

C: graph.cpp X

C: graph.cpp &gt; main()

```
1 #include <iostream>
2 #include <chrono>
3 #include <vector>
4 #include <cstdlib>
5 #include "matplotlibcpp.h"
6
7 namespace plt = matplotlibcpp;
8 using namespace std;
9 using namespace chrono;
10
11
12 void merge(int A[], int start, int mid1, int mid2, int end) {
13     int n1 = mid1 - start + 1;
14     int n2 = end - mid2 + 1;
15
16     int Left[n1];
17     int Right[n2];
18
19     for (int i = 0; i < n1; i++)
20         Left[i] = A[start + i];
21     for (int i = 0; i < n2; i++)
22         Right[i] = A[mid2 + i];
23
24     Left[n1] = INT_MAX;
25     Right[n2] = INT_MAX;
26
27     int i = 0;
28     int j = 0;
29
30     for (int k = start; k <= end; k++) {
31         if (Left[i] <= Right[j]) {
32             A[k] = Left[i];
33             i++;
34         } else {
35             A[k] = Right[j];
36             j++;
37         }
38     }
39 }
40
41 void mergerSort(int arr[], int n){
42     int len = 1;
43     while (len < n){
44         int i = 0;
45         while(i < n){
46             int L1 = i;
47             int R1 = i + len - 1;
48             int L2 = i + len;
49             int R2 = i + 2*len - 1;
50             if (L2 >= n){
51                 break;
52             }
53             if (R2 >= n){
54                 R2 = n - 1;
55             }
56             merge(arr, L1, R1, L2, R2);
57             i = i + 2 * len;
58         }
59         len = 2 * len;
60     }
61 }
62
63 void merge(int A[], int start, int mid, int end) {
64     int n1 = mid - start + 1;
65     int n2 = end - mid;
66
67     int Left[n1];
68     int Right[n2];
69
70     for (int i = 0; i < n1; i++)
71         Left[i] = A[start + i];
72     for (int i = 0; i < n2; i++)
73         Right[i] = A[mid + 1 + i];
74
75     Left[n1] = INT_MAX;
76     Right[n2] = INT_MAX;
77
78     int i = 0;
79     int j = 0;
80
81     for (int k = start; k <= end; k++) {
82         if (Left[i] <= Right[j]) {
83             A[k] = Left[i];
84             i++;
85         } else {
86             A[k] = Right[j];
87             j++;
88         }
89     }
90
91 void mergerSort(int A[], int start, int end) {
92     if (start < end) {
93         int mid = (start + end) / 2;
94         mergerSort(A, start, mid);
95         mergerSort(A, mid + 1, end);
96         merge(A, start, mid, end);
97     }
98 }
99
```

```

100 int main(){
101
102     vector<double> avg_times_1;
103     double avg;
104     for(vector<vector<int> > a : sample){
105         double time_sum;
106         for(vector<int> b : a){
107             vector<int> vector_array = b;
108             int n = vector_array.size();
109             int arr[n];
110             for (int i = 0; i < n; i++){
111                 arr[i] = vector_array[i];
112             }
113             auto start = steady_clock::now();
114             ///////////////
115
116             mergerSort(arr, 0, sizeof(arr) / sizeof(int) - 1);
117
118             ///////////////
119             auto end = steady_clock::now();
120             auto diff = duration_cast<duration<double> >(end - start);
121             double durationInSeconds = diff.count();
122             time_sum += durationInSeconds;
123         }
124         avg = time_sum / 5;
125         avg_times_1.push_back(avg);
126     }
127
128     vector<double> avg_times_2;
129     for(vector<vector<int> > a : sample){
130         double time_sum;
131         for(vector<int> b : a){
132             vector<int> vector_array = b;
133             int n = vector_array.size();
134             int arr[n];
135             for (int i = 0; i < n; i++){
136                 arr[i] = vector_array[i];
137             }
138             auto start = steady_clock::now();
139             ///////////////
140
141             mergerSort(arr, n);
142
143             ///////////////
144             auto end = steady_clock::now();
145             auto diff = duration_cast<duration<double> >(end - start);
146             double durationInSeconds = diff.count();
147             time_sum += durationInSeconds;
148         }
149         avg = time_sum / 5;
150         avg_times_2.push_back(avg);
151     }
152
153     vector<int> input_size(49-3);
154     for(int i = 3; i < 50; i++){
155         input_size.push_back(i);
156     }
157
158     // Plot the data
159     plt::plot(input_size, avg_times_1);
160
161     plt::plot(input_size, avg_times_2);
162
163     // Set plot labels
164     plt::xlabel("Input-Array-Size");
165     plt::ylabel("Time-taken");
166     plt::title("Time Complexity of Recursive and Iterative Merge Sort");
167
168     // Save the plot to a file
169     plt::save("plot.png");
170
171     // Show the plot
172     plt::show();
173
174     return 0;
175 }
176
177
178
179

```

Time Complexity of Recursive and Iterative Merge Sort

