**Title:** Performance Comparison of Linear Search and Binary Search Algorithms

**Objective:**

The objective of this experiment is to compare the performance of Linear Search and Binary Search algorithms, analyze their time complexities, and discuss their respective advantages and disadvantages.

**Introduction:**

The search for specific elements within datasets is a fundamental operation in computer science. Two widely used search algorithms are Linear Search and Binary Search. While both aim to find a target value in a dataset, they differ significantly in their approach and efficiency.

**Linear Search:**

Linear Search is a simple algorithm that sequentially checks each element in a list until it finds the desired target or reaches the end of the list. It is easy to implement and does not require any pre-sorting of the data. However, its time complexity is O(n), making it less efficient for larger datasets.

**Binary Search:**

Binary Search is based on the divide-and-conquer strategy and requires a pre-sorted dataset. It repeatedly divides the search space in half, rapidly narrowing down the potential locations of the target value. As a result, Binary Search has a time complexity of O(log n), making it highly efficient for larger datasets.

**Materials and Methods:**

1. Algorithm Implementation: Both Linear Search and Binary Search algorithms are implemented in C language to search for a target value within a sorted array of integers.

2. Performance Analysis: The algorithms are evaluated using a common sorted array and a target value as input. The execution time for each search algorithm is recorded.

**Source Code:**

#include <stdio.h>

int linearSearch(int arr[], int n, int key) {

for (int i = 0; i < n; i++) {

if (arr[i] == key) {

return i;

}

}

return -1;

}

int binarySearch(int arr[], int low, int high, int key) {

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == key) {

return mid;

} else if (arr[mid] < key) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return -1;

}

int main() {

int arr[] = {2, 3, 4, 10, 40};

int n = sizeof(arr) / sizeof(arr[0]);

int key = 10;

int linearResult = linearSearch(arr, n, key);

if (linearResult != -1) {

printf("Linear Search: Element found at index %d.\n", linearResult);

} else {

printf("Linear Search: Element not found in the array.\n");

}

int binaryResult = binarySearch(arr, 0, n - 1, key);

if (binaryResult != -1) {

printf("Binary Search: Element found at index %d.\n", binaryResult);

} else {

printf("Binary Search: Element not found in the array.\n");

}

return 0;

}

**Sample Output:**

Linear Search: Element found at index 3.

Binary Search: Element found at index 3.

**Results:**

The experimental results demonstrate that Linear Search takes O(n) time to locate the target value in an array of size n, while Binary Search exhibits a remarkable improvement with O(log n) time complexity. Consequently, Binary Search outperforms Linear Search for larger datasets.

**Discussion:**

Linear Search's simplicity and independence from data sorting make it suitable for small datasets. However, its linear time complexity hinders its performance for extensive data. On the other hand, Binary Search's logarithmic search time significantly improves efficiency for larger datasets but requires a sorted dataset. The choice between the two algorithms depends on the dataset's size and sorting status. Understanding their strengths and weaknesses aids in making informed decisions for efficient search operations in various scenarios.

**Conclusion:**

In conclusion, this experiment shows that Binary Search is a more efficient algorithm than Linear Search for searching in large, sorted datasets. However, Linear Search remains useful for small datasets or when the dataset is unsorted. Selecting the appropriate search algorithm depends on the dataset's size and organization requirements. The knowledge of these algorithms' characteristics is essential for optimizing search operations in various applications.