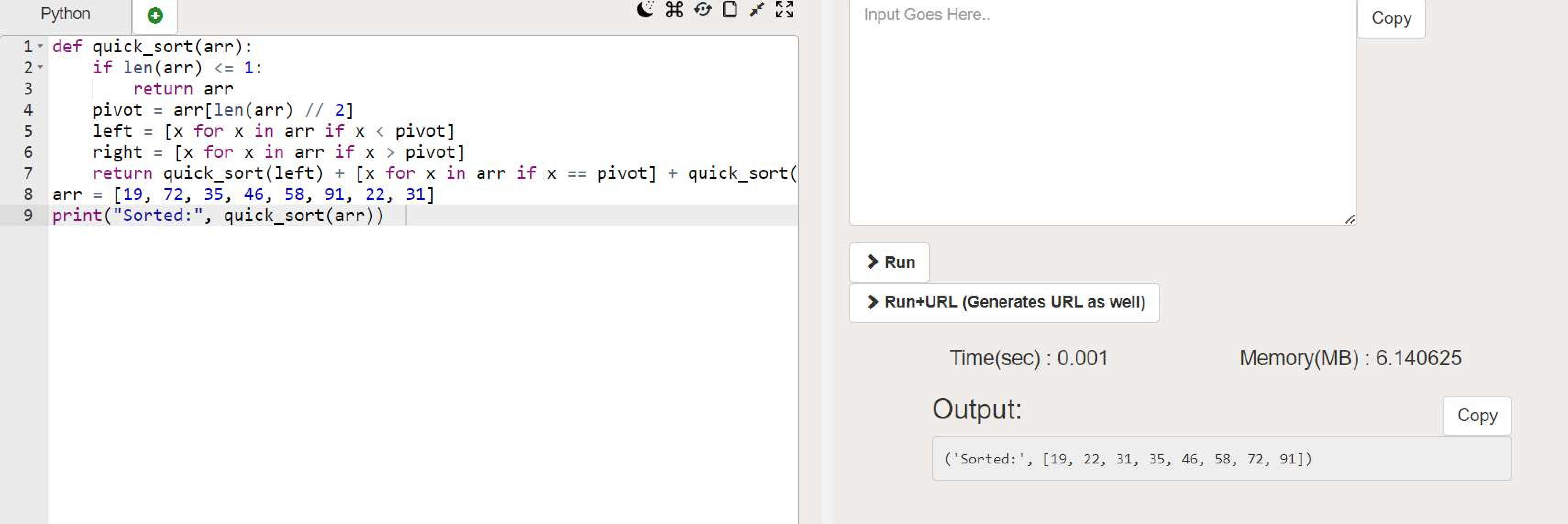
Output:

```
('Sorted:', [1, 4, 12, 23, 45, 67, 78, 89])
```



```python
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[0]
    left = [x for x in arr[1:] if x <= pivot]
    right = [x for x in arr[1:] if x > pivot]
    return quick_sort(left) + [pivot] + quick_sort(right)
arr = [10, 16, 8, 12, 15, 6, 3, 9, 5]
print("Sorted:", quick_sort(arr))
```

```
('Sorted:', [3, 5, 6, 8, 9, 10, 12, 15, 16])
```



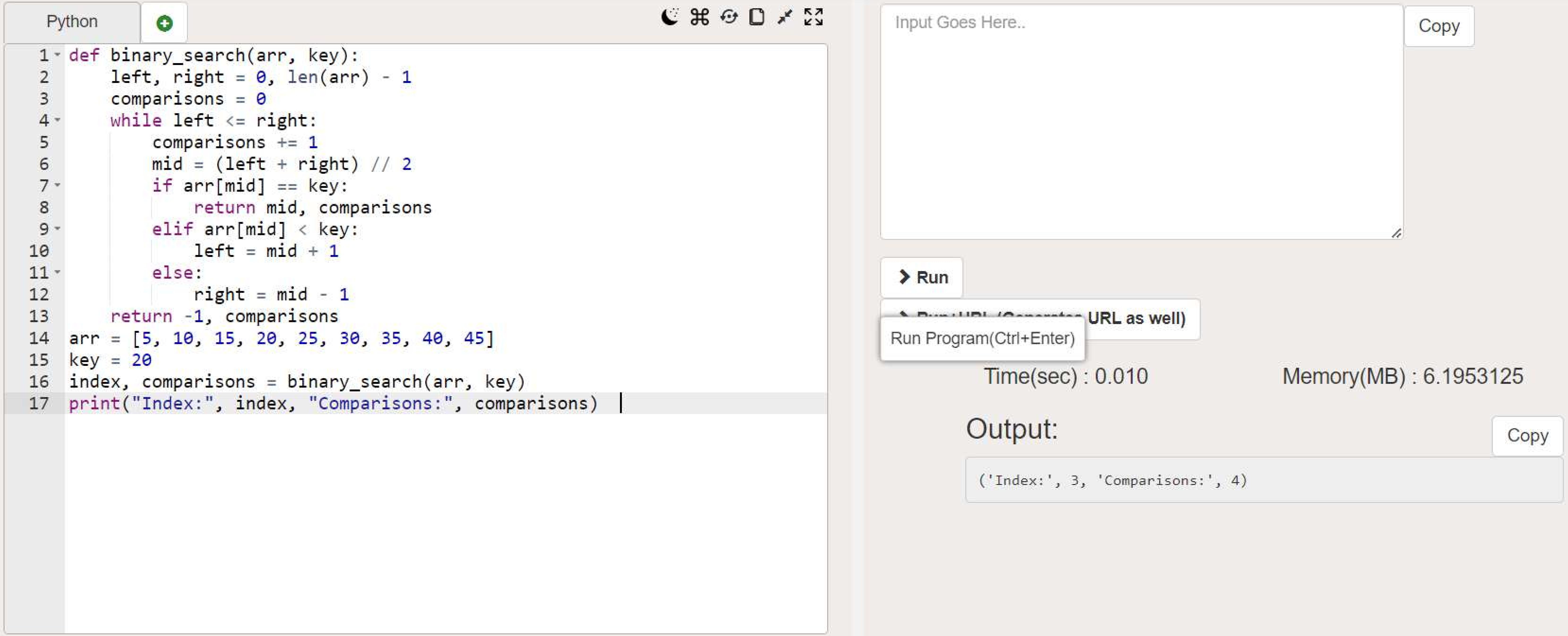
```python
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + [x for x in arr if x == pivot] + quick_sort(
arr = [19, 72, 35, 46, 58, 91, 22, 31]
print("Sorted:", quick_sort(arr))
```

