Batch: A1	**Roll No.: 16010123012**
Experiment No. 3	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Study, Implementation, and Comparative Analysis of Merge Sort and Quick Sort.

Objective: To learn the divide and conquer strategy of solving the problems of different types

CO to be achieved:

CO 2	Describe various algorithm design strategies to solve different problems and analyze Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://en.wikipedia.org/wiki/Quicksort**
4. **https://www.cs.auckland.ac.nz/~jmor159/PLDS210/qsort.html**
5. **http://www.cs.rochester.edu/~gildea/csc282/slides/C07-quicksort.pdf**
6. **http://www.sorting-algorithms.com/quick-sort**
7. **http://www.cse.ust.hk/~dekai/271/notes/L01a/quickSort.pdf**
8. **http://en.wikipedia.org/wiki/Merge_sort**
9. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/mergeSort.htm**
10. **http://www.sorting-algorithms.com/merge-sort**
11. **http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Merge_sort.html**

Pre Lab/ Prior Concepts:
Data structures, various sorting techniques

Historical Profile:
Quicksort and merge sort are divide-and-conquer sorting algorithm in which division is dynamically carried out. They are one the most efficient sorting algorithms.

New Concepts to be learned:
Number of comparisons, Application of algorithmic design strategy to any problem, Classical problem solving vs Divide-and-Conquer problem solving.

Algorithm Recursive Quick Sort:

```
void quicksort( Integer A[ ], Integer left, Integer right)
//sorts A[left.. right] by using partition() to partition A[left.. right], and then //calling itself //
twice to sort the two subarrays.
{ IF ( left < right ) then
        {       q = partition( A, left, right);
                quicksort( A, left, q–1);
                quicksort( A, q+1, right);

        }

}
```

```
Integer partition(integer AT[], Integer left, Integer right)
//This function  rarranges A[left..right] and finds and returns an integer q, such that A[left], ...,
//A[q–1] <~□pivot, A[q] = pivot, A[q+1], ..., A[right] > pivot, where pivot is the first element
of //a[left…right], before partitioning.
{
pivot = A[left]; lo = left+1; hi = right;
WHILE ( lo ≤ hi)
{       WHILE (A[hi] > pivot)                                   hi = hi – 1;
        WHILE ( lo ≤ hi and A[lo] <~pivot)              lo = lo + 1;
        IF ( lo ≤ hi) then                              swap( A[lo], A[hi]);
}
swap(pivot, A[hi]);
  RETURN hi;

}
```

The space complexity of Quick Sort: O(logn)

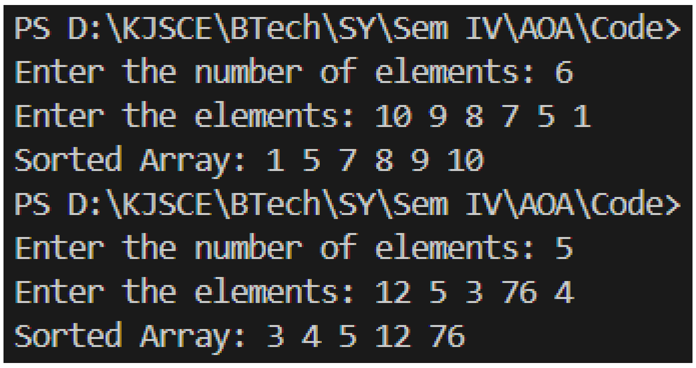
The time complexity of Quick Sort: O(nlogn)

Code:

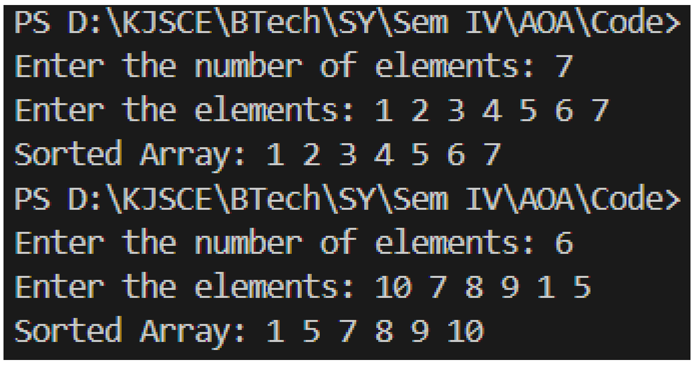
```
#include <bits/stdc++.h>
using namespace std;
```

```
int partition(int a[], int low, int high)
{
  int pivot = a[low];
  int i = low;

  for (int j = low + 1; j <= high; j++)
  {
    if (a[j] < pivot)
    {
      i++;
      swap(a[i], a[j]);
    }
  }
  swap(a[i], a[low]);
  return i;
}

void quickSort(int a[], int low, int high)
{
  if (low < high)
  {
    int q = partition(a, low, high);
    quickSort(a, low, q - 1);
    quickSort(a, q + 1, high);
  }
}

int main()
{
  int n;
  cout << "Enter the number of elements: ";
  cin >> n;
  int a[n];
  cout << "Enter the elements: ";
  for (int i = 0; i < n; i++)
  {
    cin >> a[i];
  }

  quickSort(a, 0, n - 1);
  cout << "Sorted Array: ";
  for (int i = 0; i < n; i++)
  {
    cout << a[i] << " ";
  }
}
```

Output:

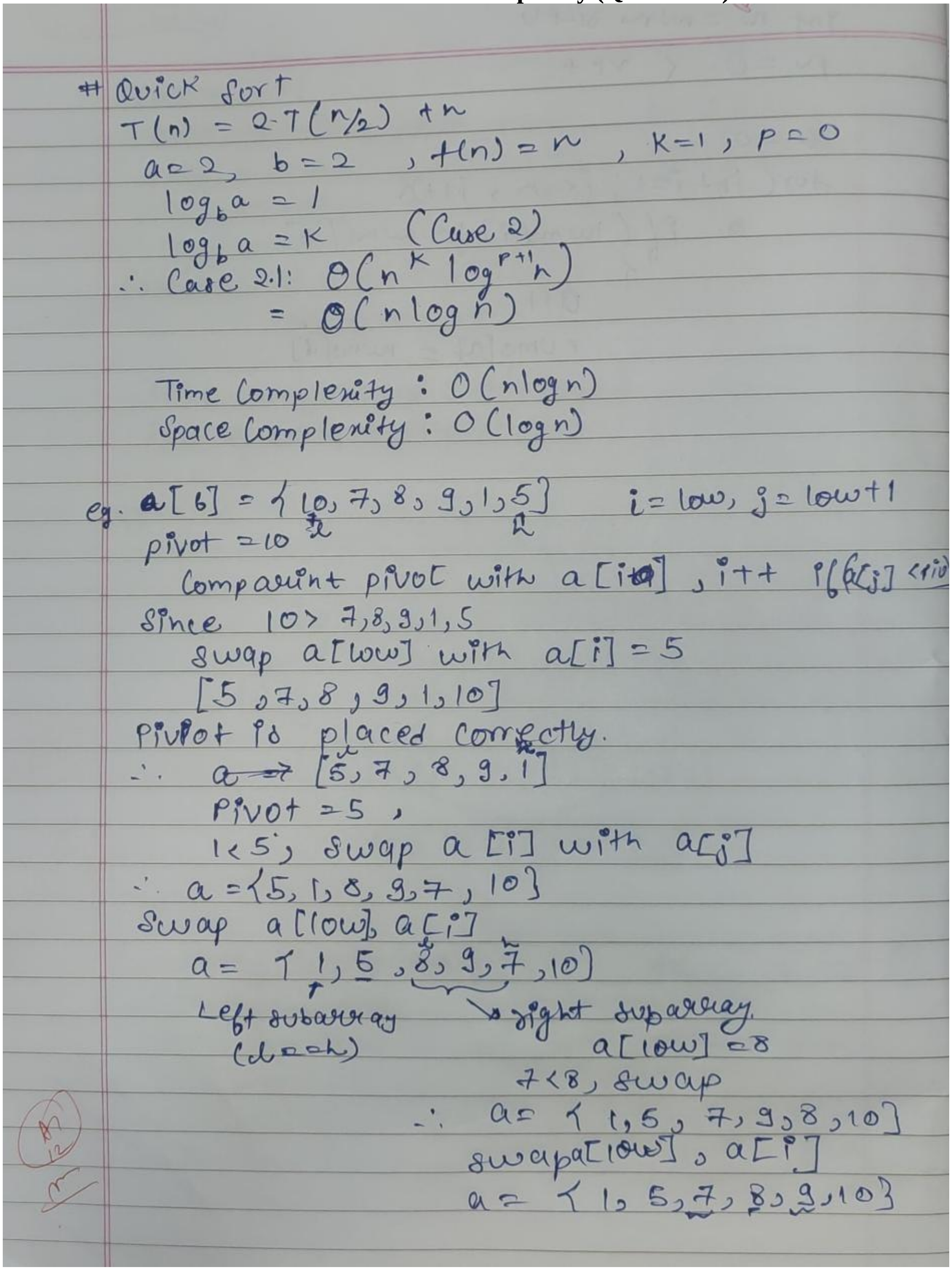


```
PS D:\KJSCE\BTech\SY\Sem IV\AOA\Code>
Enter the number of elements: 6
Enter the elements: 10 9 8 7 5 1
Sorted Array: 1 5 7 8 9 10
PS D:\KJSCE\BTech\SY\Sem IV\AOA\Code>
Enter the number of elements: 5
Enter the elements: 12 5 3 76 4
Sorted Array: 3 4 5 12 76
```



```
PS D:\KJSCE\BTech\SY\Sem IV\AOA\Code>
Enter the number of elements: 7
Enter the elements: 1 2 3 4 5 6 7
Sorted Array: 1 2 3 4 5 6 7
PS D:\KJSCE\BTech\SY\Sem IV\AOA\Code>
Enter the number of elements: 6
Enter the elements: 10 7 8 9 1 5
Sorted Array: 1 5 7 8 9 10
```

