**Rajan Gautam**

**19BCP101**

**Div. II, CE 19**

**SOT, PDPU**

**Pandit Deendayal Petroleum University**

**School of Technology**

**Design & Analysis of Algorithm (20CP209P)**

**B. Tech - Computer Science & Engineering (Sem-IV)**

**Table of Contents**

[**Lab 2 Assignment: Comparison of Quick Sort & Merge Sort. 1**](#_Toc63522507)

[**AIM: To write a C/C++ Program to implement Merge Sort & Quick Sort. 1**](#_Toc63522508)

[**ALGORITHMS: 1**](#_Toc63522509)

[**1. Merge Sort Algorithm (Pseudocode) 1**](#_Toc63522510)

[**2. Quick Sort Algorithm (Pseudocode) 1**](#_Toc63522511)

[**CODE: 2**](#_Toc63522512)

[**OUTPUT: 5**](#_Toc63522513)

[**ALL OUTPUT: 5**](#_Toc63522514)

[**Performance Comparison of Merge Sort and Quick Sort 6**](#_Toc63522515)

[**Performance Comparison of All Four Sorting Algorithms 6**](#_Toc63522516)

# Lab 2 Assignment: Comparison of Quick Sort & Merge Sort.

## AIM: To write a C/C++ Program to implement Merge Sort & Quick Sort.

## ALGORITHMS:

### Merge Sort Algorithm (Pseudocode)

**INSERTION-SORT (A, p, q, r)**

**n1 🡨 q – p + 1**

**N2 🡨 r – q**

**Create arrays L[1…n1 + 1] and R[1…n2 + 1]**

**For I 🡨 1 to n1**

**Do L[i] 🡨 A[p + I – 1]**

**For j 🡨 1 to n2**

**R[j] 🡨 A[q + j]**

**L[n1 + 1] 🡨 infinite**

**R[n2 + 1] 🡨 infinite**

**i 🡨 1**

**j 🡨 1**

**for k 🡨 p to r**

**do if L[i] ≤ R[j]**

**then A[k] 🡨 L[i]**

**i 🡨 i + 1**

**else A[k] 🡨 R[j]**

**j 🡨 j + 1**

**MERGE-SORT(A, p, r)**

**if p < r**

**then q 🡨 L(p + r) / 2｣**

MERGE-SORT(A, p, q)

**MERGE-SORT(A, q + 1, r)**

**MERGE-SORT(A, p, q, r)**

### 

### Quick Sort Algorithm (Pseudocode)

**PARTITION(A, p, r)**

**X = A[r]**

**I = p – 1**

**For j = p to r – 1**

**if A[j] ≤ x**

**i = i + 1**

**exchange A[i] with A[j]**

**exchange A[i + 1] with A[r]**

**return i + 1**

**QUICKSORT (A, p, r)**

**if p < r**

**q = PARTITION (A, p, r)**

**QUICKSORT (A, p, q – 1)**

**QUICKSORT (A, q + 1, r)**

## CODE:

1. **/\* ------------------------- 19BCP101 -----------------------\*/**
2. **/\* ----------------------- Rajan Gautam ---------------------\*/**
4. **#include <stdio.h>**
5. **#include <stdlib.h>**
6. **#include <time.h> // For Time Calculation**
8. **void merge(int arr[], int l, int m, int r)**
9. **{**
10. **int i, j, k;**
11. **int n1 = m - l + 1;**
12. **int n2 = r - m;**
14. **int L[n1], R[n2];**
16. **// Copy data to temp arrays L[] and R[]**
18. **for (i = 0; i < n1; i++)**
19. **L[i] = arr[l + i];**
20. **for (j = 0; j < n2; j++)**
21. **R[j] = arr[m + 1 + j];**
23. **i = 0;**
24. **j = 0;**
25. **k = l;**
27. **while (i < n1 && j < n2)**
28. **{**
29. **if (L[i] <= R[j])**
30. **{**
31. **arr[k] = L[i];**
32. **i++;**
33. **}**
34. **else**
35. **{**
36. **arr[k] = R[j];**
37. **j++;**
38. **}**
39. **k++;**
40. **}**
42. **// Copy the remaining elements of L[], if there are any**
43. **while (i < n1)**
44. **{**
45. **arr[k] = L[i];**
46. **i++;**
47. **k++;**
48. **}**
50. **// Copy the remaining elements of R[], if there are any**
51. **while (j < n2)**
52. **{**
53. **arr[k] = R[j];**
54. **j++;**
55. **k++;**
56. **}**
57. **}**
59. **void mergeSort(int arr[], int l, int r)**
60. **{**
61. **if (l < r) {**
62. **// Same as (l+r)/2, but avoids overflow for large l and h**
63. **int m = l + (r - l) / 2;**
65. **// Sort first and second halves**
66. **mergeSort(arr, l, m);**
67. **mergeSort(arr, m + 1, r);**
69. **merge(arr, l, m, r);**
70. **}**
71. **}**
73. **/\* FOR QUICK SORT \*/**
75. **void swap(int\* a, int\* b)**
76. **{**
77. **int t = \*a;**
78. **\*a = \*b;**
79. **\*b = t;**
80. **}**
82. **int partition (int arr[], int low, int high)**
83. **{**
84. **int pivot = arr[high];**
85. **int i = (low - 1);**
87. **for (int j = low; j <= high - 1; j++)**
88. **{**
89. **// If current element is smaller than the pivot**
90. **if (arr[j] < pivot)**
91. **{**
92. **i++;**
93. **swap(&arr[i], &arr[j]);**
94. **}**
95. **}**
96. **swap(&arr[i + 1], &arr[high]);**
97. **return (i + 1);**
98. **}**
100. **void quickSort(int arr[], int low, int high)**
101. **{**
102. **if (low < high)**
103. **{**
104. **int pi = partition(arr, low, high);**
106. **quickSort(arr, low, pi - 1);**
107. **quickSort(arr, pi + 1, high);**
108. **}**
109. **}**

