तमसो मा ज्योतिर्गमय

NIRMA
UNIVERSITY

INSTITUTE OF TECHNOLOGY

NAAC ACCREDITED 'A' GRADE

NAME: - Darji Akshatkumar Hiteshbhai

RollNo: - 23MCD001

Branch: - M.tech-CSE**(Data Science)**

Subject: - Complexity Theory & Algorithms

Practical-2

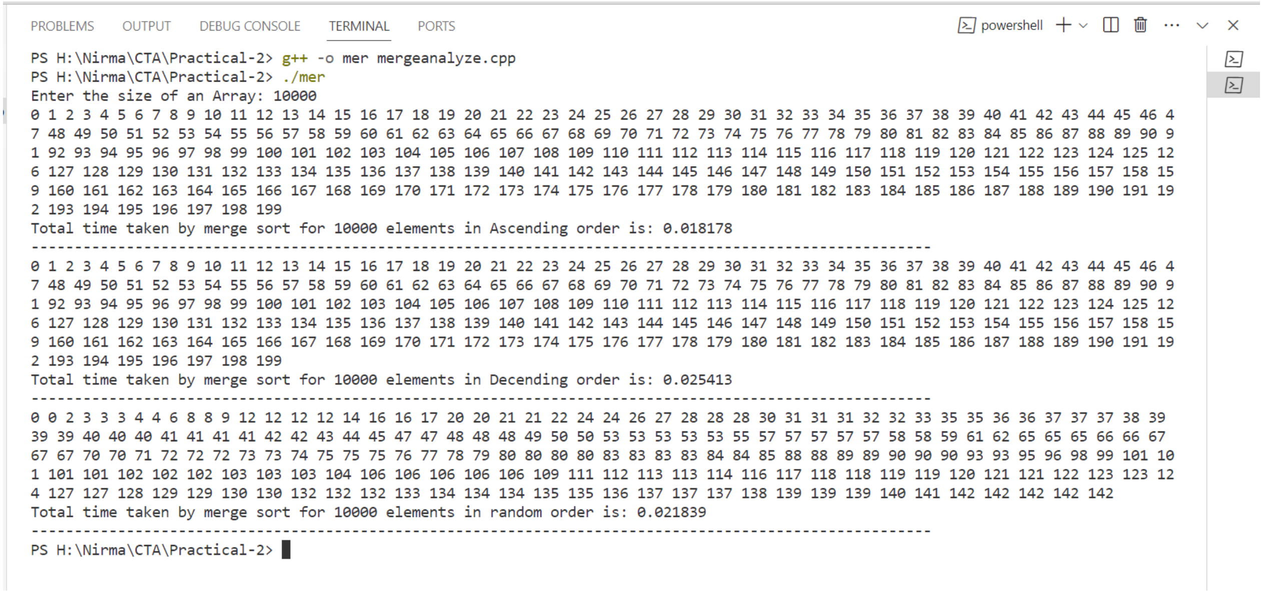
Aim: Perform Merge Sort, External Merge sort for the input size 10000, 50000 and 100000 for Ascending, Descending & Random order array. Plot the chart of the output data and do the analysis which algorithm is best and justify your reason.

Code for Merge sort-

```
#include <bits/stdc++.h>
using namespace std;
using namespace std::chrono;

void acending(vector<int> &arr, int n)
{
    for (int i = 0; i < n; i++)
    {
        arr[i] = i;
    }
}
void decending(vector<int> &arr, int n)
{
    for (int i = 0; i < n; i++)
    {
        arr[i] = n - i - 1;
    }
}
void random(vector<int> &arr, int n)
{
    for (int i = 0; i < n; i++)
    {
        arr[i] = rand() % n;
    }
}

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

    while (left <= mid && right <= high)
    {
        if (arr[left] <= arr[right])
        {
            temp.push_back(arr[left]);
            left++;
        }
        else
        {
            temp.push_back(arr[right]);
            right++;
        }
    }
    while (left <= mid)
    {
        temp.push_back(arr[left]);
        left++;
    }
    while (right <= high)
    {
        temp.push_back(arr[right]);
        right++;
    }

    for (int i = low; i <= high; i++)
    {
        arr[i] = temp[i - low];
    }
}
void mergesort(vector<int> &arr, int low, int high)
{
    if (low >= high)
        return;
    int mid = floor((low + high) / 2);
    mergesort(arr, low, mid);
    mergesort(arr, mid + 1, high);
    merge(arr, low, mid, high);
}
void mergesort_a(vector<int> &arr, int n)
{
    acending(arr, n);
    auto start = high_resolution_clock::now();
    mergesort(arr, 0, n - 1);
    auto end = high_resolution_clock::now();
    duration<double> total = end - start;
    for (int i = 0; i < 200; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
    cout << "Total time taken by merge sort for " << n << " elements in Ascending
order is: " << total.count() << endl;
    cout << "--------------------------------------------------------------------
------------------------------------";
    cout << endl;
}
void mergesort_d(vector<int> &arr, int n)
{
    decending(arr, n);
    auto start = high_resolution_clock::now();
    mergesort(arr, 0, n - 1);
    auto end = high_resolution_clock::now();
    duration<double> total = end - start;
    for (int i = 0; i < 200; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
    cout << "Total time taken by merge sort for " << n << " elements in Decending
order is: " << total.count() << endl;
    cout << "--------------------------------------------------------------------
------------------------------------";
    cout << endl;
}
void mergesort_r(vector<int> &arr, int n)
{
    random(arr, n);
    auto start = high_resolution_clock::now();
    mergesort(arr, 0, n - 1);
    auto end = high_resolution_clock::now();
    duration<double> total = end - start;
    for (int i = 0; i < 200; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
    cout << "Total time taken by merge sort for " << n << " elements in random
order is: " << total.count() << endl;
    cout << "--------------------------------------------------------------------
------------------------------------";
    cout << endl;
}
int main()
{
    int n;
    cout << "Enter the size of an Array: ";
    cin >> n;
    vector<int> arr(n);

    for (int i = 0; i <= 3; i++)
    {
        if (i == 1)
        {
            mergesort_a(arr, n);
        }
        if (i == 2)
        {
            mergesort_d(arr, n);
        }
        if (i == 3)
        {
            mergesort_r(arr, n);
        }
    }
}
```

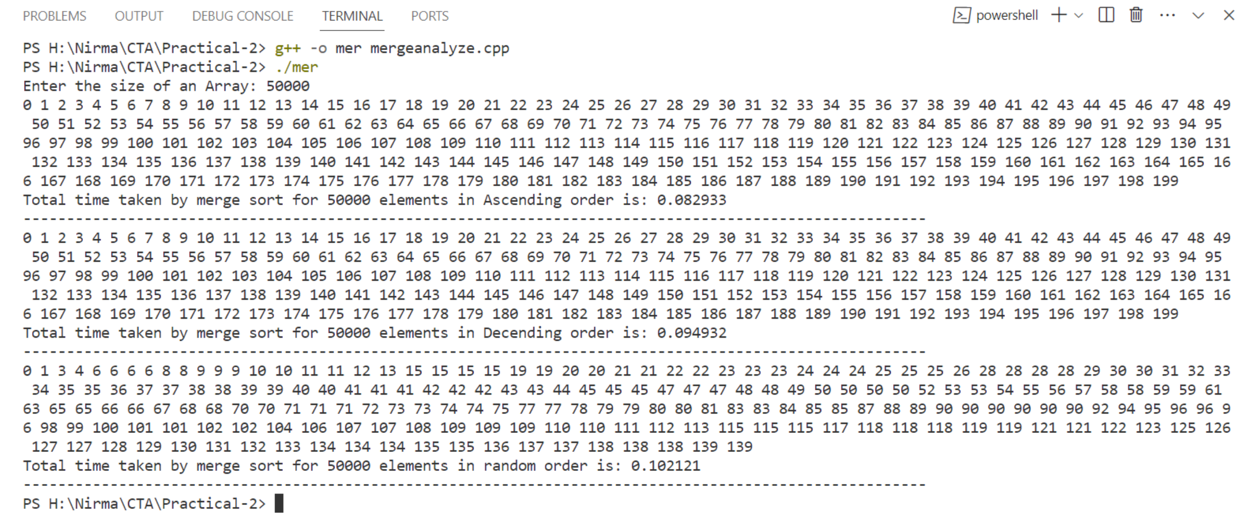
- **Output**

For array size = 10000



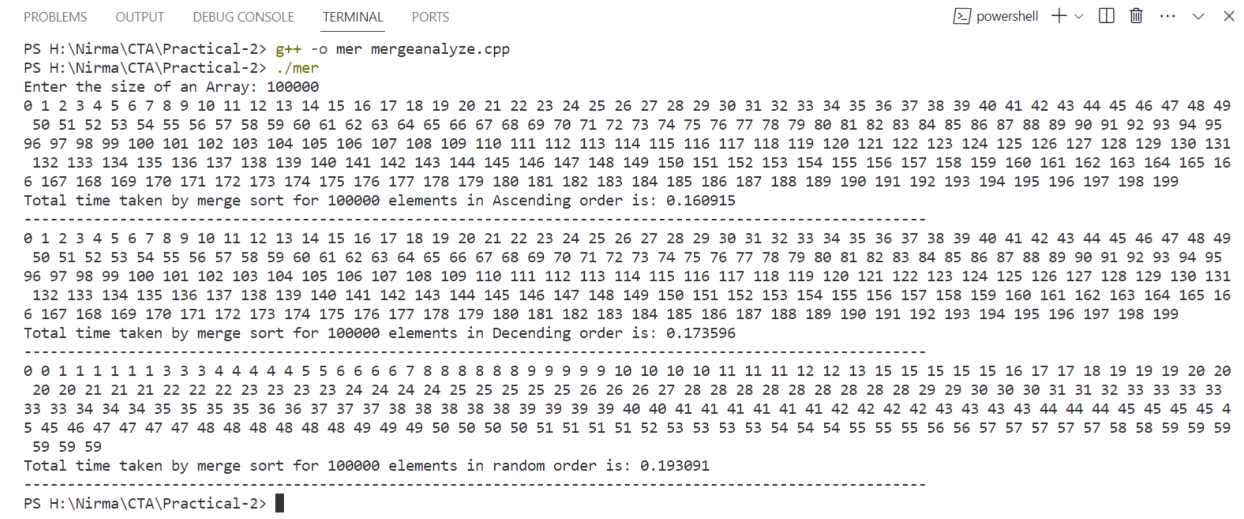
```
PS H:\Nirma\CTA\Practical-2> g++ -o mer mergeanalyze.cpp
PS H:\Nirma\CTA\Practical-2> ./mer
Enter the size of an Array: 10000
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 4
7 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 9
1 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 12
6 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 15
9 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 19
2 193 194 195 196 197 198 199
Total time taken by merge sort for 10000 elements in Ascending order is: 0.018178
--------------------------------------------------------------------------------------------------------
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 4
7 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 9
1 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 12
6 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 15
9 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 19
2 193 194 195 196 197 198 199
Total time taken by merge sort for 10000 elements in Decending order is: 0.025413
--------------------------------------------------------------------------------------------------------
0 0 2 3 3 3 4 4 6 8 8 9 12 12 12 12 14 16 16 17 20 20 21 21 22 24 24 26 27 28 28 28 30 31 31 31 32 32 33 35 35 36 36 37 37 37 38 39
39 39 40 40 40 41 41 41 41 42 42 43 44 45 47 47 48 48 48 49 50 50 53 53 53 53 53 55 57 57 57 57 57 58 58 59 61 62 65 65 65 66 66 67
67 67 70 70 71 72 72 72 73 73 74 75 75 75 76 77 78 79 80 80 80 80 83 83 83 83 84 84 85 88 88 89 89 90 90 90 93 93 95 96 98 99 101 10
1 101 101 102 102 102 103 103 103 104 106 106 106 106 106 109 111 112 113 113 114 116 117 118 118 119 119 120 121 121 122 123 123 12
4 127 127 128 129 129 130 130 132 132 132 133 134 134 134 135 135 136 137 137 137 138 139 139 139 140 141 142 142 142 142 142
Total time taken by merge sort for 10000 elements in random order is: 0.021839
--------------------------------------------------------------------------------------------------------
PS H:\Nirma\CTA\Practical-2> █
```

