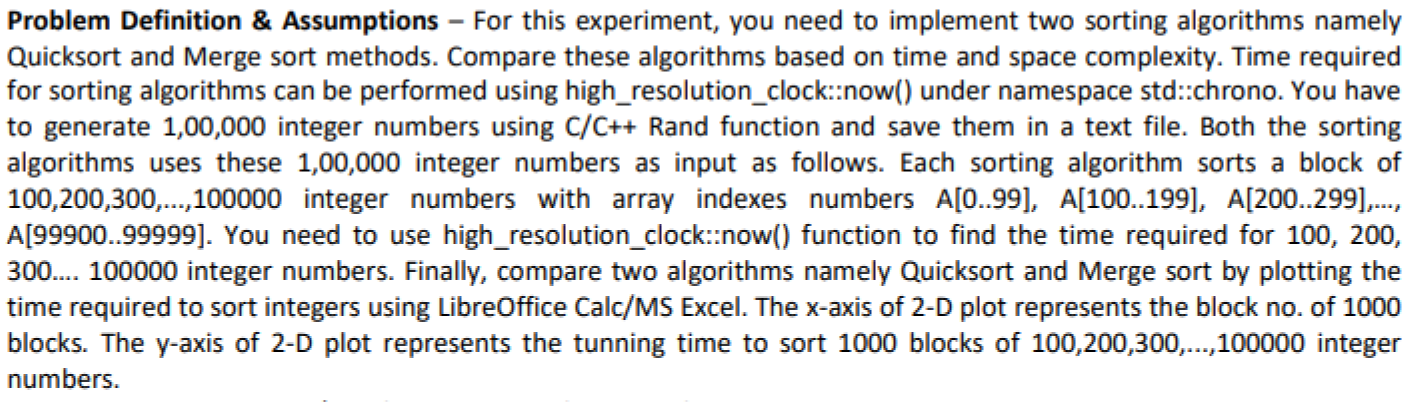
DAA EXPERIMENT NO. 2

NAME: Glen Dsouza CSE DS (BATCH A)

UID: 2022701002

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.

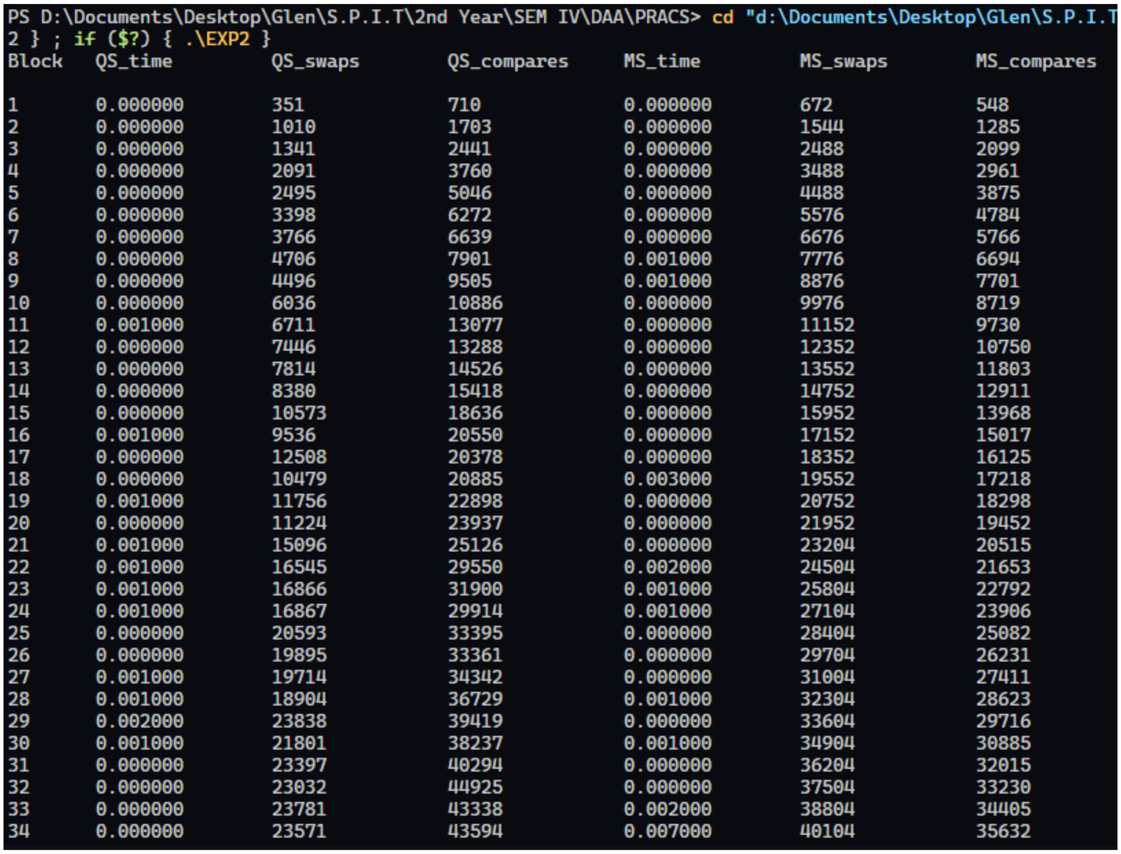
Step 14: Close the file after reading.

