```cpp
#include <iostream>
#include <fstream>
#include <ctime>
#include <cstdlib>
#include <iomanip>

using namespace std;

const int ARR_SIZE = 1000;
const int ID_MIN = 1000;
const int ID_MAX = 99999;

//class definition
class Student {
    public:
        Student(int sid = 0, double sgpa = 0.0){
            id = sid;
            gpa = sgpa;
        }
        int id;
        double gpa;
};

void display(Student array[]){
    for (int index = 0; index < ARR_SIZE; index++){
        cout << array[index].id << ":" << array[index].gpa << " ";
    }
}
//bubble sort
void sortBubble(Student array[]){
    for (int pass = 0; pass < ARR_SIZE; pass++){
        for (int index = 0; index < ARR_SIZE - pass; index++){
            // check if element at index is less than element at
            // index + 1
            if (array[index].id > array[index + 1].id){
                // perform swap
                int temp = array[index].id;
                double tempGpa = array[index].gpa;
                array[index].id = array[index + 1].id;
                array[index].gpa = array[index + 1].gpa;
                array[index + 1].id = temp;
                array[index + 1].gpa = tempGpa;
            }
        }
    }
}
//swap two elements in the array
void swapValues(Student &student1, Student &student2){
    int temp = student1.id;
    double tempGpa = student1.gpa;
    student1.id = student2.id;
    student1.gpa = student2.gpa;
    student2.id = temp;
    student2.gpa = tempGpa;
}

//move element that are less than the pivot to the left
//and move element greater than the pivot to the right
int qPartition(Student array[], int lowIndex, int highIndex){
    int pivotItem = array[highIndex].id;

    // index of the place where element will be placed
    int putIndex = lowIndex - 1;

    for (int currentIndex = lowIndex; currentIndex < highIndex; currentIndex++){
        // if element at currentIndex is less than the pivotItem, move it to left
```

```
side
 67          if (array[currentIndex].id <= pivotItem){
 68              putIndex++;
 69              swapValues(array[currentIndex], array[putIndex]);
 70          }
 71      }
 72
 73      // once the loop completes, move the pivotItem to its right place
 74      swapValues(array[highIndex], array[putIndex + 1]);
 75
 76      // return the index of pivotItem
 77      return putIndex + 1;
 78  }
 79
 80  //quicksort
 81  void qsort(Student array[], int lowIndex, int highIndex){
 82      if (lowIndex < highIndex){
 83          int partitionIndex = qPartition(array, lowIndex, highIndex);
 84
 85          // sort the left and right side of partitionIndex
 86          qsort(array, lowIndex, partitionIndex - 1);
 87          qsort(array, partitionIndex + 1, highIndex);
 88      }
 89  }
 90
 91  int main(){
 92      // create an array of 1M elements
 93      Student students[ARR_SIZE];
 94      // fill with random values: id = ID MIN - ID MAX)
 95      // initialize the seed
 96      srand(time(0));
 97
 98      for (int index = 0; index < ARR_SIZE; index++){
 99          int id = ID_MIN + (rand() % (ID_MAX - ID_MIN + 1));
100          double gpa = (rand() % 41) / 10.0;
101
102          students[index] = Student(id, gpa);
103      }
104
105      // display unsorted only for smaller inputs
106      //display(students);
107
108      cout << "Choose:\n1 = bubble sort\n2 = qsort\n";
109      int choice;
110      cin >> choice;
111      clock_t start_time, end_time;
112
113      // start measuring time
114      start_time = clock();
115
116      // sort by selected algorithm
117      if (choice == 1){
118          sortBubble(students);
119      }
120      else if(choice == 2){
121          qsort(students, 0, ARR_SIZE - 1);
122      }
123
124      // stop measuring time
125      end_time = clock() - start_time;
126
127      // display sorted array only for smaller inputs
128      //display(students);
129
130      // display table
131      int total_seconds = ((double)end_time / CLOCKS_PER_SEC);
```

```
132
133       // display the resultant time
134       cout << "Total time taken: " << total_seconds << endl;
135   }
```