Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri (W), Mumbai – 400 058.

Name – Advika Kharat
UID – 2021300060
Subject – DAA
Class – SE Comps A

Experiment No. 2

Aim – Experiment on finding the running time of an algorithm.

Details – The understanding of running time of algorithms is explored by implementing two basic sorting algorithms namely Merge and Quick Sort. These algorithms work as follows.

Merge Sort - A sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array. In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.

Quick Sort - Like Merge Sort, QuickSort is a Divide and Conquer algorithm. It picks an element as a pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

- Always pick the first element as a pivot.
- Always pick the last element as a pivot (implemented below)
- Pick a random element as a pivot.
- Pick the median as the pivot.

The key process in quicksort is a partition(). The target of partitions is, given an array and an element x of an array as the pivot, put x at its correct position in a sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void merge(int arr[], int l,
           int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
```

```c
    }
}

void mergeSort(int arr[],
               int l, int r)
{
    if (l < r)
    {

        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

int main()
{
    FILE *fp;
    fp = fopen("Experiment2.txt", "w");
    int arr[100000];

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

    for (int k = 0; k < arr_size; k++)
    {
        arr[k] = rand() % 100 + 1;
    }

    fprintf(fp, "Unsorted array is \n");
    for (int i = 0; i < arr_size; i++)
        fprintf(fp, "%d ", arr[i]);
    printf("\n");

    printf("\nMerge Sort :  \n");
    fprintf(fp, "\nMerge Sort :  \n");

    for (int k = 100; k <= arr_size; k += 100)
    {
        clock_t start, end;
        double cpu_time_used;
        start = clock();

        mergeSort(arr, 0, k - 1);

        end = clock();
        cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
```

```
        fprintf(fp, "\nTime taken for %d elements %lf seconds.", k,
cpu_time_used);
        printf("\n %d elements %lf ", k, cpu_time_used);
    }
    printf("\n");
    printf("\nQuick Sort :  \n");
    fprintf(fp, "\n");
    fprintf(fp, "\nQuick Sort :  \n");

    for (int k = 100; k <= arr_size; k += 100)
    {
        clock_t start, end;
        double cpu_time_used;
        start = clock();

        quicksort(arr, 0, k - 1);

        end = clock();
        cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;

        fprintf(fp, "\nTime taken for %d elements %lf seconds", k,
cpu_time_used);
        printf("\n %d elements %lf", k, cpu_time_used);
    }
    printf("\n");
    return 0;
}

void quicksort(int number[25], int first, int last)
{
    int i, j, pivot, temp;
    if (first < last)
    {
        pivot = first;
        i = first;
        j = last;
        while (i < j)
        {
            while (number[i] <= number[pivot] && i < last)
                i++;
            while (number[j] > number[pivot])
                j--;
            if (i < j)
            {
                temp = number[i];
                number[i] = number[j];
                number[j] = temp;
            }
```

