NAME	**OM CHANDRA**
UID NO.	**2021700014**
EXPERIMENT NO.	**2**

AIM:	Experiment on finding the running time of an algorithm.
Program 1	
PROBLEM STATEMENT :	For this experiment, you need to implement two sorting algorithms namely Quick and Merge sort methods. Compare these algorithms based on time and space complexity. Time required to sorting algorithms can be performed using high_resolution_clock::now() under namespace std::chrono. You have togenerate1,00,000 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 integers numbers with array indexes numbers A[0..99], A[0..199], A[0..299],..., A[0..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 Insertion and Selection by plotting the time required to sort 100000 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 running time to sort 1000 blocks of 100,200,300,...,100000 integer numbers. Note – You have to use C/C++ file processing functions for reading and writing randomly generated 100000 integer numbers.

ALGORITHM/ THEORY:	1: Start. 2: Include the required libraries stdio.h, stdlib.h, time.h, and limits.h. 3: Define two sorting functions as per problem statemebt quick_sort and merge_sort. 4: In the main function, using file handling open the file for writing. 5: Generate 1000 blocks of 100 random numbers each and store them in the file. 6: Close the file after writing. 7: Open the file for reading. 8: For each block of 100 elements, read the elements from the file into two arrays. 9: Sort the elements of array using the quick_sort function. 10: Use clock() to measure the time taken by the algorithm, and store the value inside a variable.

	11: Sort the elements of array using the merge_sort function. 12: Use clock() to measure the time taken by the algorithm, and store the value inside a variable. 13: Display the number of blocks and time taken by both of the algorithm to sort a specific blocks. 14: Repeat the process for 1000 blocks. 15: Close the file after reading. 16: Stop.

PROGRAM:

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

unsigned long long noOfComparison1,noOfComparison2;


void quickSort(int arr[], int left, int right,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;
      }
    }
    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,qs_compares);
quickSort(arr, p + 1, right,qs_compares);
  }
```

```
}
erge(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)
        {
                                noOfComparison2++;
        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++;
}
```

```
}       ergeSort(int arr[], int l, int r)

void m noOfComparison2=0;
{       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() {

unsigned long long noOfComparison_quickSort,noOfCompariosn_mergeSort;

                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 \t No of Comparison in QuickSort \t No of
Comparison in MergeSort \n");
  for(int block=0;block<1000;block++) {

        clock_t t1,t2;   int
arr[(block+1)*100];    int
arr1[(block+1)*100];
        int qs_compares = 0;
                                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, &qs_compares);
        //noOfComparison_quickSort = noOfComparison1;
        t1 = clock() - t1;
                                                double quick_sort_time =
((double)t1)/CLOCKS_PER_SEC;
```


```
t2 = clock();
size = sizeof(arr1) / sizeof(arr1[0]);
mergeSort(arr1, 0, size - 1);
noOfCompariosn_mergeSort = noOfComparison2; t2
= clock() - t2;
double merge_sort_time = ((double)t2)/CLOCKS_PER_SEC;
        printf("%d \t %f \t %f%26d%26llu\n",(block+1),quick_sort_time,
merge_sort_time,qs_compares,noOfCompariosn_mergeSort);
```

```
    }       fclose(fp);