For array size = 50000



```
PS H:\Nirma\CTA\Practical-2> g++ -o mer mergeanalyze.cpp
PS H:\Nirma\CTA\Practical-2> ./mer
Enter the size of an Array: 50000
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131
 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 16
6 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
Total time taken by merge sort for 50000 elements in Ascending order is: 0.082933
--------------------------------------------------------------------------------------------------------
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131
 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 16
6 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
Total time taken by merge sort for 50000 elements in Decending order is: 0.094932
--------------------------------------------------------------------------------------------------------
0 1 3 4 6 6 6 6 8 8 9 9 9 10 10 11 11 12 13 15 15 15 15 19 19 20 20 21 21 22 22 23 23 23 24 24 24 25 25 25 26 28 28 28 28 29 30 30 31 32 33
 34 35 35 36 37 37 38 38 39 39 40 40 41 41 41 42 42 42 43 43 44 45 45 45 47 47 47 48 48 49 50 50 50 50 52 53 53 54 55 56 57 58 58 59 59 61
63 65 65 66 66 67 68 68 70 70 71 71 71 72 73 73 74 74 75 77 77 78 79 79 80 80 81 83 83 84 85 85 87 88 89 90 90 90 90 90 90 92 94 95 96 96 9
6 98 99 100 101 101 102 102 104 106 107 107 108 109 109 109 110 110 111 112 113 115 115 115 117 118 118 118 119 119 121 121 122 123 125 126
 127 127 128 129 130 131 132 133 134 134 134 135 135 136 137 137 138 138 138 139 139
Total time taken by merge sort for 50000 elements in random order is: 0.102121
--------------------------------------------------------------------------------------------------------
PS H:\Nirma\CTA\Practical-2> █
```

For array size = 100000

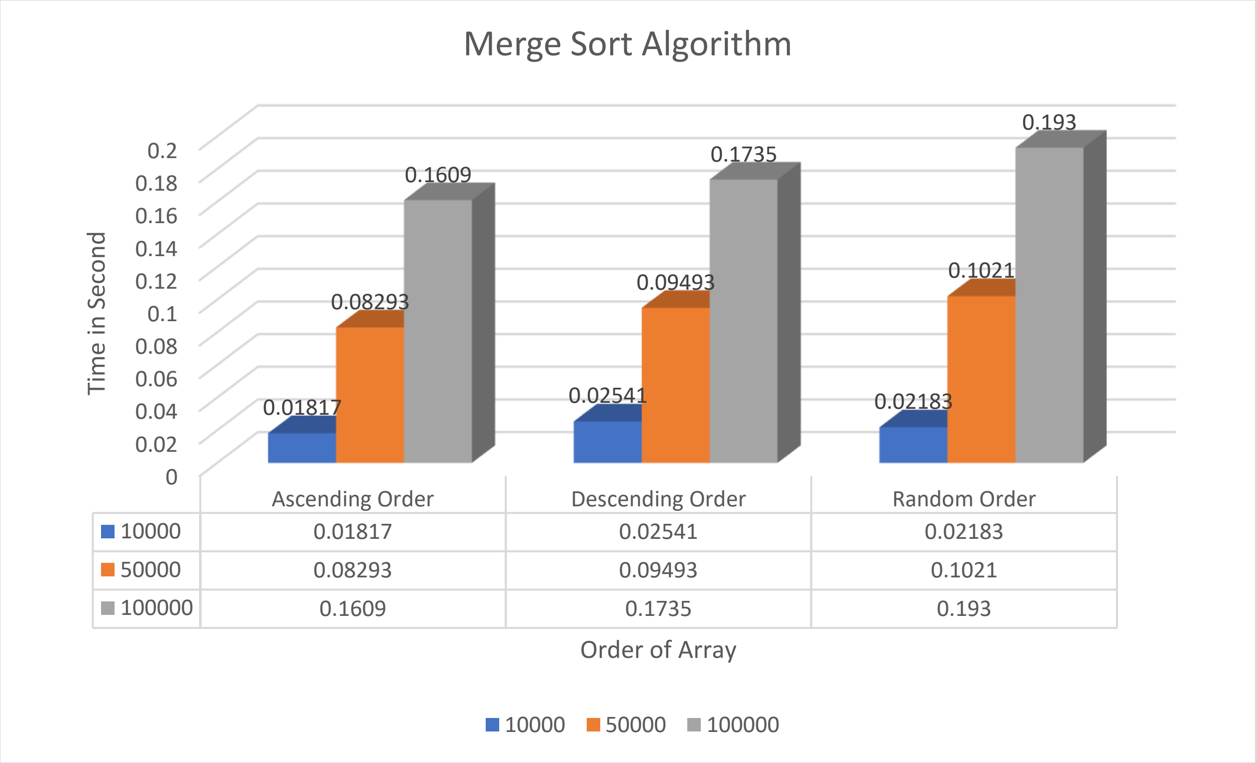


```
PS H:\Nirma\CTA\Practical-2> g++ -o mer mergeanalyze.cpp
PS H:\Nirma\CTA\Practical-2> ./mer
Enter the size of an Array: 100000
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131
 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 16
6 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
Total time taken by merge sort for 100000 elements in Ascending order is: 0.160915
--------------------------------------------------------------------------------------------------------
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131
 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 16
6 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
Total time taken by merge sort for 100000 elements in Decending order is: 0.173596
--------------------------------------------------------------------------------------------------------
0 0 1 1 1 1 1 1 3 3 3 4 4 4 4 4 5 5 6 6 6 6 7 8 8 8 8 8 8 9 9 9 9 9 10 10 10 10 11 11 11 12 12 13 15 15 15 15 15 16 17 17 18 19 19 19 20 20
 20 20 21 21 21 22 22 22 23 23 23 23 24 24 24 24 25 25 25 25 25 26 26 26 27 28 28 28 28 28 28 28 28 29 29 30 30 30 31 31 32 33 33 33 33
33 33 34 34 34 35 35 35 35 36 36 37 37 37 38 38 38 38 38 39 39 39 39 40 40 41 41 41 41 41 41 42 42 42 42 43 43 43 43 44 44 44 45 45 45 45 4
5 45 46 47 47 47 47 48 48 48 48 48 48 49 49 49 50 50 50 50 51 51 51 51 52 53 53 53 53 54 54 54 55 55 55 56 56 57 57 57 57 57 58 58 59 59 59
 59 59 59
Total time taken by merge sort for 100000 elements in random order is: 0.193091
--------------------------------------------------------------------------------------------------------
PS H:\Nirma\CTA\Practical-2> █
```

- **Graphs & Output Data**

Here are the One tables that describes the output generated by the above code for ascending, descending and random array for size 10000, 50000 & 100000.

	Ascending Order	Descending Order	Random Order
10000	**0.01817**	**0.02541**	**0.02183**
50000	**0.08293**	**0.09493**	**0.1021**
100000	**0.1609**	**0.1735**	**0.193**



	Ascending Order	Descending Order	Random Order
■ 10000	0.01817	0.02541	0.02183
■ 50000	0.08293	0.09493	0.1021
■ 100000	0.1609	0.1735	0.193

- In above graph x-axis contains the time in second, y-axis contains the order of array for 10000, 50000, and 100000 input size.

- **Analysis**

  Merge sort is dividing and conquer algorithm so from my data as per the array size increase the time taken by merge sort algorithm increases in all three cases that is ascending, descending and random order array. Now for the 10000 elements array for ascending order it takes lesser time than all because the no of swaps and comparisons are less then other order because in other order the no of swaps and comparisons are more then the ascending order.

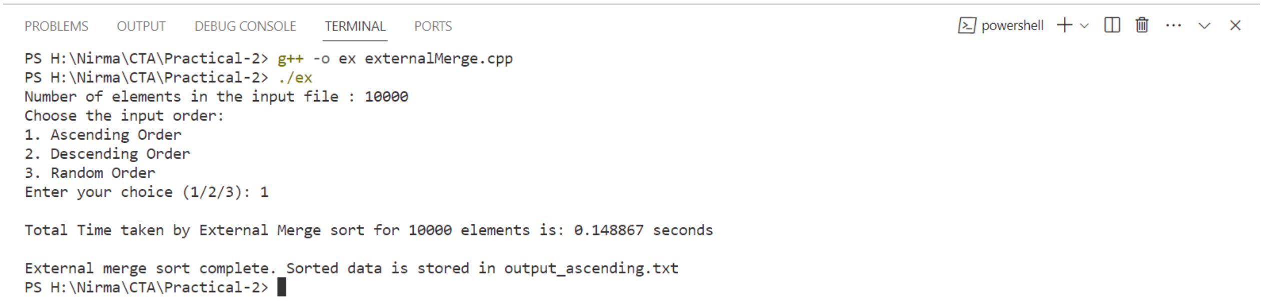
Code for External Merge sort-

```
#include <bits/stdc++.h>
using namespace std;
using namespace std::chrono;

const int CHUNK_SIZE = 1000;

// Function to generate a random input file with n elements
void generateAscendingInput(const string &inputFile, int n)
{
    ofstream outFile(inputFile);
    if (!outFile)
    {
        cerr << "Error: Cannot open output file for writing." << endl;
        return;
    }

    for (int i = 1; i <= n; ++i)
    {
        outFile << i << endl;
    }

    outFile.close();
}

void generateDescendingInput(const string &inputFile, int n)
{
    ofstream outFile(inputFile);
    if (!outFile)
    {
        cerr << "Error: Cannot open output file for writing." << endl;
        return;
    }
    for (int i = n; i >= 1; --i)
    {
        outFile << i << endl;
    }

    outFile.close();
}

void generateRandomInput(const string &inputFile, int n)
{
    ofstream outFile(inputFile);
    if (!outFile)
    {
        cerr << "Error: Cannot open output file for writing." << endl;
        return;
    }

    srand(static_cast<unsigned int>(time(nullptr)));

    for (int i = 0; i < n; ++i)
    {
        int value = rand() % 10000; // Adjust the range as needed
        outFile << value << endl;
    }

    outFile.close();
}

// Function to merge sorted chunks
void mergeChunks(const vector<string> &chunkFiles, const string &outputFile)
{
    vector<ifstream> chunkStreams;
    for (const auto &chunkFile : chunkFiles)
    {
        chunkStreams.emplace_back(chunkFile);
    }

    vector<int> currentValues(chunkStreams.size());
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> minHeap;

    // Initialize currentValues with the first element from each chunk
    for (int i = 0; i < chunkStreams.size(); ++i)
    {
        if (chunkStreams[i] >> currentValues[i])
        {
            minHeap.emplace(currentValues[i], i);
        }
    }

    ofstream outFile(outputFile);

    while (!minHeap.empty())
    {
        auto [value, chunkIndex] = minHeap.top();
        minHeap.pop();
        outFile << value << endl;

        if (chunkStreams[chunkIndex] >> currentValues[chunkIndex])
        {
            minHeap.emplace(currentValues[chunkIndex], chunkIndex);
        }
    }

    for (const auto &chunkFile : chunkFiles)
    {
        remove(chunkFile.c_str()); // Clean up temporary chunk files
    }
}

// Function to perform external merge sort
double externalMergeSort(const string &inputFile, const string &outputFile, int
maxElementsPerChunk)
{
    ifstream inFile(inputFile);
    if (!inFile)
    {
        cerr << "Error: Cannot open input file." << endl;
        return -1.0;
    }

    int chunkNumber = 0;
    vector<string> chunkFiles;

    auto start_time = high_resolution_clock::now(); // Start measuring time

    while (!inFile.eof())
    {
        vector<int> chunkData;
        chunkData.reserve(maxElementsPerChunk);
        for (int i = 0; i < maxElementsPerChunk; ++i)
        {
            int value;
            if (inFile >> value)
            {
                chunkData.push_back(value);
            }
            else
            {
                break;
            }
        }

        sort(chunkData.begin(), chunkData.end());
        string chunkFile = "chunk_" + to_string(chunkNumber) + ".tmp";
        ofstream chunkOutFile(chunkFile);

        for (int value : chunkData)
        {
            chunkOutFile << value << endl;
        }

        chunkFiles.push_back(chunkFile);
        ++chunkNumber;
    }

    inFile.close();

    // Merge sorted chunks
    mergeChunks(chunkFiles, outputFile);

    auto end_time = high_resolution_clock::now(); // Stop measuring time
    auto total_time = duration<double>(end_time - start_time);

    return total_time.count();
}

int main()
{
    int maxElementsPerChunk = CHUNK_SIZE;
    int totalElements;
    cout << "Number of elements in the input file : ";
    cin >> totalElements;

    cout << "Choose the input order:" << endl;
    cout << "1. Ascending Order" << endl;
    cout << "2. Descending Order" << endl;
    cout << "3. Random Order" << endl;
    int choice;
    cout << "Enter your choice (1/2/3): ";
    cin >> choice;

    string inputFile, outputFile;
    switch (choice)
    {
    case 1:
        inputFile = "input_ascending.txt";
        outputFile = "output_ascending.txt";
        generateAscendingInput(inputFile, totalElements);
        break;
    case 2:
        inputFile = "input_descending.txt";
        outputFile = "output_descending.txt";
        generateDescendingInput(inputFile, totalElements);
        break;
    case 3:
        inputFile = "input_random.txt";
        outputFile = "output_random.txt";
        generateRandomInput(inputFile, totalElements);
        break;
    default:
        cout << "Invalid choice. Exiting." << endl;
        return 1;
    }

    double elapsedTime = externalMergeSort(inputFile, outputFile,
maxElementsPerChunk);

    cout << endl
         << "Total Time taken by External Merge sort for " << totalElements << "
elements is: " << elapsedTime << " seconds" << endl;
    cout << endl
         << "External merge sort complete. Sorted data is stored in " <<
outputFile << endl;

    return 0;
}
```

