DIVIDE AND CONQUER

EXPERIMENT :1

Aim: To sort an unsorted array using the Merge Sort algorithm.

Procedure:

□ Divide the array into two halves.

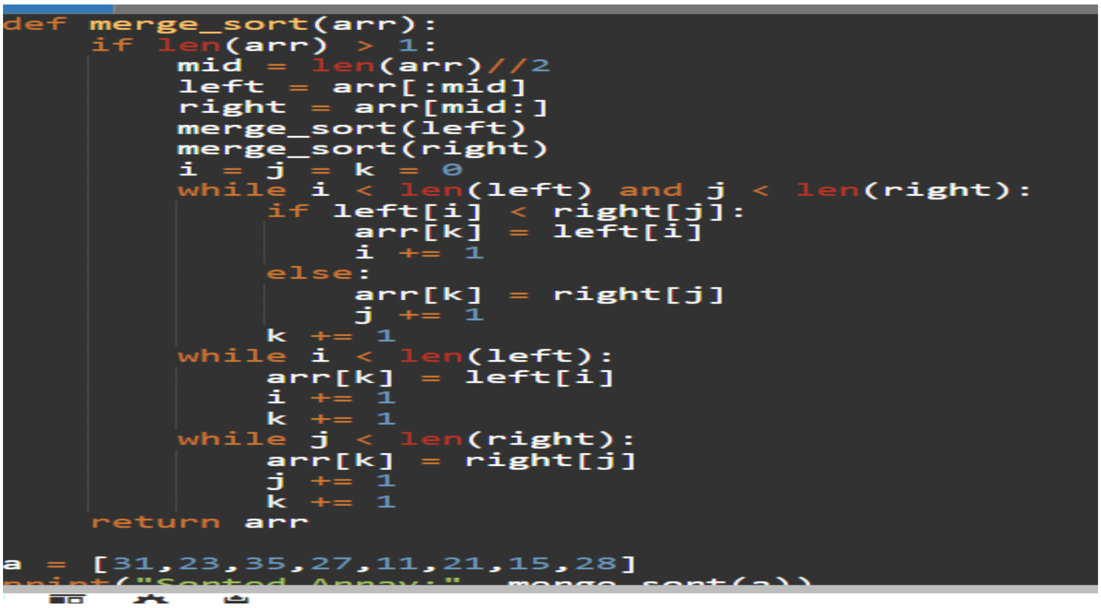
□ Recursively sort both halves.

□ Merge them in sorted order.

□ Return final sorted list.

□ Print sorted result.

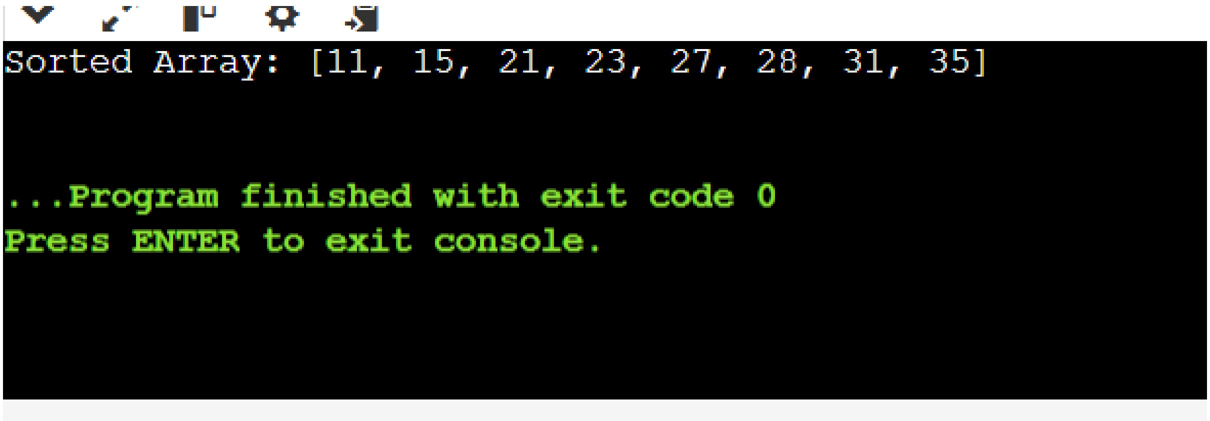
PROGRAM:



```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        left = arr[:mid]
        right = arr[mid:]
        merge_sort(left)
        merge_sort(right)
        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                arr[k] = left[i]
                i += 1
            else:
                arr[k] = right[j]
                j += 1
            k += 1
        while i < len(left):
            arr[k] = left[i]
            i += 1
            k += 1
        while j < len(right):
            arr[k] = right[j]
            j += 1
            k += 1
    return arr

a = [31,23,35,27,11,21,15,28]
print("Sorted Array:", merge_sort(a))
```

