```
    ∨s_code

              € graph.cpp ×
                                                                                                                                                                                                                                                                                                                                                                                                                          ▷ ∨ 🕲 🎞 …
                 G graph.cpp > 分 main()
                             ph.6pp / @ manit)
#include <iostream>
#include <chrono>
#include <vector>
#include <cstdlib>
#include "matplotlibcpp.h"
                              namespace plt = matplotlibcpp;
using namespace std;
using namespace chrono;
<del>LL</del>
                               void merge(int A[), int start, int mid1, int mid2, int end) {
   int n1 = mid1 - start + 1;
   int n2 = end - mid2 + 1;
                                        int Right[n2];
                                       for (int i = 0; i < n1; i++)

Left[i] = A[start + i];

for (int i = 0; i < n2; i++)

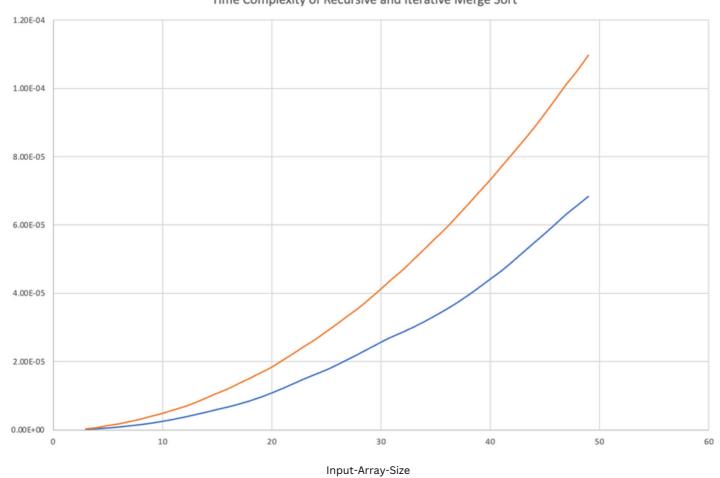
Right[i] = A[mid2 + i];
                                       Left[n1] = INT_MAX;
Right[n2] = INT_MAX;
                                        int i = 0;
int j = 0;
                                        for (int k = start; k <= end; k++) {
    if(Left[i] <= Right[j]) {
        A[k] = Left[i];
}</pre>
                                                 } else {
    A[k] = Right[j];
    j++;
@
0
                              hale(1 < n)/{
   int L1 = i;
   int R1 = i + len - 1;
   int L2 = i + len;
   int R2 = i + 2*len - 1;
   if (L2 >= n){
      break;
   }
}
H
                                                                 R2 = n - 1;
                                                          merge(arr, L1, R1, L2, R2);
i = i + 2 * len;
                               void merge(int A[], int start, int mid, int end) {
   int n1 = mid - start + 1;
   int n2 = end - mid;
                                       for (int i = 0; i < n1; i++)
  Left[i] = A[start + i];
for (int i = 0; i < n2; i++)
  Right[i] = A[mid + 1 + i];</pre>
                                        Left[n1] = INT_MAX;
Right[n2] = INT_MAX;
                                        for (int k = start; k <= end; k++) {
    if(Left[i] <= Right[j]) {
        A[k] = Left[i];
        i++;
    } else {
        A[k] = Right[j];
    }
}</pre>
<del>u</del>
                               void mergerSort(int A[], int start, int end) {
                                      if (start < end) {
   int mid = (start + end) / 2;
   mergerSort(A, start, mid);
   mergerSort(A, mid + 1, end);
   merge(A, start, mid, end);</pre>
```

```
double avg;
for(vector<vector<int> > a : sample){
    double time_sum;
    for(vector<int> b : a){
        vector<int> vector_array = b;
}
                                                    vector<ant> vector<array = 0;
int n =vector<array.size();
int arr[n];
for (int i = 0; i < n; i++){
    arr[i] = vector_array[i];
}</pre>
H
                                                     mergerSort(arr, 0, sizeof(arr) / sizeof(int) - 1); |
                                                    auto end = steady_clock::now();
auto diff = duration_cast<duration<double> >(end - start);
                                                    double durationInSeconds = diff.count();
time_sum += durationInSeconds;
                                            avg = time_sum / 5;
avg_times_1.push_back(avg);
                                    vector<double> avg_times_2;
for(vector<vector<int> > a : sample){
   double time_sum;
   for(vector<int> b : a){
                                                    int arr[n];

for (int i = 0; i < n; i++){

    arr[i] = vector_array[i];
8
                                                     mergerSort(arr, n);
                                                    auto end = steady_clock::now();
auto diff = duration_cast<duration<double> >(end - start);
                                                     double durationInSeconds = diff.count();
time_sum += durationInSeconds;
                                            avg = time_sum / 5;
avg_times_2.push_back(avg);
                                    vector<int> input_size(49-3);
for(int i = 3; i < 50; i++){
   input_size.push_back(i);</pre>
                                    // Set plot labels
plt::xlabel("Input-Array-Size");
plt::ylabel("Time-taken");
plt::title("Time Complexity of Recursive and Iterative Merge Sort");
<del>L</del>
                                    // Save the plot to a file
plt::save("plot.png");
                                    // Show the plot
plt::show();
8
```

Time Complexity of Recursive and Iterative Merge Sort



Time--taken