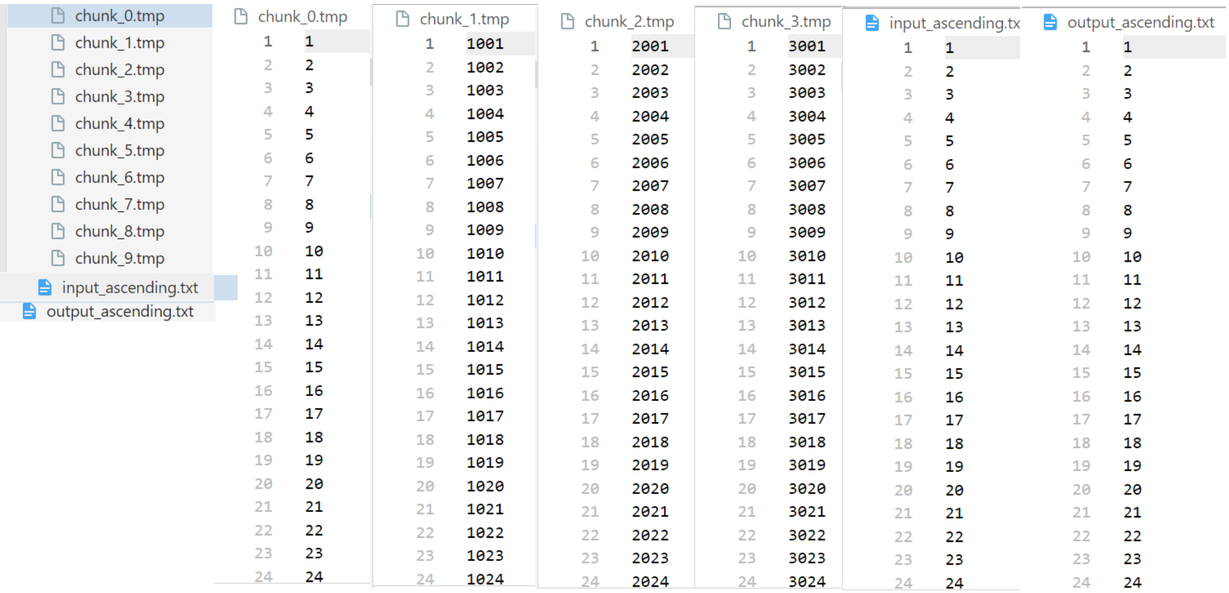
- **Output**

For array size = 10000

Ascending Order



```
PS H:\Nirma\CTA\Practical-2> g++ -o ex externalMerge.cpp
PS H:\Nirma\CTA\Practical-2> ./ex
Number of elements in the input file : 10000
Choose the input order:
1. Ascending Order
2. Descending Order
3. Random Order
Enter your choice (1/2/3): 1

Total Time taken by External Merge sort for 10000 elements is: 0.148867 seconds

External merge sort complete. Sorted data is stored in output_ascending.txt
PS H:\Nirma\CTA\Practical-2>
```



chunk_0.tmp

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

chunk_1.tmp

```
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
```

chunk_2.tmp

```
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
```

chunk_3.tmp

```
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
```

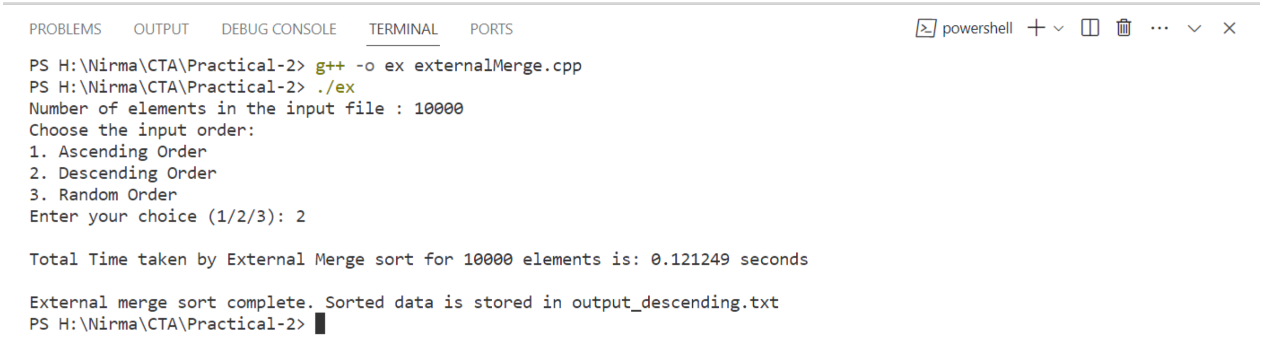
input_ascending.tx

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

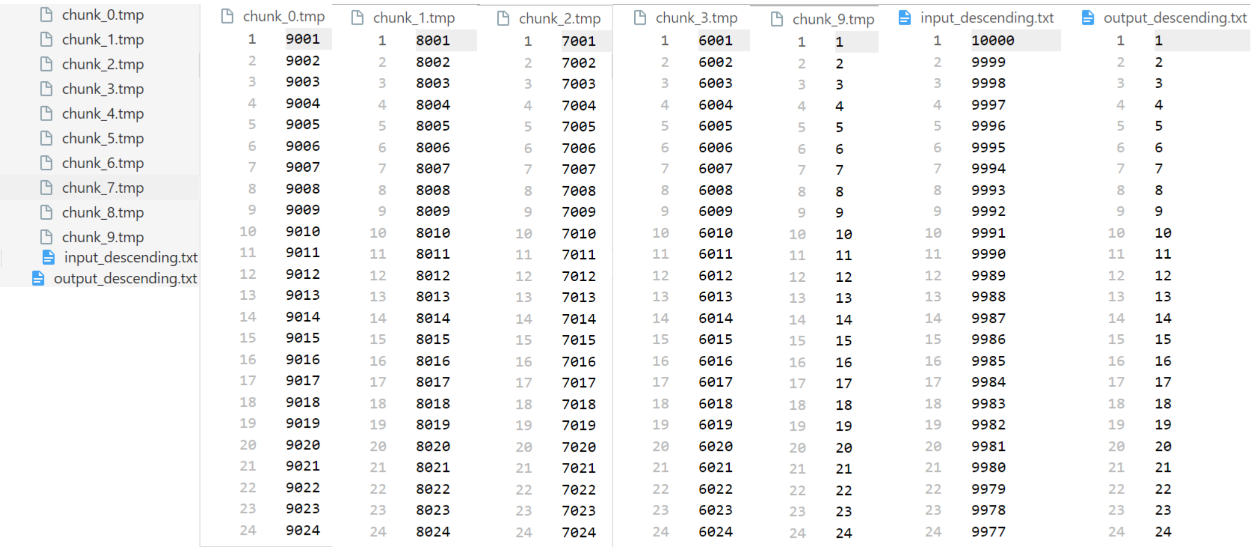
output_ascending.txt

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

Descending Order

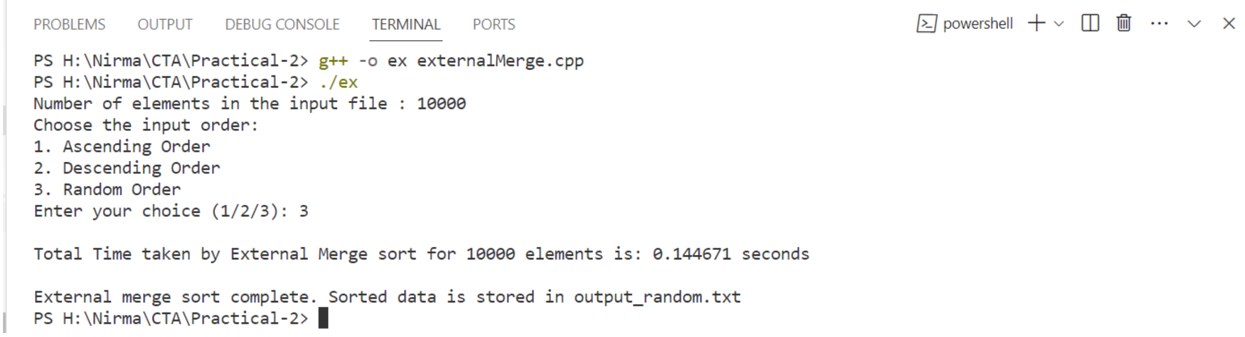


```
PS H:\Nirma\CTA\Practical-2> g++ -o ex externalMerge.cpp
PS H:\Nirma\CTA\Practical-2> ./ex
Number of elements in the input file : 10000
Choose the input order:
1. Ascending Order
2. Descending Order
3. Random Order
Enter your choice (1/2/3): 2

Total Time taken by External Merge sort for 10000 elements is: 0.121249 seconds

External merge sort complete. Sorted data is stored in output_descending.txt
PS H:\Nirma\CTA\Practical-2>
```