112. **int main()**
113. **{**
114. **printf("<--------------- Sorting --------------->\n\n");**
116. **int n = 1000, it = 0;**
117. **double time1[20], time2[20]; // To store the time values**
119. **printf(" Array \t Merge(s) \t Quick(s) \n\n");**
121. **while(it++ < 10)**
122. **{**
123. **long int a[n], b[n];**
124. **for (int i = 0; i < n ; i++)**
125. **{**
126. **// Generating Random Integer Array for each algorithm**
128. **a[i] = (rand() % n);**
129. **b[i] = (rand() % n);**
130. **}**

133. **// For time calculation**
134. **clock\_t start, end;**

137. **// For Merge Sort Algorithm**
138. **start = clock();**
139. **mergeSort(a, 0, n-1);**
140. **end = clock();**
142. **time1[it] = ((double)(end - start)/CLOCKS\_PER\_SEC);**

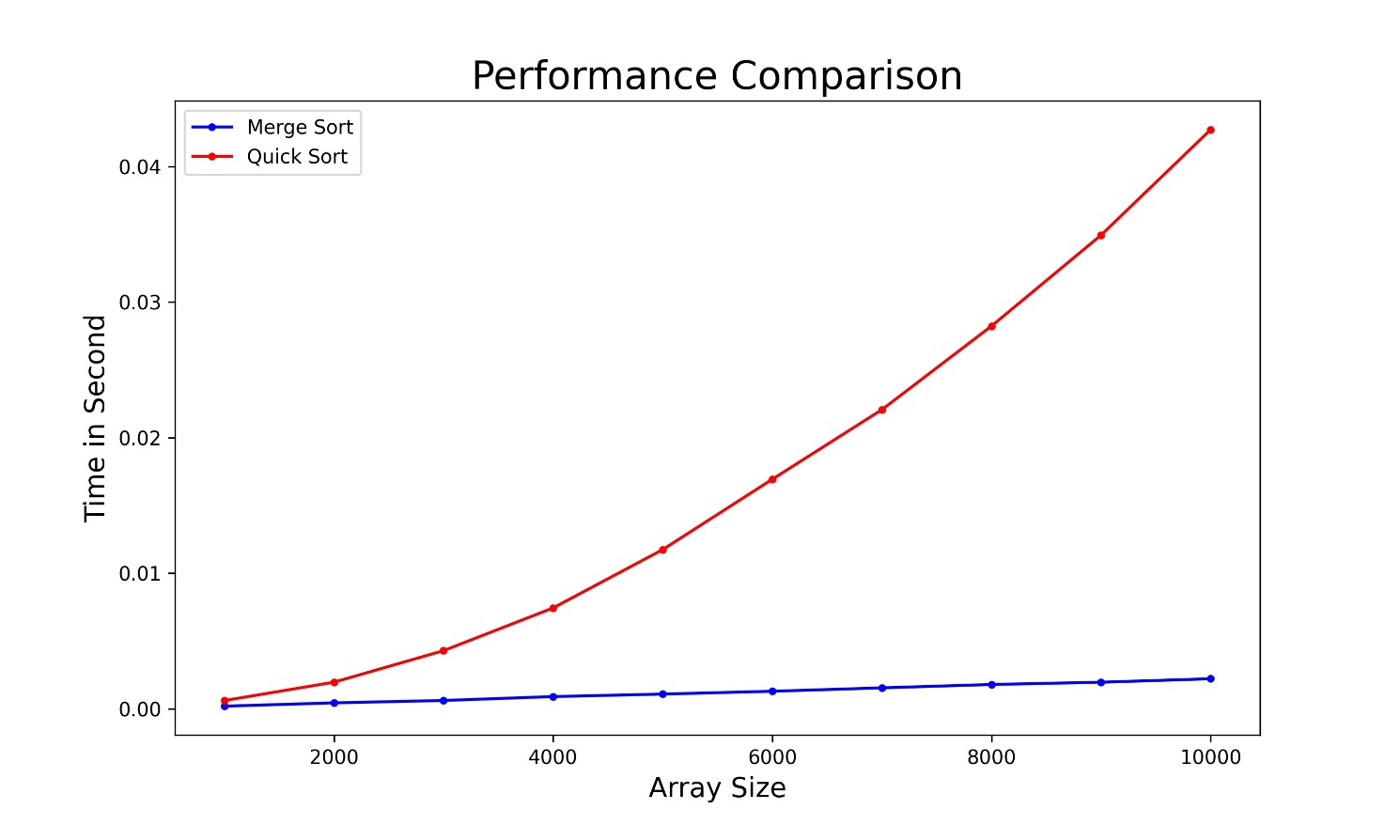
145. **// For Quick Sort Algorithm**
146. **start = clock();**
147. **quickSort(b, 0, n-1);**
148. **end = clock();**
150. **time2[it] = ((double)(end - start)/CLOCKS\_PER\_SEC);**
152. **// Printing the table of array size, time taken by Bubble Sort and Insertion Algorithm**
153. **printf(" %d \t %f \t %f\n", n, time1[it], time2[it]);**