```
        }
        temp = number[pivot];
        number[pivot] = number[j];
        number[j] = temp;
        quicksort(number, first, j - 1);
        quicksort(number, j + 1, last);
    }
}
```

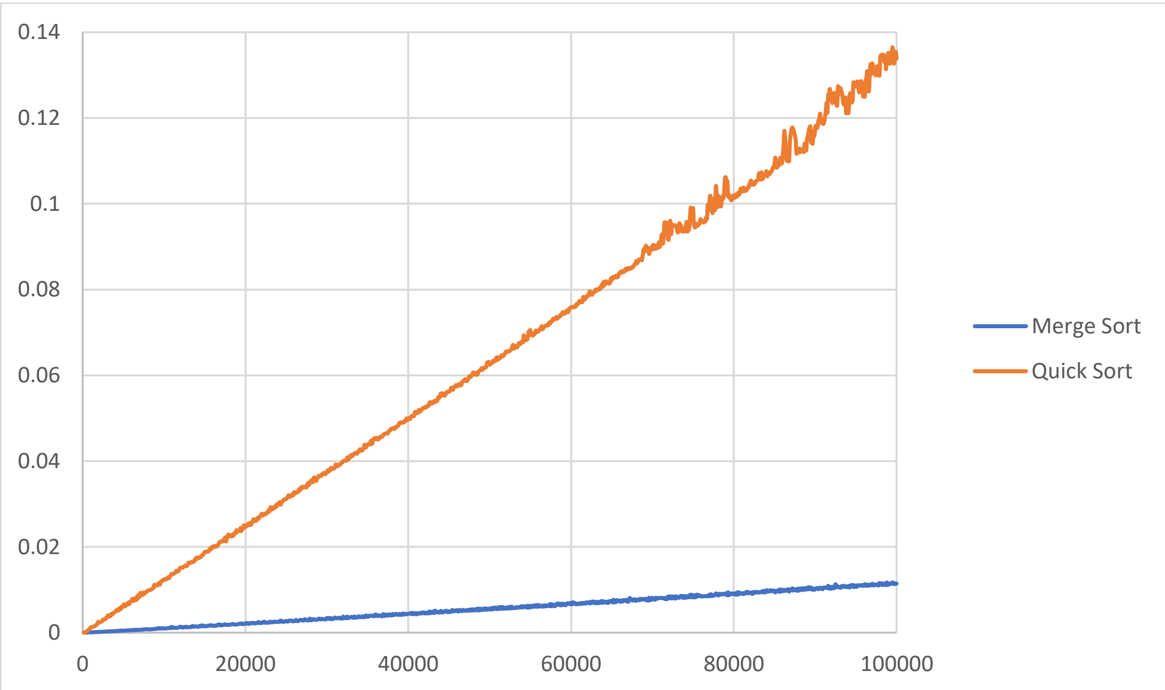
Output:

```
Unsorted array is
42 68 35 1 70 25 79 59 63 65 6 46 82 28 62 92 96 43 28 37 92 5 3 54 93 83 22 17 19 96 48 27 72 39 70 13 68
 54 13 9 33 46 14 57 22 59 47 83 82 45 97 23 30 62 36 51 74 67 45 60 93 40 54 25 55 11 46 50 87 14 75 23 6
42 30 41 14 75 2 78 16 84 14 93 25 2 93 60 71 29 28 85 76 87 99 71 88 48 5 4 22 64 7 64 11 72 90 41 65 43
 7 69 19 3 8 8 82 13 37 31 15 10 85 57 91 94 97 53 55 46 9 49 92 13 32 15 40 59 23 5 96 53 70 80 39 24 19
9 82 58 51 15 99 59 62 64 57 8 79 90 36 66 76 87 87 34 61 31 49 29 93 34 41 67 36 11 100 38 93 83 29 53 70
94 7 79 49 7 72 67 97 98 21 95 30 89 10 85 70 79 18 16 27 85 69 7 29 98 19 91 100 86 87 100 21 11 72 14 16
 84 68 37 26 19 38 29 20 78 38 92 57 96 61 2 94 33 37 81 76 8 85 75 20 91 77 42 52 30 91 75 73 92 90 88 91
6 82 15 68 94 97 62 47 39 71 39 77 56 18 72 61 53 59 34 56 66 27 2 35 60 94 49 74 93 94 34 38 4 91 12 95 6
 57 62 13 12 60 52 39 71 25 8 8 85 52 2 90 59 42 91 14 13 56 61 94 73 3 77 44 74 67 52 76 29 33 62 70 4 30
8 78 18 41 43 1 1 59 77 55 36 31 32 96 38 24 23 28 30 12 12 55 81 54 94 22 86 29 44 1 56 72 71 53 76 32 41
70 94 78 15 5 85 44 96 12 94 93 99 91 100 91 94 75 50 75 15 74 70 19 66 34 39 12 53 64 9 50 89 46 32 21 7
65 12 41 51 5 84 67 42 71 46 7 89 57 8 67 58 79 100 75 56 92 91 33 59 97 18 91 68 8 35 47 63 3 40 27 96 7
 42 100 68 9 45 70 82 80 9 8 60 7 50 76 27 25 8 75 85 29 47 46 82 15 50 69 39 8 94 5 6 98 18 56 42 44 34 9
66 87 20 60 14 36 29 2 69 70 57 32 79 75 96 44 75 17 25 12 86 23 72 58 96 75 39 84 59 96 23 84 32 4 33 1 9
2 4 91 87 27 4 5 1 69 24 48 97 67 31 66 53 1 18 74 87 69 98 80 25 29 61 59 26 68 13 35 53 70 63 71 12 36 9
1 71 46 26 49 75 75 44 30 37 64 17 59 35 2 62 56 16 75 83 40 83 25 1 76 38 77 1 25 65 38 96 26 30 52 89 95
2 32 25 8 26 2 84 70 89 65 64 25 35 71 52 98 25 12 68 66 41 80 56 29 62 33 28 61 40 67 38 49 49 5 49 66 22
 90 82 75 60 51 47 61 28 34 22 62 19 47 51 64 62 47 59 95 72 46 3 47 21 55 54 61 87 20 36 17 14 36 93 19 6
 38 2 77 5 74 62 98 96 26 86 63 30 61 19 74 80 10 63 25 75 83 76 16 44 47 33 69 85 62 73 87 67 90 52 5 34
3 45 50 86 56 73 90 59 53 30 41 61 47 28 45 23 37 74 55 34 57 52 92 35 32 23 80 3 77 81 70 89 30 86 3 11 2
2 41 26 25 47 1 39 32 22 18 25 85 16 45 53 20 71 68 20 75 88 31 46 17 45 100 63 86 16 22 3 42 5 74 34 53 1
 71 15 11 56 33 66 19 21 38 65 10 4 45 51 42 2 45 28 42 11 97 25 84 99 41 32 71 55 36 4 46 10 15 33 18 31
68 98 10 42 32 100 99 75 88 60 63 46 30 60 25 68 67 69 28 95 50 97 43 24 97 18 12 69 49 82 34 75 89 85 69
 59 6 78 59 12 82 51 71 81 54 28 60 17 98 19 19 78 82 66 53 14 72 77 87 75 43 42 10 98 63 49 43 49 93 37 2
 64 14 32 87 61 67 50 23 61 84 74 67 7 47 75 9 47 71 21 67 70 79 28 39 85 60 68 79 31 22 92 91 94 45 10 8
 40 36 7 40 74 70 74 45 28 96 37 96 3 17 8 23 65 49 9 73 9 91 58 48 32 87 97 7 68 15 91 68 6 12 84 54 40 2
```

```
27 34 72 54 67 46 37 11 68 17 23 40 33 96 65 51 39 88 5 78 10 78 27 89 39 5 96 86 51 78 56 100 90 4 36 88
73 57 95 60 84 72 1 89 90 44 82 48 62 54 80 14 45 69 47 11 64 48 8 89 26 6 46 90 13 22 90 48 32 79 44 76 7
6 92 54 3 3 23 95 5 71 63 19 46 1 57 67 2 73 93 11 98 9 100 6 46 15 97 61 9 14 69 95 85 40 37 44 83 57 29
0 22 49 73 23 29 81 68 49 24 71 81 32 45 2 34 81 100 55 48 39 27 24 75 12 100 27 83 96 70 2 6 92 11 89 76
1 61 45 20 48 76 62 15 84 7 81 71 16 19 1 15 44 13 94 59 80 9 10 38 24 54 54 61 57 73 8 15 89 5 12 54 7 46
1 3 62 1 2 55 52 1 78 2 81 54 30
Merge Sort :