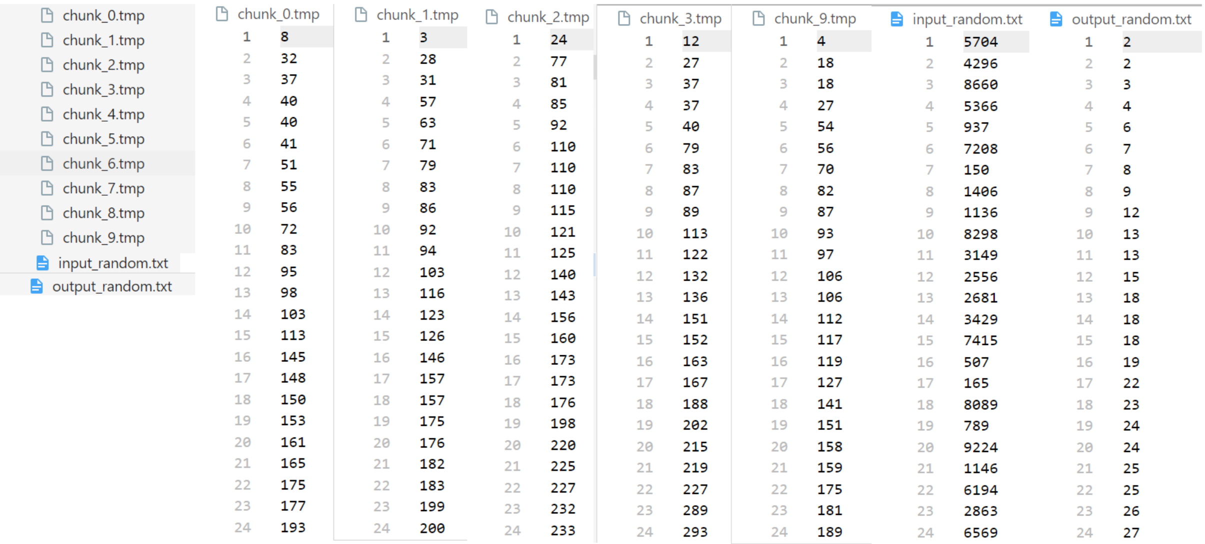

chunk_0.tmp
chunk_1.tmp
chunk_2.tmp
chunk_3.tmp
chunk_4.tmp
chunk_5.tmp
chunk_6.tmp
chunk_7.tmp
chunk_8.tmp
chunk_9.tmp
input_descending.txt
output_descending.txt

chunk_0.tmp	chunk_1.tmp	chunk_2.tmp	chunk_3.tmp	chunk_9.tmp	input_descending.txt	output_descending.txt
9001	8001	7001	6001	1	10000	1
9002	8002	7002	6002	2	9999	2
9003	8003	7003	6003	3	9998	3
9004	8004	7004	6004	4	9997	4
9005	8005	7005	6005	5	9996	5
9006	8006	7006	6006	6	9995	6
9007	8007	7007	6007	7	9994	7
9008	8008	7008	6008	8	9993	8
9009	8009	7009	6009	9	9992	9
9010	8010	7010	6010	10	9991	10
9011	8011	7011	6011	11	9990	11
9012	8012	7012	6012	12	9989	12
9013	8013	7013	6013	13	9988	13
9014	8014	7014	6014	14	9987	14
9015	8015	7015	6015	15	9986	15
9016	8016	7016	6016	16	9985	16
9017	8017	7017	6017	17	9984	17
9018	8018	7018	6018	18	9983	18
9019	8019	7019	6019	19	9982	19
9020	8020	7020	6020	20	9981	20
9021	8021	7021	6021	21	9980	21
9022	8022	7022	6022	22	9979	22
9023	8023	7023	6023	23	9978	23
9024	8024	7024	6024	24	9977	24

Random Order



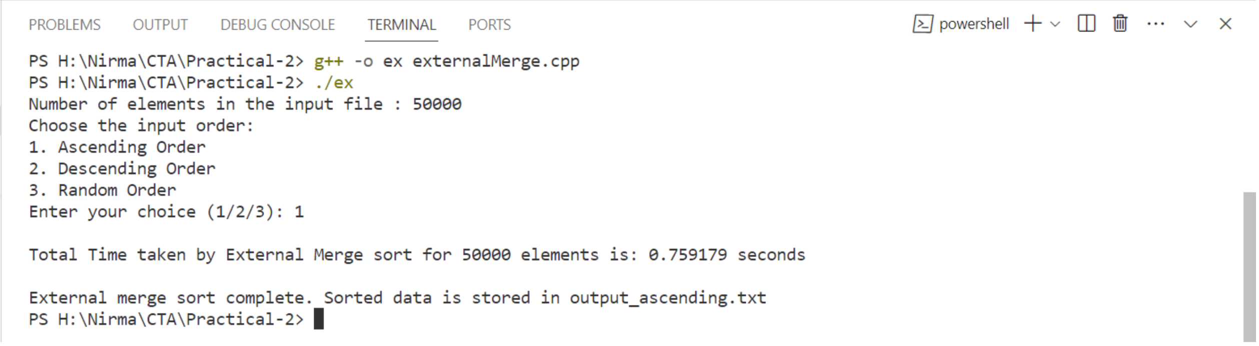
```
PS H:\Nirma\CTA\Practical-2> g++ -o ex externalMerge.cpp
PS H:\Nirma\CTA\Practical-2> ./ex
Number of elements in the input file : 10000
Choose the input order:
1. Ascending Order
2. Descending Order
3. Random Order
Enter your choice (1/2/3): 3

Total Time taken by External Merge sort for 10000 elements is: 0.144671 seconds

External merge sort complete. Sorted data is stored in output_random.txt
PS H:\Nirma\CTA\Practical-2> 
```


chunk_0.tmp
chunk_1.tmp
chunk_2.tmp
chunk_3.tmp
chunk_4.tmp
chunk_5.tmp
chunk_6.tmp
chunk_7.tmp
chunk_8.tmp
chunk_9.tmp
input_random.txt
output_random.txt

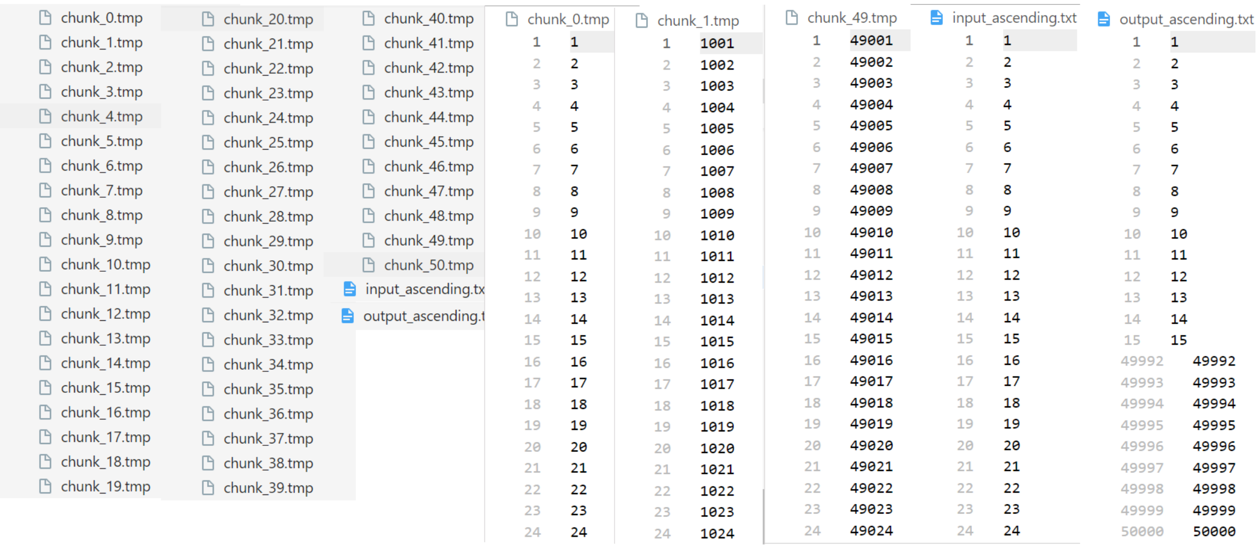
chunk_0.tmp	chunk_1.tmp	chunk_2.tmp	chunk_3.tmp	chunk_9.tmp	input_random.txt	output_random.txt
8	3	24	12	4	5704	2
32	28	77	27	18	4296	2
37	31	81	37	18	8660	3
40	57	85	37	27	5366	4
40	63	92	40	54	937	6
41	71	110	79	56	7208	7
51	79	110	83	70	150	8
55	83	110	87	82	1406	9
56	86	115	89	87	1136	12
72	92	121	113	93	8298	13
83	94	125	122	97	3149	13
95	103	140	132	106	2556	15
98	116	143	136	106	2681	18
103	123	156	151	112	3429	18
113	126	160	152	117	7415	18
145	146	173	163	119	507	19
148	157	173	167	127	165	22
150	157	176	188	141	8089	23
153	175	198	202	151	789	24
161	176	220	215	158	9224	24
165	182	225	219	159	1146	25
175	183	227	227	175	6194	25
177	199	232	289	181	2863	26
193	200	233	293	189	6569	27

For array size = 50000

Ascending Order



```
PS H:\Nirma\CTA\Practical-2> g++ -o ex externalMerge.cpp
PS H:\Nirma\CTA\Practical-2> ./ex
Number of elements in the input file : 50000
Choose the input order:
1. Ascending Order
2. Descending Order
3. Random Order
Enter your choice (1/2/3): 1

Total Time taken by External Merge sort for 50000 elements is: 0.759179 seconds

External merge sort complete. Sorted data is stored in output_ascending.txt
PS H:\Nirma\CTA\Practical-2>
```



```
chunk_0.tmp     chunk_20.tmp    chunk_40.tmp
chunk_1.tmp     chunk_21.tmp    chunk_41.tmp
chunk_2.tmp     chunk_22.tmp    chunk_42.tmp
chunk_3.tmp     chunk_23.tmp    chunk_43.tmp
chunk_4.tmp     chunk_24.tmp    chunk_44.tmp
chunk_5.tmp     chunk_25.tmp    chunk_45.tmp
chunk_6.tmp     chunk_26.tmp    chunk_46.tmp
chunk_7.tmp     chunk_27.tmp    chunk_47.tmp
chunk_8.tmp     chunk_28.tmp    chunk_48.tmp
chunk_9.tmp     chunk_29.tmp    chunk_49.tmp
chunk_10.tmp    chunk_30.tmp    chunk_50.tmp
chunk_11.tmp    chunk_31.tmp    input_ascending.tx
chunk_12.tmp    chunk_32.tmp    output_ascending.t
chunk_13.tmp    chunk_33.tmp
chunk_14.tmp    chunk_34.tmp
chunk_15.tmp    chunk_35.tmp
chunk_16.tmp    chunk_36.tmp
chunk_17.tmp    chunk_37.tmp
chunk_18.tmp    chunk_38.tmp
chunk_19.tmp    chunk_39.tmp
```

chunk_0.tmp

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

chunk_1.tmp

```
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
```

chunk_49.tmp

```
49001
49002
49003
49004
49005
49006
49007
49008
49009
49010
49011
49012
49013
49014
49015
49016
49017
49018
49019
49020
49021
49022
49023
49024
```

