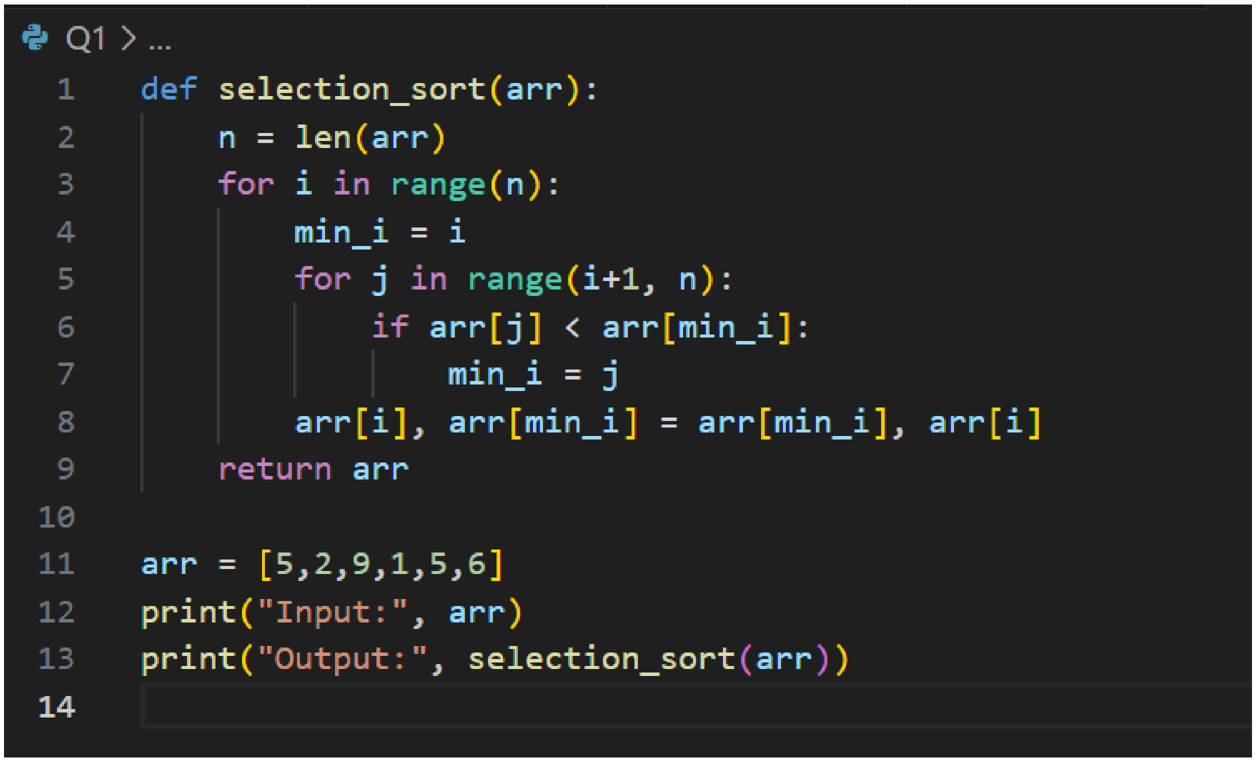
TOPIC 2 - BRUTE FORCE

Q1. Selection Sort Implementation

Aim:
To sort an array in ascending order using the selection sort algorithm.
Algorithm:

1. Start from index 0.
2. Find the smallest element in the unsorted portion.
3. Swap it with the first unsorted element.
4. Move the boundary rightwards.
5. Repeat until fully sorted.