}
```

RESULT:

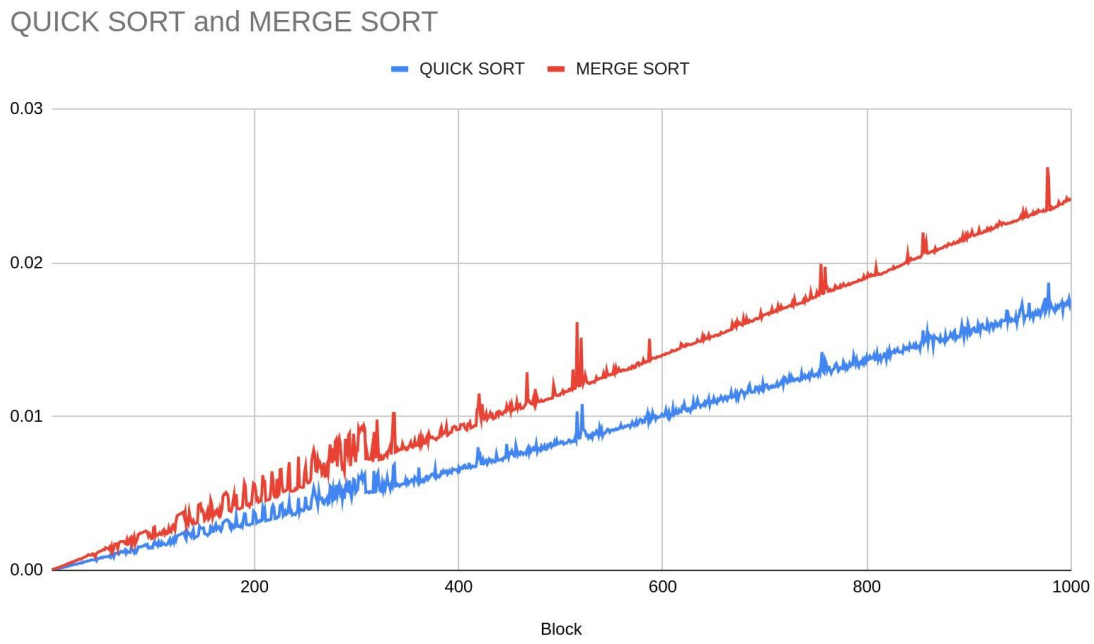
Block	QUICK SORT	MERGE SORT	No of Comparison in QuickSort	No of Comparison
1	0.000008	0.000038	619	187
2	0.000018	0.000029	1608	376
3	0.000027	0.000046	2424	579
4	0.000036	0.000062	3627	772
5	0.000047	0.000083	4644	968
6	0.000057	0.000097	6128	1171
7	0.000069	0.000113	7634	1377
8	0.000080	0.000137	8758	1577
9	0.000095	0.000146	9302	1782
10	0.000127	0.000185	11362	1979
11	0.000142	0.000264	11871	2181
12	0.000156	0.000293	12856	2379
13	0.000176	0.000299	16094	2580
14	0.000191	0.000323	17058	2777
15	0.000198	0.000363	16999	2976
16	0.000220	0.000370	18689	3175
17	0.000190	0.000328	20282	3371
18	0.000259	0.000461	20912	3575
19	0.000276	0.000473	23993	3770
20	0.000328	0.000495	23824	3971
21	0.000303	0.000527	24658	4173
22	0.000332	0.000581	27218	4372
23	0.000334	0.000553	28552	4568
24	0.000375	0.000608	29315	4770
25	0.000365	0.000626	30026	4971
26	0.000392	0.000679	34456	5172
27	0.000432	0.000679	34145	5366
28	0.000443	0.000704	35067	5576
29	0.000460	0.000743	36121	5779
30	0.000459	0.000813	39177	5975
31	0.000498	0.000834	39363	6176
32	0.000469	0.000776	43068	6382
33	0.000514	0.000854	42102	6568
34	0.000560	0.000906	42612	6767
35	0.000547	0.000902	47895	6977
36	0.000551	0.000913	48967	7169
37	0.000620	0.001002	46824	7376
38	0.000629	0.001017	49325	7571
39	0.000624	0.001022	52995	7773
40	0.000639	0.001005	52560	7973
41	0.000650	0.000960	51599	8177
42	0.000747	0.001138	61075	8375
43	0.000679	0.001149	61119	8573
44	0.000750	0.001258	62161	8776
45	0.000736	0.001173	61383	8973
46	0.000759	0.001194	62401	9170
47	0.000839	0.001291	67265	9373
48	0.000810	0.001320	63404	9565
49	0.000796	0.001284	68803	9780
50	0.000848	0.001345	73653	9972
51	0.000862	0.001347	72344	10175
52	0.000916	0.001367	77849	10379
53	0.000872	0.001434	82218	10576
54	0.000865	0.001529	82711	10777

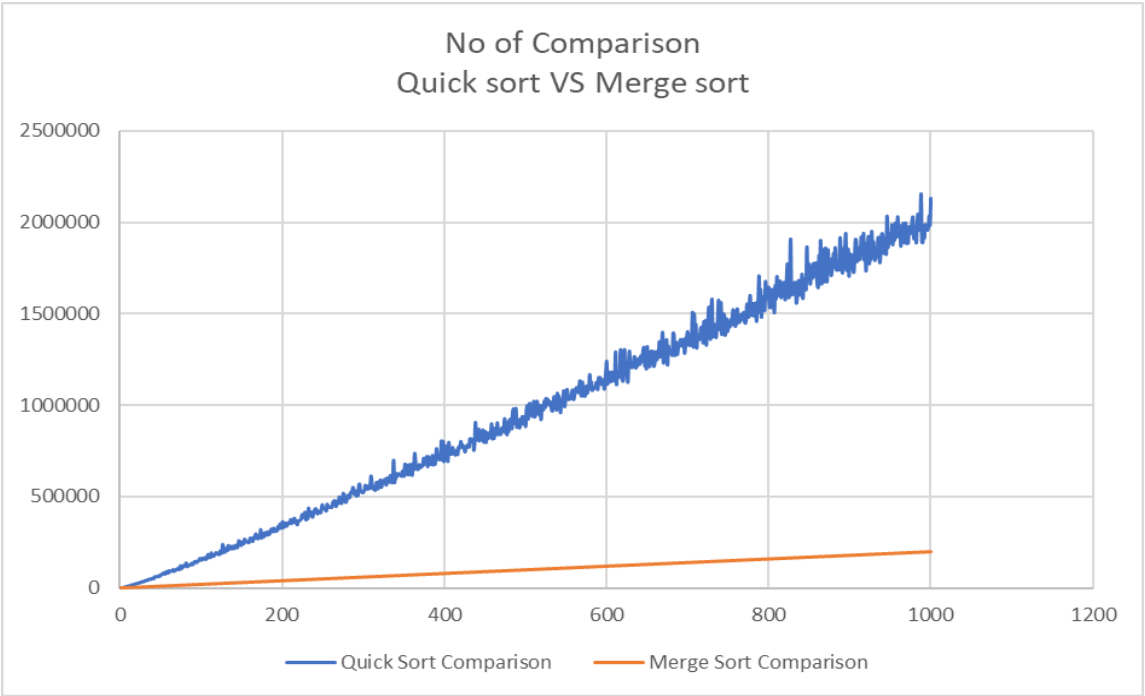