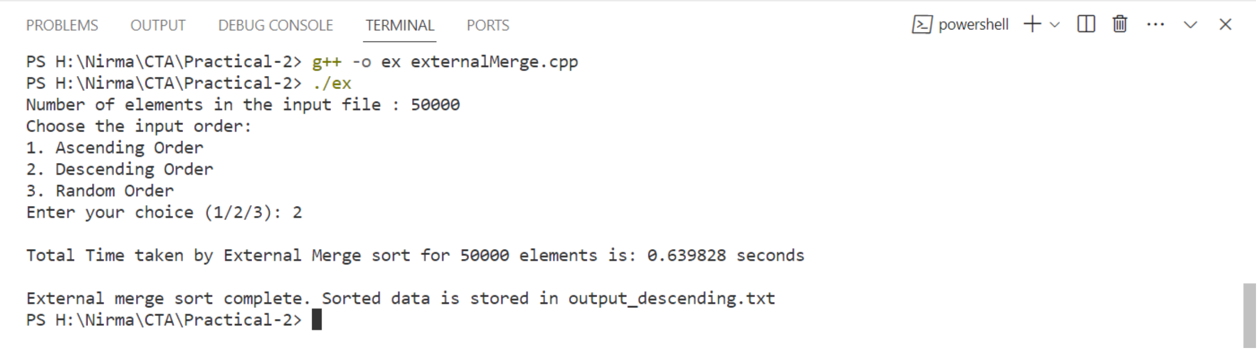
input_ascending.txt

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

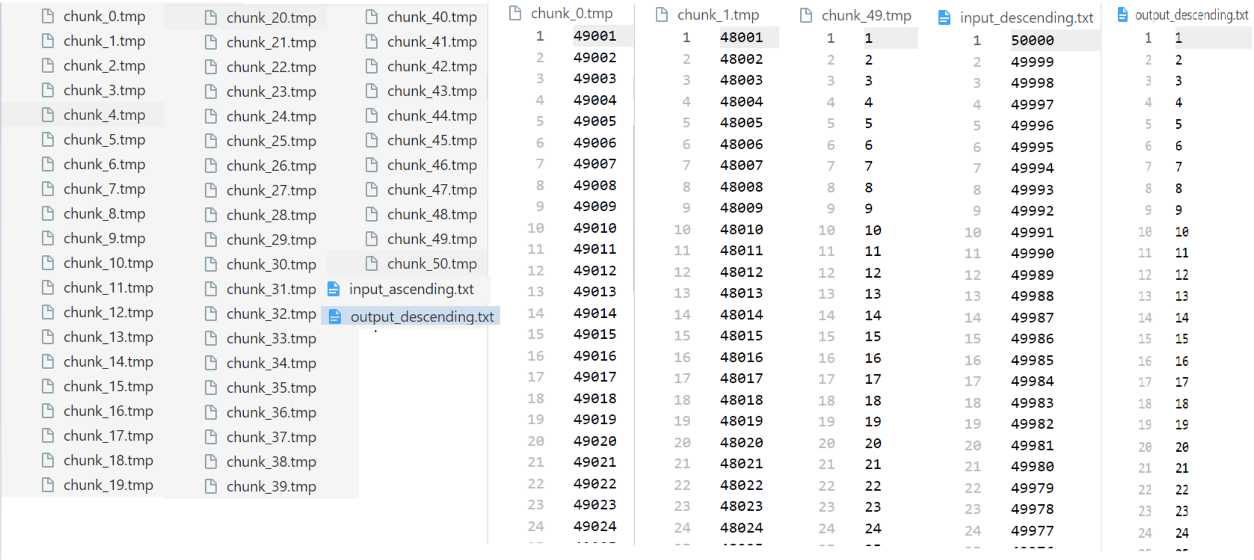
output_ascending.txt

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
49992
49993
49994
49995
49996
49997
49998
49999
50000
```

Descending Order



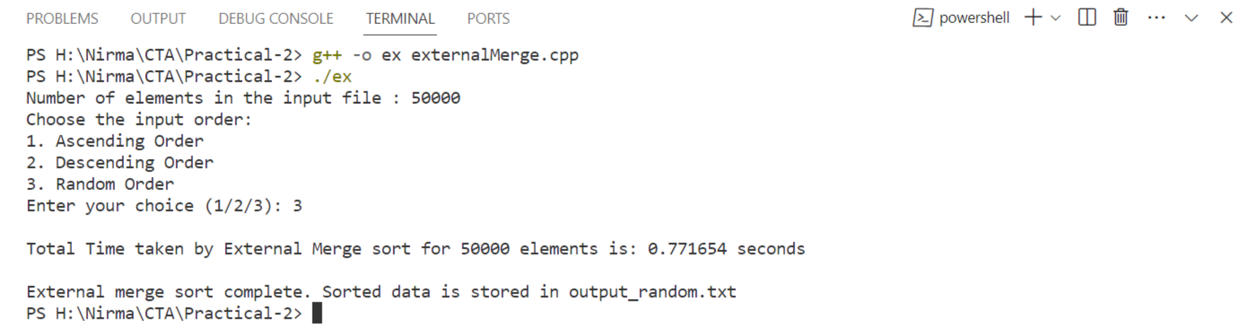
```
PS H:\Nirma\CTA\Practical-2> g++ -o ex externalMerge.cpp
PS H:\Nirma\CTA\Practical-2> ./ex
Number of elements in the input file : 50000
Choose the input order:
1. Ascending Order
2. Descending Order
3. Random Order
Enter your choice (1/2/3): 2

Total Time taken by External Merge sort for 50000 elements is: 0.639828 seconds

External merge sort complete. Sorted data is stored in output_descending.txt
PS H:\Nirma\CTA\Practical-2>
```



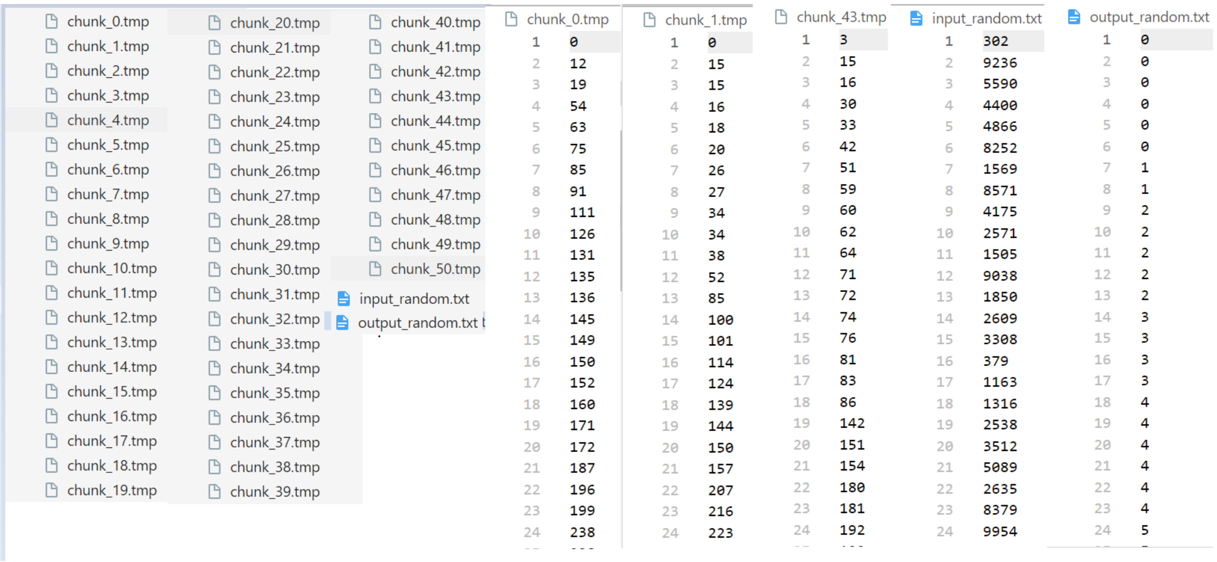
```
chunk_0.tmp     chunk_20.tmp    chunk_40.tmp
chunk_1.tmp     chunk_21.tmp    chunk_41.tmp
chunk_2.tmp     chunk_22.tmp    chunk_42.tmp
chunk_3.tmp     chunk_23.tmp    chunk_43.tmp
chunk_4.tmp     chunk_24.tmp    chunk_44.tmp
chunk_5.tmp     chunk_25.tmp    chunk_45.tmp
chunk_6.tmp     chunk_26.tmp    chunk_46.tmp
chunk_7.tmp     chunk_27.tmp    chunk_47.tmp
chunk_8.tmp     chunk_28.tmp    chunk_48.tmp
chunk_9.tmp     chunk_29.tmp    chunk_49.tmp
chunk_10.tmp    chunk_30.tmp    chunk_50.tmp
chunk_11.tmp    chunk_31.tmp    input_ascending.txt
chunk_12.tmp    chunk_32.tmp    output_descending.txt
chunk_13.tmp    chunk_33.tmp
chunk_14.tmp    chunk_34.tmp
chunk_15.tmp    chunk_35.tmp
chunk_16.tmp    chunk_36.tmp
chunk_17.tmp    chunk_37.tmp
chunk_18.tmp    chunk_38.tmp
chunk_19.tmp    chunk_39.tmp
```

```
chunk_0.tmp  chunk_1.tmp  chunk_49.tmp  input_descending.txt  output_descending.txt
49001        48001        1             50000                 1
49002        48002        2             49999                 2
49003        48003        3             49998                 3
49004        48004        4             49997                 4
49005        48005        5             49996                 5
49006        48006        6             49995                 6
49007        48007        7             49994                 7
49008        48008        8             49993                 8
49009        48009        9             49992                 9
49010        48010        10            49991                 10
49011        48011        11            49990                 11
49012        48012        12            49989                 12
49013        48013        13            49988                 13
49014        48014        14            49987                 14
49015        48015        15            49986                 15
49016        48016        16            49985                 16
49017        48017        17            49984                 17
49018        48018        18            49983                 18
49019        48019        19            49982                 19
49020        48020        20            49981                 20
49021        48021        21            49980                 21
49022        48022        22            49979                 22
49023        48023        23            49978                 23
49024        48024        24            49977                 24
```

Random Order



```
PS H:\Nirma\CTA\Practical-2> g++ -o ex externalMerge.cpp
PS H:\Nirma\CTA\Practical-2> ./ex
Number of elements in the input file : 50000
Choose the input order:
1. Ascending Order
2. Descending Order
3. Random Order
Enter your choice (1/2/3): 3

Total Time taken by External Merge sort for 50000 elements is: 0.771654 seconds

External merge sort complete. Sorted data is stored in output_random.txt
PS H:\Nirma\CTA\Practical-2> 
```

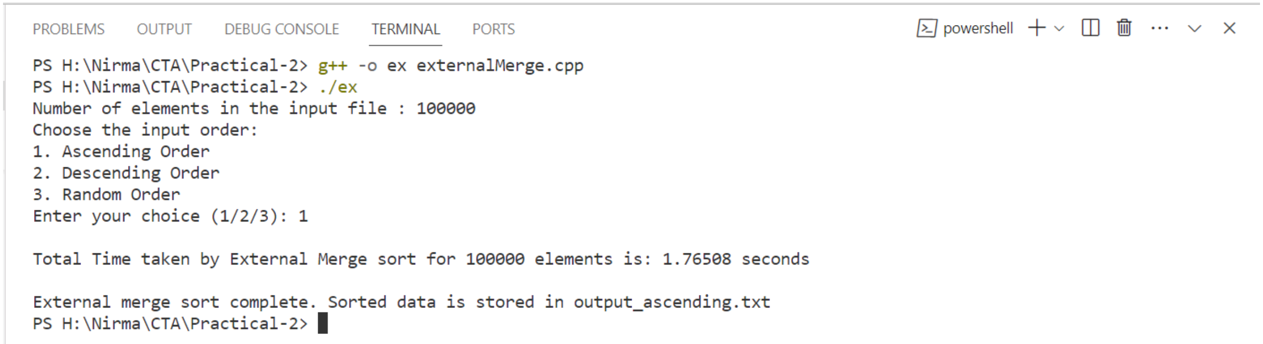


