BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	Balla Mahadev Shrikrishna
UID no.	2023300010
Experiment No.	2A

AIM:	Experiment based on divide and conquer approach.
Program 1	
PROBLEM STATEMENT :	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 the C/C++ Rand function and save them in a text file. Both the sorting algorithms use 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[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 Quicksort and Merge sort by plotting the time required to sort integers using LibreOffice Calc/MS Excel. The x-axis of the 2-D plot represents the block no. of 1000 blocks. The y-axis of the 2-D plot represents the running time to sort 1000 blocks of 100,200,300,...,100000 integer numbers.
PROGRAM:	#include <bits/stdc++.h> using namespace std; using namespace chrono; #define NUM_COUNT 100000 #define OUTPUT_FILE "random_numbers.txt" #define TIME_RESULT_FILE "sorting_times.csv" using namespace std;

```
void generate_random_numbers()
{
        ofstream file(OUTPUT_FILE);
        if (!file) {
        cerr << "Error opening file for writing!" << endl;
        return;
        }
        for (int i = 0; i < NUM_COUNT; i++) {
        file << rand() % 1000000 << "\n";
        }
        file.close();
}

void read_numbers(vector<int>& arr, int size) {
        ifstream file(OUTPUT_FILE);
        if (!file) {
        cerr << "Error opening file for reading!" << endl;
        return;
        }
        arr.resize(size);
        for (int i = 0; i < size; i++) {
        file >> arr[i];
        }
        file.close();
}

void merge(vector<int>& arr, int left, int mid, int right) {
        vector<int> temp;
        int i = left, j = mid + 1;

        while (i <= mid && j <= right)
        temp.push_back((arr[i] <= arr[j]) ? arr[i++] : arr[j++]);

        while (i <= mid) temp.push_back(arr[i++]);
        while (j <= right) temp.push_back(arr[j++]);

        for (int i = left; i <= right; i++)
        arr[i] = temp[i - left];
```

```
}

void merge_sort(vector<int>& arr, int left, int right) {
        if (left >= right) return;

        int mid = left + (right - left) / 2;
        merge_sort(arr, left, mid);
        merge_sort(arr, mid + 1, right);
        merge(arr, left, mid, right);
}

int partition(vector<int>& arr, int low, int high) {
        int pivot = arr[low];
        int i = low + 1;
        for (int j = low + 1; j <= high; j++) {
        if (arr[j] < pivot) {
        swap(arr[i], arr[j]);
        i++;
        }
        }
        swap(arr[low], arr[i - 1]);
        return i - 1;
}

void quick_sort(vector<int>& arr, int low, int high) {
        if (low < high) {
        int pi = partition(arr, low, high);
        quick_sort(arr, low, pi - 1);
        quick_sort(arr, pi + 1, high);
        }
}

void perform_experiment() {
        ofstream file(TIME_RESULT_FILE);
        if (!file) {
        cerr << "Error opening file for writing results!" << endl;
        return;
        }
```

```
        file << "Block Size,QuickSort Random (ms),MergeSort Random
(ms),QuickSort Best (ms),MergeSort Best (ms),QuickSort Worst
(ms),MergeSort Worst (ms)\n";

        for (int block_size = 100; block_size <= NUM_COUNT; block_size
+= 100) {
        vector<int> arr1, arr2;
        read_numbers(arr1, block_size);
        arr2 = arr1;

        auto start = high_resolution_clock::now();
        quick_sort(arr1, 0, block_size - 1);
        auto end = high_resolution_clock::now();
        double quicksort_random = duration<double, milli>(end -
start).count();

        start = high_resolution_clock::now();
        merge_sort(arr2, 0, block_size - 1);
        end = high_resolution_clock::now();
        double mergesort_random = duration<double, milli>(end -
start).count();

        sort(arr1.begin(), arr1.end()); // Best case sorted input
        start = high_resolution_clock::now();
        quick_sort(arr1, 0, block_size - 1);
        end = high_resolution_clock::now();
        double quicksort_best = duration<double, milli>(end - start).count();

        start = high_resolution_clock::now();
        merge_sort(arr2, 0, block_size - 1);
        end = high_resolution_clock::now();
        double mergesort_best = duration<double, milli>(end -
start).count();

        sort(arr1.rbegin(), arr1.rend()); // Worst case reversed input
        start = high_resolution_clock::now();
        quick_sort(arr1, 0, block_size - 1);
        end = high_resolution_clock::now();
```

```
        double quicksort_worst = duration<double, milli>(end -
start).count();

        start = high_resolution_clock::now();
        merge_sort(arr2, 0, block_size - 1);
        end = high_resolution_clock::now();
        double mergesort_worst = duration<double, milli>(end -
start).count();

        file << block_size << "," << quicksort_random << "," <<
mergesort_random << "," << quicksort_best << "," << mergesort_best <<
"," << quicksort_worst << "," << mergesort_worst << "\n";

        cout << "Block Size: " << block_size << " | QuickSort Random: "
<< quicksort_random << " ms | MergeSort Random: " <<
mergesort_random
        << " ms | QuickSort Best: " << quicksort_best << " ms | MergeSort
Best: " << mergesort_best
        << " ms | QuickSort Worst: " << quicksort_worst << " ms |
MergeSort Worst: " << mergesort_worst << " ms" << endl;
        }
        file.close();
}

int main() {
        srand(time(nullptr));
        generate_random_numbers();
        perform_experiment();
        cout << "Experiment complete. Results saved in " <<
TIME_RESULT_FILE << "." << endl;
        return 0;
}
```

