RAJSHAHI UNIVERSITY OF ENGINEERING & TECHNOLOGY

LAB REPORT-03

COURSE NAME: SESSIONAL BASED ON CSE-2201
COURSE CODE: CSE-2102

SUBMITTED TO-
DR. MD. ALI HOSSAIN
ASSOCIATE PROFESSOR
Department of Computer Science & Engineering
Rajshahi University of Engineering & Technology

SUBMITTED BY-
SRABONTI DEB
Roll-1803163
Section - C
Department of Computer Science & Engineering
Rajshahi University of Engineering & Technology

SUBMISSION DATE – 12 MARCH, 2021

Title: Finding the time and space complexity of the merge sort approach and comparing it with the performance of quick sort algorithm for (i) Best case (ii) Average case and (iii) worst case.

Introduction: Merge sort is a divide and conquer algorithm. The merge sort function repeatedly divides the array into two halves until we reach a stage where we try to perform mergesort on a subarray of size.

Quick sort is also a divide and conquer algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. That's why, sometimes it is called as 'partition exchange sort'.

Description:

Algorithm of Merge Sort:

MergeSort (var a as array)

1. if (n = 1)
2. return a
3. var l_1 as array = a[0] . . . a[n/2]
4. var l_2 as array = a[n/2+1] . . . . a[n]
5. l_1 = MergeSort (l_1)
6. l_2 = MergeSort(l_2)

7. return merge (l_1, l_2)

Merge (var a as array, var b as array)

1. var c as array.
2. while (a and b have elements)
3. if (a[0] > b[0]
4. ~~add~~ add b[0] to the end of c
5. remove b[0] from b.
6. else
7. add a[0] to the end of c
8 remove a[0] from a
9. end while.
10. while (a has elements)
11. add a[0] to the end of c.
12. remove a[0] from a.
13. end while
14. while (b has elements).
15. add b[0]. to the end of c
16. remove b[0] from b.
17. ~~whi~~ end while.
18. return c

<u>Algorithm of quick sort:</u>

QuickSort (A, p, r)

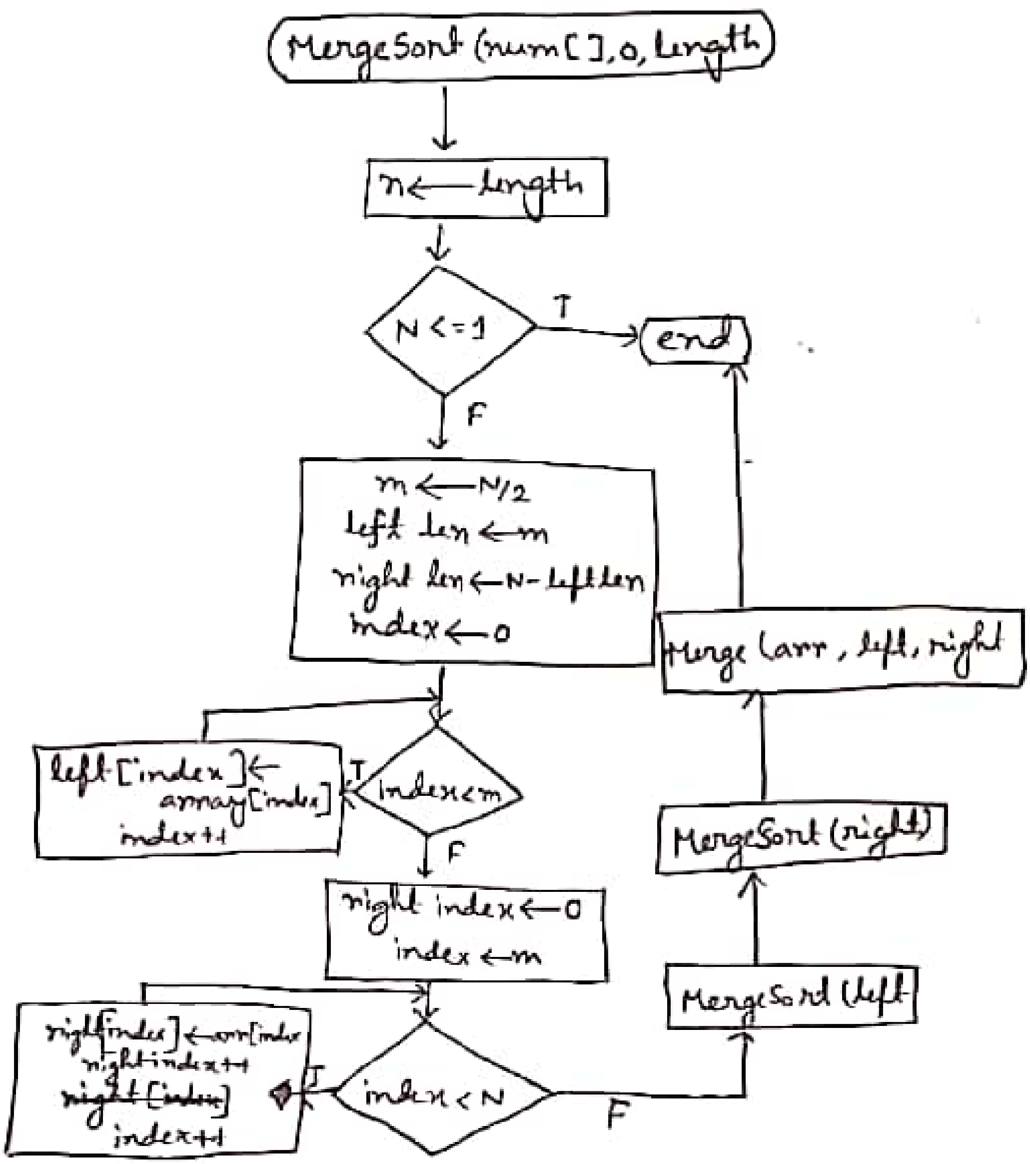
1. if q < r
2. then q ⟵ partition (A, P, r)
3. QuickSort (A, P, q-1)
4. QuickSort (A, q+1, r)

