Name: Kamel Mohamed Mohamed Kamel Abdalla

Department: Artificial Intelligence

ID: 808967917

Counting Sort

This Python code implements the Counting Sort algorithm, which is an efficient sorting technique for sorting integers within a specific range. Here's a brief overview of the key components:

1. ***Imports**: The code imports the **sys** module for handling system-level operations and **List** from the **typing** module for type hinting.*
2. ***Input Function (read_input)**: This function prompts the user to enter an unsorted array of integers. It reads the input, splits it into individual string elements, converts them to integers, and returns the list. If the input contains non-integer values, the program exits with an error message.*
3. ***Counting Sort Function (counting_sort)**: This function implements the Counting Sort algorithm:*
    - ***Step 1**: It determines the minimum and maximum values in the input list to establish the range of the counting array.*
    - ***Step 2**: It initializes a counting array (**counter**) with zeros, sized to accommodate the range of input values.*
    - ***Step 3**: It populates the counting array by counting occurrences of each integer in the input list, adjusting for the minimum value to use zero-based indexing.*
    - ***Step 4**: It modifies the counting array to store cumulative counts, which helps in determining the positions of elements in the sorted output.*
    - ***Step 5**: It constructs the sorted output list by iterating through the input list in reverse order, placing each element in its correct position based on the counting array.*
4. ***Main Function**: The script's entry point calls the **read_input** function to get the unsorted array, applies the **counting_sort** function to sort the array, and then prints the sorted result.*

Usage

To use this code, simply run the script and input a space-separated list of integers when prompted. The program will output the sorted list of integers. Counting Sort is particularly efficient for sorting integers when the range of input values is not significantly larger than the number of elements to be sorted.

Time complexity = O(m + n)

Space complexity = O(m)

where,

m = range of elements

n = number of elements

Code Defines Description:

1- def read_input()

#Reads input from standard input, removes leading/trailing whitespace, and casts the values to their appropriate datatypes.

# Returns: List[int]: List of integers representing the unsorted array

2-def counting_sort()

#Sorts the elements of input list using the counting sort algorithm and returns a new sorted list.

# Algorithm:

1. Find the minimum and maximum elements to find the range.
2. Initialize empty list (counter) with size as the range,

   for counts of possible elements.
3. Iterate over the unsorted array and store the counts of elements

   in the counter list.
4. Updated counter list to have cumulative counts from minimum to maximum

element. This will give the position of element in sorted array.

5. Build the sorted array using the updated counter list.

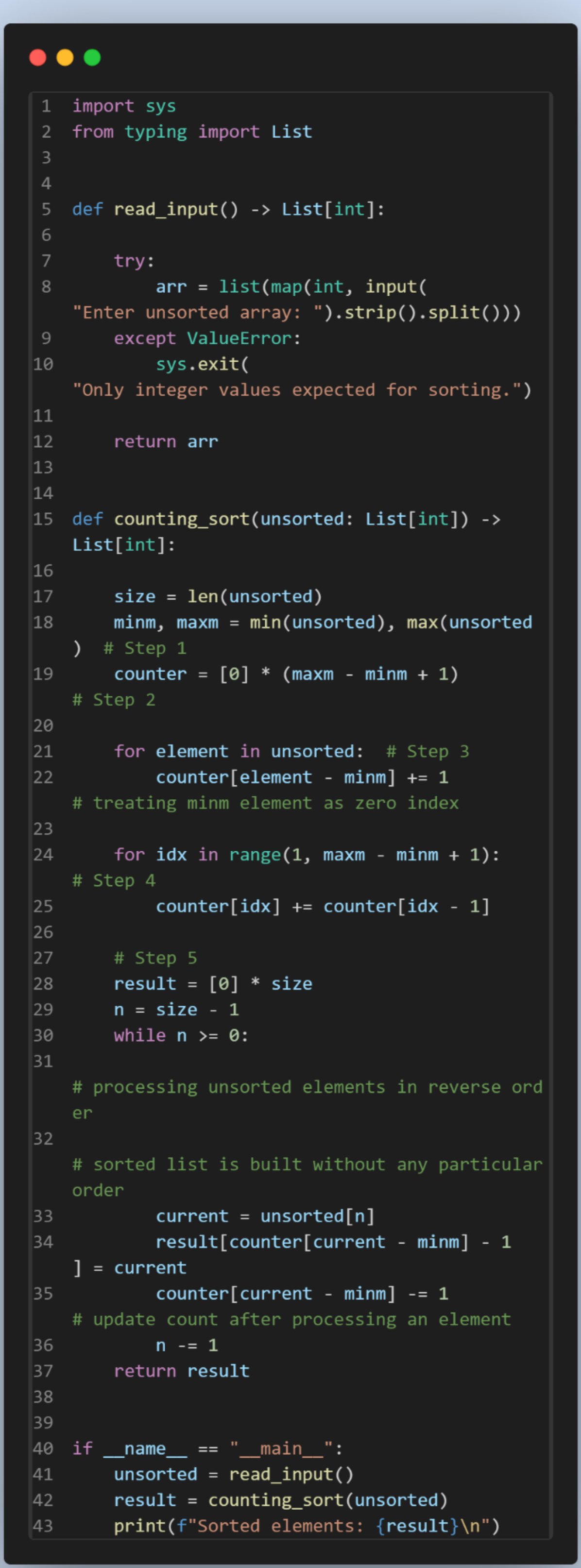
# Args:

unsorted (List[int]): Unsorted array

# Returns:

List[int]: Sorted version of input array

A Snap of The Code



```python
import sys
from typing import List


def read_input() -> List[int]:

    try:
        arr = list(map(int, input("Enter unsorted array: ").strip().split()))
    except ValueError:
        sys.exit("Only integer values expected for sorting.")

    return arr


def counting_sort(unsorted: List[int]) -> List[int]:

    size = len(unsorted)
    minm, maxm = min(unsorted), max(unsorted)  # Step 1
    counter = [0] * (maxm - minm + 1)  # Step 2

    for element in unsorted:  # Step 3
        counter[element - minm] += 1  # treating minm element as zero index

    for idx in range(1, maxm - minm + 1):  # Step 4
        counter[idx] += counter[idx - 1]

    # Step 5
    result = [0] * size
    n = size - 1
    while n >= 0:
        # processing unsorted elements in reverse order
        # sorted list is built without any particular order
        current = unsorted[n]
        result[counter[current - minm] - 1] = current
        counter[current - minm] -= 1  # update count after processing an element
        n -= 1
    return result


if __name__ == "__main__":
    unsorted = read_input()
    result = counting_sort(unsorted)
    print(f"Sorted elements: {result}\n")
```