Time(sec) : 0.001

Memory(MB) : 6.140625

Output:

```
('Sorted:', [19, 22, 31, 35, 46, 58, 72, 91])
```

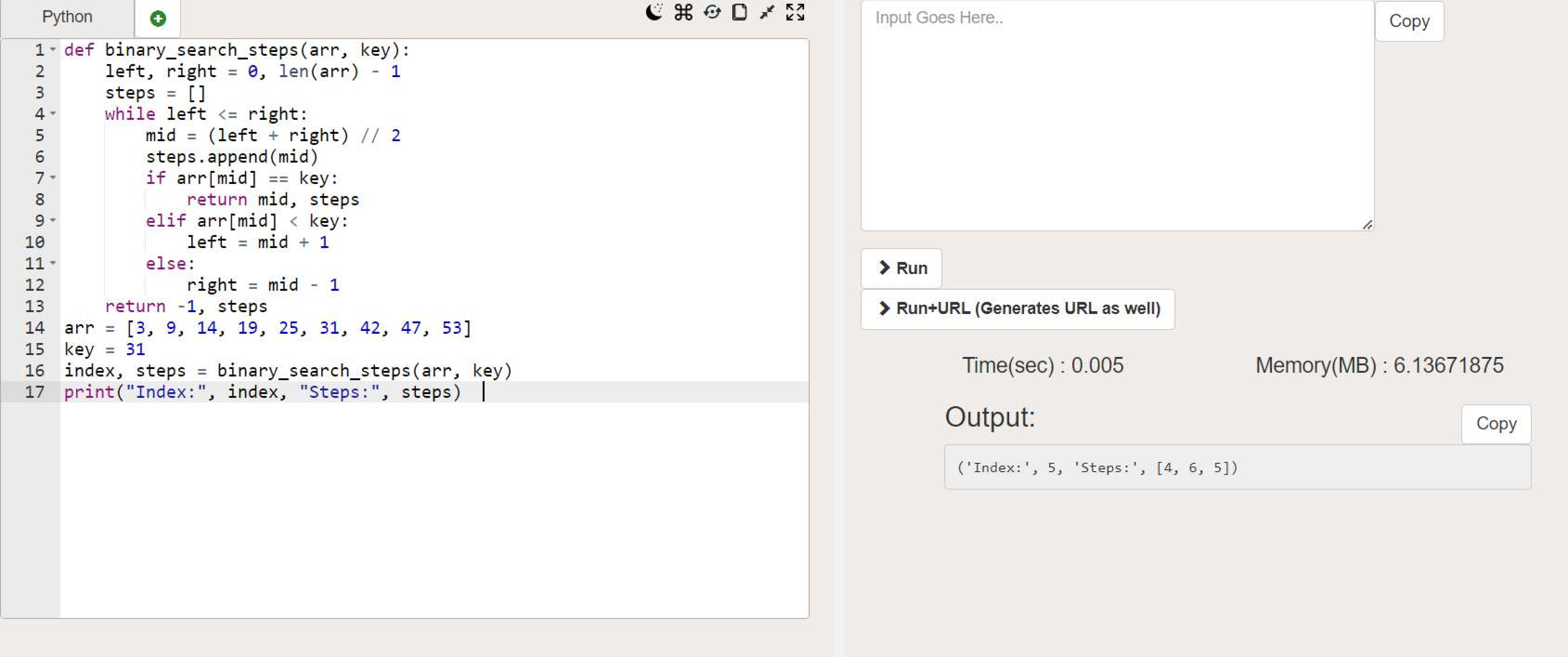


```python
def binary_search(arr, key):
    left, right = 0, len(arr) - 1
    comparisons = 0
    while left <= right:
        comparisons += 1
        mid = (left + right) // 2
        if arr[mid] == key:
            return mid, comparisons
        elif arr[mid] < key:
            left = mid + 1
        else:
            right = mid - 1
    return -1, comparisons
arr = [5, 10, 15, 20, 25, 30, 35, 40, 45]
key = 20
index, comparisons = binary_search(arr, key)
print("Index:", index, "Comparisons:", comparisons)
```