Step 15: Stop.

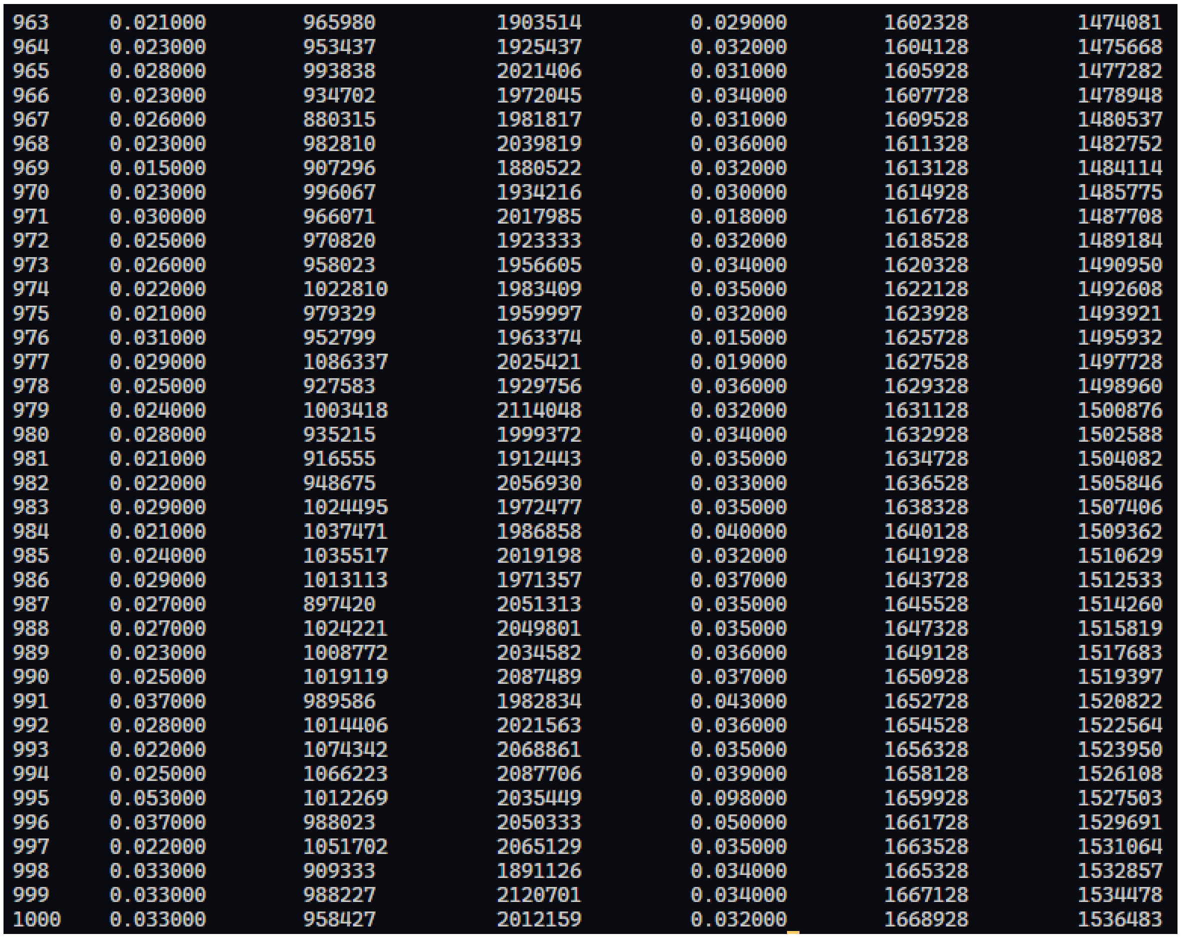
CODE:

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

void quickSort(int arr[], int left, int right, int *qs_swaps, int *qs_compares) {
  if (left < right) {
    int pivot = arr[right];
    int i = left - 1;
    for (int j = left; j < right; j++) {
      (*qs_compares)++;
      if (arr[j] < pivot) {
        i++;
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
        (*qs_swaps)++;
      }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[right];
    arr[right] = temp;
    (*qs_swaps)++; //icrement no. of swaps

    int p = i + 1; // p is the pivot element

    quickSort(arr, left, p - 1, qs_swaps, qs_compares);
    quickSort(arr, p + 1, right, qs_swaps, qs_compares);
  }
}

void merge(int arr[], int l, int m, int r, int *ms_swaps, int *ms_compares) {
    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) {
        (*ms_compares)++;
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else  {
            arr[k] = R[j];
            j++;
        }
        (*ms_swaps)++;
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
        (*ms_swaps)++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
        (*ms_swaps)++;
    }
}

void mergeSort(int arr[], int l, int r, int *ms_swaps, int *ms_compares) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m, ms_swaps, ms_compares);
        mergeSort(arr, m + 1, r, ms_swaps, ms_compares);
        merge(arr, l, m, r, ms_swaps, ms_compares);
    }
}

void main() {
    FILE *fp;

    fp  = fopen ("random.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("random.txt", "r");
   printf("Block\t%-8s\t%-8s\t%-8s\t%-8s\t%-8s\t%-8s\n\n","QS_time", "QS_swaps",
"QS_compares","MS_time","QS_swaps","MS_compares");
   for(int block=0;block<1000;block++) {
    int qs_swaps = 0, qs_compares = 0, ms_swaps = 0, ms_compares = 0;
    clock_t t,t1;