```
chunk_0.tmp     chunk_20.tmp    chunk_40.tmp
chunk_1.tmp     chunk_21.tmp    chunk_41.tmp
chunk_2.tmp     chunk_22.tmp    chunk_42.tmp
chunk_3.tmp     chunk_23.tmp    chunk_43.tmp
chunk_4.tmp     chunk_24.tmp    chunk_44.tmp
chunk_5.tmp     chunk_25.tmp    chunk_45.tmp
chunk_6.tmp     chunk_26.tmp    chunk_46.tmp
chunk_7.tmp     chunk_27.tmp    chunk_47.tmp
chunk_8.tmp     chunk_28.tmp    chunk_48.tmp
chunk_9.tmp     chunk_29.tmp    chunk_49.tmp
chunk_10.tmp    chunk_30.tmp    chunk_50.tmp
chunk_11.tmp    chunk_31.tmp    input_random.txt
chunk_12.tmp    chunk_32.tmp    output_random.txt
chunk_13.tmp    chunk_33.tmp
chunk_14.tmp    chunk_34.tmp
chunk_15.tmp    chunk_35.tmp
chunk_16.tmp    chunk_36.tmp
chunk_17.tmp    chunk_37.tmp
chunk_18.tmp    chunk_38.tmp
chunk_19.tmp    chunk_39.tmp
```

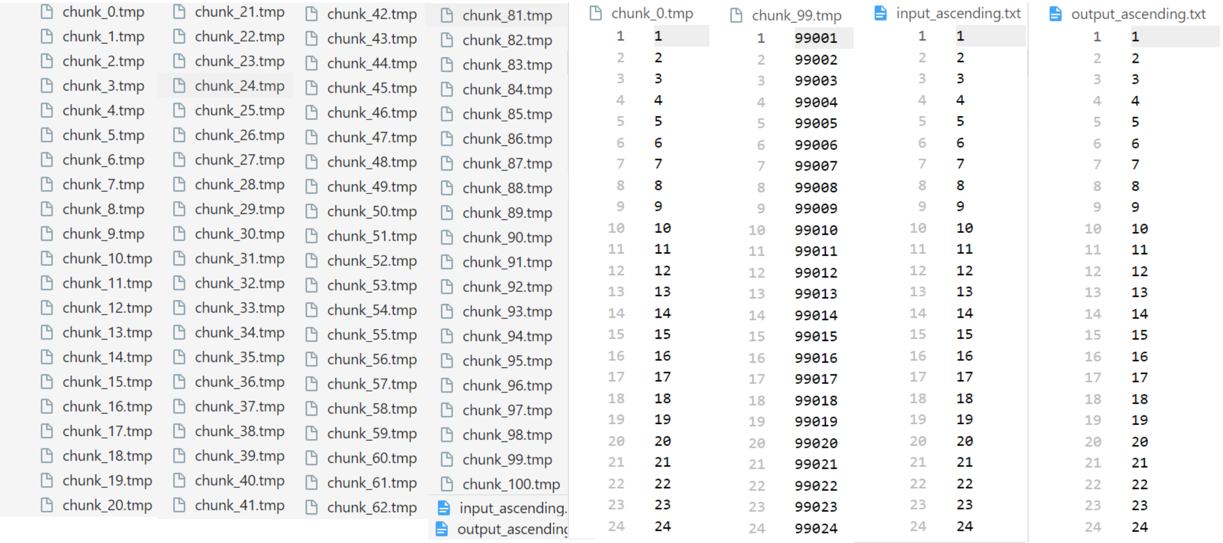
```
chunk_0.tmp  chunk_1.tmp  chunk_43.tmp  input_random.txt  output_random.txt
0            0            3             302               0
12           15           15            9236              0
19           15           16            5590              0
54           16           30            4400              0
63           18           33            4866              0
75           20           42            8252              0
85           26           51            1569              1
91           27           59            8571              1
111          34           60            4175              2
126          34           62            2571              2
131          38           64            1505              2
135          52           71            9038              2
136          85           72            1850              2
145          100          74            2609              3
149          101          76            3308              3
150          114          81            379               3
152          124          83            1163              3
160          139          86            1316              4
171          144          142           2538              4
172          150          151           3512              4
187          157          154           5089              4
196          207          180           2635              4
199          216          181           8379              4
238          223          192           9954              5
```

For array size = 100000

Ascending Order



```
PS H:\Nirma\CTA\Practical-2> g++ -o ex externalMerge.cpp
PS H:\Nirma\CTA\Practical-2> ./ex
Number of elements in the input file : 100000
Choose the input order:
1. Ascending Order
2. Descending Order
3. Random Order
Enter your choice (1/2/3): 1

Total Time taken by External Merge sort for 100000 elements is: 1.76508 seconds

External merge sort complete. Sorted data is stored in output_ascending.txt
PS H:\Nirma\CTA\Practical-2> 
```



chunk_0.tmp
chunk_1.tmp
chunk_2.tmp
chunk_3.tmp
chunk_4.tmp
chunk_5.tmp
chunk_6.tmp
chunk_7.tmp
chunk_8.tmp
chunk_9.tmp
chunk_10.tmp
chunk_11.tmp
chunk_12.tmp
chunk_13.tmp
chunk_14.tmp
chunk_15.tmp
chunk_16.tmp
chunk_17.tmp
chunk_18.tmp
chunk_19.tmp
chunk_20.tmp
chunk_21.tmp
chunk_22.tmp
chunk_23.tmp
chunk_24.tmp
chunk_25.tmp
chunk_26.tmp
chunk_27.tmp
chunk_28.tmp
chunk_29.tmp
chunk_30.tmp
chunk_31.tmp
chunk_32.tmp
chunk_33.tmp
chunk_34.tmp
chunk_35.tmp
chunk_36.tmp
chunk_37.tmp
chunk_38.tmp
chunk_39.tmp
chunk_40.tmp
chunk_41.tmp
chunk_42.tmp
chunk_43.tmp
chunk_44.tmp
chunk_45.tmp
chunk_46.tmp
chunk_47.tmp
chunk_48.tmp
chunk_49.tmp
chunk_50.tmp
chunk_51.tmp
chunk_52.tmp
chunk_53.tmp
chunk_54.tmp
chunk_55.tmp
chunk_56.tmp
chunk_57.tmp
chunk_58.tmp
chunk_59.tmp
chunk_60.tmp
chunk_61.tmp
chunk_62.tmp
chunk_81.tmp
chunk_82.tmp
chunk_83.tmp
chunk_84.tmp
chunk_85.tmp
chunk_86.tmp
chunk_87.tmp
chunk_88.tmp
chunk_89.tmp
chunk_90.tmp
chunk_91.tmp
chunk_92.tmp
chunk_93.tmp
chunk_94.tmp
chunk_95.tmp
chunk_96.tmp
chunk_97.tmp
chunk_98.tmp
chunk_99.tmp
chunk_100.tmp
input_ascending.
output_ascending

chunk_0.tmp

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

chunk_99.tmp

```
99001
99002
99003
99004
99005
99006
99007
99008
99009
99010
99011
99012
99013
99014
99015
99016
99017
99018
99019
99020
99021
99022
99023
99024
```

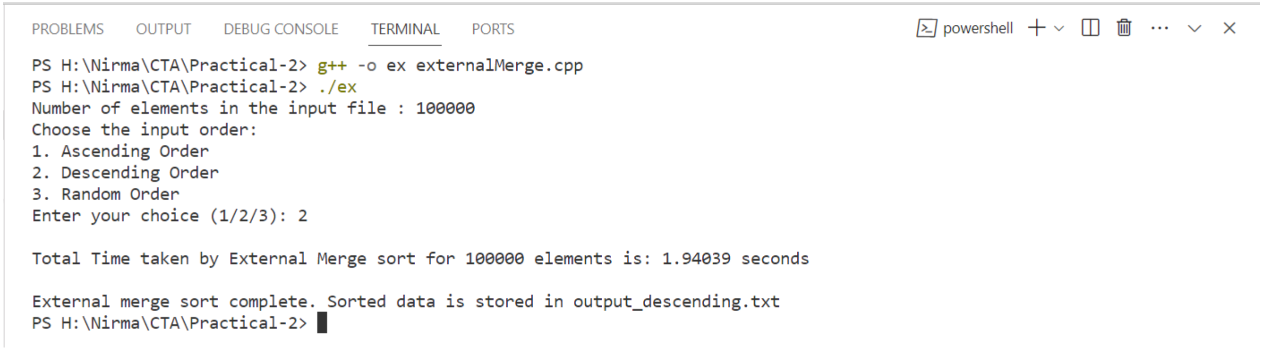
input_ascending.txt

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

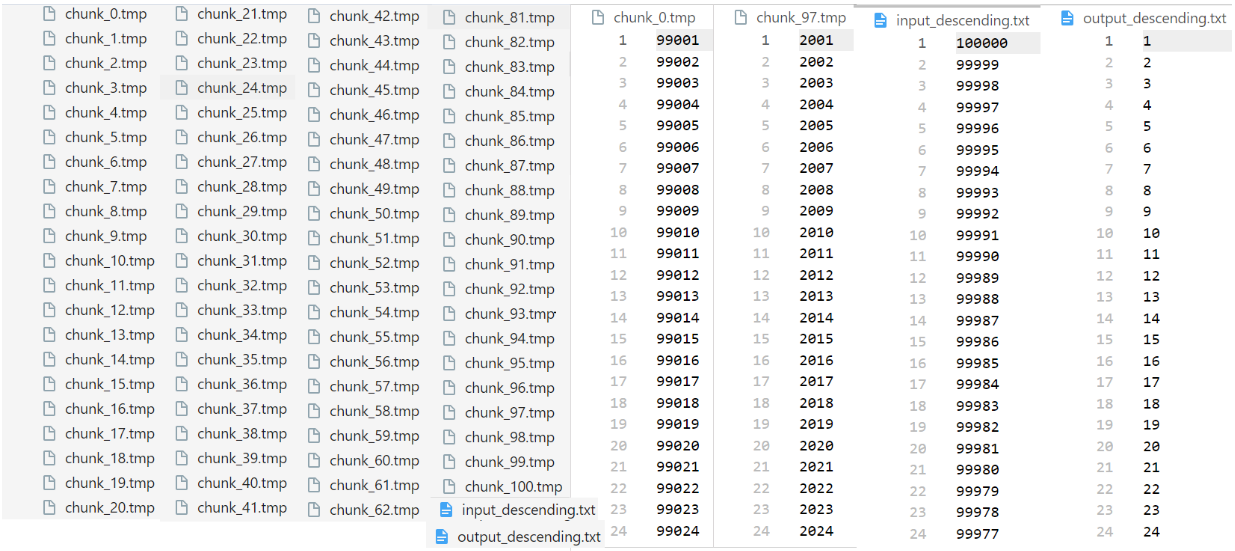
output_ascending.txt

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

Descending Order



```
PS H:\Nirma\CTA\Practical-2> g++ -o ex externalMerge.cpp
PS H:\Nirma\CTA\Practical-2> ./ex
Number of elements in the input file : 100000
Choose the input order:
1. Ascending Order
2. Descending Order
3. Random Order
Enter your choice (1/2/3): 2