RESULT:

```
PS C:\Mahadev\S.E\Sem 4\DAA\Lab\Lab Sessions\exp2> g++ exp2a.cpp
PS C:\Mahadev\S.E\Sem 4\DAA\Lab\Lab Sessions\exp2> ./a.exe
Block Size: 100 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 0 ms | MergeSort Best: 0 ms | QuickSort Worst: 0 ms | MergeSort Worst: 2.712 ms
Block Size: 200 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 0 ms | MergeSort Best: 1.508 ms | QuickSort Worst: 0 ms | MergeSort Worst: 0 ms
Block Size: 300 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 0 ms | MergeSort Best: 0 ms | QuickSort Worst: 0 ms | MergeSort Worst: 0 ms
Block Size: 400 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 0 ms | MergeSort Best: 0 ms | QuickSort Worst: 4.123 ms | MergeSort Worst: 0.338 ms
Block Size: 500 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 4.501 ms | MergeSort Best: 0 ms | QuickSort Worst: 3.643 ms | MergeSort Worst: 0 ms
Block Size: 600 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 5.453 ms | MergeSort Best: 0 ms | QuickSort Worst: 2.713 ms | MergeSort Worst: 0 ms
Block Size: 700 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 0 ms | MergeSort Best: 0 ms | QuickSort Worst: 6.772 ms | MergeSort Worst: 0 ms
Block Size: 800 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 7.72 ms | MergeSort Best: 0 ms | QuickSort Worst: 0 ms | MergeSort Worst: 0 ms
Block Size: 900 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 0 ms | MergeSort Best: 0 ms | QuickSort Worst: 0 ms | MergeSort Worst: 0 ms
Block Size: 1000 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 7.703 ms | MergeSort Best: 0 ms | QuickSort Worst: 2.006 ms | MergeSort Worst: 0 ms
Block Size: 1100 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 8.596 ms | MergeSort Best: 0 ms | QuickSort Worst: 7.413 ms | MergeSort Worst: 0 ms
Block Size: 1200 | QuickSort Random: 0 ms | MergeSort Random: 2.377 ms | QuickSort Best: 7.317 ms | MergeSort Best: 0 ms | QuickSort Worst: 6.881 ms | MergeSort Worst: 0 ms
Block Size: 1300 | QuickSort Random: 0 ms | MergeSort Random: 2.172 ms | QuickSort Best: 5.011 ms | MergeSort Best: 7.032 ms | QuickSort Worst: 1.321 ms | MergeSort Worst: 8.01 ms
Block Size: 1400 | QuickSort Random: 0 ms | MergeSort Random: 9.107 ms | QuickSort Best: 7.641 ms | MergeSort Best: 0 ms | QuickSort Worst: 7.793 ms | MergeSort Worst: 0 ms
Block Size: 1500 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 20.801 ms | MergeSort Best: 2.004 ms | QuickSort Worst: 10.223 ms | MergeSort Worst: 0 ms
Block Size: 1600 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 34.882 ms | MergeSort Best: 0 ms | QuickSort Worst: 30.601 ms | MergeSort Worst: 0 ms
Block Size: 1700 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 49.288 ms | MergeSort Best: 8.795 ms | QuickSort Worst: 32.17 ms | MergeSort Worst: 0 ms
Block Size: 1800 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 57.62 ms | MergeSort Best: 0 ms | QuickSort Worst: 23.985 ms | MergeSort Worst: 8.318 ms
Block Size: 1900 | QuickSort Random: 0 ms | MergeSort Random: 4.016 ms | QuickSort Best: 41.833 ms | MergeSort Best: 0 ms | QuickSort Worst: 41.346 ms | MergeSort Worst: 0 ms
Block Size: 2000 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 56.271 ms | MergeSort Best: 8.918 ms | QuickSort Worst: 27.086 ms | MergeSort Worst: 3.75 ms
Block Size: 2100 | QuickSort Random: 0 ms | MergeSort Random: 8.761 ms | QuickSort Best: 44.762 ms | MergeSort Best: 0 ms | QuickSort Worst: 33.649 ms | MergeSort Worst: 0 ms
Block Size: 2200 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 49.365 ms | MergeSort Best: 8.191 ms | QuickSort Worst: 37.867 ms | MergeSort Worst: 3.843 ms
Block Size: 2300 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 89.232 ms | MergeSort Best: 0 ms | QuickSort Worst: 24.924 ms | MergeSort Worst: 0 ms
Block Size: 2400 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 74.292 ms | MergeSort Best: 0 ms | QuickSort Worst: 24.297 ms | MergeSort Worst: 0 ms
Block Size: 2500 | QuickSort Random: 0 ms | MergeSort Random: 0 ms | QuickSort Best: 74.877 ms | MergeSort Best: 0 ms | QuickSort Worst: 41.405 ms | MergeSort Worst: 0.426 ms
Block Size: 2600 | QuickSort Random: 0 ms | MergeSort Random: 7.103 ms | QuickSort Best: 58.101 ms | MergeSort Best: 0 ms | QuickSort Worst: 41.351 ms | MergeSort Worst: 0.357 ms
Block Size: 2700 | QuickSort Random: 0 ms | MergeSort Random: 2.305 ms | QuickSort Best: 40.805 ms | MergeSort Best: 8.33 ms | QuickSort Worst: 24.794 ms | MergeSort Worst: 0 ms
```