53	0.000872	0.001434	82218	10576
54	0.000865	0.001529	82711	10777
55	0.000908	0.001445	76394	10973
56	0.000886	0.001431	85924	11175
57	0.000963	0.001584	90790	11366
58	0.000959	0.001599	82413	11572
59	0.000953	0.001552	82723	11772
60	0.001111	0.001636	95104	11971
61	0.000946	0.001560	91281	12158
62	0.001165	0.001696	91271	12367
63	0.001027	0.001723	92065	12562
64	0.001103	0.001715	93309	12765
65	0.001111	0.001837	101053	12968
66	0.001056	0.001909	91787	13164
67	0.001069	0.001831	95100	13360
68	0.001131	0.001956	97402	13568
69	0.001132	0.002037	102115	13766
70	0.001323	0.001955	105465	13956
71	0.001190	0.001966	103994	14159
72	0.001104	0.001985	100080	14370
73	0.001274	0.001932	112834	14575
74	0.001383	0.002232	120966	14765
75	0.001187	0.001996	104540	14968
76	0.001285	0.002180	120873	15164
77	0.001401	0.002289	114364	15370
78	0.001489	0.002445	117099	15565
79	0.001391	0.002255	124798	15765
80	0.001452	0.002336	121745	15972
81	0.001520	0.002230	137431	16164
82	0.001466	0.002382	116882	16366
83	0.001501	0.002279	121282	16568
84	0.001411	0.002301	122098	16764
85	0.001444	0.002444	124715	16961
86	0.001587	0.002518	124943	17159
87	0.001533	0.002514	131629	17358
88	0.001573	0.002432	134819	17563
89	0.001733	0.002483	127390	17756
90	0.001599	0.002655	133609	17958
91	0.001540	0.002518	143059	18165
92	0.001640	0.002666	142676	18353
93	0.001647	0.002759	146441	18559
94	0.001631	0.002658	139220	18754
95	0.001742	0.002720	146823	18964
96	0.001809	0.002847	147787	19162
97	0.001797	0.002804	152316	19363
98	0.001781	0.002895	162303	19567
99	0.001731	0.002755	156141	19764
100	0.001810	0.002944	152988	19968
101	0.001920	0.002955	162547	20158
102	0.001866	0.003118	159266	20358
103	0.001854	0.003011	157842	20556
104	0.002009	0.003054	166461	20771
105	0.001848	0.002999	157162	20964
106	0.001935	0.003086	158385	21162
107	0.001934	0.003098	170590	21351