CODE:



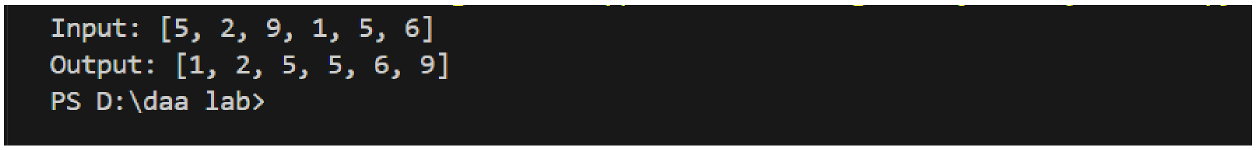
Q1 > ...

```python
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_i = i
        for j in range(i+1, n):
            if arr[j] < arr[min_i]:
                min_i = j
        arr[i], arr[min_i] = arr[min_i], arr[i]
    return arr

arr = [5,2,9,1,5,6]
print("Input:", arr)
print("Output:", selection_sort(arr))

```

Input & Output



```
Input: [5, 2, 9, 1, 5, 6]
Output: [1, 2, 5, 5, 6, 9]
PS D:\daa lab>
```

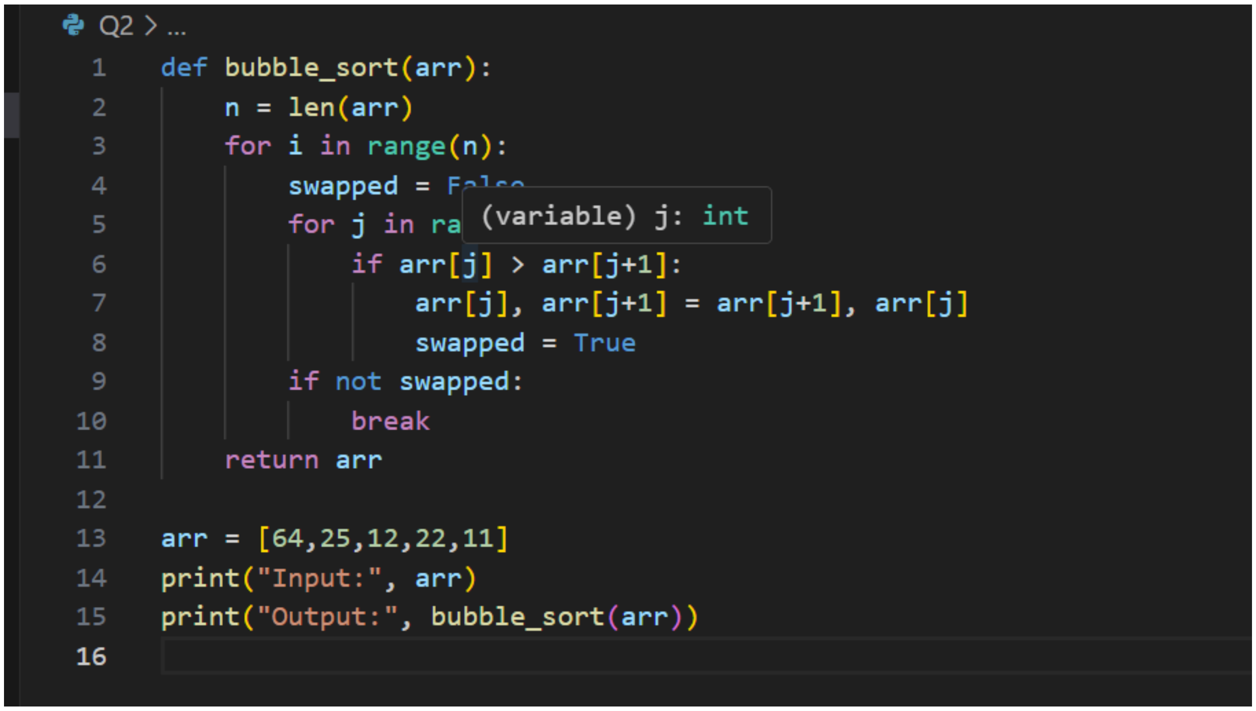
Result: Array sorted successfully using brute force selection sort.

Q2. Bubble Sort Optimization

Aim:
To implement bubble sort and stop early when the array becomes sorted.
Algorithm:

1. Traverse the array multiple times.
2. Swap adjacent elements if out of order.
3. Track if a swap occurred in the pass.
4. Stop if no swaps occurred.
5. Print sorted array.