Derivation of best case and worst-case time complexity (Quick Sort):



# Quick sort

$T(n) = 2 \cdot T(n/2) + n$

$a = 2$, $b = 2$, $f(n) = n$, $k = 1$, $p = 0$

$\log_b a = 1$

$\log_b a = k$ (Case 2)

$\therefore$ Case 2.1: $\Theta(n^k \log^{p+1} n)$

$= \Theta(n \log n)$

Time Complexity : $O(n \log n)$

Space Complexity : $O(\log n)$

eg. $a[6] = \{\underset{i}{10}, 7, 8, 9, 1, \underset{h}{5}\}$ $i = low$, $j = low + 1$

pivot $= 10$

Comparint pivot with $a[i]$, $i{+}{+}$ if $a[j] <$ pivot

Since $10 > 7, 8, 9, 1, 5$

swap $a[low]$ with $a[i] = 5$

$[5, 7, 8, 9, 1, 10]$

Pivot is placed correctly.

$\therefore$ ~~a →~~ $[5, 7, 8, 9, 1]$

Pivot $= 5$,

$1 < 5$, swap $a[i]$ with $a[j]$

$\therefore a = \{5, 1, 8, 9, 7, 10\}$

Swap $a[low]$, $a[i]$

$a = \{1, 5, \underbrace{8, 9, 7}, 10\}$

Left subarray (done) → right subarray

$a[low] = 8$

$7 < 8$, swap

$\therefore a = \{1, 5, 7, 9, 8, 10\}$

swap $a[low]$, $a[i]$

$a = \{1, 5, 7, 8, 9, 10\}$

A7 12

[illegible]

CONCLUSION:
I have successfully implemented the Quick Sort algorithm using the divide and conquer technique. It achieves a time complexity of O(nlog n) in the best and average cases, while the worst case occurs when partitions are highly unbalanced or when the array is already sorted and the pivot is the first or last element, leading to $O(n^2)$ complexity. The space complexity is O(logn) due to recursive calls, but it is more memory-efficient than Merge Sort. Quick Sort is widely used in databases, search engines, and system libraries due to its speed and efficiency.