Time taken for 100 elements 0.000000 seconds.
Time taken for 200 elements 0.000000 seconds.
Time taken for 300 elements 0.000000 seconds.
Time taken for 400 elements 0.000000 seconds.
Time taken for 500 elements 0.000000 seconds.
Time taken for 600 elements 0.000000 seconds.
Time taken for 700 elements 0.000000 seconds.
Time taken for 800 elements 0.000000 seconds.
Time taken for 900 elements 0.003000 seconds.
Time taken for 1000 elements 0.000000 seconds.
Time taken for 1100 elements 0.000000 seconds.
Time taken for 1200 elements 0.000000 seconds.
Time taken for 1300 elements 0.000000 seconds.
Time taken for 1400 elements 0.000000 seconds.
Time taken for 1500 elements 0.000000 seconds.
Time taken for 1600 elements 0.000000 seconds.
Time taken for 1700 elements 0.000000 seconds.
Time taken for 1800 elements 0.000000 seconds.
Time taken for 1900 elements 0.000000 seconds.
Time taken for 2000 elements 0.000000 seconds.
Time taken for 2100 elements 0.000000 seconds.
Time taken for 2200 elements 0.000000 seconds.
Time taken for 2300 elements 0.000000 seconds.
Time taken for 2400 elements 0.000000 seconds.
Time taken for 2500 elements 0.000000 seconds.
Time taken for 2600 elements 0.000000 seconds.
Time taken for 2700 elements 0.000000 seconds.
Time taken for 2800 elements 0.000000 seconds.
Time taken for 2900 elements 0.000000 seconds.
Time taken for 3000 elements 0.002000 seconds.
Time taken for 3100 elements 0.000000 seconds.
```

```
Time taken for 99100 elements 0.009000 seconds.
Time taken for 99200 elements 0.011000 seconds.
Time taken for 99300 elements 0.008000 seconds.
Time taken for 99400 elements 0.015000 seconds.
Time taken for 99500 elements 0.009000 seconds.
Time taken for 99600 elements 0.009000 seconds.
Time taken for 99700 elements 0.015000 seconds.
Time taken for 99800 elements 0.010000 seconds.
Time taken for 99900 elements 0.008000 seconds.
Time taken for 100000 elements 0.013000 seconds.

Quick Sort :

Time taken for 100 elements 0.000000 seconds
Time taken for 200 elements 0.000000 seconds
Time taken for 300 elements 0.000000 seconds
Time taken for 400 elements 0.000000 seconds
Time taken for 500 elements 0.000000 seconds
Time taken for 600 elements 0.007000 seconds
Time taken for 700 elements 0.000000 seconds
Time taken for 800 elements 0.003000 seconds
Time taken for 900 elements 0.000000 seconds
Time taken for 1000 elements 0.003000 seconds
Time taken for 1100 elements 0.002000 seconds
Time taken for 1200 elements 0.002000 seconds
Time taken for 1300 elements 0.001000 seconds
Time taken for 1400 elements 0.002000 seconds
Time taken for 1500 elements 0.002000 seconds
Time taken for 1600 elements 0.001000 seconds
Time taken for 1700 elements 0.001000 seconds
Time taken for 1800 elements 0.000000 seconds
Time taken for 1900 elements 0.009000 seconds
Time taken for 2000 elements 0.000000 seconds
Time taken for 2100 elements 0.000000 seconds
Time taken for 2200 elements 0.003000 seconds
Time taken for 2300 elements 0.000000 seconds
Time taken for 2400 elements 0.002000 seconds
Time taken for 2500 elements 0.002000 seconds
Time taken for 2600 elements 0.001000 seconds
```

Graph:



Conclusion : From the graph, we can see that quick sort takes around 0.14 seconds to sort an array of 100,00 random numbers while merge sort only takes 0.01 seconds. Hence, it is obvious that merge sort is much quicker and more efficient than quick sort is.