    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);
    t = clock();
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1, &qs_swaps, &qs_compares);
    t = clock() - t;
    t1 = clock();
    n = sizeof(arr1) / sizeof(arr1[0]);
    mergeSort(arr1, 0, n - 1, &ms_swaps, &ms_compares);
    t1 = clock() - t1;
    double time_taken_quick_sort = ((double)t)/CLOCKS_PER_SEC;
    double time_taken_merge_sort = ((double)t1)/CLOCKS_PER_SEC;
    printf("%d\t%-8f\t%-8d\t%-8d\t%-8f\t%-8d\t%-8d\n",(block+1),time_taken_quick_sort, qs_swaps, qs_compares,
time_taken_merge_sort, ms_swaps, ms_compares);
   }
    fclose(fp);
}
```

OUTPUT:

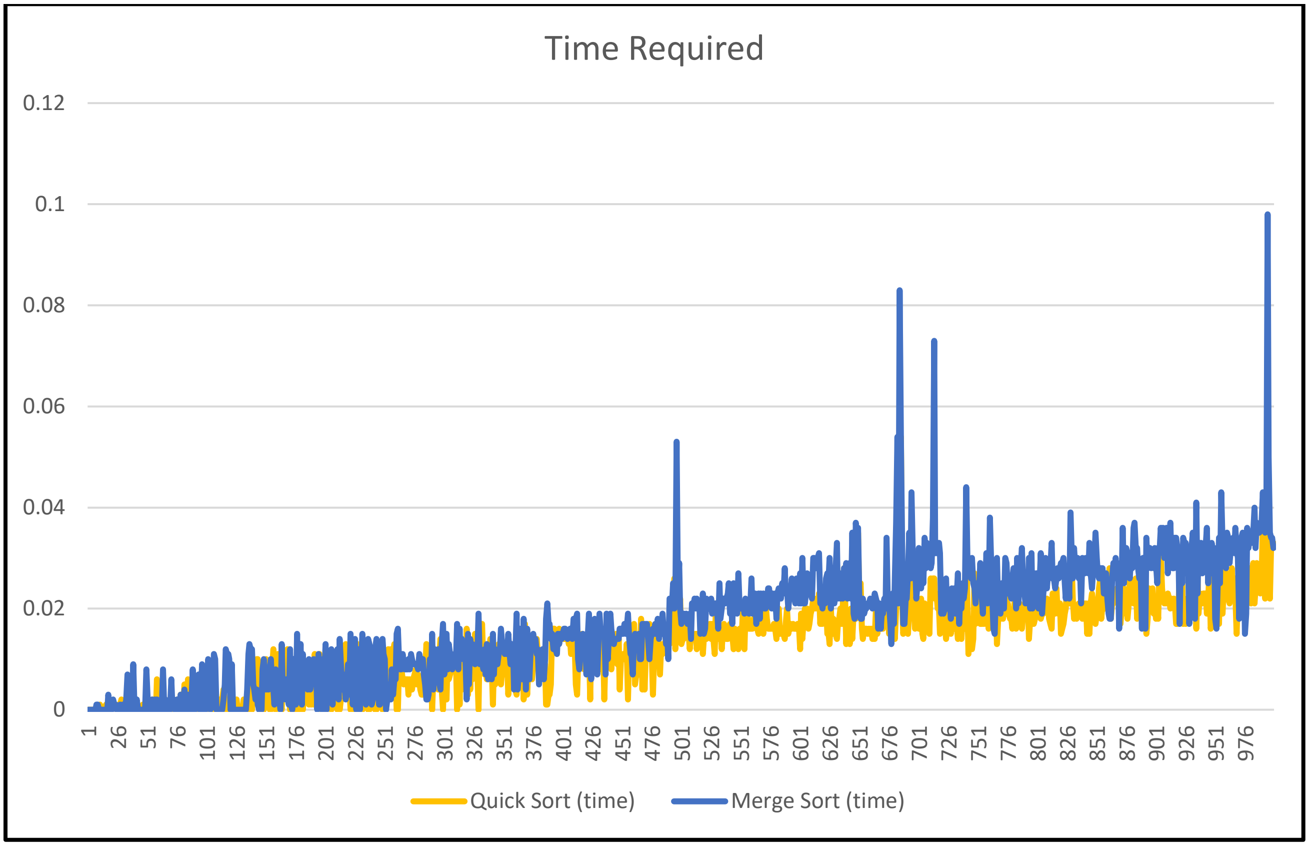


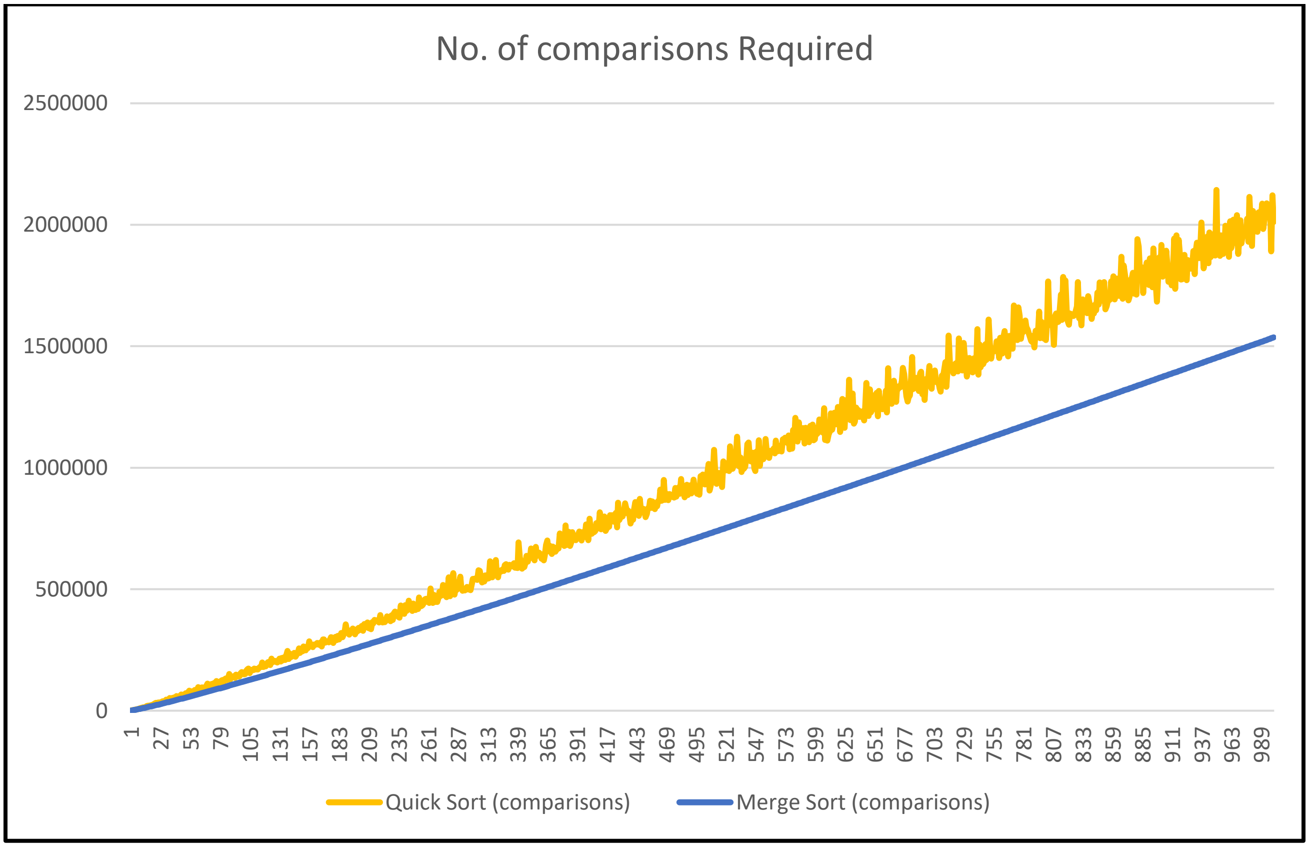
```
PS D:\Documents\Desktop\Glen\S.P.I.T\2nd Year\SEM IV\DAA\PRACS> cd "d:\Documents\Desktop\Glen\S.P.I.T
2 } ; if ($?) { .\EXP2 }
Block   QS_time         QS_swaps        QS_compares     MS_time         MS_swaps        MS_compares