OUTPUT:



```
Sorted Array: [11, 15, 21, 23, 27, 28, 31, 35]


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

RESULT:

Merge Sort was successfully implemented and sorted the array in ascending order.

EXPERIMENT:2

AIM: To implement Merge Sort and count the number of comparisons made.

PROCEDURE:

□Implement merge sort with a counter variable.

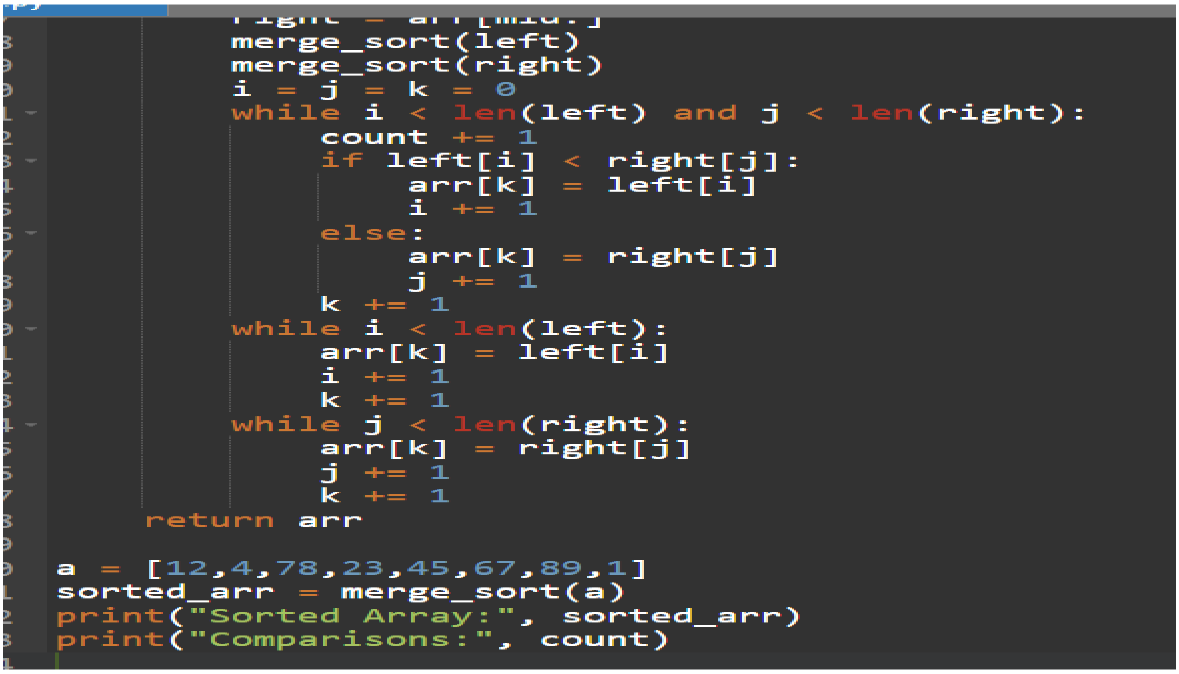
□ Increment counter for each comparison.

□ Sort the array using merge logic.

□ Print both sorted array and comparison count.

□ Verify correctness.

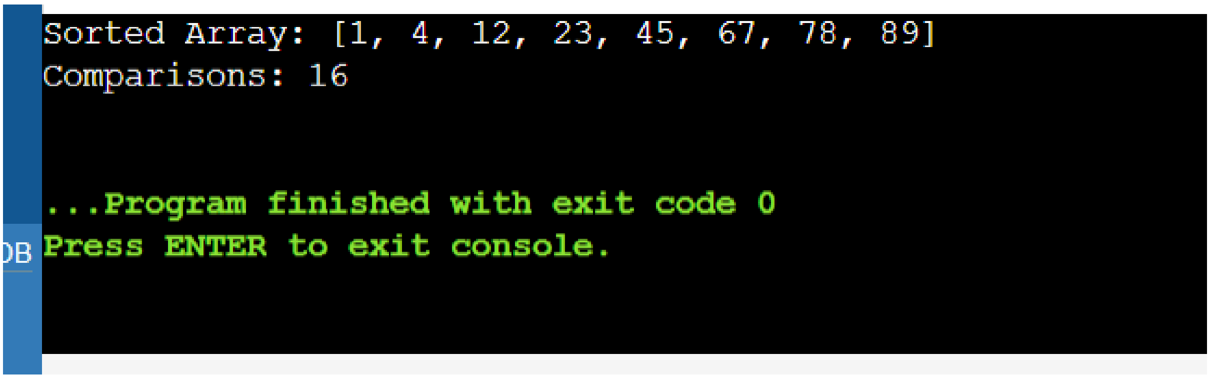
PROGRAM:



```
        right = arr[mid:]
        merge_sort(left)
        merge_sort(right)
        i = j = k = 0
        while i < len(left) and j < len(right):
            count += 1
            if left[i] < right[j]:
                arr[k] = left[i]
                i += 1
            else:
                arr[k] = right[j]
                j += 1
            k += 1
        while i < len(left):
            arr[k] = left[i]
            i += 1
            k += 1
        while j < len(right):
            arr[k] = right[j]
            j += 1
            k += 1
    return arr

a = [12,4,78,23,45,67,89,1]
sorted_arr = merge_sort(a)
print("Sorted Array:", sorted_arr)
print("Comparisons:", count)
```

OUTPUT:



```
Sorted Array: [1, 4, 12, 23, 45, 67, 78, 89]
Comparisons: 16


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

RESULT:

Merge Sort successfully sorted the array and counted the number of element comparisons.

EXPERIMENT:3

AIM: To find maximum and minimum values in a sorted array.

PROCEDURE:

□ Create an ascending order array.

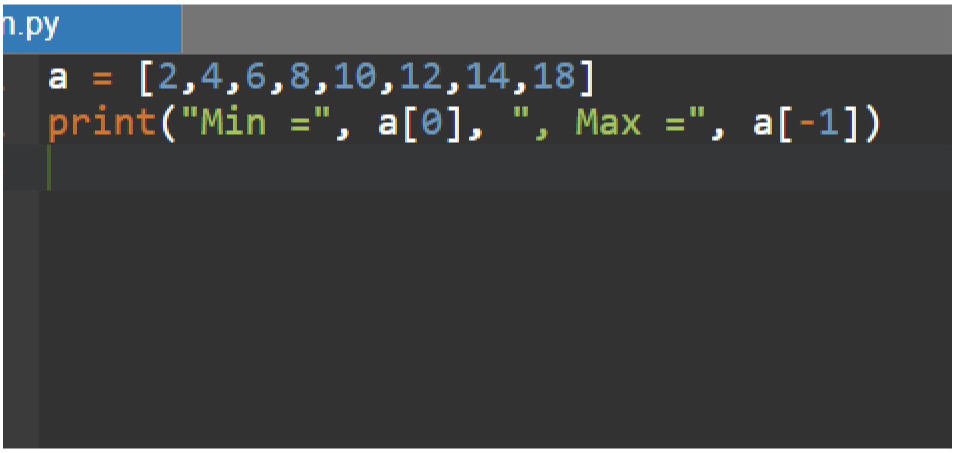
□ Since sorted, min is first and max is last element.

□ Use indexing or built-in functions.

□ Print min and max values.

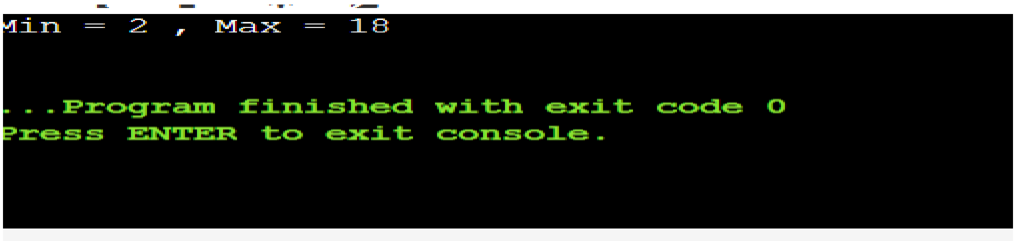
□ Test with given inputs.