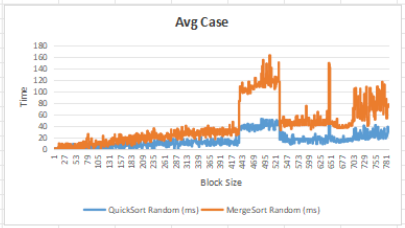
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

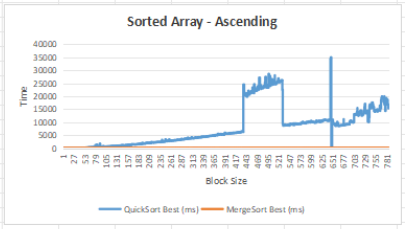
```
Block Size: 75900 | QuickSort Random: 30.923 ms | MergeSort Random: 69.514 ms | QuickSort Best: 14147.7 ms | MergeSort Best: 67.601 ms | QuickSort Worst: 12817.5 ms | MergeSort Worst: 41.798 ms
Block Size: 76000 | QuickSort Random: 26.812 ms | MergeSort Random: 107.888 ms | QuickSort Best: 14595.3 ms | MergeSort Best: 41.995 ms | QuickSort Worst: 13026.7 ms | MergeSort Worst: 60.602 ms
Block Size: 76100 | QuickSort Random: 37.865 ms | MergeSort Random: 76.433 ms | QuickSort Best: 14296.2 ms | MergeSort Best: 42.116 ms | QuickSort Worst: 12817.3 ms | MergeSort Worst: 42.002 ms
Block Size: 76200 | QuickSort Random: 17.038 ms | MergeSort Random: 84.979 ms | QuickSort Best: 14666.3 ms | MergeSort Best: 86.825 ms | QuickSort Worst: 12824.5 ms | MergeSort Worst: 43.062 ms
Block Size: 76300 | QuickSort Random: 24.023 ms | MergeSort Random: 102.914 ms | QuickSort Best: 14622.1 ms | MergeSort Best: 42.514 ms | QuickSort Worst: 14481.9 ms | MergeSort Worst: 59.809 ms
Block Size: 76400 | QuickSort Random: 17.508 ms | MergeSort Random: 70.117 ms | QuickSort Best: 15216.5 ms | MergeSort Best: 43.506 ms | QuickSort Worst: 15946.2 ms | MergeSort Worst: 62.511 ms
Block Size: 76500 | QuickSort Random: 23.278 ms | MergeSort Random: 67.618 ms | QuickSort Best: 16399.2 ms | MergeSort Best: 42.994 ms | QuickSort Worst: 14361.8 ms | MergeSort Worst: 45.004 ms
Block Size: 76600 | QuickSort Random: 27.686 ms | MergeSort Random: 95.582 ms | QuickSort Best: 14376.5 ms | MergeSort Best: 42.106 ms | QuickSort Worst: 12367.1 ms | MergeSort Worst: 43.008 ms
Block Size: 76700 | QuickSort Random: 18.522 ms | MergeSort Random: 66.326 ms | QuickSort Best: 15359.3 ms | MergeSort Best: 45.001 ms | QuickSort Worst: 16024.3 ms | MergeSort Worst: 63.183 ms
Block Size: 76800 | QuickSort Random: 31.082 ms | MergeSort Random: 99.701 ms | QuickSort Best: 18471.2 ms | MergeSort Best: 49.585 ms | QuickSort Worst: 16534.8 ms | MergeSort Worst: 63.713 ms
Block Size: 76900 | QuickSort Random: 32.999 ms | MergeSort Random: 71.524 ms | QuickSort Best: 19307.5 ms | MergeSort Best: 85.672 ms | QuickSort Worst: 17243.5 ms | MergeSort Worst: 59.685 ms
Block Size: 77000 | QuickSort Random: 24.903 ms | MergeSort Random: 101.091 ms | QuickSort Best: 19914.8 ms | MergeSort Best: 64.509 ms | QuickSort Worst: 17283.1 ms | MergeSort Worst: 54.994 ms
