Lab Manual: Linear Time Sorting Algorithms

1. Introduction

This lab manual covers linear time sorting algorithms: Counting Sort, Radix Sort, and Bucket Sort. These algorithms are efficient in specific scenarios and achieve sorting in linear time under certain constraints.

2. Algorithms

2.1 Counting Sort

Problem Definition:
Sort an array of integers in the range 0 to k using Counting Sort.

Input:
An array of non-negative integers A[0...n-1] where each A[i] ≤ k.

Desired Output:
A sorted array in non-decreasing order.

Solution:
Counting Sort counts the occurrences of each value, then computes the position of each element in the output array.

```
def counting_sort(arr):
    k = max(arr)
    count = [0] * (k + 1)
    output = [0] * len(arr)

    for num in arr:
        count[num] += 1

    for i in range(1, len(count)):
        count[i] += count[i - 1]

    for num in reversed(arr):
        output[count[num] - 1] = num
        count[num] -= 1

    return output
```

Time Complexity:
Best, Average, and Worst Case: $O(n + k)$

Space Complexity:
$O(n + k)$

2.2 Radix Sort

Problem Definition:
Sort an array of integers using digit-wise sorting.

Input:
An array of non-negative integers A[0...n-1].

Desired Output:
A sorted array in non-decreasing order.

Solution:
Radix Sort processes digits from least significant to most significant using a stable sort (like Counting Sort) on each digit.

```
def counting_sort_exp(arr, exp):
    n = len(arr)
    output = [0] * n
    count = [0] * 10

    for num in arr:
        index = (num // exp) % 10
        count[index] += 1

    for i in range(1, 10):
        count[i] += count[i - 1]

    for num in reversed(arr):
        index = (num // exp) % 10
        output[count[index] - 1] = num
        count[index] -= 1

    return output

def radix_sort(arr):
    max_val = max(arr)
    exp = 1
    while max_val // exp > 0:
        arr = counting_sort_exp(arr, exp)
        exp *= 10
    return arr
```

Time Complexity:
O(d*(n + b)), where d is the number of digits and b is the base.

Space Complexity:
O(n + b)

2.3 Bucket Sort

Problem Definition:
Sort an array of real numbers uniformly distributed in [0, 1).

Input:
An array of real numbers A[0...n-1] ∈ [0, 1).

Desired Output:
A sorted array in non-decreasing order.

Solution:
Bucket Sort divides the interval into buckets, distributes input among them, sorts each bucket, and concatenates the results.

```
def bucket_sort(arr):
    n = len(arr)
    buckets = [[] for _ in range(n)]

    for num in arr:
        index = int(n * num)
        buckets[index].append(num)

    for bucket in buckets:
        bucket.sort()

    result = []
    for bucket in buckets:
        result.extend(bucket)

    return result
```

Time Complexity:
Average Case: O(n), Worst Case: O(n^2)

Space Complexity:
O(n)

3. Proof of Correctness & Lower Bound

All sorting algorithms described maintain stability and correctness by relying on fundamental principles:
- Counting Sort places elements based on their frequency counts.
- Radix Sort uses stable sort at each digit position.
- Bucket Sort assumes uniform distribution.

Comparison-based sorting has a lower bound of $\Omega(n \log n)$, but these non-comparison-based algorithms bypass this bound under certain constraints.

4. Result

Each algorithm was implemented and tested with sample data. Their performance matches theoretical expectations when constraints are met.

5. Lab Problems

Problem 1: Student Score Normalization

Scenario:
A university wants to sort the scores (0 to 100) of thousands of students to assign percentile ranks. Use Counting Sort to efficiently sort the scores.

Input:
An array of integer scores in the range [0, 100].
Output:
Sorted array of scores in ascending order.

Problem 2: Sorting Sensor Readings from a Weather Station

Scenario:
A weather station records temperature values as floating point numbers in the range [0, 1) to model humidity levels over time. Sort this data to analyze climate patterns.

Input:
An array of float values in [0, 1).
Output:
Sorted array of float values in ascending order using Bucket Sort.