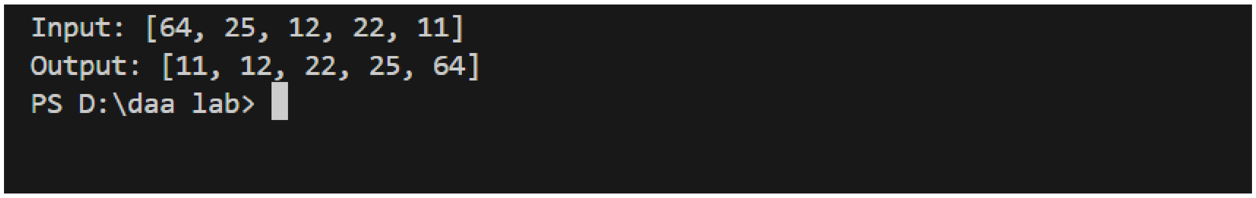
CODE:



```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in ra (variable) j: int
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr

arr = [64,25,12,22,11]
print("Input:", arr)
print("Output:", bubble_sort(arr))
```

Input & Output:



```
Input: [64, 25, 12, 22, 11]
Output: [11, 12, 22, 25, 64]
PS D:\daa lab>
```

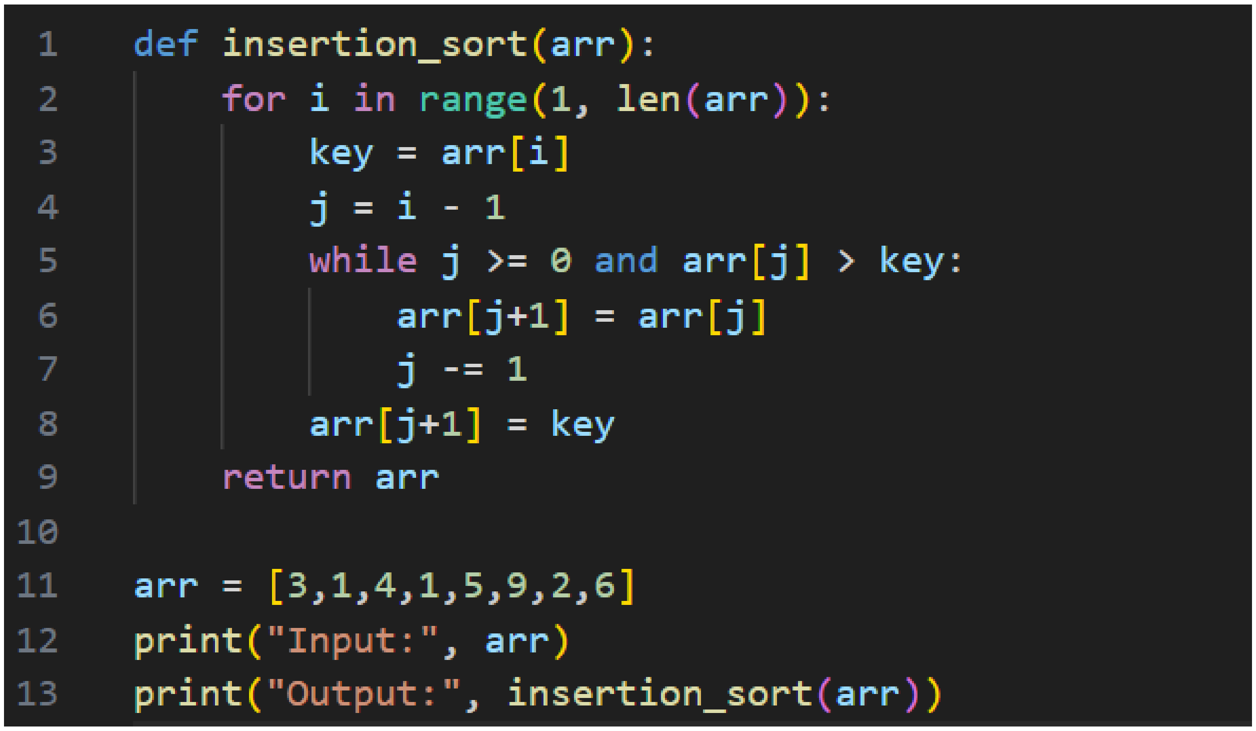
Result: Optimized bubble sort executes fewer passes.

