#include <iostream>  
#include <chrono>  
using namespace std;  
using namespace std::chrono;  
  
template <typename T>  
  
T QuickSort(T arr, int left, int right) {  
 int pivot, i = left, j = right;  
  
 if (left >= right)  
 return arr;  
  
 pivot = (left + right) / 2;  
  
 while (i < j) {  
 while (i < pivot && arr[i] <= arr[pivot])  
 i++;  
 while (j > pivot && arr[j] >= arr[pivot])  
 j--;  
  
 if (i < j) {  
 swap(arr[j], arr[i]);  
 if (i == pivot)  
 pivot = j;  
 else if (j == pivot)  
 pivot = i;  
 }  
 }  
 QuickSort(arr, left, pivot);  
 QuickSort(arr, pivot + 1, right);  
}  
  
  
int main() {  
 using std::chrono::high\_resolution\_clock;  
 using std::chrono::duration\_cast;  
 using std::chrono::duration;  
 using std::chrono::milliseconds;  
  
  
 int n;  
 cout << "Enter size of Array: ";  
 cin >> n;  
 double arr[n];  
 for (int i = 0; i < n; i++)  
 cin >> arr[i];  
  
 int length = n;  
  
 auto start = high\_resolution\_clock::now();  
 QuickSort(arr, 0, length - 1);  
  
 for (int i = 0; i < length; i++)  
 cout << arr[i] << " ";  
  
  
 auto stop = high\_resolution\_clock::now();  
 duration<double, std::milli> ms\_double = stop - start;  
 cout << endl << ms\_double.count() << " ms" << endl;  
  
 return 0;  
}

QuickSort is a divide and conquer algorithm and that is why it uses binary technique. It divides array into two left and right parts. As we know it takes log n times to divide. Then, we multiply it by n where n is a size of array;

Overall Time Complexity: O(n log n);

The worst case time complexity of a typical implementation of QuickSort is O(n2). The worst case occurs when the picked pivot is always an extreme (smallest or largest) element.