PROGRAM:



```
a = [2,4,6,8,10,12,14,18]
print("Min =", a[0], ", Max =", a[-1])
```

OUTPUT:



```
Min = 2 , Max = 18

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

RESULT:

Program correctly identifies the smallest and largest values in the sorted array.

EXPERIMENT:4

AIM: To perform insertion sort and handle duplicate values correctly.

PROCEDURE:

□ Iterate through array elements.

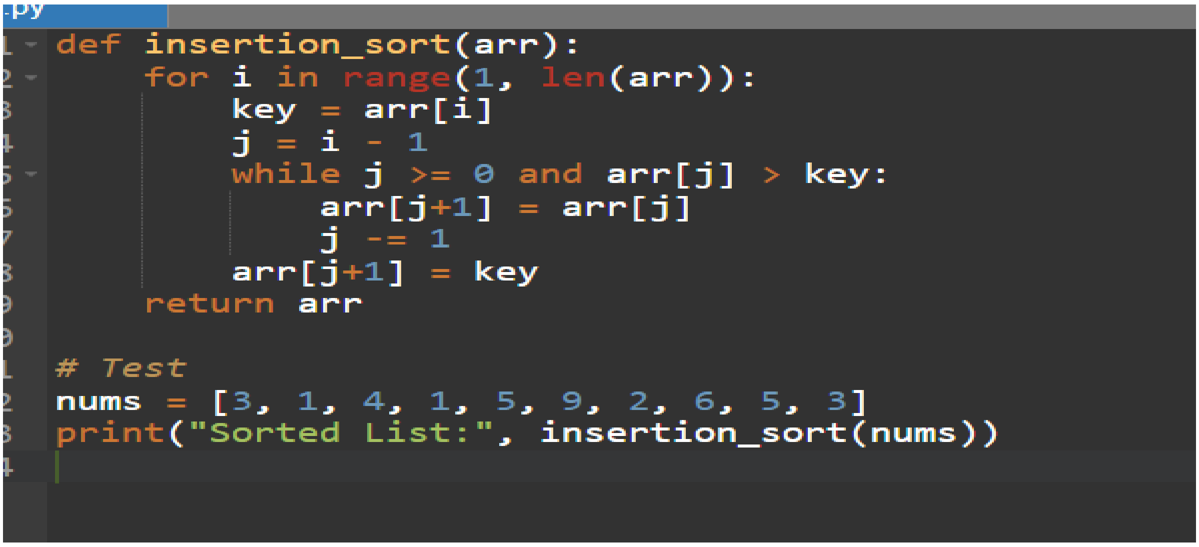
□ Insert each element into its correct position.

□ Duplicates remain in their relative order (stable sort).

□ Works well for small datasets.

□ Display sorted array.

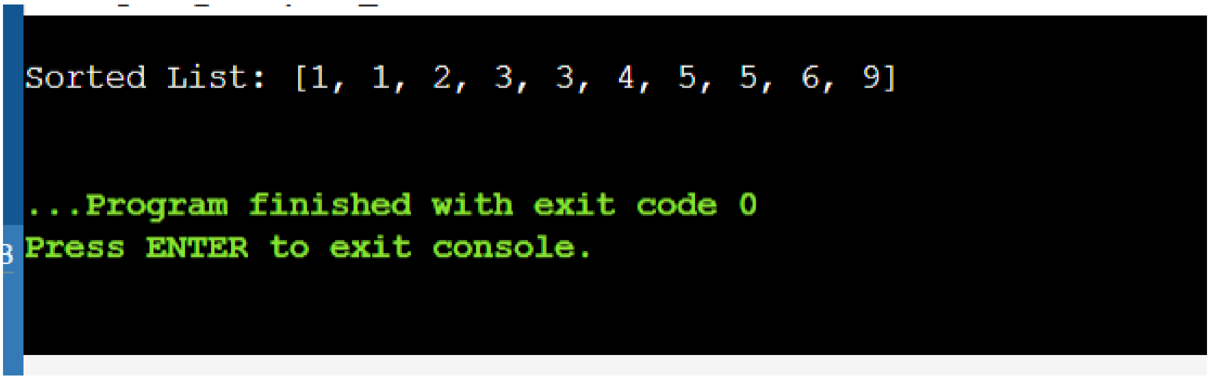
PROGRAM:



```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

# Test
nums = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]
print("Sorted List:", insertion_sort(nums))
```

OUTPUT:



```
Sorted List: [1, 1, 2, 3, 3, 4, 5, 5, 6, 9]


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

RESULT:

Insertion Sort correctly handled duplicate elements and preserved their relative order.

EXPERIMENT:5

AIM: To find the kth missing positive number from a sorted array.

PROCEDURE:

□ Initialize counter for missing numbers.

