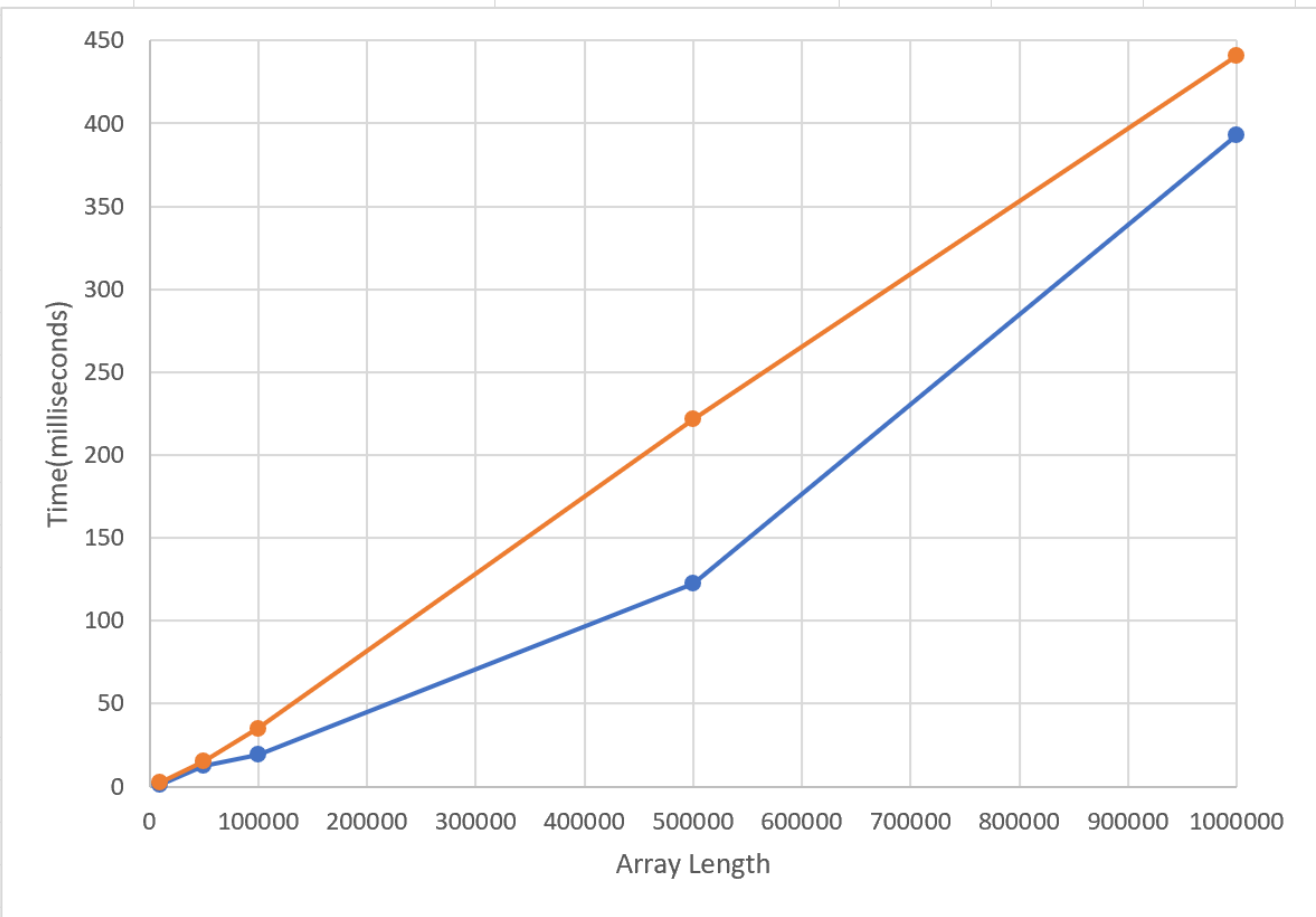
| array length | non - recursive merge sort | recursive merge sort |
|---|---|---|
| 10000 | 1.2 | 2.4 |
| 50000 | 12.8 | 15 |
| 100000 | 19.2 | 35.1 |
| 500000 | 122.2 | 221.4 |
| 1000000 | 393.2 | 441 |



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

void array_display(int array[], int size) {
    for (int i = 0; i < size; i++) {
        cout << array[i] << " ";
    }
    cout << endl;
}

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

int min(int x, int y) { return (x<y)? x :y; }

void mergeSort(int arr[], int n)
{
   int curr_size;
   int left_start;

   for (curr_size=1; curr_size<=n-1; curr_size = 2*curr_size)
   {
       for (left start=0; left start<n-1; left start += 2*curr size)
       {
           int mid = min(left_start + curr_size - 1, n-1);

           int right_end = min(left_start + 2*curr_size - 1, n-1);

           merge(arr, left_start, mid, right_end);
       }
   }
}

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

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        cout <<" "<< A[i];
    cout <<"\n";
}

int main()
{
    srand(time(NULL));
    int size = 10;
    int array[size];

    for (int i = 0; i < size; i++) {
        array[i] = rand() % 100;
    }

    //int size = sizeof(array) / sizeof(array[0]);
    array_display(array, size);

    auto start = high_resolution_clock::now();
    mergeSort(array, size);

    auto end = high_resolution_clock::now();

    array_display(array, size);

    auto elapsed_time_ms = duration_cast<milliseconds>(end - start);
    cout << "Time taken by mergeSort() in milliseconds: " << elapsed_time_ms.count() <<
endl;
    return 0;
}
```

```cpp
#include <iostream>
#include <chrono>
#include <cstdlib>
#include <ctime>
using namespace std;
using namespace std::chrono;

void array_display(int array[], int size) {
    for (int i = 0; i < size; i++) {
        cout << array[i] << " ";
    }
    cout << endl;
}

void merge(int array[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int L[n1], R[n2];

    for (int i = 0; i < n1; i++) {
        L[i] = array[left + i];
    }
    for (int j = 0; j < n2; j++) {
        R[j] = array[mid + 1 + j];
    }

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            array[k++] = L[i++];
        } else {
            array[k++] = R[j++];
        }
    }
    while (i < n1) {
        array[k++] = L[i++];
    }
    while (j < n2) {
        array[k++] = R[j++];
    }
}

void mergeSort(int array[], int left, int right) {
    if (left >= right) {
        return;
    }
    int mid = left + (right - left) / 2;
    mergeSort(array, left, mid);
    mergeSort(array, mid + 1, right);
    merge(array, left, mid, right);
}

int main() {

    srand(time(NULL));
    int size = 10000;
    int array[size];

    for (int i = 0; i < size; i++) {
        array[i] = rand() % 100;
    }

    //int size = sizeof(array) / sizeof(array[0]);
    array_display(array, size);

    auto start = high_resolution_clock::now();
    mergeSort(array, 0, size - 1);
    auto end = high_resolution_clock::now();

    array_display(array, size);

    auto elapsed_time_ms = duration_cast<milliseconds>(end - start);
    cout << "Time taken by mergeSort() in milliseconds: " << elapsed_time_ms.count() <<
endl;
    return 0;
}
```