(To sort an entire array A, we initially call is Quicksort (A, 1, length[A])) partition (A, p, r)

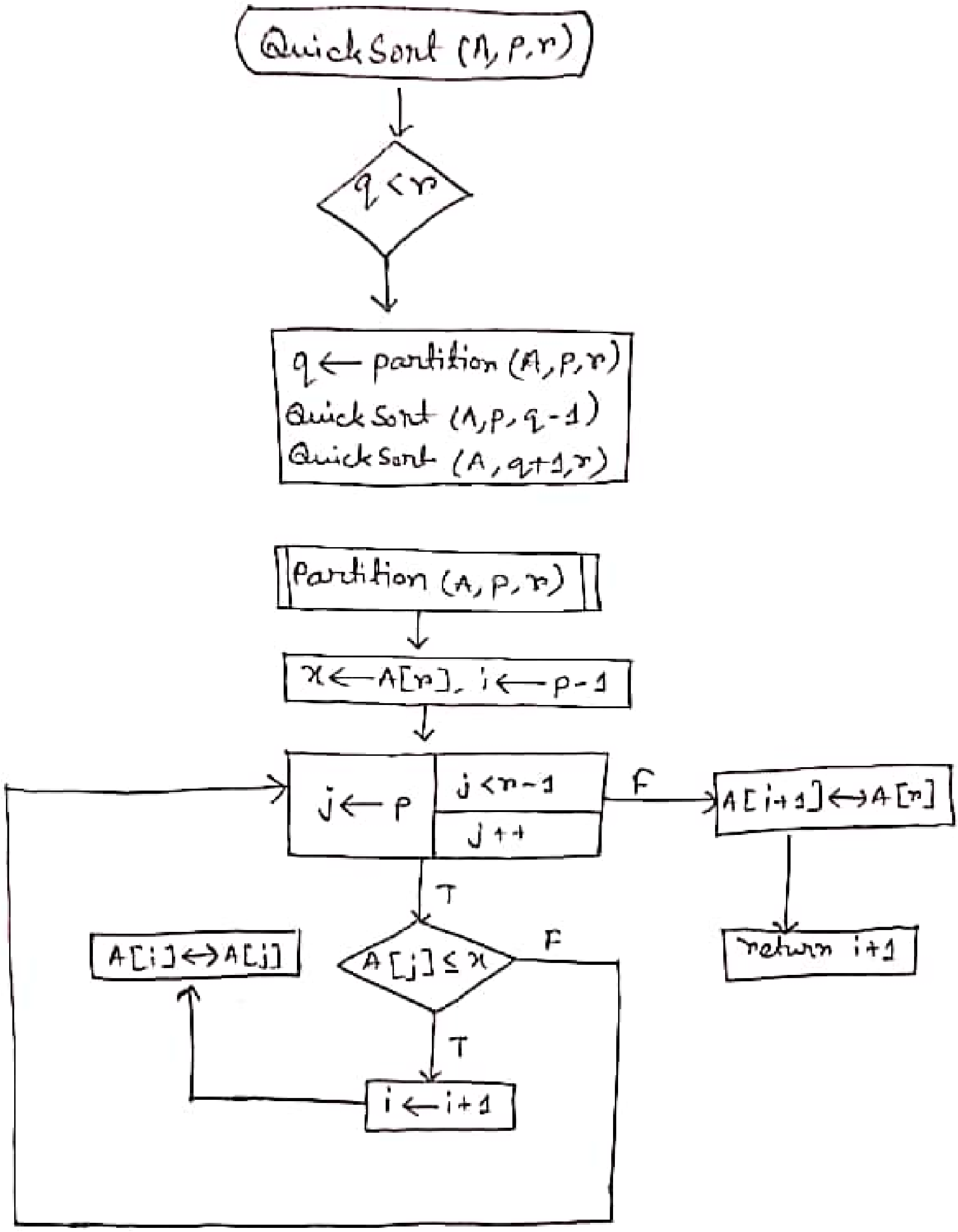
1. $x \leftarrow A[r]$
2. $i \leftarrow p-1$
3. for $j \leftarrow p$ to $r-1$.
4. do if $A[j] \leq x$.
5.   then $i \leftarrow i+1$.
6.   exchange $A[i] \leftrightarrow A[j]$
7. exchange $A[i+1] \longleftrightarrow A[r]$
8. return $i+1$.

