# DAA EXPERIMENT NO. 2

**NAME:** Rahul Chalwadi

**UID:** 2022701002

CSE DS (BATCH A(d1))

**AIM:** Experiment based on divide and conquer approach.

**Problem Definition & Assumptions** – For this experiment, you need to implement two sorting algorithms namely Quicksort and Merge sort methods. Compare these algorithms based on time and space complexity. Time required for sorting algorithms can be performed using high_resolution_clock::now() under namespace std::chrono. You have to generate 1,00,000 integer numbers using C/C++ Rand function and save them in a text file. Both the sorting algorithms uses these 1,00,000 integer numbers as input as follows. Each sorting algorithm sorts a block of 100,200,300,...,100000 integer numbers with array indexes numbers A[0..99], A[100..199], A[200..299],..., A[99900..99999]. You need to use high_resolution_clock::now() function to find the time required for 100, 200, 300.... 100000 integer numbers. Finally, compare two algorithms namely Quicksort and Merge sort by plotting the time required to sort integers using LibreOffice Calc/MS Excel. The x-axis of 2-D plot represents the block no. of 1000 blocks. The y-axis of 2-D plot represents the tunning time to sort 1000 blocks of 100,200,300,...,100000 integer numbers.

## ALGORITHM:

<u>Quick Sort Function:</u>

**Step 1:** Start.

**Step 2:** Check if the left index is less than the right index.

**Step 3:** Select the last element of the array (arr[right]) as the pivot element.

**Step 4:** Initialize a variable i to left - 1.

**Step 5:** Iterate over the sub-array from left to right-1. a. If the current element (arr[j]) is less than the pivot element, increment i and swap arr[i] and arr[j].

**Step 6:** Swap arr[i+1] and arr[right] to place the pivot element in its correct position.

**Step 7:** Set p to i + 1, the index of the pivot element.

**Step 8:** Recursively call quickSort() on the left sub-array, from left to p-1.

**Step 9:** Recursively call quickSort() on the right sub-array, from p+1 to right.

**Step 10:** Stop.

<u>Merge Sort Function:</u>

**Step 1:** Start.

**Step 2:** Declare an array and left, right, mid variable.

**Step 3:** Perform merge function.

```
mergesort(array,left,right)
mergesort (array, left, right)
if left > right
return
mid= (left+right)/2
mergesort(array, left, mid)
mergesort(array, mid+1, right)
merge(array, left, mid, right)
```

**Step 4:** Stop.

Main Function:

**Step 1:** Start

**Step 3:** In the main function, open a file "exp2.txt" for writing and initialize the random number generator with srand((unsigned int) time(NULL)).

**Step 4:** Generate 1000 blocks of 100 random numbers each and store them in the file.

**Step 5:** Close the file after writing.

**Step 6:** Open the file "exp2.txt" for reading.

**Step 7:** For each block of 100 elements, read the elements from the file into two arrays arr and arr1.

**Step 8:** Sort the elements in the arr using the quick_sort function.

**Step 9:** Measure the time taken for sorting using the clock() function and store it in the time_taken_quick_sort variable.

**Step 10:** Sort the elements in the arr1 using the merge_sort function.

**Step 11:** Measure the time taken for sorting using the clock() function and store it in the time_taken_merge_sort variable.

**Step 12:** Print the block number, time taken for quick sort, and time taken for merge sort.

**Step 13:** Repeat the process for 1000 blocks.

**Step 14:** Close the file after reading.

**Step 15:** Stop.

**CODE:**

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

void quickSort(int arr[], int left, int right) {
  if (left < right) {
        int pivot = arr[right];
        int i = left - 1;
        for (int j = left; j < right; j++) {
        if (arr[j] < pivot) {
        i++;
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
        }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[right];
        arr[right] = temp;
        int p = i + 1; // p is the pivot element
        quickSort(arr, left, p - 1);
        quickSort(arr, p + 1, right);
  }
}

void merge(int arr[], int l, int m, int r)
```

```
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    // Create temp arrays
    int L[n1], R[n2];