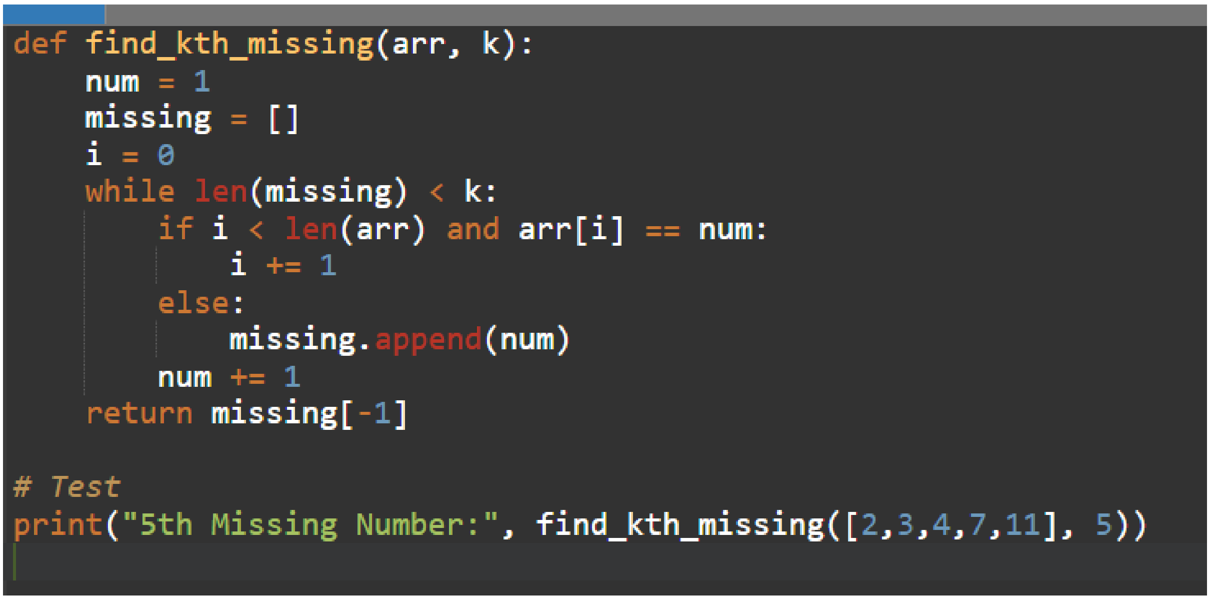
□ Loop through positive integers starting from 1.

□ Skip numbers found in array.

□ Stop when kth missing number is found.

□ Display the result.

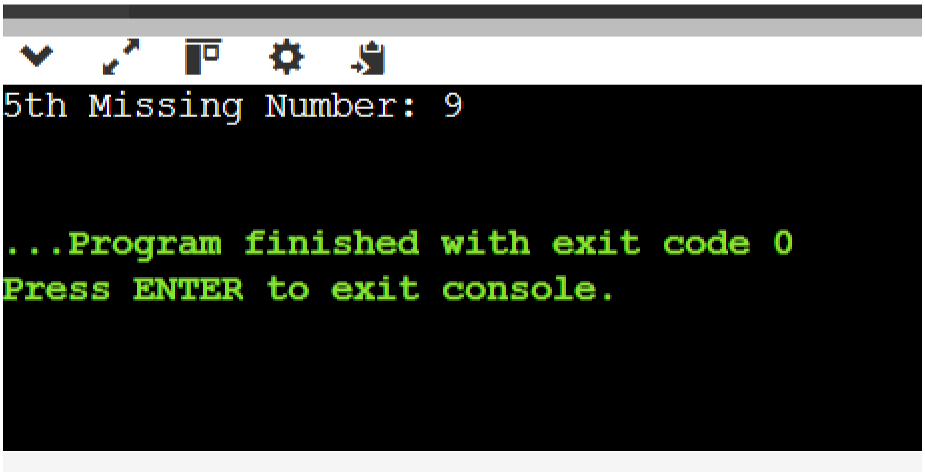
PROGRAM:



```python
def find_kth_missing(arr, k):
    num = 1
    missing = []
    i = 0
    while len(missing) < k:
        if i < len(arr) and arr[i] == num:
            i += 1
        else:
            missing.append(num)
        num += 1
    return missing[-1]

# Test
print("5th Missing Number:", find_kth_missing([2,3,4,7,11], 5))
```

OUTPUT:



```
5th Missing Number: 9


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

RESULT:

The algorithm correctly found the kth missing positive integer from the given sorted array