```
950     0.014610        0.021624        1896120         189947
951     0.015489        0.022547        1888247         190161
952     0.014642        0.021466        1987171         190357
953     0.015157        0.022344        1877085         190563
954     0.015597        0.022099        1977236         190763
955     0.015359        0.023300        1901370         190961
956     0.015322        0.023290        1995640         191164
957     0.015553        0.022210        1945432         191358
958     0.015522        0.023724        1921079         191556
959     0.015331        0.022308        2030258         191758
960     0.015486        0.022784        1962076         191961
961     0.015519        0.023630        1985227         192162
962     0.015734        0.022980        1971708         192354
963     0.015226        0.023305        1869103         192563
964     0.015427        0.022519        1950345         192768
965     0.015321        0.022863        1920065         192963
966     0.015672        0.023255        1995043         193164
967     0.015286        0.022775        1890539         193357
968     0.015160        0.022171        1950966         193564
969     0.014438        0.021558        1994373         193760
970     0.014936        0.022203        1883263         193958
971     0.014861        0.022787        1941165         194156
972     0.015194        0.021642        1885788         194356
973     0.015214        0.021854        1975044         194565
974     0.015164        0.021799        1938065         194761
975     0.014465        0.022182        1940916         194961
976     0.014881        0.021602        1971921         195165
977     0.015355        0.023363        2013274         195358
978     0.015729        0.023910        2028071         195565
979     0.015091        0.022917        1911038         195762
980     0.015236        0.023251        1913732         195962
981     0.015274        0.022780        1945382         196163
982     0.014813        0.023637        1887764         196357
983     0.015542        0.023890        1982774         196556
984     0.015174        0.022640        2044538         196761
985     0.016164        0.022999        1958051         196965
986     0.016077        0.023832        1995734         197166
987     0.015105        0.023066        2009956         197368
988     0.016421        0.023385        2153876         197562
989     0.015438        0.023142        1892779         197768
990     0.015452        0.023038        1888597         197963
991     0.015148        0.022387        1957562         198166
992     0.014489        0.022460        1914048         198365
993     0.015266        0.022875        1986576         198560
994     0.015334        0.023488        1958678         198767
995     0.015867        0.022582        1966345         198964
996     0.015230        0.022900        1956705         199169
997     0.015321        0.023694        1979996         199361
998     0.016542        0.024407        2035221         199564
999     0.016760        0.024629        1985069         199767
1000    0.016315        0.023412        2130936         199968


...Program finished with exit code 0
Press ENTER to exit console.
```

Chart:


QUICK SORT and MERGE SORT
QUICK SORT
MERGE SORT
0.03
0.02
0.01
0.00
200
400
600
800
1000
Block


No of Comparison
Quick sort VS Merge sort
2500000
2000000
1500000
1000000
500000
0
0
200
400
600
800
1000
1200
Quick Sort Comparison
Merge Sort Comparison

Observation:	We plotted the graph for time taken by Quick sort and Merge sort. On the X axis we have a number of blocks from 0 to 1000 and on the Y axis we have time in seconds. By observing the graph it is clear that time taken to sort all the blocks for Merge sort is more than time taken by Quick sort. Also in second graph on X axis we have number of blocks and on Y axis we have number of comparisons. And we observed that in Quick sort number of comparisons are more as compare to Merge sort.
CONCLUSION:	Thus, we have found the running time of quick sort and merge sort on each block, and plotted a 2-D chart which shows the comparison of both algorithm's running time. Also shows the number of comparisons each algorithm made using graph. And we found out that the time taken by Quick sort is less then Merge sort but number of comparisons done by Quick sort is greater than Merge Sort.