    // Copy data to temp arrays
    // L[] and R[]
    for (i = 0; i < n1; i++)
    L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
    R[j] = arr[m + 1 + j];

    // Merge the temp arrays back
    // into arr[l..r]
    // Initial index of first subarray
    i = 0;

    // Initial index of second subarray
    j = 0;

    // Initial index of merged subarray
    k = l;
    while (i < n1 && j < n2)
    {
    if (L[i] <= R[j])
```

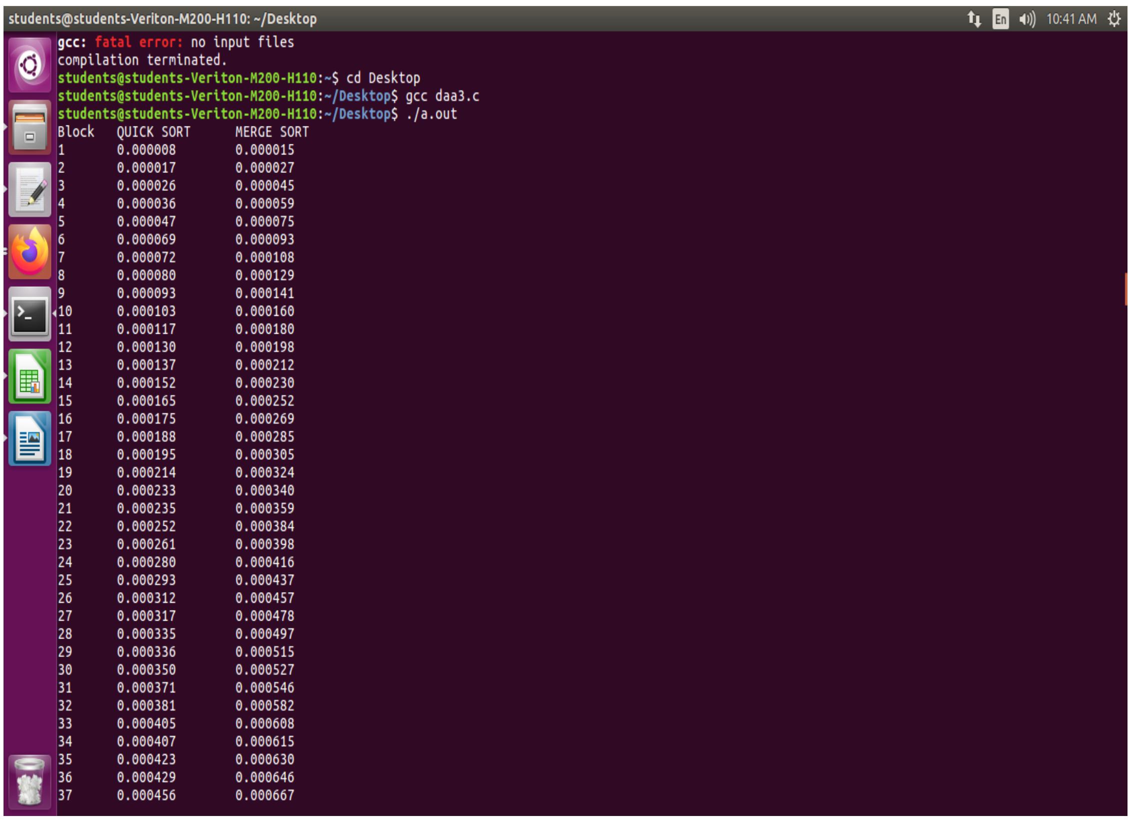
```
{
arr[k] = L[i];
i++;
}
else
{
arr[k] = R[j];
j++;
}
k++;
}

// Copy the remaining elements
// of L[], if there are any
while (i < n1) {
arr[k] = L[i];
i++;
k++;
}

// Copy the remaining elements of
// R[], if there are any
while (j < n2)
{
arr[k] = R[j];
j++;
k++;
}
```

```
}

void mergeSort(int arr[], int l, int r)
{
	if (l < r)
	{
	// Same as (l+r)/2, but avoids
	// overflow for large l and h
	int m = l + (r - l) / 2;