Block Size: 77100 | QuickSort Random: 29.724 ms | MergeSort Random: 116.018 ms | QuickSort Best: 19697.3 ms | MergeSort Best: 77.511 ms | QuickSort Worst: 17266.5 ms | MergeSort Worst: 66.885 ms
Block Size: 77200 | QuickSort Random: 20.023 ms | MergeSort Random: 88.723 ms | QuickSort Best: 19560.8 ms | MergeSort Best: 107.815 ms | QuickSort Worst: 17515.7 ms | MergeSort Worst: 55.721 ms
Block Size: 77300 | QuickSort Random: 21.097 ms | MergeSort Random: 76.627 ms | QuickSort Best: 19249.7 ms | MergeSort Best: 55.571 ms | QuickSort Worst: 15931.3 ms | MergeSort Worst: 42.711 ms
Block Size: 77400 | QuickSort Random: 17.694 ms | MergeSort Random: 59.963 ms | QuickSort Best: 17819.4 ms | MergeSort Best: 50.556 ms | QuickSort Worst: 16185.6 ms | MergeSort Worst: 64.23 ms
Block Size: 77500 | QuickSort Random: 29.998 ms | MergeSort Random: 81.52 ms | QuickSort Best: 18890.3 ms | MergeSort Best: 60.998 ms | QuickSort Worst: 17141.1 ms | MergeSort Worst: 58.132 ms
Block Size: 77600 | QuickSort Random: 35.093 ms | MergeSort Random: 111.253 ms | QuickSort Best: 19975.7 ms | MergeSort Best: 57.003 ms | QuickSort Worst: 16444.5 ms | MergeSort Worst: 50.612 ms
Block Size: 77700 | QuickSort Random: 24.676 ms | MergeSort Random: 75.108 ms | QuickSort Best: 16841.3 ms | MergeSort Best: 44.024 ms | QuickSort Worst: 12674.7 ms | MergeSort Worst: 52.803 ms
Block Size: 77800 | QuickSort Random: 26.015 ms | MergeSort Random: 75.023 ms | QuickSort Best: 14359.6 ms | MergeSort Best: 43.697 ms | QuickSort Worst: 12739.5 ms | MergeSort Worst: 55.145 ms
Block Size: 77900 | QuickSort Random: 18.286 ms | MergeSort Random: 84.586 ms | QuickSort Best: 14581.9 ms | MergeSort Best: 44.539 ms | QuickSort Worst: 13833.9 ms | MergeSort Worst: 51.985 ms
Block Size: 78000 | QuickSort Random: 21.264 ms | MergeSort Random: 62.131 ms | QuickSort Best: 19298.2 ms | MergeSort Best: 58.534 ms | QuickSort Worst: 17002.2 ms | MergeSort Worst: 74.695 ms
Block Size: 78100 | QuickSort Random: 24.994 ms | MergeSort Random: 86.617 ms | QuickSort Best: 18381.9 ms | MergeSort Best: 59.625 ms | QuickSort Worst: 15752.8 ms | MergeSort Worst: 49.069 ms
Block Size: 78200 | QuickSort Random: 22.028 ms | MergeSort Random: 53.002 ms | QuickSort Best: 16854.8 ms | MergeSort Best: 99.54 ms | QuickSort Worst: 15126 ms | MergeSort Worst: 71.873 ms
Block Size: 78300 | QuickSort Random: 26.002 ms | MergeSort Random: 72.795 ms | QuickSort Best: 18661.9 ms | MergeSort Best: 67.008 ms | QuickSort Worst: 15261.4 ms | MergeSort Worst: 65.997 ms