1       0.000000        351             710             0.000000        672             548
2       0.000000        1010            1703            0.000000        1544            1285
3       0.000000        1341            2441            0.000000        2488            2099
4       0.000000        2091            3760            0.000000        3488            2961
5       0.000000        2495            5046            0.000000        4488            3875
6       0.000000        3398            6272            0.000000        5576            4784
7       0.000000        3766            6639            0.000000        6676            5766
8       0.000000        4706            7901            0.001000        7776            6694
9       0.000000        4496            9505            0.001000        8876            7701
10      0.000000        6036            10886           0.000000        9976            8719
11      0.001000        6711            13077           0.000000        11152           9730
12      0.000000        7446            13288           0.000000        12352           10750
13      0.000000        7814            14526           0.000000        13552           11803
14      0.000000        8380            15418           0.000000        14752           12911
15      0.000000        10573           18636           0.000000        15952           13968
16      0.001000        9536            20550           0.000000        17152           15017
17      0.000000        12508           20378           0.000000        18352           16125
18      0.000000        10479           20885           0.003000        19552           17218
19      0.001000        11756           22898           0.000000        20752           18298
20      0.000000        11224           23937           0.000000        21952           19452
21      0.001000        15096           25126           0.000000        23204           20515
22      0.001000        16545           29550           0.002000        24504           21653
23      0.001000        16866           31900           0.001000        25804           22792
24      0.001000        16867           29914           0.001000        27104           23906
25      0.000000        20593           33395           0.000000        28404           25082
26      0.000000        19895           33361           0.000000        29704           26231
27      0.001000        19714           34342           0.000000        31004           27411
28      0.001000        18904           36729           0.001000        32304           28623
29      0.002000        23838           39419           0.000000        33604           29716
30      0.001000        21801           38237           0.001000        34904           30885
31      0.000000        23397           40294           0.000000        36204           32015
32      0.000000        23032           44925           0.000000        37504           33230
33      0.000000        23781           43338           0.002000        38804           34405
34      0.000000        23571           43594           0.007000        40104           35632
```

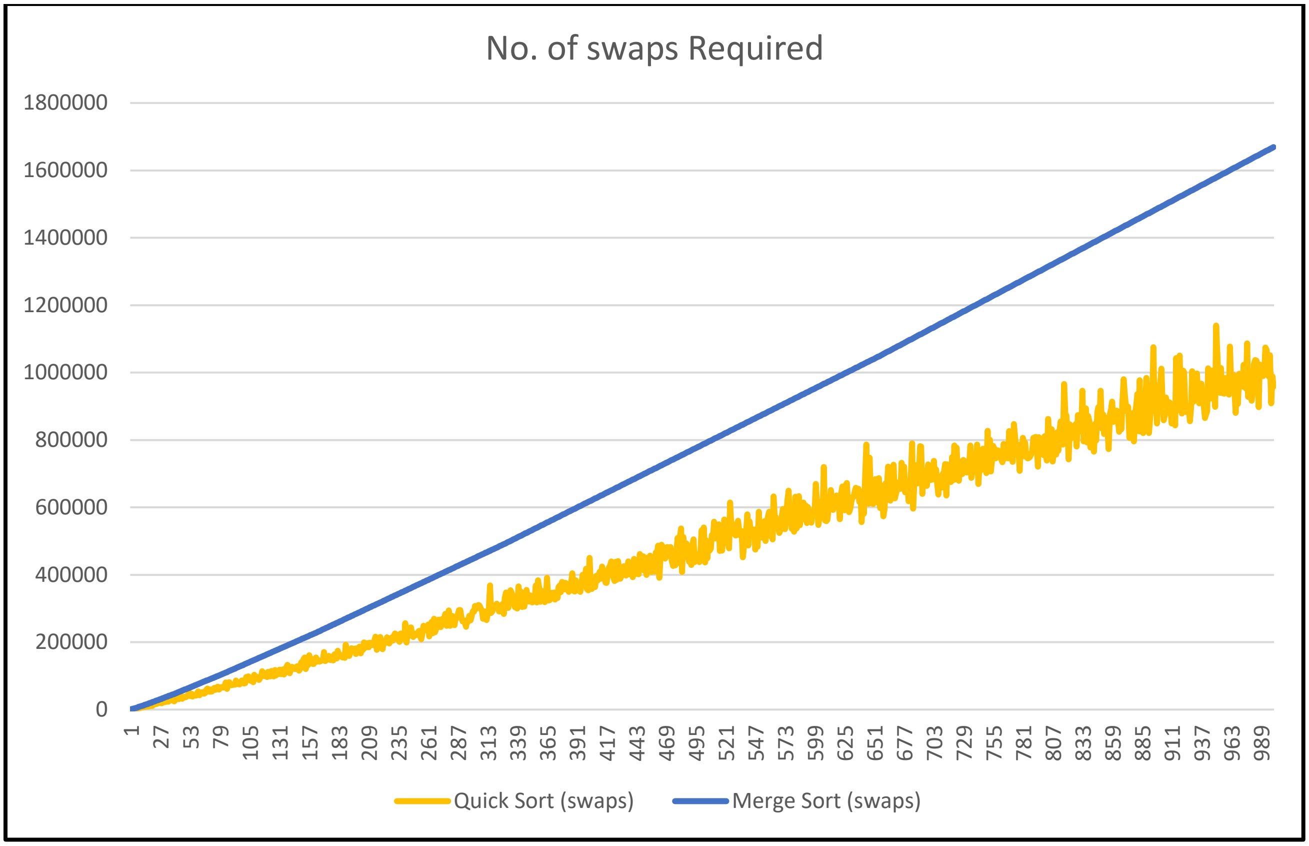