	// Sort first and second halves
	mergeSort(arr, l, m);
	mergeSort(arr, m + 1, r);

	merge(arr, l, m, r);
	}
}


void main() {
	FILE *fp;
	fp  = fopen ("EXP2.txt", "w");
	srand((unsigned int) time(NULL));
	for(int block=0;block<1000;block++) {
	for(int i=0;i<100;i++) {
	int number = (int)(((float) rand() / (float)(RAND_MAX))*100000);
	fprintf(fp,"%d ",number);
	}
```

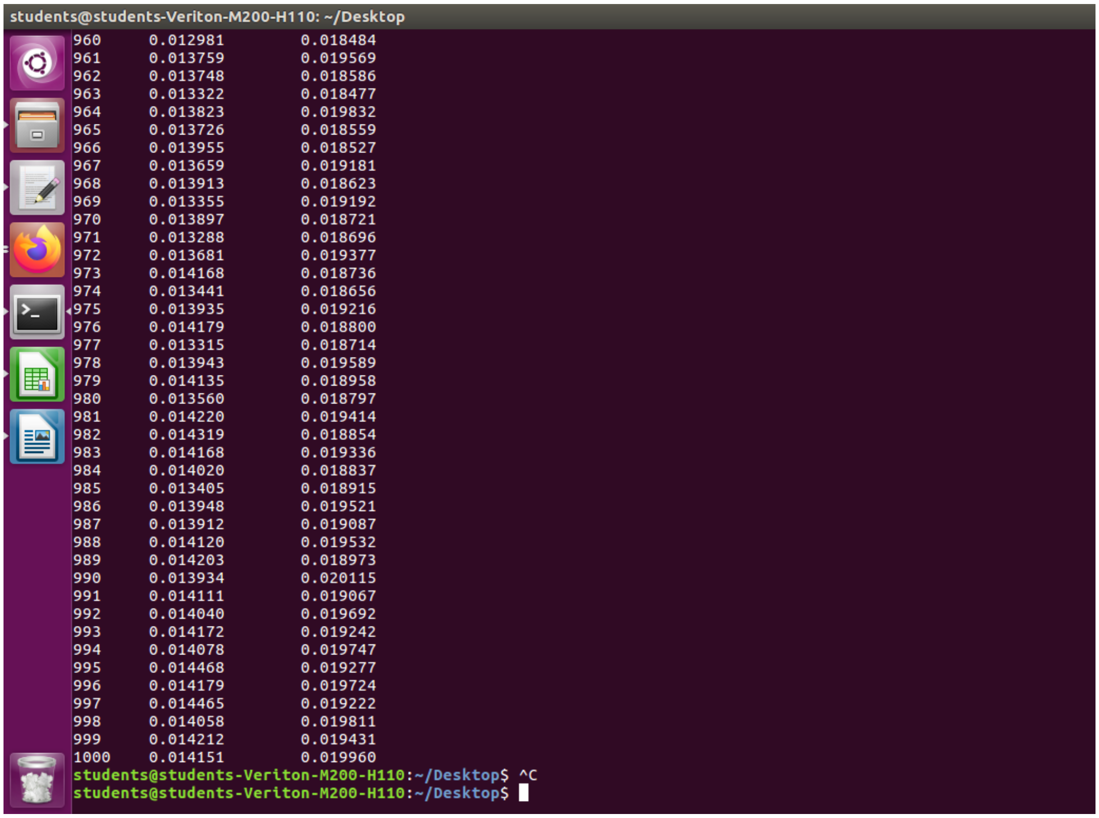
```
        fputs("\n",fp);
        }
        fclose (fp);
fp = fopen("EXP2.txt", "r");
printf("Block\tQUICK SORT\tMERGE SORT\n");
for(int block=0;block<1000;block++) {
        clock_t t1,t2;
        int arr[(block+1)*100];
        int arr1[(block+1)*100];
        for(int i=0;i<(block+1)*100;i++){
        fscanf(fp, "%d", &arr[i]);
        arr1[i] = arr[i];
        }
        fseek(fp, 0, SEEK_SET);