Total Time taken by External Merge sort for 100000 elements is: 1.94039 seconds

External merge sort complete. Sorted data is stored in output_descending.txt
PS H:\Nirma\CTA\Practical-2> 
```



```
chunk_0.tmp
chunk_1.tmp
chunk_2.tmp
chunk_3.tmp
chunk_4.tmp
chunk_5.tmp
chunk_6.tmp
chunk_7.tmp
chunk_8.tmp
chunk_9.tmp
chunk_10.tmp
chunk_11.tmp
chunk_12.tmp
chunk_13.tmp
chunk_14.tmp
chunk_15.tmp
chunk_16.tmp
chunk_17.tmp
chunk_18.tmp
chunk_19.tmp
chunk_20.tmp
chunk_21.tmp
chunk_22.tmp
chunk_23.tmp
chunk_24.tmp
chunk_25.tmp
chunk_26.tmp
chunk_27.tmp
chunk_28.tmp
chunk_29.tmp
chunk_30.tmp
chunk_31.tmp
chunk_32.tmp
chunk_33.tmp
chunk_34.tmp
chunk_35.tmp
chunk_36.tmp
chunk_37.tmp
chunk_38.tmp
chunk_39.tmp
chunk_40.tmp
chunk_41.tmp
chunk_42.tmp
chunk_43.tmp
chunk_44.tmp
chunk_45.tmp
chunk_46.tmp
chunk_47.tmp
chunk_48.tmp
chunk_49.tmp
chunk_50.tmp
chunk_51.tmp
chunk_52.tmp
chunk_53.tmp
chunk_54.tmp
chunk_55.tmp
chunk_56.tmp
chunk_57.tmp
chunk_58.tmp
chunk_59.tmp
chunk_60.tmp
chunk_61.tmp
chunk_62.tmp
chunk_81.tmp
chunk_82.tmp
chunk_83.tmp
chunk_84.tmp
chunk_85.tmp
chunk_86.tmp
chunk_87.tmp
chunk_88.tmp
chunk_89.tmp
chunk_90.tmp
chunk_91.tmp
chunk_92.tmp
chunk_93.tmp
chunk_94.tmp
chunk_95.tmp
chunk_96.tmp
chunk_97.tmp
chunk_98.tmp
chunk_99.tmp
chunk_100.tmp
input_descending.txt
output_descending.txt
```

chunk_0.tmp

```
99001
99002
99003
99004
99005
99006
99007
99008
99009
99010
99011
99012
99013
99014
99015
99016
99017
99018
99019
99020
99021
99022
99023
99024
```

chunk_97.tmp

```
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
```

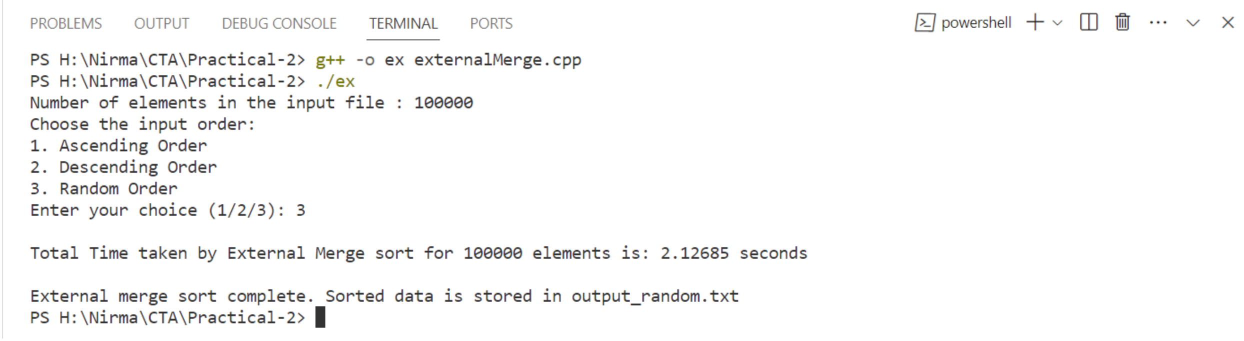
input_descending.txt

```
100000
99999
99998
99997
99996
99995
99994
99993
99992
99991
99990
99989
99988
99987
99986
99985
99984
99983
99982
99981
99980
99979
99978
99977
```

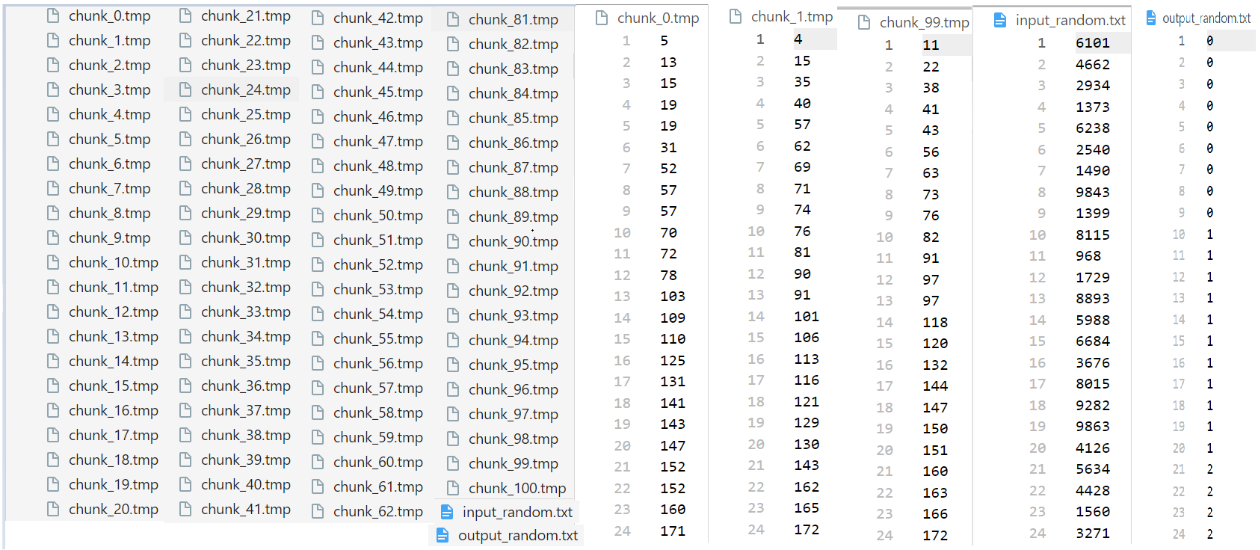
output_descending.txt

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

Random Order



```
PS H:\Nirma\CTA\Practical-2> g++ -o ex externalMerge.cpp
PS H:\Nirma\CTA\Practical-2> ./ex
Number of elements in the input file : 100000
Choose the input order:
1. Ascending Order
2. Descending Order
3. Random Order
Enter your choice (1/2/3): 3

Total Time taken by External Merge sort for 100000 elements is: 2.12685 seconds

External merge sort complete. Sorted data is stored in output_random.txt
PS H:\Nirma\CTA\Practical-2>
```



```
chunk_0.tmp
chunk_1.tmp
chunk_2.tmp
chunk_3.tmp
chunk_4.tmp
chunk_5.tmp
chunk_6.tmp
chunk_7.tmp
chunk_8.tmp
chunk_9.tmp
chunk_10.tmp
chunk_11.tmp
chunk_12.tmp
chunk_13.tmp
chunk_14.tmp
chunk_15.tmp
chunk_16.tmp
chunk_17.tmp
chunk_18.tmp
chunk_19.tmp
chunk_20.tmp
chunk_21.tmp
chunk_22.tmp
chunk_23.tmp
chunk_24.tmp
chunk_25.tmp
chunk_26.tmp
chunk_27.tmp
chunk_28.tmp
chunk_29.tmp
chunk_30.tmp
chunk_31.tmp
chunk_32.tmp
chunk_33.tmp
chunk_34.tmp
chunk_35.tmp
chunk_36.tmp
chunk_37.tmp
chunk_38.tmp
chunk_39.tmp
chunk_40.tmp
chunk_41.tmp
chunk_42.tmp
chunk_43.tmp
chunk_44.tmp
chunk_45.tmp
chunk_46.tmp
chunk_47.tmp
chunk_48.tmp
chunk_49.tmp
chunk_50.tmp
chunk_51.tmp
chunk_52.tmp
chunk_53.tmp
chunk_54.tmp
chunk_55.tmp
chunk_56.tmp
chunk_57.tmp
chunk_58.tmp
chunk_59.tmp
chunk_60.tmp
chunk_61.tmp
chunk_62.tmp
chunk_81.tmp
chunk_82.tmp
chunk_83.tmp
chunk_84.tmp
chunk_85.tmp
chunk_86.tmp
chunk_87.tmp
chunk_88.tmp
chunk_89.tmp
chunk_90.tmp
chunk_91.tmp
chunk_92.tmp
chunk_93.tmp
chunk_94.tmp
chunk_95.tmp
chunk_96.tmp
chunk_97.tmp
chunk_98.tmp
chunk_99.tmp
chunk_100.tmp
input_random.txt
output_random.txt
```

chunk_0.tmp

```
5
13
15
19
19
31
52
57
57
70
72
78
103
109
110
125
131
141
143
147
152
152
160
171
```

chunk_1.tmp

```
4
15
35
40
57
62
69
71
74
76
81
90
91
101
106
113
116
121
129
130
143
162
165
172
```

chunk_99.tmp

```
11
22
38
41
43
56
63
73
76
82
91
97
97
118
120
132
144
147
150
151
160
163
166
172
```

input_random.txt

```
6101
4662
2934
1373
6238
2540
1490
9843
1399
8115
968
1729
8893
5988
6684
3676
8015
9282
9863
4126
5634
4428
1560
3271
```

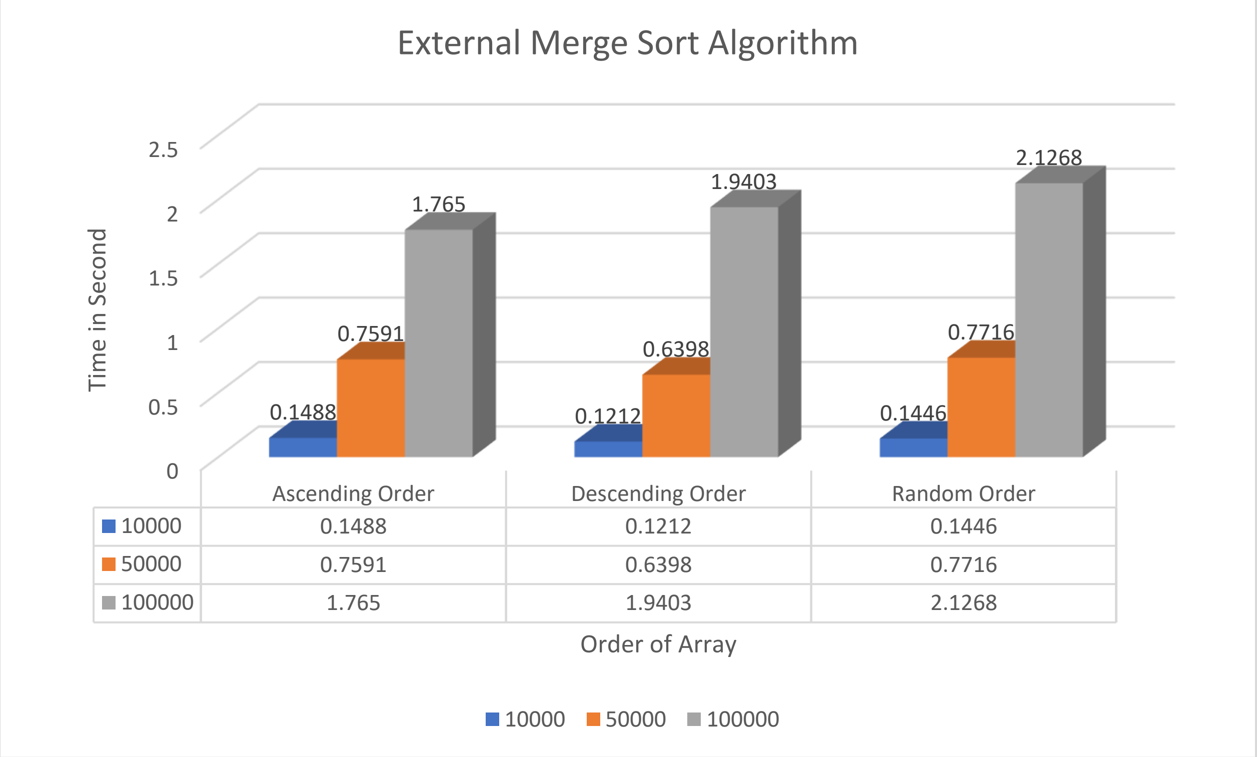
output_random.txt