```
963     0.021000        965980          1903514         0.029000        1602328         1474081
964     0.023000        953437          1925437         0.032000        1604128         1475668
965     0.028000        993838          2021406         0.031000        1605928         1477282
966     0.023000        934702          1972045         0.034000        1607728         1478948
967     0.026000        880315          1981817         0.031000        1609528         1480537
968     0.023000        982810          2039819         0.036000        1611328         1482752
969     0.015000        907296          1880522         0.032000        1613128         1484114
970     0.023000        996067          1934216         0.030000        1614928         1485775
971     0.030000        966071          2017985         0.018000        1616728         1487708
972     0.025000        970820          1923333         0.032000        1618528         1489184
973     0.026000        958023          1956605         0.034000        1620328         1490950
974     0.022000        1022810         1983409         0.035000        1622128         1492608
975     0.021000        979329          1959997         0.032000        1623928         1493921
976     0.031000        952799          1963374         0.015000        1625728         1495932
977     0.029000        1086337         2025421         0.019000        1627528         1497728
978     0.025000        927583          1929756         0.036000        1629328         1498960
979     0.024000        1003418         2114048         0.032000        1631128         1500876
980     0.028000        935215          1999372         0.034000        1632928         1502588
981     0.021000        916555          1912443         0.035000        1634728         1504082
982     0.022000        948675          2056930         0.033000        1636528         1505846
983     0.029000        1024495         1972477         0.035000        1638328         1507406
984     0.021000        1037471         1986858         0.040000        1640128         1509362
985     0.024000        1035517         2019198         0.032000        1641928         1510629
986     0.029000        1013113         1971357         0.037000        1643728         1512533
987     0.027000        897420          2051313         0.035000        1645528         1514260
988     0.027000        1024221         2049801         0.035000        1647328         1515819
989     0.023000        1008772         2034582         0.036000        1649128         1517683
990     0.025000        1019119         2087489         0.037000        1650928         1519397
991     0.037000        989586          1982834         0.043000        1652728         1520822
992     0.028000        1014406         2021563         0.036000        1654528         1522564
993     0.022000        1074342         2068861         0.035000        1656328         1523950
994     0.025000        1066223         2087706         0.039000        1658128         1526108
995     0.053000        1012269         2035449         0.098000        1659928         1527503
996     0.037000        988023          2050333         0.050000        1661728         1529691
997     0.022000        1051702         2065129         0.035000        1663528         1531064
998     0.033000        909333          1891126         0.034000        1665328         1532857
999     0.033000        988227          2120701         0.034000        1667128         1534478
1000    0.033000        958427          2012159         0.032000        1668928         1536483
```

RESULT:


Time Required
0.12
0.1
0.08
0.06
0.04
0.02
0
1
26
51
76
101
126
151
176
201
226
251
276
301
326
351
376
401
426
451
476
501
526
551
576
601
626
651
676
701
726
751
776
801
826
851
876
901
926
951
976
Quick Sort (time)
Merge Sort (time)


No. of comparisons Required
2500000
2000000
1500000
1000000
500000
0
1
27
53
79
105
131
157
183
209
235
261
287
313
339
365
391
417
443
469
495
521
547
573
599
625
651
677
703
729
755
781
807
833
859
885
911
937
963
989
Quick Sort (comparisons)
Merge Sort (comparisons)



RESULT ANALYSIS:

The 1st graph is representation of amount of time (in seconds) required to sort block of integers using Quick sort & Merge sort algorithm.

The 2nd graph is representation of no. of comparisons required to sort block of integers using Quick sort & Merge sort algorithm.

The 3rd graph is representation of no. of swaps required to sort block of integers using Quick sort & Merge sort algorithm.

In the above 3 graphs, (time values, no. of comparisons, no. of swaps) 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 1000th block using quick sort is approx. 0.019 seconds and using merge sort is 0.031 seconds.

Merge sort requires comparatively less no. of comparisons but more no. of swaps than Quick sort

From the above 3 graphs, we can conclude Quick sort and Merge sort require almost similar with little variation of time. Comparatively, merge sort requires slightly more time than quick sort.

CONCLUSION: In this experiment quick sort & merge sort were implemented and their runtime, no. of comparisons and no. of swaps across 1000 block of 100 integers was plotted on a graph.