156. **// Incrementing the value of n by 1000**
157. **n += 1000;**
158. **}**
159. **return 0;**
160. **}**

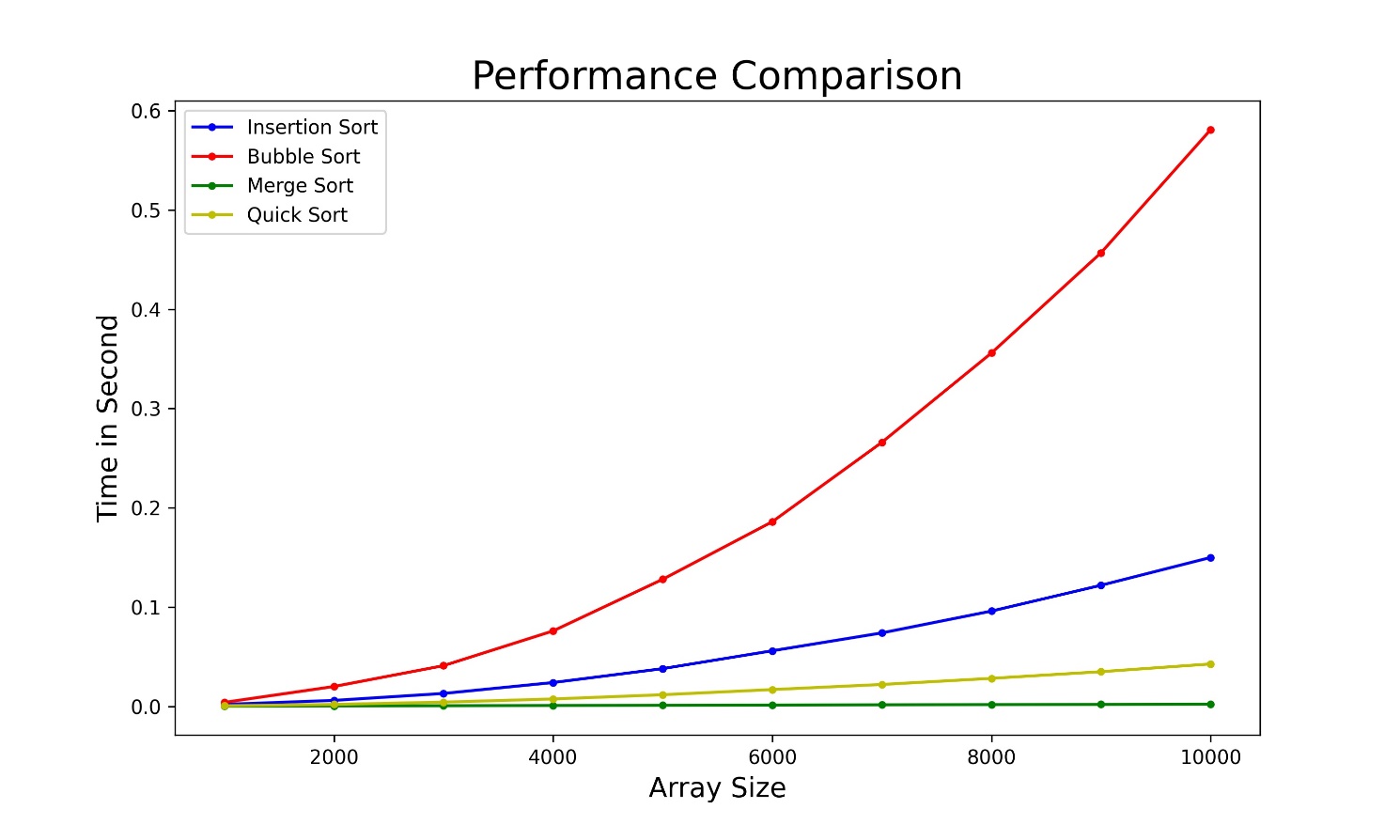
## OUTPUT:

## ALL OUTPUT:

## Performance Comparison of Merge Sort and Quick Sort



## Performance Comparison of All Four Sorting Algorithms



**Link:** <https://github.com/rgautam320/Design-and-Analysis-of-Algorithm-Lab/tree/master/Lab_2_Sorting>