```
0
0
0
0
0
0
0
0
0
1
1
1
1
1
1
1
1
1
1
1
2
2
2
2
```

- **Graphs & Output Data**

Here are the One tables that describes the output generated by the above code for ascending, descending and random array for size 10000, 50000 & 100000.

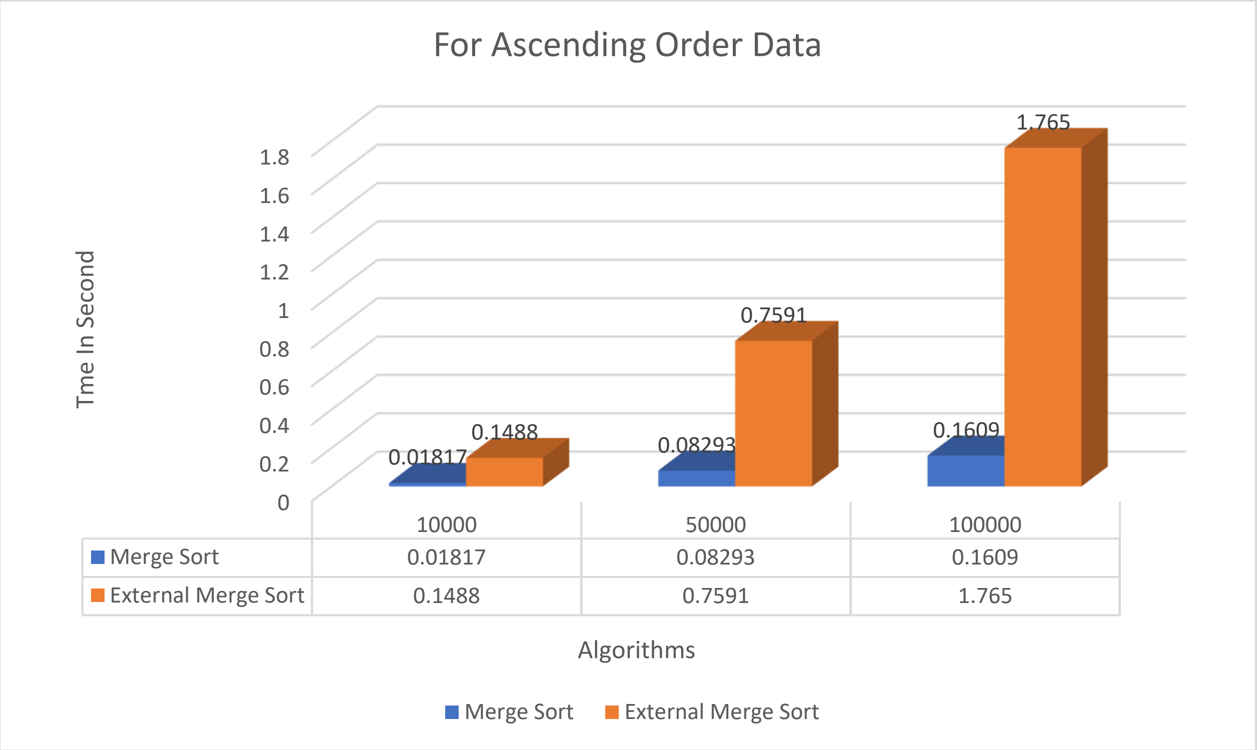
	Ascending Order	Descending Order	Random Order
10000	0.1488	0.1212	0.1446
50000	0.7591	0.6398	0.7716
100000	1.765	1.9403	2.1268

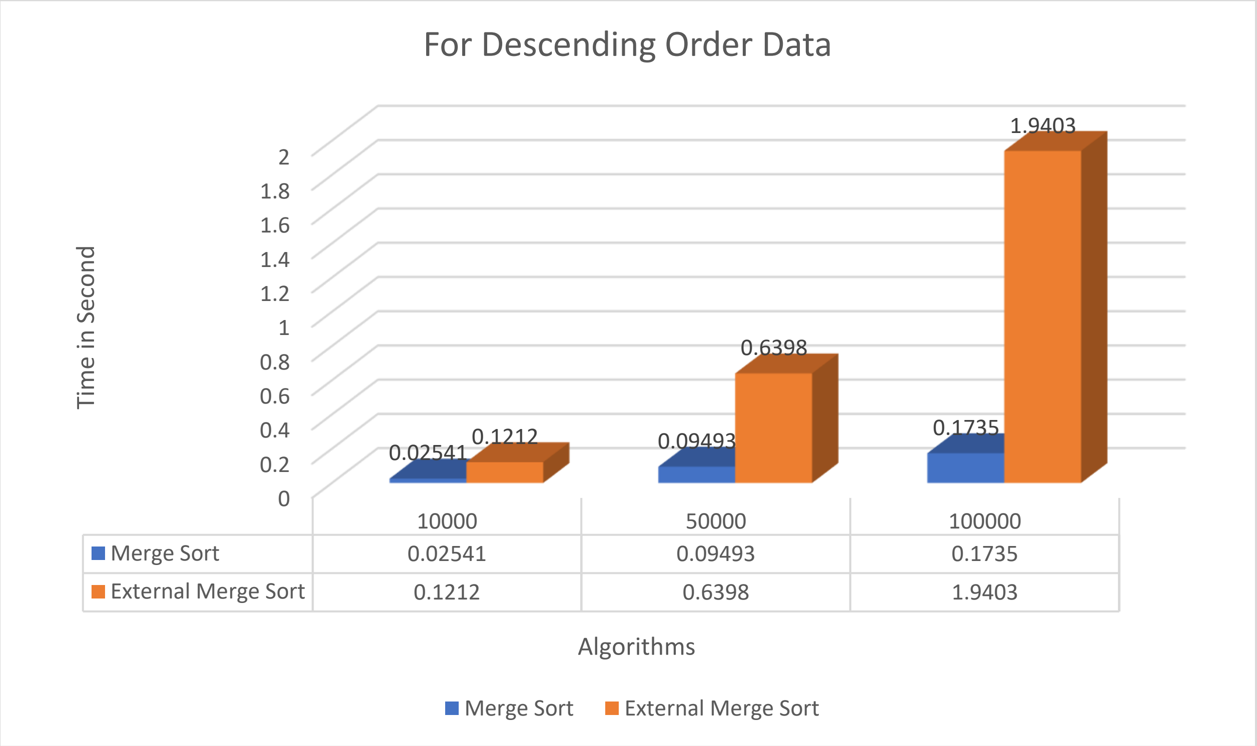


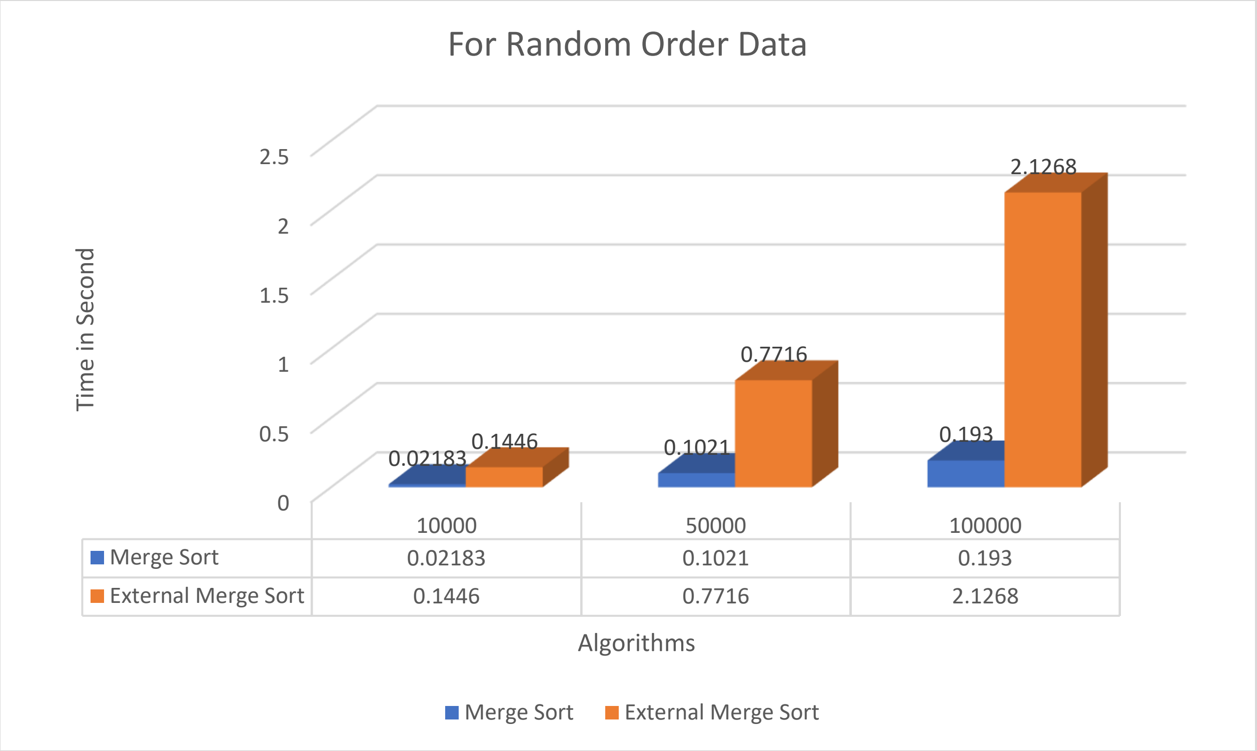
- **Analysis**

From above code the conclusion is as per the order the i/o operations are increases with the array size and it also depends on the order of an array if array is sorted so the no of i/o operations are less then the descending and random order data.

- **Comparison of Merge Sort and External Merge Sort**







	10000	50000	100000
🟦 Merge Sort	0.02183	0.1021	0.193
🟧 External Merge Sort	0.1446	0.7716	2.1268

- Conclusion

Merge sort is used for the in-memory data which can easily fit into the RAM of our Hardware device wherein there are large dataset which can not fit into the main memory so in such cases we use the external merge sort where the data is divided into chunks and sorted that into memory and then merge them. In summary, if you have smaller dataset which can easily fit into main memory then merge sort is efficient but if you have larger data set then external merge sort is efficient.

External merge sort is slower than in memory algorithm like merge sort because read and write operations to external storage to access the elements from file and it is also dependent on the hardware specifications.