        //CALLING QUICKSORT
        t1 = clock();
        int size = sizeof(arr) / sizeof(arr[0]);
        quickSort(arr, 0, size - 1);
        t1 = clock() - t1;



        t2 = clock();
        size = sizeof(arr1) / sizeof(arr1[0]);
        mergeSort(arr1, 0, size - 1);
        t2 = clock() - t2;
        double quick_sort_time = ((double)t1)/CLOCKS_PER_SEC;
        double merge_sort_time = ((double)t2)/CLOCKS_PER_SEC;
```

```
        printf("%d\t%f\t%f\n",(block+1), quick_sort_time, merge_sort_time);
    }
        fclose(fp);
}
```

**OUTPUT:**

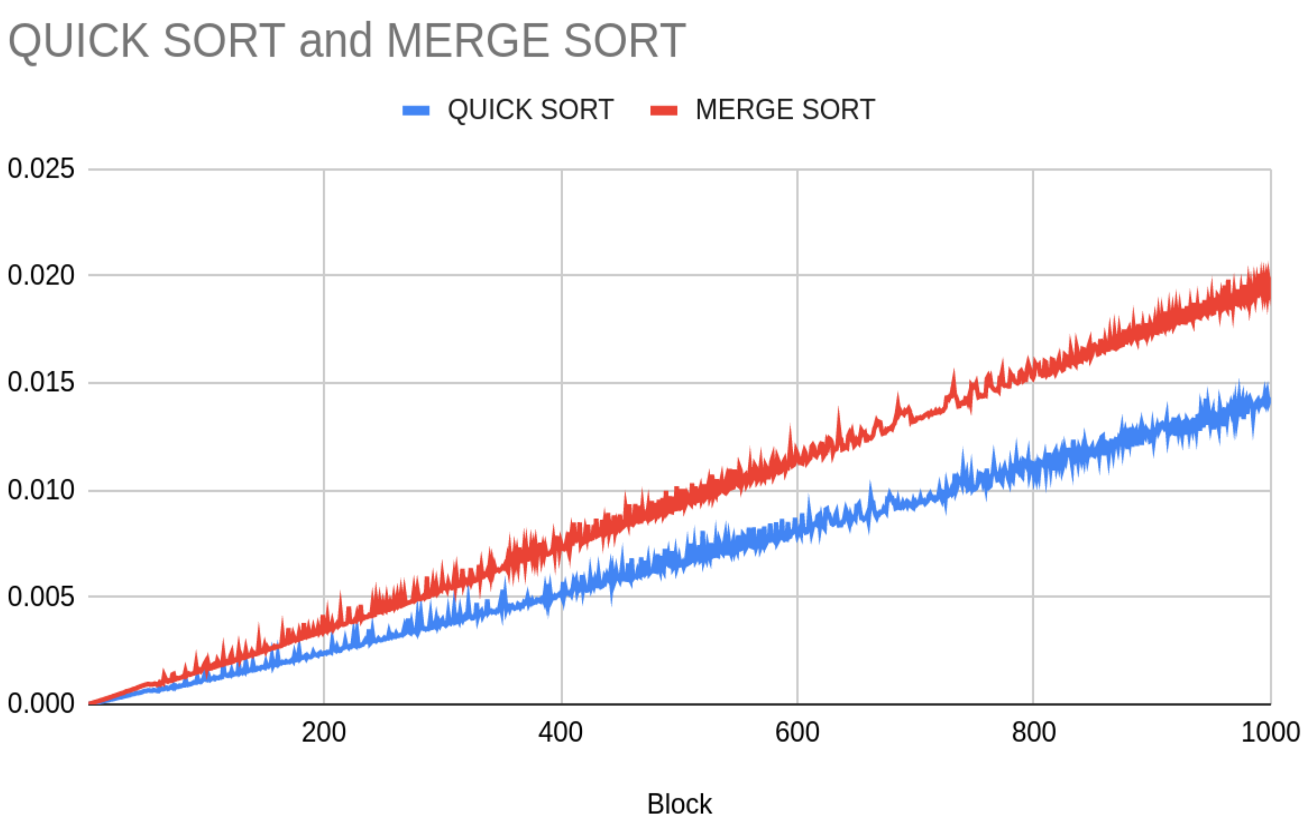