Q3. Insertion Sort with Duplicates

Aim:
To sort an array with duplicate values while maintaining relative order.
Algorithm:

1. Iterate from index 1 to n − 1.
2. Store current value in `key`.
3. Compare `key` with previous elements.
4. Shift elements greater than key.
5. Insert `key` in its correct position.

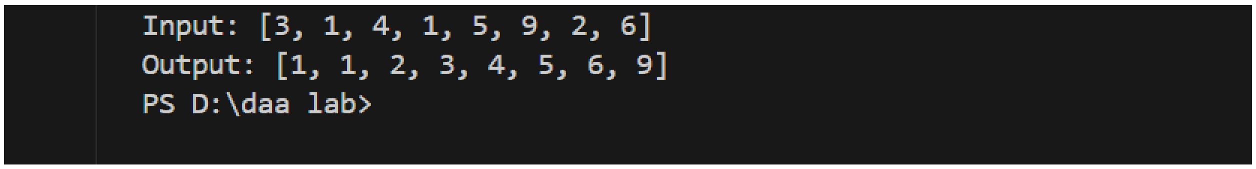
CODE:



```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key
    return arr

arr = [3,1,4,1,5,9,2,6]
print("Input:", arr)
print("Output:", insertion_sort(arr))
```

Input & Output:



```
Input: [3, 1, 4, 1, 5, 9, 2, 6]
Output: [1, 1, 2, 3, 4, 5, 6, 9]
PS D:\daa lab>
```

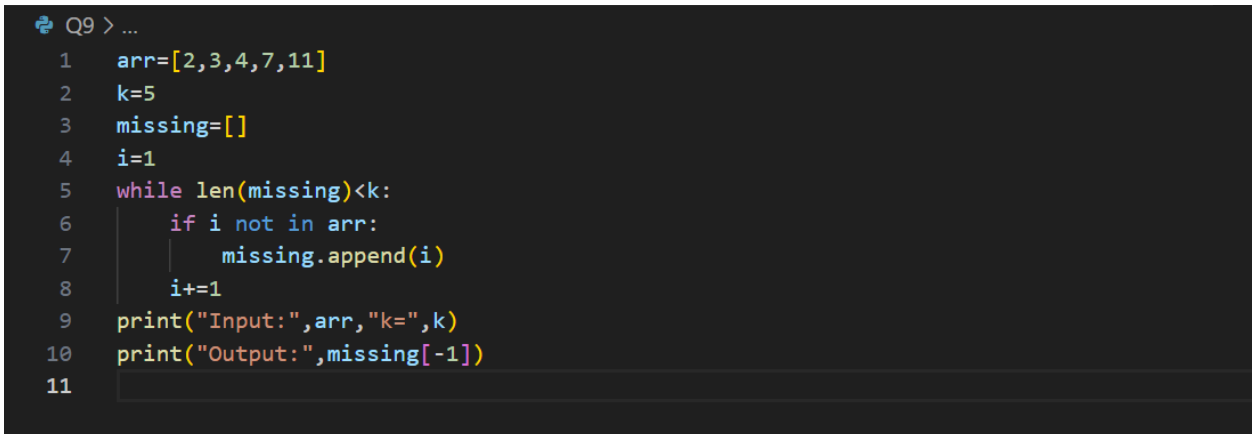
Result: Stable sort achieved with duplicates handled properly.

Q4. Missing Positive Number

Aim:
To find the k-th missing positive integer in a sorted list.
Algorithm:

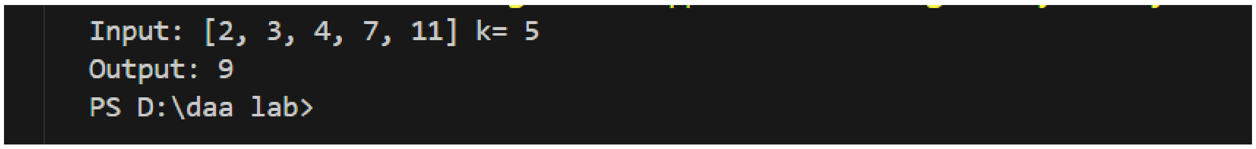
1. Read sorted list and k.
2. Generate missing numbers by comparison.
3. Count until k-th missing number found.
4. Return that number.
5. End.