Block Size: 78400 | QuickSort Random: 38.047 ms | MergeSort Random: 72.436 ms | QuickSort Best: 17573.2 ms | MergeSort Best: 82.488 ms | QuickSort Worst: 16744 ms | MergeSort Worst: 90.339 ms
Block Size: 78500 | QuickSort Random: 34.374 ms | MergeSort Random: 74.956 ms | QuickSort Best: 16803.7 ms | MergeSort Best: 56.082 ms | QuickSort Worst: 16046.6 ms | MergeSort Worst: 64.578 ms
Block Size: 78600 | QuickSort Random: 29.665 ms | MergeSort Random: 77.195 ms | QuickSort Best: 15324.1 ms | MergeSort Best: 69.619 ms | QuickSort Worst: 14180.8 ms | MergeSort Worst: 43.688 ms
Block Size: 78700 | QuickSort Random: 30.668 ms | MergeSort Random: 76.959 ms | QuickSort Best: 15636 ms | MergeSort Best: 53.981 ms | QuickSort Worst: 13684.8 ms | MergeSort Worst: 65.771 ms
Block Size: 78800 | QuickSort Random: 31.247 ms | MergeSort Random: 104.217 ms | QuickSort Best: 16685.1 ms | MergeSort Best: 58.1 ms | QuickSort Worst: 14243 ms | MergeSort Worst: 72.603 ms
Block Size: 78900 | QuickSort Random: 23.086 ms | MergeSort Random: 79.852 ms | QuickSort Best: 16408.8 ms | MergeSort Best: 43.996 ms | QuickSort Worst: 14733.1 ms | MergeSort Worst: 64.658 ms
Block Size: 79000 | QuickSort Random: 23.112 ms | MergeSort Random: 81.614 ms | QuickSort Best: 16254.4 ms | MergeSort Best: 43.911 ms | QuickSort Worst: 13983.7 ms | MergeSort Worst: 44.824 ms
Block Size: 79100 | QuickSort Random: 24.926 ms | MergeSort Random: 110.506 ms | QuickSort Best: 16052.8 ms | MergeSort Best: 71.776 ms | QuickSort Worst: 14143.4 ms | MergeSort Worst: 49.519 ms
Block Size: 79200 | QuickSort Random: 34.005 ms | MergeSort Random: 91.871 ms | QuickSort Best: 15795.1 ms | MergeSort Best: 76.972 ms | QuickSort Worst: 14970.3 ms | MergeSort Worst: 88.256 ms
Block Size: 79300 | QuickSort Random: 37.022 ms | MergeSort Random: 85.716 ms | QuickSort Best: 17506.1 ms | MergeSort Best: 58.596 ms | QuickSort Worst: 16094.4 ms | MergeSort Worst: 73.595 ms
Block Size: 79400 | QuickSort Random: 27.576 ms | MergeSort Random: 86.847 ms | QuickSort Best: 17602.3 ms | MergeSort Best: 59.513 ms | QuickSort Worst: 14685.6 ms | MergeSort Worst: 58.008 ms
Block Size: 79500 | QuickSort Random: 25.431 ms | MergeSort Random: 94.577 ms | QuickSort Best: 17098 ms | MergeSort Best: 78.405 ms | QuickSort Worst: 15035.9 ms | MergeSort Worst: 79.472 ms
PS C:\Mahadev\S.E\Sem 4\DAA\Lab\Lab Sessions\exp2>
```