2) Flow chart:

Flow chart of Merge Sort



Flow chart of Quick Sort:



Time Complexity:

Case → Merge Sort → Quick Sort

Worst Case → O(nlogn) → O(n^2)

Average Case → O(nlogn) → O(nlogn)

Best Case → O(nlogn) → O(nlogn)

Sample Output:

For 4000 inputs:

Time required for quick sort: 1.783ms

Time required for merge Sort: ~~1.027ms~~ 1.044ms

Sorted list: 0 5 10 36 66. . . . . .

For 5000 inputs:

Time required for quick sort: 2.772ms

Time required for merge Sort: 1.405ms.

Sorted list: 5 7 13 16 . . . . .

For 6000 inputs:

Time required for quick Sort: 1.915ms

Time required for ~~quick~~ merge sort: 0.868ms.

Sorted list: 0 24 31 47 60. . . . . .

For 7000 inputs:

Time required for quick Sort: 2.022ms.

Time required for merge Sort: 1.07ms.

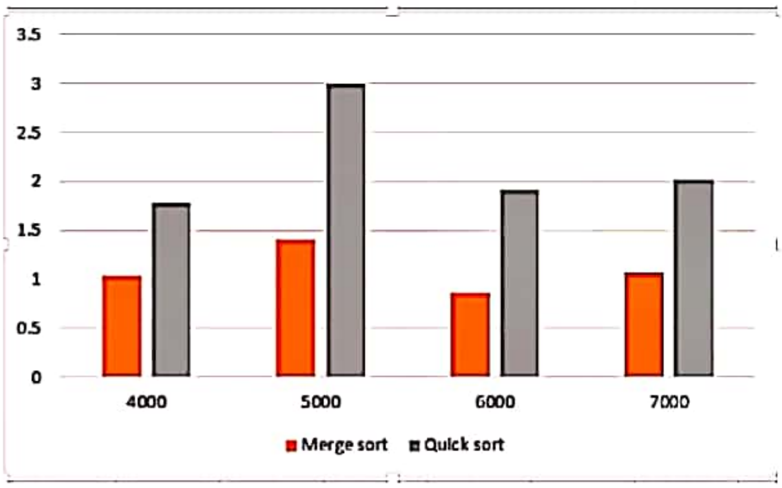
Sorted list: 0 12 25 32 . . . .

Data Table and Graph:

Data Table:

number of inputs	Merge sort	Quick sort
4000	1.044	1.783
5000	1.405	2.992
6000	0.868	1.9165
7000	1.07	2.022

Graph:



Conclusion: From the output of the code we can realise that, merge sort is faster that quick sort. We ~~is~~ can ~~see the~~ visualize the difference of complexity in the graph.