CODE:



```
Q9 > ...
arr=[2,3,4,7,11]
k=5
missing=[]
i=1
while len(missing)<k:
    if i not in arr:
        missing.append(i)
    i+=1
print("Input:",arr,"k=",k)
print("Output:",missing[-1])
```

Input & Output:



```
Input: [2, 3, 4, 7, 11] k= 5
Output: 9
PS D:\daa lab>
```

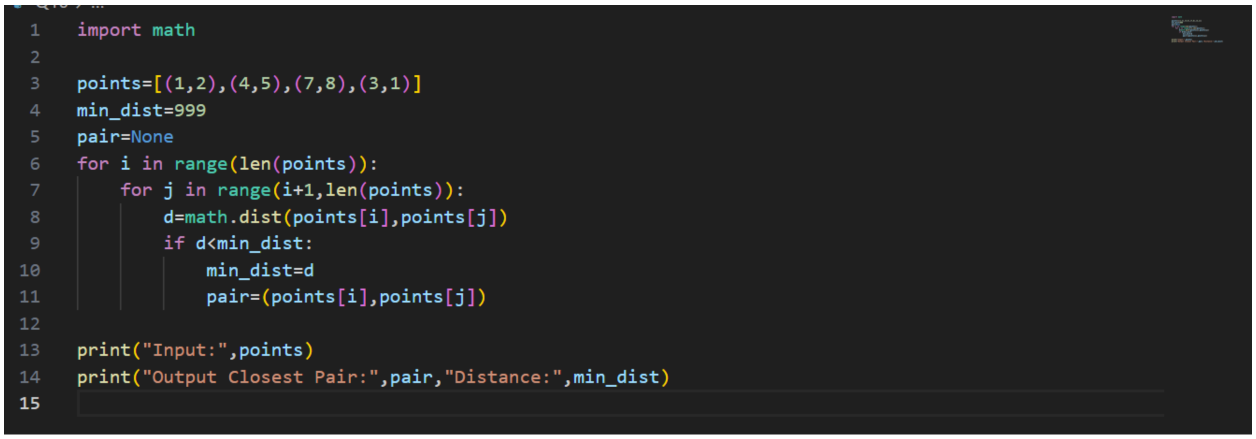
Result: Fifth missing integer correctly identified as 9.

Q5. Closest Pair of Points

Aim:
To find the closest pair of 2D points using brute force.
Algorithm:

1. Input list of points.
2. Compute distance between every pair.
3. Store minimum distance and pair.
4. Compare all pairs.
5. Output closest pair.

CODE:



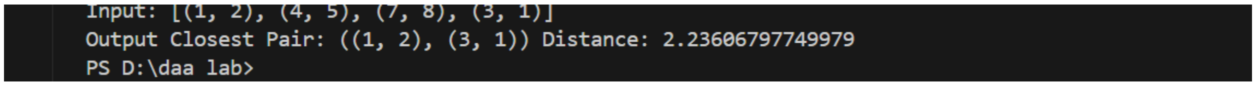
```python
import math

points=[(1,2),(4,5),(7,8),(3,1)]
min_dist=999
pair=None
for i in range(len(points)):
    for j in range(i+1,len(points)):
        d=math.dist(points[i],points[j])
        if d<min_dist:
            min_dist=d
            pair=(points[i],points[j])

print("Input:",points)
print("Output Closest Pair:",pair,"Distance:",min_dist)

```

Input & Output:



```
Input: [(1, 2), (4, 5), (7, 8), (3, 1)]
Output Closest Pair: ((1, 2), (3, 1)) Distance: 2.23606797749979
PS D:\daa lab>
```

Result: Brute-force method finds correct closest pair.