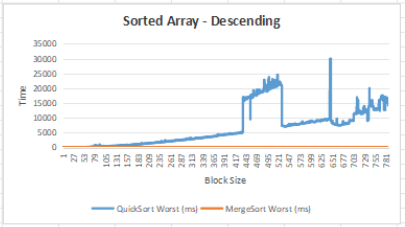
Ln 21, Col 16 Spaces: 4 UTF-8 CRLF {} C++ Go Live Quokka Win32 Prettier

EXCEL OUTPUT:

Block Size	QuickSort Random (ms)	MergeSort Random (ms)	QuickSort Best (ms)	MergeSort Best (ms)	QuickSort Worst (ms)	MergeSort Worst (ms)
100	0	0	0	0	0	2.712
200	0	0	0	1.508	0	0
300	0	0	0	0	0	0
400	0	0	0	0	4.123	0.338
500	0	0	4.501	0	3.643	0
600	0	0	5.453	0	2.713	0
700	0	0	0	0	6.772	0
800	0	0	7.72	0	0	0
900	0	0	0	0	0	0
1000	0	0	7.703	0	2.006	0
1100	0	0	8.596	0	7.413	0
1200	0	2.377	7.317	0	6.881	0
1300	0	2.172	5.011	7.032	1.321	8.01
1400	0	9.107	7.641	0	7.793	0
1500	0	0	20.801	2.004	10.223	0
1600	0	0	34.882	0	30.601	0
1700	0	0	49.288	8.795	32.17	0
1800	0	0	57.62	0	23.985	8.318
1900	0	4.016	41.833	0	41.346	0
2000	0	0	56.271	8.918	27.086	3.75
2100	0	8.761	44.762	0	33.649	0
2200	0	0	49.365	8.191	37.867	3.843
2300	0	0	89.232	0	24.924	0
2400	0	0	74.292	0	24.297	0
2500	0	0	74.877	0	41.405	0.426
2600	0	7.103	58.101	0	41.351	0.357
2700	0	2.305	40.805	8.33	24.794	0
2800	0	2.003	58.932	1.502	23.786	8.257
2900	0	0	57.342	8.391	41.428	0
3000	0	0	67.445	1.502	31.924	3.322
3100	0	9.045	56.592	0	35.975	2.003
3200	0	0	70.519	2.003	41.073	0
3300	0	4.458	78.339	0	44.174	4.994
3400	0	9.012	73.258	0	50.479	0
3500	0	0	90.601	0	49.739	0
3600	0	4.491	83.878	0	49.733	0
3700	0	6.641	97.226	0	57.31	0
3800	0	8.073	107.605	0	98.652	9.07
3900	7.908	0.512	148.339	0	81.911	0
4000	0	6.083	98.965	0	67.148	0
4100	1.505	0.511	105.579	0	66.917	0
4200	0	8.11	114.856	8.326	75.868	8.036
4300	0	0	140.514	0	66.346	8.433
4400	2.01	4.419	105.714	0	66.843	0
4500	1.007	0	133.837	7.62	82.002	0
4600	0	0	115.311	8.324	90.148	0
4700	0	0	132.408	0	108.086	0
4800	8.572	0.894	163.633	1.09	80.871	8.121
4900	2.54	2.089	153.782	0	124.738	8.152







sorting_times

BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

CONCLUSION:

Name: Balla Mahadev Shrikrishna
VID: 2023300010
Division: A
Batch: A

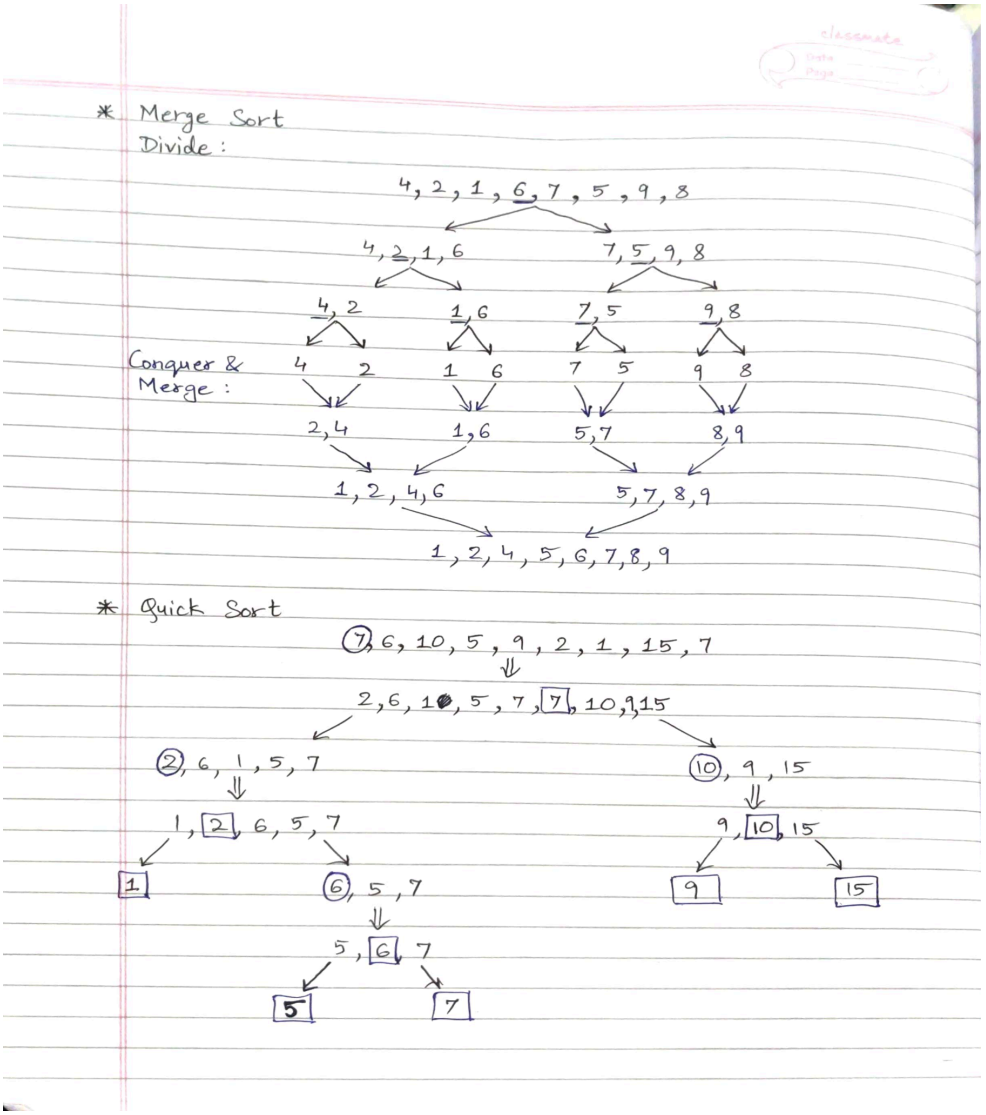
Exp-2A

* Merge Sort:
  - Time Complexity –
    - Best, Worst, Avg Case: $O(n \log n)$
  - Space Complexity –
    - $O(n)$ due to auxiliary space used for merging

* Quick Sort:
  - Time Complexity –
    - Best Case, Avg Case: $O(n \log n)$
      - when pivot divides the array into two nearly equal halves.
    - Worst Case: $O(n^2)$
      - when the pivot is smallest or largest element, leading to highly unbalanced partitions.
  - Space Complexity –
    - $O(\log n)$ due to the recursion stack.

* Merge sort is more consistent & stable but uses more memory.

* Quick sort is faster in practice & uses less memory but can have poor worst-case performance if not implemented carefully.



The avg case ran well in time but the issue occurred due to sorted cases because of which I couldn't record the run time for higher block sizes.