```
students@students-Veriton-M200-H110: ~/Desktop                10:41 AM
gcc: fatal error: no input files
compilation terminated.
students@students-Veriton-M200-H110:~$ cd Desktop
students@students-Veriton-M200-H110:~/Desktop$ gcc daa3.c
students@students-Veriton-M200-H110:~/Desktop$ ./a.out
Block   QUICK SORT      MERGE SORT
1       0.000008        0.000015
2       0.000017        0.000027
3       0.000026        0.000045
4       0.000036        0.000059
5       0.000047        0.000075
6       0.000069        0.000093
7       0.000072        0.000108
8       0.000080        0.000129
9       0.000093        0.000141
10      0.000103        0.000160
11      0.000117        0.000180
12      0.000130        0.000198
13      0.000137        0.000212
14      0.000152        0.000230
15      0.000165        0.000252
16      0.000175        0.000269
17      0.000188        0.000285
18      0.000195        0.000305
19      0.000214        0.000324
20      0.000233        0.000340
21      0.000235        0.000359
22      0.000252        0.000384
23      0.000261        0.000398
24      0.000280        0.000416
25      0.000293        0.000437
26      0.000312        0.000457
27      0.000317        0.000478
28      0.000335        0.000497
29      0.000336        0.000515
30      0.000350        0.000527
31      0.000371        0.000546
32      0.000381        0.000582
33      0.000405        0.000608
34      0.000407        0.000615
35      0.000423        0.000630
36      0.000429        0.000646
37      0.000456        0.000667
```



```
students@students-Veriton-M200-H110: ~/Desktop
960     0.012981        0.018484
961     0.013759        0.019569
962     0.013748        0.018586
963     0.013322        0.018477
964     0.013823        0.019832
965     0.013726        0.018559
966     0.013955        0.018527
967     0.013659        0.019181
968     0.013913        0.018623
969     0.013355        0.019192
970     0.013897        0.018721
971     0.013288        0.018696
972     0.013681        0.019377
973     0.014168        0.018736
974     0.013441        0.018656
975     0.013935        0.019216
976     0.014179        0.018800
977     0.013315        0.018714
978     0.013943        0.019589
979     0.014135        0.018958
980     0.013560        0.018797
981     0.014220        0.019414
982     0.014319        0.018854
983     0.014168        0.019336
984     0.014020        0.018837
985     0.013405        0.018915
986     0.013948        0.019521
987     0.013912        0.019087
988     0.014120        0.019532
989     0.014203        0.018973
990     0.013934        0.020115
991     0.014111        0.019067
992     0.014040        0.019692
993     0.014172        0.019242
994     0.014078        0.019747
995     0.014468        0.019277
996     0.014179        0.019724
997     0.014465        0.019222
998     0.014058        0.019811
999     0.014212        0.019431
1000    0.014151        0.019960
students@students-Veriton-M200-H110:~/Desktop$ ^C
students@students-Veriton-M200-H110:~/Desktop$
```

**RESULT:**



**RESULT ANALYSIS**

The following graph is representation of amount of time (in seconds) required to sort block of integers using Quick sort & Merge sort algorithm.

In the above graph, time values of sorting algorithm are plotted on y-axis against no. of blocks on x-axis. The maximum no. of block is 1000 on X-axis.

Maximum amount of time required to sort $1000^{th}$ block using quick sort is approx. 0.019 seconds and using merge sort is 0.031 seconds.

Quick sort and Merge sort require almost similar with little variation of time. Comparatively, merge sort requires slightly more time than quick sort.

**CONCLUSION:** By performing this experiment i understood about the working and implementation of merge sort and quick sort algorithm and verified its time complexity as well.