Time(sec) : 0.010    Memory(MB) : 6.1953125

Output:

```
('Index:', 3, 'Comparisons:', 4)
```

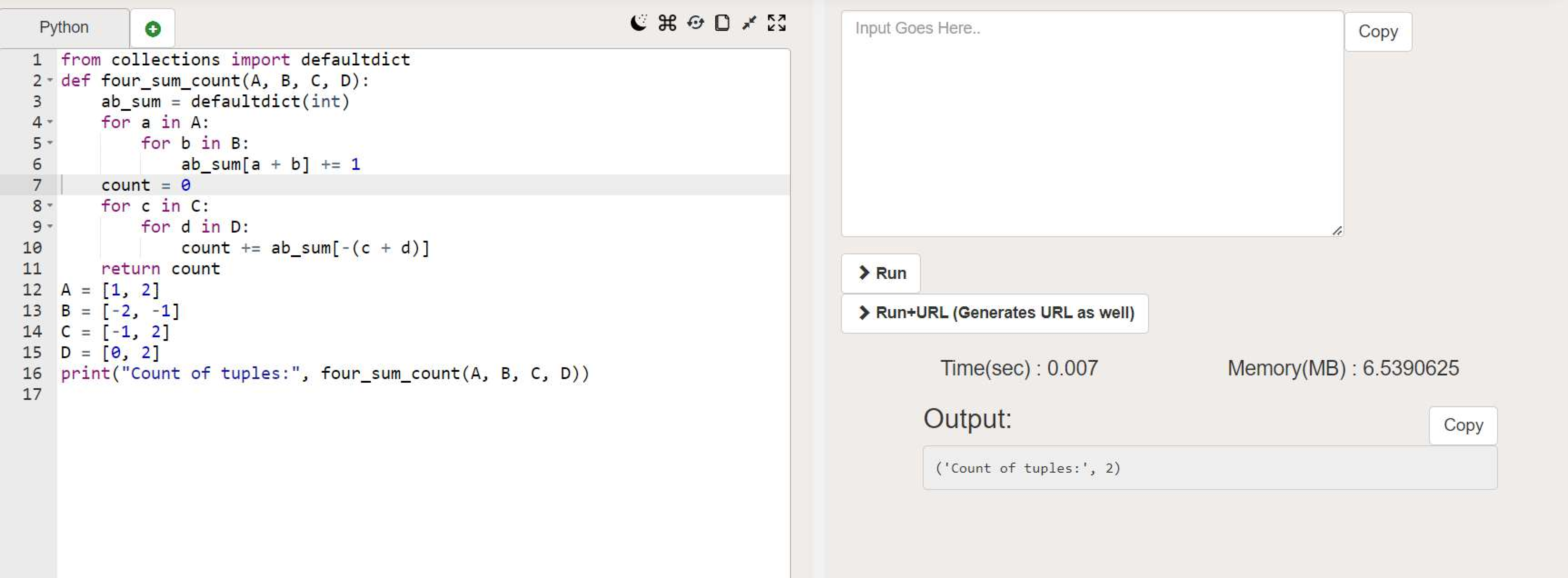


```python
def binary_search_steps(arr, key):
    left, right = 0, len(arr) - 1
    steps = []
    while left <= right:
        mid = (left + right) // 2
        steps.append(mid)
        if arr[mid] == key:
            return mid, steps
        elif arr[mid] < key:
            left = mid + 1
        else:
            right = mid - 1
    return -1, steps
arr = [3, 9, 14, 19, 25, 31, 42, 47, 53]
key = 31
index, steps = binary_search_steps(arr, key)
print("Index:", index, "Steps:", steps)
```

Time(sec) : 0.005    Memory(MB) : 6.13671875

Output:

```
('Index:', 5, 'Steps:', [4, 6, 5])
```



```python
from collections import defaultdict
def four_sum_count(A, B, C, D):
    ab_sum = defaultdict(int)
    for a in A:
        for b in B:
            ab_sum[a + b] += 1
    count = 0
    for c in C:
        for d in D:
            count += ab_sum[-(c + d)]
    return count
A = [1, 2]
B = [-2, -1]
C = [-1, 2]
D = [0, 2]
print("Count of tuples:", four_sum_count(A, B, C, D))
```

Time(sec) : 0.007    Memory(MB) : 6.5390625

Output:

```
('Count of tuples:', 2)
```